AVR. Учебный курс. Операционная система. Установка

AVR. Учебный курс | 10 Июль 2008 | DI HALT | 23 Comments

Ядро у нас есть, теперь осталось это все хозяйство запихать на МК. Для этого всего лишь надо рассовать нужные части кода в исходник. Показывать буду на примере ATmega8. Для других МК разница минимальная. Может быть с таймером что нибудь помудрить придется, но не более того.

Например, недавно, вкорячивал ту же схему на ATmega168, так пришлось подправить инициализацию таймера — регистры там зовутся по другому. Пришлось изменить макрос OUTI — так как многие привычные уже регистры перестали загружаться через комадну OUT — выпали из диапазона, только через LDS/STS ну и, собственно, все хлопоты. Потратил минут 20 на переименование регистров и заработало.

Итак. Есть у нас совершенно новый пустой файл NewMega8-rtos.asm

```
; Добавляем в него первым же делом инклюдник восьмой меги:

                .include "m8def.inc"    ; Используем ATMega8

; Следом я добавляю файл с моими макроопределениями в котором записаны все символические имена
; для ресурсов, вроде регистров, портов, отдельных пинов. ИМХО удобней все держать по разным
; файлам, но тут уже дело за вашими привычками.

                .include "define.asm"

;Потом файл с макросами, он тоже отдельный и кочует из программы в программу. Именно там всякие
; левые самодельные команды вроде OUTI прописаны.

                .include "macro.asm"    ; Все макросы у нас тут

; Следом идет файл макросов ядра ОС. Он должен располагаться в начале программы, иначе компилятор
; не поймет. Именно там прописаны макросы таймерной службы, добавления задачи и таймера, там же
; заныкан стандартный макрос инциализации UART и многое другое.

                .include "kernel_macro.asm"

;Дальше прописывается сегмент оперативной памяти в котором заранее определены
; все очереди задач и таймеров.
```

```
                .DSEG
                .equ    TaskQueueSize = 11      ; Размер очереди событий
TaskQueue:      .byte   TaskQueueSize           ; Адрес очереди сотытий в SRAM
                .equ    TimersPoolSize = 5      ; Количество таймеров
TimersPool:     .byte   TimersPoolSize*3        ; Адреса информации о таймерах

; Следом уже идет код, начинается кодовый сегмент. Надо заметить, что у меня вся таблица
; Прерываний спрятана в vectors.asm и вместо таблицы в коде .include "vectors.asm" это удобно

                .CSEG
                .ORG    0x0000          ; Проц стартует с нуля, но дальше идут вектора
                RJMP    Reset

                .ORG    INT0addr        ; External Interrupt Request 0
                RETI
                .ORG    INT1addr        ; External Interrupt Request 1
                RETI

        .ORG    OC2addr                 ; Timer/Counter2 Compare Match
        RJMP    OutComp2Int             ;<<<<<<<< Прерывание ОС!!!

                .ORG    OVF2addr        ; Timer/Counter2 Overflow
                RETI
                .ORG    ICP1addr        ; Timer/Counter1 Capture Event
                RETI
                .ORG    OC1Aaddr        ; Timer/Counter1 Compare Match A
                RETI
                .ORG    OC1Baddr        ; Timer/Counter1 Compare Match B
                RETI
```

```
53                  .ORG    OVF1addr        ; Timer/Counter1 Overflow
54                  RETI
55                  .ORG    OVF0addr        ; Timer/Counter0 Overflow
56                  RETI
57                  .ORG    SPIaddr         ; Serial Transfer Complete
58                  RETI
59                  .ORG    URXCaddr        ; USART, Rx Complete
60                  RJMP    Uart_RCV
61                  .ORG    UDREaddr        ; USART Data Register Empty
62                  RETI
63                  .ORG    UTXCaddr        ; USART, Tx Complete
64                  RJMP    Uart_TMT
65                  .ORG    ADCCaddr        ; ADC Conversion Complete
66                  RETI
67                  .ORG    ERDYaddr        ; EEPROM Ready
68                  RETI
69                  .ORG    ACIaddr         ; Analog Comparator
70                  RETI
71                  .ORG    TWIaddr         ; 2-wire Serial Interface
72                  RETI
73                  .ORG    SPMRaddr        ; Store Program Memory Ready
74                  RETI
75                  .ORG    INT_VECTORS_SIZE        ; Конец таблицы прерываний
```

После таблицы векторов идут обработчики прерываний. Они короткие, поэтому их размещаю в начале. Первым же обработчиком идет обработчик прерывания от таймера на котором висит таймерная служба ОС. Под таймерную службу желательно отдать самый стремный таймер, на который не завязана ШИМ или еще какая полезная служба. В идеале бы под это дело пустить Timer0, как самый лоховский. Но он

не умеет считать от 0 до регистра сравнения, только от нуля до 255, впрочем, в обработчик прерывания можно добавить предварительную загрузку таймера0 нужным значением и не расходовать более навороченный таймер. Мне было лень, я повесил все на Таймер2, а в регистр сравнения прописал такое значение, чтобы прерывание было ровно один раз в 1мс. Разумеется, выставив предварительно нужный делитель.

```
; Interrupts procs
; Output Compare 2 interrupt  - прерывание по совпадению TCNT2 и OCR2
; Main Timer Service - Служба Таймеров Ядра - Обработчик прерывания

OutComp2Int:    TimerService    ; Служба таймера OS
                                ; Весь код обработчика в виде одного макроса
                                ; Просто вставил и все. Куда угодно. Можно извратиться
                                ; Подать импульсы с нужной частотой на какой-нибудь
                                ; INT0 и службу таймеров повесить на его прерывание
                                ; Разумеется, в таблице векторов из вектора  прописан
                                ;переход сюда
                RETI            ; выходим из прерывания

Uart_RCV:       RETI            ; Другие прерывания если нужны
Uart_TMT:       RETI
```

То все было обязательной подготовкой и разметкой адресов и памяти, а вот тут уже начинается сама программа. Именно отсюда стартует проц. Вписываем следующий код:

```
Reset:          OUTI    SPL,low(RAMEND)             ; Первым делом инициализируем стек
                OUTI    SPH,High(RAMEND)
```

Все инициализации у меня спрятана в **.include «init.asm»**, но тут я распишу ее полностью.

```
; init.asm
; Очистка памяти
RAM_Flush:      LDI     ZL,Low(SRAM_START)
                LDI     ZH,High(SRAM_START)
                CLR     R16
Flush:          ST      Z+,R16
                CPI     ZH,High(RAMEND)
                BRNE    Flush

                CPI     ZL,Low(RAMEND)
                BRNE    Flush

                CLR     ZL
                CLR     ZH

; Init RTOS                     ; В исходнике все сделано
;               INIT_RTOS       ; вот так вот, одним макросом. Макрос описан
;                               ; в файле kernel_macro.asm
;                               ; Но я распишу тут подробно. Там идет настройка
;                               ; Таймера в работу

; Содержимое макроса INIT_RTOS

                OUTI    SREG, 0         ; Сброс всех флагов
```

```
25
26                  rcall ClearTimers       ; Очистить список таймеров РТОС
27                  rcall ClearTaskQueue    ; Очистить очередь событий РТОС
28                  sei                     ; Разрешить обработку прерываний
29
30  ; Настройка таймера 2 - Основной таймер для ядра
31
32                  .equ    MainClock       = 8000000               ; CPU Clock
33                  .equ    TimerDivider    = MainClock/64/1000     ; 1 mS
34
35                  OUTI    TCCR2,1<<CTC2|4<<CS20   ; Установить режим CTC и предделитель =64
36                  OUTI    TCNT2,0                 ; Установить начальное значение счётчиков
37
38
39                  ldi OSRG,low(TimerDivider)
40                  out OCR2,OSRG                   ; Установить значение в регистр сравнения
41  ; Конец макроса INIT_RTOS
42
43
44                  OUTI    TIMSK,1<<OCF2           ; Разрешить прерывание по сравнению
45
46  ; Инициализация остальной периферии
47                  USART_INIT
48  ; Конец init.asm
```

После инициализации идет секция запуска фоновых приложений. Добавим пока меточку про запас, с нас не убудет.

```
1
```

```
Background:     NOP             ; Пока тут ничего нет
```

Главный цикл. Его надо скопировать без изменений, как есть.

```
1
2
3
4
5
6
7
8
```

```
Main:   SEI                             ; Разрешаем прерывания.
        wdr                             ; Reset Watch DOG
        rcall   ProcessTaskQueue        ; Обработка очереди процессов
        rcall   Idle                    ; Простой Ядра
        rjmp    Main                    ; Основной цикл микроядра РТОС

; В Idle можно сунуть что нибудь простое, быстрое и некритичное.
; Но я обычно оставляю его пустым.
```

После главного цикла вставляется шаблон под секцию задач. Именно сюда вписывается наш исполняемый код. Тут творится самое интересное

```
1
2
3
4
5
6
7
```

```
Idle:           RET
;-----------------------------------------------------------------------
Task1:          RET
;-----------------------------------------------------------------------
Task2:          RET
;-----------------------------------------------------------------------
Task3:          RET
```

```
;-------------------------------------------------------------------------
Task4:          RET
;-------------------------------------------------------------------------
Task5:          RET
;-------------------------------------------------------------------------
Task6:          RET
;-------------------------------------------------------------------------
Task7:          RET
;-------------------------------------------------------------------------
Task8:          RET
;-------------------------------------------------------------------------
Task9:          RET

; А после секции задач вставляем шаблонную таблицу переходов и код ядра
                .include "kerneldef.asm"        ; Подключаем настройки ядра
                .include "kernel.asm"           ; Подклчюаем ядро ОС

TaskProcs:      .dw Idle                ; [00]
                .dw Task1               ; [01]
                .dw Task2               ; [02]
                .dw Task3               ; [03]
                .dw Task4               ; [04]
                .dw Task5               ; [05]
                .dw Task6               ; [06]
                .dw Task7               ; [07]
                .dw Task8               ; [08]
                .dw Task9               ; [09]
```

Готово! Можно компилировать, пока это пустой проект. Но ненадолго.

Итак, теперь то же самое, но по пунктам.
Установка AVR OS

- Создаем пустой проект
- Вставляем файлы макроопределений
- Вставляем разметку памяти под очереди задач/таймеров
- Вставляем таблицу векторов прерываний
- Прописываем в таблице векторов прерываний переход на обработчик таймера по переполнению
- Добавляем обработчик прерываний
- Прописываем стартовую метку и инициализацию стека
- Инициализация всего что только можно — портов, периферии, обнуление ОЗУ, обнуление очередей, запуск таймера ОС.
- Добавляем секцию фоновых задач
- Добавляем код главного цикла
- Добавляем шаблонную сетку задач
- Добавляем код ядра и таблицу переходов.
- Пишем наш код
- . . .
- PROFIT

В следующий раз я покажу практический пример работы с этой ОС. В котором будет красочно показано ради чего, собственно, этот геморрой и почему мне он так нравится :)

Assembler AVR RTOS

23 thoughts on “AVR. Учебный курс. Операционная система. Установка”

Medved

9 Апрель 2009 в 21:41

Вот бы поглядеть на этот геморой в деле!!! Очень хочу увидеть!!))))

---

nestandart

9 Апрель 2009 в 21:44

чем таким красивым так красиво исходники в веб оформляешь ?

★ DI HALT

9 Апрель 2009 в 23:01

Плагин для вордпресса

WP-Syntax 0.9.1
Syntax highlighting using GeSHi supporting a wide range of popular languages. От Ryan McGeary.

Syrok

9 Апрель 2009 в 23:43

зачем он такие большие табы делает…
беру свои слова обратно, иначе метки не влезут…

★ DI HALT

9 Апрель 2009 в 23:50

Эт я делаю по три таба от края.

goodic

10 Апрель 2009 в 17:09

для 8-ки код выложить можешь?

★ DI HALT

10 Апрель 2009 в 21:34

На нее и будет

Aspiring

19 Январь 2011 в 16:00

Я извиняюсь, но по моему здесь выставляется предделитель не 64, а 256
CSn2 CSn1 CSn0 Description
0 1 1 clk /64 (From prescaler)
1 0 0 clk /256 (From prescaler)

; Настройка таймера 2 — Основной таймер для ядра
.equ MainClock = 8000000 ; CPU Clock
.equ TimerDivider = MainClock/64/1000 ; 1 mS
OUTI TCCR2,1<<CTC2|4<<CS20 ; Установить режим CTC и предделитель =64 OUTI TCNT2,0 ; Установить начальное значение счётчиков

rumatavz

15 Ноябрь 2011 в 19:06

Тут ошибка
>.equ TimerDivider = MainClock/64/1000 ; 1 mS

Пусть TimerDivider = 2
Тогда будет так
TimerDivider = 0
задержка
TimerDivider = 1
задержка
TimerDivider = 2
задержка
TimerDivider = 0 и прерывание

Те 3 задержки а не 2, как ожидается. Т.о. при малых величинах TimerDivider частота срабатывания таймера будет отличатся от ожидаемого.

---

andykay

23 Ноябрь 2012 в 3:25

Всем привет. Исходя из текста урока я не очень понял, для чего нужна таблица векторов. Если можно — в общих чертах уточните для чайников.

★ DI HALT

23 Ноябрь 2012 в 5:36

При прерывании происходит фиксированный переход по конкретному вектору прерывания, жестко заданны физический адрес, а оттуда мы перенаправляем уже туда, куда нам нужно.

Т.е. проц знает ,что если пришел байт в UART надо перейти в такую то ячейку таблицы векторов, а вот что туда положит программер и куда потом оно перейдет это уже дело программиста.

TideAriel

29 Июль 2013 в 15:59

А почему все подключаемые файлы .asm а не .inc ?

★ DI HALT

29 Июль 2013 в 16:13

А почему они должны быть именно inc? обычно в инк нет исполняемого кода, только дефайны всякие.

TideAriel

29 Июль 2013 в 17:40

ну просто .asm как мне кажется модуль с ассемблерными командами, а вот .inc это заголовочный файл, который просто включает на место директивы .include свое содержимое. Разницы как я понял нет в случае простого включения файла в другой файл, но как то логичней мне кажется .inc использовать.

TideAriel

29 Июль 2013 в 17:43

просто когда я вижу файл с расширением .asm, то мне кажется, что он должен ассемблироваться как отдельный объектный модуль. А когда я вижу файл .inc, то я сразу понимаю, что этот файл просто для вставки в другой файл.

Biggy

2 Август 2013 в 14:04

У меня такая проблема. Скопировал саму PTOC с пинбоарда, у меня как раз мега 16. Но всю свою программу выкинул. Оставил только то, что здесь написано. При компиляции выдает такую лажу

\Sys-RTOS\Sys_RTOS.asm(94): error: Invalid character: ‘ò’ (0xf2)

и таких 59 символов. Ниче не могу понять, что за фигня. Реально ничего нету, а такая фигня.

★ DI HALT

2 Август 2013 в 14:06

эм…. а можешь скриншот студии на этот участок показать? На что она тебе там тычет? Еще рекомендую смотреть выше и ниже что там есть.

Biggy

2 Август 2013 в 14:52

а все нашел, при копипастинге не зацепил ;

Wan-Derer

23 Октябрь 2013 в 12:53

Вот такой пример пришёл на ум.
Предположим, есть ряд из 8 (всего-то) кнопок, из которых собирается байт данных, используемый в программе. Ненажатая кнопа — 1, нажатая — 0. Может быть нажата одна кнопа, несколько или ни одной. Надо бороться с дребезгом. Решение «в лоб» с помощью RTOS:
1. Берём байт с порта
2. Накидываем на него промежуточную маску (по ИЛИ, в нужных битах единички)
3. Анализируем результат, ноль в бите номер X запускает задачу X, которая:
— поднимает бит X в промежуточной маске
— кладёт бит X в выходном байте
— инициирует задачу Y через время успокоения дребезга
4. Идём на начало

Задача Y:
— кладёт бит X в промежуточной маске
— поднимает бит X в выходном байте

Итого надо 8 задач X, 8 задач Y, плюс опрос клавы тоже лучше по прерыванию, плюс поднятие флага готовности выходного байта после того как отработают все X. Слишком жирно или нормально? :)

★ DI HALT

24 Октябрь 2013 в 11:41

Да можно завести просто промежуточную переменную и опрос кнопки делать раз в 10мс, например.

Т.е. считали байт. Сунули в переменную. Вызвали задачу повторно. Даже не проверяя кто там что нажал — считали еще раз через 10мс. Сравнили с переменной — если не изменился. Считаем, что нажатие есть. Анализируем его побитно или кейсом — делаем действия. Все на конечном автомате.

Тут все сильно упрощается, для геймпада, например, такой метод не годится, т.к. быстрое нажатие одновременно нескольких кнопок вызовет эффект дребезга. Ну в этом случае делаешь сравнение на дребезг побитно.

Wan-Derer

24 Октябрь 2013 в 12:37

Тут же какая проблема. Вероятна ситуация когда нажата сначала одна кнопка, а потом — до окончания её дребезга нажимается вторая. Ну хорошо, с кнопками маловероятно, а вот с каким-то устройством с релейными GPO — вполне. Т.е. необходимо контролировать дребезг каждого бита, а не байта в целом: сначала зажат бит1 и мы перестали его опрашивать на время дребезга, но в это время зажат бит2. Потом прошло время дребезга бита1 и мы снова его опрашиваем — но только его т.к. бит2 ещё дребезжит и надо подождать.
У тебя же получается что после нажатия надо какое-то время не опрашивать весь байт…
Впрочем, возможно я загоняюсь :) Проще выставить требование к управляющему ус-ву чтобы оно держало бит зажатым, скажем

200-300 мс, тогда всё становится проще :)
Кстати, каково типичное время дребезга маленьких реле или герконов?

★ DI HALT

24 Октябрь 2013 в 15:02

Крайне маленькое, миллисекунды. На обычной китайской тактовой кнопке я как то хотел увидеть дребезг, чтобы сделать скриншот для статьи и… не увидел. Я выкрутил осциллограф на минимальную временную задержку и там что то телепалось в микровеличинах, так, что уже фронты начали заваливаться. Пришлось для эффектного кадра стучать по кнопке отверткой. :) У реле, а тем пачке герконов, с этим еще лучше.

Celeron

13 Февраль 2014 в 18:07

Смотрите также проект: **«AVRASM: Диспетчер задач RTOS 2.0 (псевдо кооперативная ОС)»**

В котором Автор: отрефакторил код представленного здесь «Диспетчера задач RTOS», оптимизировал и универсализировал, добавил новые фичи, декларировал чёткое API, и опубликовал на GitHub… Фактически, весь код был переписан сызнова, по прототипу DI HALTa.