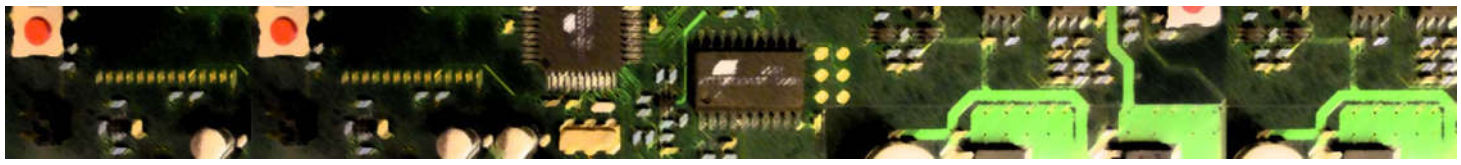

(/)

Разделы

Новости (/index.php/novosti.html)
Встраиваемые системы (/index.php/embedded-programming.html)
Программирование AVR (/index.php/programming-avr.html)
Программирование ARM (/index.php/programmirovanie-arm.html)
Инструменты/ технологии (/index.php/instruments-technologies.html)
Как подключить (/index.php/how-connection.html)
Компоненты (/index.php/electronic-components.html)
RTOS (/index.php/rtos.html)
Софт (/index.php/iar-embedded-workbench.html)
Проекты (/index.php/projects-avr.html)
Ссылки (/index.php/links.html)

1wire (/index.php/programming-avr/itemlist/tag/1wire.html)

arm (/index.php/programming-avr/itemlist/tag/arm.html)

avr (/index.php/programming-avr/itemlist/tag/avr.html)

avr программатор (/index.php/programming-avr/itemlist/tag/avr программатор.html)

ds18b20 (/index.php/programming-avr/itemlist/tag/ds18b20.html)

eeprom (/index.php/programming-avr/itemlist/tag/eeprom.html)

i2c (/index.php/programming-avr/itemlist/tag/i2c.html)

IAR (/index.php/programming-avr/itemlist/tag/IAR.html)

lcd (/index.php/programming-avr/itemlist/tag/lcd.html)

tsop (/index.php/programming-avr/itemlist/tag/tsop.html)

twi (/index.php/programming-avr/itemlist/tag/twi.html)

Программные таймеры. Часть 1

25/10/2009 - 21:50 | Pavel Bobkov (/index.php/programming-avr/itemlist/user/800-pavelbobkov.html)

Исходный код библиотеки

Для наглядности я объединил хедер и сишный файл.

```
//состояния таймера - неработающий, активный, отработавший
enum StateTimer {IDLE, ACTIVE, DONE};

//структура программного таймера
typedef struct{
  unsigned int time;          //через какое время запустить
  unsigned int period;        //период повторения
  enum StateTimer state;      //текущее состояние
  void (*pFunc)(void);        //указатель на функцию
}SoftTimer;

//максимальное число таймеров
#define MAX_TIMERS  4

//число созданных таймеров
unsigned char AmountTimers = 0;

//массив указателей на таймеры
SoftTimer* SoftTimers[MAX_TIMERS];

//функция создания программного таймера
void CreateTimer(SoftTimer *CurSoftTimer, unsigned int time, unsigned int period , enum
StateTimer state, void (*pFunc)(void)){
  SoftTimers[AmountTimers] = CurSoftTimer;
  CurSoftTimer->time = time;
  CurSoftTimer->period = period;
  CurSoftTimer->state = state;
  CurSoftTimer->pFunc = pFunc;
  AmountTimers++;
}

//функция проверки таймеров
void CheckTimer(void){
  for(unsigned char i = 0; i < AmountTimers; i++){
    if(SoftTimers[i]->state == ACTIVE){
      if(SoftTimers[i]->time == 0){
        SoftTimers[i]->pFunc();
        if (SoftTimers[i]->period != 0) SoftTimers[i]->time = (SoftTimers[i]->period-1);
        else SoftTimers[i]->state = DONE;
      }
      else(SoftTimers[i]->time)--;
    }
  }
}
```

Пояснение к коду

Структура программного таймера

Программный таймер должен:
- уметь запускаться сразу или через заданное время
- работать в режиме однократного или переодического запуска
- иметь три состояния – остановлен, активный, закончил работу
- сигнализировать об окончании счета

На основании этих требований определяем структуру **SoftTimer.**

```
//состояния таймера - неработающий, активный, отработавший
enum StateTimer {IDLE, ACTIVE, DONE};

//структура программного таймера
typedef struct{
  unsigned int time;          //через какое время запустить
```

алгоритмы (/index.php/programming-avr/itemlist/tag/алгоритмы.html)
библиотеки (/index.php/programming-avr/itemlist/tag/библиотеки.html)
датчик (/index.php/programming-avr/itemlist/tag/датчик.html)
драйвер (/index.php/programming-avr/itemlist/tag/драйвер.html)
интерфейс (/index.php/programming-avr/itemlist/tag/интерфейс.html)
компоненты (/index.php/programming-avr/itemlist/tag/компоненты.html)
макросы (/index.php/programming-avr/itemlist/tag/макросы.html)
оу (/index.php/programming-avr/itemlist/tag/оу.html)
программирование микроконтроллеров (/index.php/programming-avr/itemlist/tag/программирование микроконтроллеров.html)
расчет (/index.php/programming-avr/itemlist/tag/расчет.html)
семисегментный индикатор (/index.php/programming-avr/itemlist/tag/семисегментный индикатор.html)
си (/index.php/programming-avr/itemlist/tag/си.html)
событийная система (/index.php/programming-avr/itemlist/tag/событийная система.html)
схемотехника (/index.php/programming-avr/itemlist/tag/схемотехника.html)
таймер (/index.php/programming-avr/itemlist/tag/таймер.html)
управление (/index.php/programming-avr/itemlist/tag/управление.html)
устройства (/index.php/programming-avr/itemlist/tag/устройства.html)
учебный курс avr (/index.php/programming-avr/itemlist/tag/учебный курс avr.html)
шим (/index.php/programming-avr/itemlist/tag/шим.html)

```
  unsigned int period;        //период повторения
  enum StateTimer state;      //текущее состояние
  void (*pFunc)(void);        //указатель на функцию
}SoftTimer;
```

Для временнЫх переменных я выбрал тип данных **int**. Это более расточительно в плане ресурсов, но зато позоляет получать временные интервалы большой длительности.

time – определяет через какое время таймер сработает

period – определяет период работы таймера. Если период равен 0, то таймер будет работать в режиме однократного запуска.

state – хранит текущее состояние таймера, она типа enum. Можно было определить режимы работы с помощью define, но так более правильно.

Функция, на которую в структуре определен указатель, будет запускаться по окончании счета таймера.

Массив указателей на таймеры

Объявляем массив указателей на структуры таймеров и переменную для хранения общего числа программных таймеров.

```
//максимальное число таймеров
#define MAX_TIMERS  4

//число созданных таймеров
unsigned char AmountTimers = 0;

//массив указателей на таймеры
SoftTimer* SoftTimers[MAX_TIMERS];
```

Создание таймера

Перед использованием таймера, его нужно создать – объявить переменную типа **SoftTimer** и присвоить полям ее структуры конкретные значения. Инициализацию программного таймера выполняет функция **CreateTimer**.

```
void CreateTimer(SoftTimer *CurSoftTimer, unsigned int time, unsigned int period , enum StateTimer state, void (*pFunc)(void)){
  SoftTimers[AmountTimers] = CurSoftTimer;
  CurSoftTimer->time = time;
  CurSoftTimer->period = period;
  CurSoftTimer->state = state;
  CurSoftTimer->pFunc = pFunc;
  AmountTimers++;
}
```

Функции передаются **-** адрес переменной типа **SoftTimer** и значения ее полей. Внутри функции адрес программного таймера записывается в массив указателей на таймеры. При этом в качестве индекса массива используется переменная **AmountTimers**, значение которой перед выходом из функции увеличивается на единицу. Таким образом указатель на первый таймер запишется в нулевой элемент массива, указатель на следующий таймер запишется во второй элемент...ну и так далее.

Функция опроса таймеров

Функция опроса таймеров запускается в прерывании аппаратного таймера/счетчика

```
//функция проверки таймеров
void CheckTimer(void){
  for(unsigned char i = 0; i < AmountTimers; i++){
    if (SoftTimers[i]->state == ACTIVE){
      if (SoftTimers[i]->time == 0){
        SoftTimers[i]->pFunc();
        if (SoftTimers[i]->period != 0) SoftTimers[i]->time = (SoftTimers[i]->period-1);
        else SoftTimers[i]->state = DONE;
      }
      else(SoftTimers[i]->time)–;
    }
  }
}
```

В цикле for последовательно опрашиваются программные таймеры. Если состояние таймера – активный и счетчик таймера **time** равен нулю **-** запускается его функция. Если таймер периодический, то затем он перезапускается, если нет, ему присваивается состояние **-** отработавший. Если таймер "активный", но его счетчик еще не равен нулю, выполняется декремент счетчика.

Тестовый проект

Итак, необходимый минимум готов. Можно проверить программные таймеры/счетчики в деле. Для этих целей я написал простой проект – 4 программных таймера запускающихся одновременно, но с разной частотой. Функции таймеров инвертируют биты порта B и по этим сигналам легко проверить работу алгоритма, правда для этого нужен осциллограф. Привожу содержимое main.c. **Полная версия проекта для IARa и WINAVR в конце статьи**.

```
//программирование микроконтроллеров AVR на Си
//тестовый проект с программными таймерами
//Pashgan    ChipEnable.ru
```

```
#include <ioavr.h>
#include <intrinsics.h>
#include "Timers.h"
#include "bits_macros.h"

//объявляем переменные типа SoftTimer
SoftTimer timer1;
SoftTimer timer2;
SoftTimer timer3;
SoftTimer timer4;

//расписываем функции программных таймеров
void Clk1(void){
  InvBit(PORTB, 0);
}

void Clk2(void){
  InvBit(PORTB, 1);
}

void Clk3(void){
  InvBit(PORTB, 2);
}

void Clk4(void){
  InvBit(PORTB, 3);
}
```

```
int main( void )
{
  //инициализация порта
  PORTB = 0x00;
  DDRB = 0xff;

  //инициализация таймера T0 - прерывания каждую ms
  TIMSK = (1<<OCIE0);
  TCCR0=(1<<WGM01)|(0<<WGM00)|(0<<COM01)|(0<<COM00)|(0<<CS02)|(1<<CS01)|
(1<<CS00);
  TCNT0 = 0;
  OCR0 = 0x7d;

  //инициализируем таймеры
  CreateTimer(&timer1, 0, 1, ACTIVE, Clk1);
  CreateTimer(&timer2, 0, 2, ACTIVE, Clk2);
  CreateTimer(&timer3, 0, 3, ACTIVE, Clk3);
  CreateTimer(&timer4, 0, 4, ACTIVE, Clk4);

   //разрешаем прерывания и тупим в цикле
  __enable_interrupt();
  while(1);
  return 0;
}
//*********************************
//Прерывание таймера/счетчика T0
#pragma vector = TIMER0_COMP_vect
__interrupt void Timer0CompVect(void)
{
  CheckTimer();
}
```

Заключение

Алгоритм, конечно, расточительный. Во-первых в прерывании запускается функция. Во-вторых перебор таймеров происходит последовательно - чем больше программных таймеров, тем больше времени это займет. В-третьих функции программных таймеров тоже запускаются в прерывании. Как можно оптимизировать? Не использовать много таймеров. Сделать функцию CheckTimer встраиваемой. Период работы аппаратного таймера выбрать большим. Функции программных таймеров сделать короткими.

Напоследок картинка с осциллографа