Лекция 9: “Периферийные устройства МК: универсальные таймеры/счетчики, аналого-цифровые преобразователи”

Гончаров Олег Игоревич

Факультет вычислительной математики и кибернетики,
Московский государственный университет имени М.В. Ломоносова

2013

Универсальный таймер/счетчик

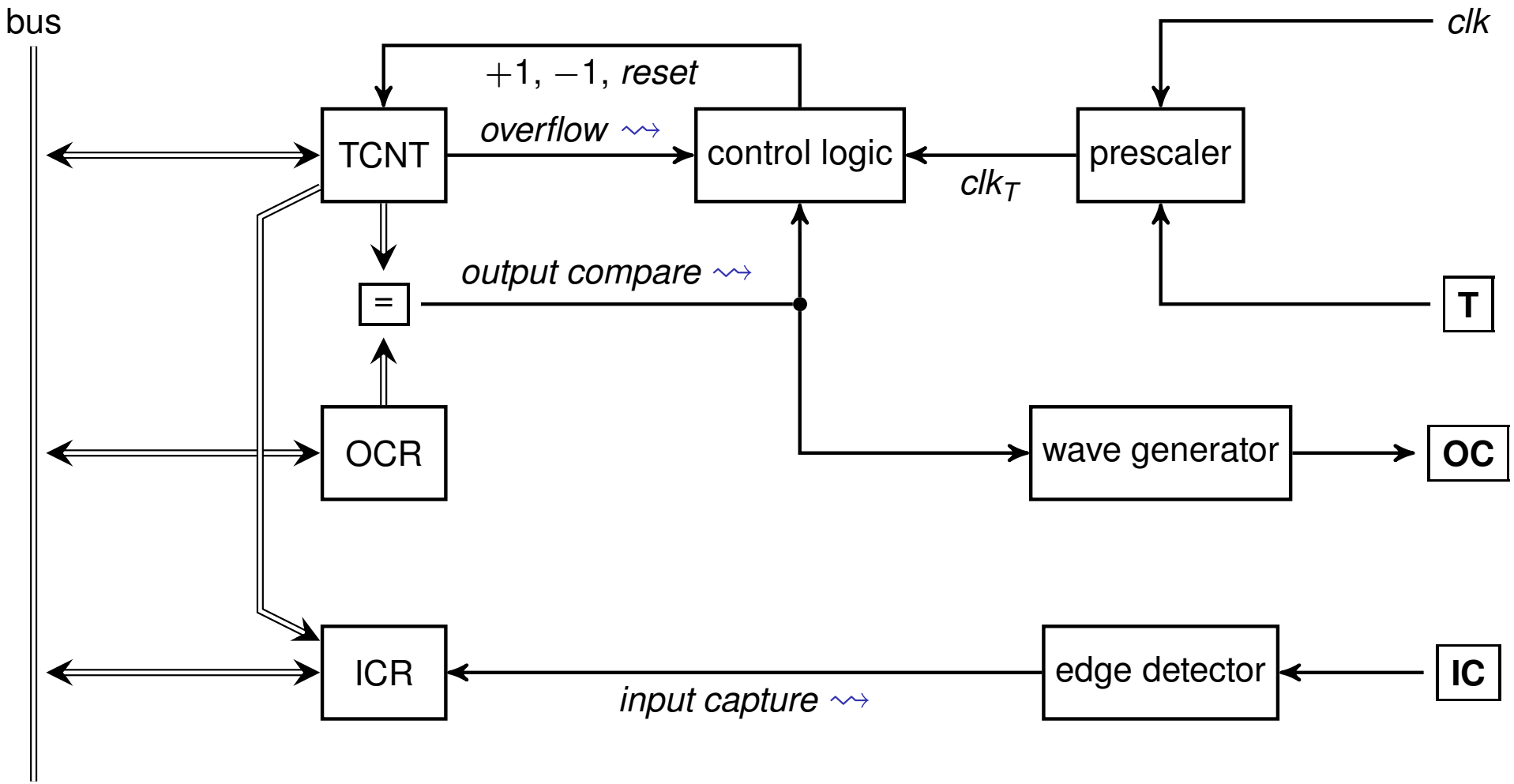
Основные функции:

- Режим счетчика: подсчет числа внешних событий (импульсов на выводе МК),
  - аппаратное декодирование сигнала относительного датчика перемещения (энкодера)
- Режим таймера:
  - генерация прерывания через заданный промежуток времени,
  - генерация прерываний с заданным периодом.
- Генерация ШИМ сигнала.
- Фиксация точного времени внешнего события.

Часто таймер/счетчик реализует только часть этих функций.

Разрешение таймера и максимальная длительность периода зависит от настроек и тактовой частоты МК.

Структура универсального таймера/счетчика



Универсальный таймер/счетчик

- Счетчик:
  - получая импульс clk_T управляющее устройство (control logic) увеличивает *TCNT* на 1,
  - при переполнении формируется прерывание *overflow* ⇝
- Делитель: формирует clk_T
  - получает импульсы от тактового генератора *clk*, либо с вывода **T** (внешний источник),
  - делит частоту на заданное настраиваемое число, например, на 1, 8, 32, 128, 1024.
- Таймер:
  - при *OCR* равном *TCNT* формируется прерывание *output compare* ⇝.
- Генератор ШИМ сигнала:
  - ШИ-модулятор (wave generator) формирует сигнал на выводе **OC** со скважностью задаваемой *OCR*.
- Захват внешних событий:
  - по изменению уровня на выводе **IC** содержимое *TCNT* копируется в *ICR*, и формируется прерывание *input capture* ⇝.

AVR: Пример использования таймера/счетчика

Режим таймера

Обработчик прерываний будет вызываться с интервалом в 1 секунду.

```
#include <avr/io.h>
#include <avr/interrupt.h>

/* Timer initialization */
void  timer_init(void) {
     /* CTC mode: Clear Timer on Compare match.
     If TCNT1 == OCR1A counter register TCNT1 is cleared.
     clkT = clk/1024 = 16000000/1024 = 15625;
     Interrupts: Compare Match A on TCNT1 == OCR1A */
     TCCR1A = 0;
     TCCR1B = _BV(WGM12) | _BV(CS10) |  _BV(CS12);

     OCR1A = 15625; // 1 second delay
     /*Enable Output Compare  Match A interrupt*/
     TIMSK1 |= _BV(OCIE1A);
}

/* interrupt handler */
ISR(TIMER1_COMPA_vect) {
...
}
```

AVR: Пример использования таймера/счетчика

ШИМ модуляция

Формирование ШИМ сигнала с частотой около 31 кГц.
ШИ-модулятор может выдавать сигналы разных типов.

```
#include <avr/io.h>
#include <avr/interrupt.h>

/* Init PWM output on pin PC5/OC1A. */
void pwm_init(void) {
    /* Simple 9-bit PWM.
    freq_pwm = clkT / 2^9 = clk / 512 = 16000000 / 512 = 31 kHz. */
    TCCR1A = _BV(WGM11);
    TCCR1B = _BV(WGM12) | _BV(CS10);
/* Configure waveform generator */
TCCR1A |= _BV(COM1A1);

    OCR1A = 0x0000;
}

/* Set PWM value. */
void set_pwm(uint16_t pwm) {
    OCR1A = (pwm <= 0x01ff) ? pwm : 0x01ff;
}
```

Аналого-цифровой преобразователь

Преобразование аналогового сигнала в цифровой.

Входной аналоговый сигнал:

- напряжение на входе по отношению к нулю (земле) V_{in},
- балансный вход, разность напряжений на двух входах $V_+ - V_-$.

Выхоной цифровой сигнал:

- квантование по времени:
  - фиксация входного сигнала в определенные моменты времени,
  - длительность цикла преобразования T_{adc};
- квантование по уровню:
  - разрядность результата преобразования N_{bit},
  - ограниченный диапазон входных сигналов максимальным (эталонным) напряжением V_{aref}.

Результат преобразования подвержен возмушению:

- неидеальности работы АЦП,
- шум, воздействующий на аналоговую часть АЦП: входные контуры, V_{cc}, наводки от ЦПУ и т.п.

Математическая модель АЦП

В идеальный результат преобразования будет

$$ADC = \text{round}\,\frac{V_{in}\cdot 2^{N_{bit}}}{V_{aref}} \text{ или } ADC = \text{round}\,\frac{(V_+ - V_-)\cdot 2^{N_{bit}-1}}{V_{aref}}.$$

В случае балансного входа один бит “потрачен” на знак.

Чувствительность преобразователя

$$V_{LSB} = \frac{V_{aref}}{2^{N_bit}}.$$

Точность определяется различными неидеальностями:

- смещение,
- отличие коэффициента усилинеия от 1,
- интегральная ошибка (?),
- разностная ошибка (?),
- ошибка квантования.

Устройство модуля АЦП

Здесь схема

Устройство модуля АЦП

Порядок работы преобразователя:

1. запуск,
2. фиксация входного значения напряжения *sample hold,*
3. *значение счетчика растет от* 0 *до* $2^{N_{bit}} - 1$*, результат ЦАП сравнивается с напряжением, при смене знака значение счетчика помещается в регистр ADC.*
4. *прерывание* ADC completed $\rightsquigarrow$

Устройство модуля АЦП

Особенности функционирования АЦП:

- мультиплексор позволяет выбрать нужные входы;
- делитель определяет длительность преобразования T_{adc} и задержку фиксации напряжения входа:

  меньше длительность — меньше точность;
- возможно задание разных источников V_{aref}:
  - внешний источник,
  - напряжение питания;
- запуск преобразования:
  - вручную,
  - по событию (флаг прерывания) — детеменированное время фиксации входного сигнала,
  - непрерывный (по событию *ADC completed* $\rightsquigarrow$) — *минимальная задержка.*
- подавление шума: отключение ЦПУ на время преобразования.

AVR: Пример использования АЦП

Непрерывный режим работы АЦП, результат преобразование записывается в разделяемую переменную.

```
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/atomic.h>

volatile uint16_t adc_result;

void adc_init_free(void) {
    /* Varef = Vcc, result is right aligned */
    ADMUX = _BV(REFS0);
    /*select input: single channel, PA0 (ADC0): MUX[4:0] = 0, MUX5 = 0
        */
    DDRA &= ~_BV(PA0); /* input mode */
    PORTA &= ~_BV(PA0); /* pull up disabled */
    DIDR0 |= _BV(ADC0D); /* disable input logic */
    /* prescaler factor is 128, ADCclk = 16 MHz / 128 = 125 kHz,
    Tadc = 13*(1/ADCclk) = 104 mcs */
    ADCSRA = _BV(ADPS0) | _BV(ADPS1) | _BV(ADPS2);
    /* free running mode: ADC is triggered by ADC Complete flag */
    ADCSRA |= _BV(ADATE);
    ADCSRB &= ~(_BV(ADTS0) | _BV(ADTS1) | _BV(ADTS2));
    ...
```

AVR: Пример использования АЦП (продолжение)

```
    ...
    /* enable ADC Complete interrupt */
    ADCSRA |= _BV(ADIE);
    /* enable ADC and start conversation */
    ADCSRA |= _BV(ADEN);
    ADCSRA |= _BV(ADSC);
}

/* interrupt handler */
ISR(ADC_vect) {
    analog_in = ADC; // interrupts are disabled
}

int main() {
    adc_init_free();
    sei();
    ...
    ATOMIC_BLOCK(ATOMIC_FORCEON) { /* cli() */
     analog = adc_result;
    }                              /* sei() */
    ...
}
```