Embedded Systems Workshop 1

02/04/2019

Agenda

- Scaling concepts
- Set up development environment (virtualbox)
- Set up hardware
- Explore some features
  - Basic I/O
  - Interrupts
  - Timer/counters
  - So many features…
  - Low power & sleep modes

Scaling – Energy vs Size vs Computational Power

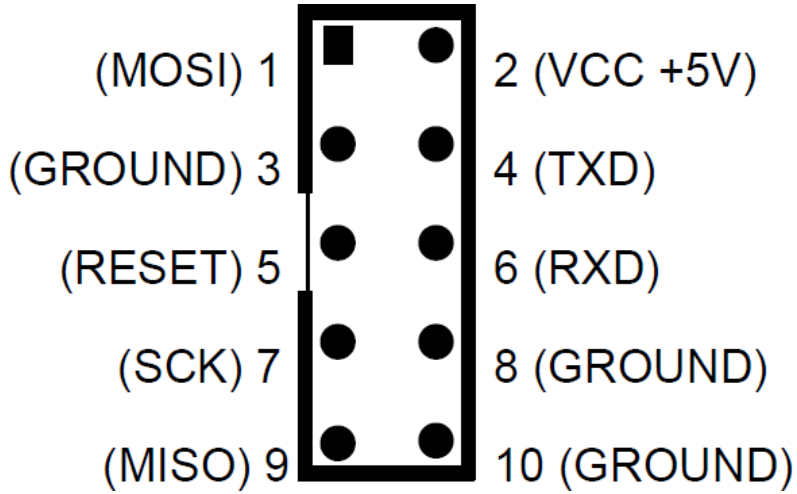
- If you are going to produce waste heat, have a good reason for doing so
- If you are going to take up space, have a good reason for doing so
- Unless otherwise constrained, choose low power & small size
  - Often, you can get away with more than you realize

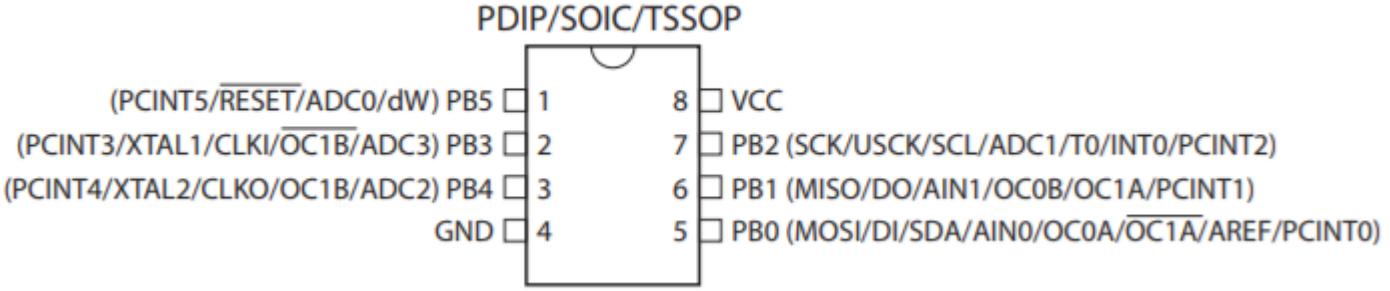
Set up dev environment

- Download and install Virtualbox
- You may need to update driver for USBASP (the programmer)
  - Download and install zadig (Windows)
  - Plug in programmer, run zadig
  - Install libusbk driver

Set up hardware

- Plug ATTINY85 into breadboard
- Wire programmer
  - VCC → VCC
  - GND → GND
  - MOSI → MOSI
  - MISO → MISO
  - RESET → RESET
  - SCK → SCK





Set up dev environment (cont.)

- Make sure programmer is unplugged
- Open “AVR Tool OS” virtual machine
- Plug in programmer to computer
- “plug in programmer” to virtual machine
  - Devices -> USB -> (usbasp device)

Compiling and Flashing Code

- We’re using AVRDUDE to program our chips
- First compile the code using `sudo ./compile_script`
- Then flash the code onto the chip using `sudo ./upload_script`

Basic I/O & bit manipulation

- We have one I/O register to work with, and just 6 pins - Port B (PB5:PB0)
- To configure a pin as an input or output, use DDRB:
  - Set PB3 as an output: `DDRB |= (1<<3)`
  - Set PB3 as an input: `DDRB &= ~(1<<3)`
  - Set all of PORTB as outputs: `DDRB = 0b11111111`
- To toggle an output pin on or off:
  - PB3 On: `PORTB |= (1<<3)`
  - PB3 Off: `PORTB &= ~(1<<3)`
  - PB3 Toggle: `PORTB ^= (1<<3)`
- Read a pin
  - Read PB3 (boolean): `uint8_t result = PINB&(1<<3)`

Example – Blinky

- Blink the LED at different speeds based on the logical input from an IO pin
- Note that PINB is only read every 500 or 200 ms

```
1  // basic IO and bit manipulation
2
3  #include <avr/io.h>
4  #include <util/delay.h>
5
6
7
8  // return a 1 if the pin is high
9  // return a 0 if the pin is low
10 uint8_t read_pin(uint8_t pin){
11 s
12     PORTB |=
13     if(PINB&(1<<pin))
14         return 1;
15     else
16         return 0;
17 }
18
19
20 int main(){
21
22     // set PB3 as an output
23     // use PB4 as an input - leave as a zero
24     DDRB = (1<<PB3);
25
26     // main loop
27     while(1){
28
29         // if PB4 is high, blink at one speed
30         if(read_pin(PB4)){
31             // toggle PB3 and delay
32             PORTB ^= (1<<PB3);
33             _delay_ms(500);
34         }
35         else{
36             // blink at another speed
37             PORTB ^= (1<<PB3);
38             _delay_ms(200);
39         }
40     }
41
-- INSERT --
```

Interrupts

ATtiny85

Vector Number	Interrupt definition	Vector name
1	External Interrupt Request 0	`INT0_vect`
2	Pin Change Interrupt Request 0	`PCINT0_vect`
3	Timer/Counter1 Compare Match A	`TIMER1_COMPA_vect`
4	Timer/Counter1 Overflow	`TIMER1_OVF_vect`
5	Timer/Counter0 Overflow	`TIMER0_OVF_vect`
6	EEPROM Ready	`EE_RDY_vect`
7	Analog Comparator	`ANA_COMP_vect`
8	ADC Conversion Complete	`ADC_vect`
9	Timer/Counter1 Compare Match B	`TIMER1_COMPB_vect`
10	Timer/Counter0 Compare Match A	`TIMER0_COMPA_vect`
11	Timer/Counter0 Compare Match B	`TIMER0_COMPB_vect`
12	Watchdog Time-out	`WDT_vect`
13	USI Start Condition	`USI_START_vect`
14	USI Overflow	`USI_OVF_vect`

Interrupts

ATtiny85

Vector Number	Interrupt definition	Vector name
1	External Interrupt Request 0	INT0_vect
2	Pin Change Interrupt Request 0	PCINT0_vect
3	Timer/Counter1 Compare Match A	TIMER1_COMPA_vect
4	Timer/Counter1 Overflow	TIMER1_OVF_vect
5	Timer/Counter0 Overflow	TIMER0_OVF_vect
6	EEPROM Ready	EE_RDY_vect
7	Analog Comparator	ANA_COMP_vect
8	ADC Conversion Complete	ADC_vect
9	Timer/Counter1 Compare Match B	TIMER1_COMPB_vect
10	Timer/Counter0 Compare Match A	TIMER0_COMPA_vect
11	Timer/Counter0 Compare Match B	TIMER0_COMPB_vect
12	Watchdog Time-out	WDT_vect
13	USI Start Condition	USI_START_vect
14	USI Overflow	USI_OVF_vect

Interrupts

- Implement ‘button’ behavior using interrupts (INT0) instead of the previous approach
- Reference – page 49 of the datasheet

MCUCR – MCU Control Register

The External Interrupt Control Register A contains control bits for interrupt sense control.

Bit	7	6	5	4	3	2	1	0	
0x35	BODS	PUD	SE	SM1	SM0	BODSE	ISC01	ISC00	MCUC
Read/Write	R	R/W	R/W	R/W	R/W	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 9-2. Interrupt 0 Sense Control

ISC01	ISC00	Description
0	0	The low level of INT0 generates an interrupt request.
0	1	Any logical change on INT0 generates an interrupt request.
1	0	The falling edge of INT0 generates an interrupt request.
1	1	The rising edge of INT0 generates an interrupt request.

GIMSK – General Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
0x3B	–	INT0	PCIE	–	–	–	–	–	GIMSK
Read/Write	R	R/W	R/W	R	R	R	R	R	
Initial Value	0	0	0	0	0	0	0	0	

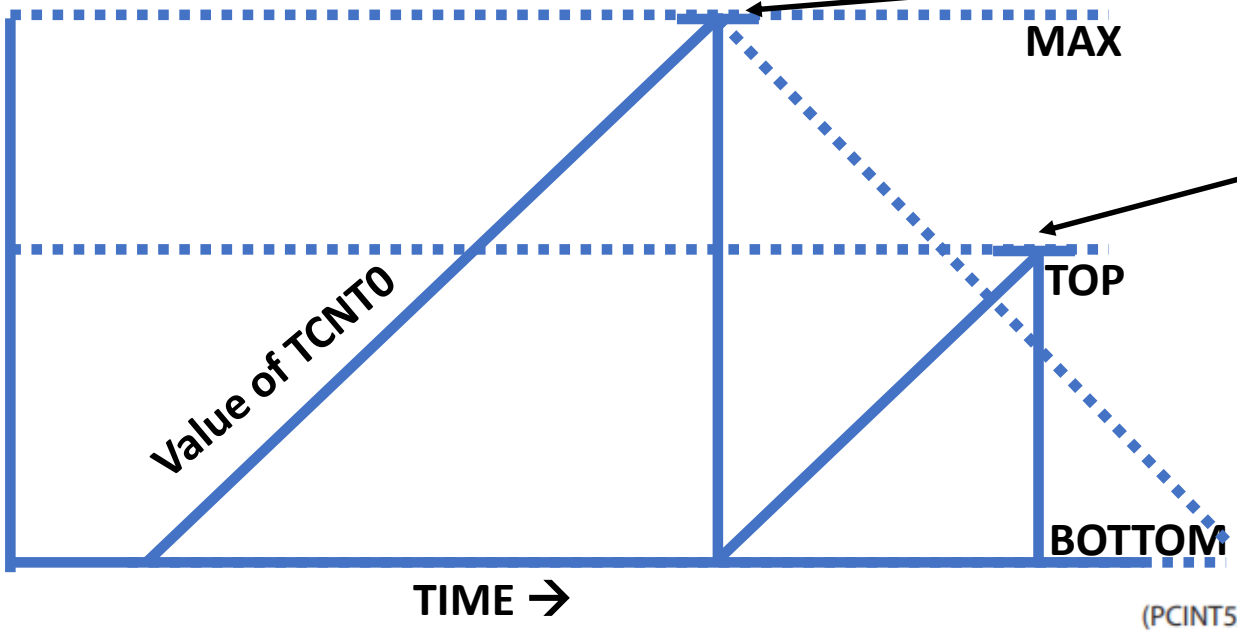
Example – Blinky again

- Blink the LED at different speeds based on the logical input from an IO pin
- Note that changes to the input pin are noticed as soon as they occur
- However, the delay time still will not be updated until the next cycle

```c
// basic IO and bit manipulation

#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

// global flag that will be set during an external interrupt
volatile uint8_t ext_flag = 0;

// Interrupt Service Routine
ISR(INT0_vect){

    // when INT0 is triggered, toggle the flag
    if(ext_flag)
        ext_flag = 0;
    else
        ext_flag = 1;
}

int main(){

    // set PB3 as an output
    DDRB = (1<<PB3);

    // set INT0 to trigger on logical change
    MCUCR = (1<<ISC00);
    // enable INT0
    GIMSK = (1<<INT0);
    // enable interrupts
    sei();

    // main loop
    while(1){

        // choose blinking speed based on ext_flag
        PORTB ^= (1<<PB3);
        if(ext_flag)
            _delay_ms(100);
        else
            _delay_ms(500);
    }
"main.c" 43L, 787C written
```

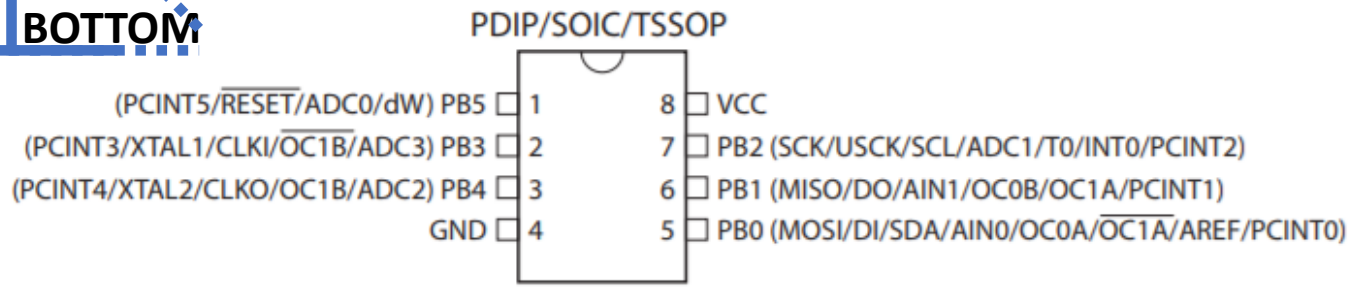
Timer/Counters

- Reference – page 85 of datasheet
- Many, many, many ways to use these



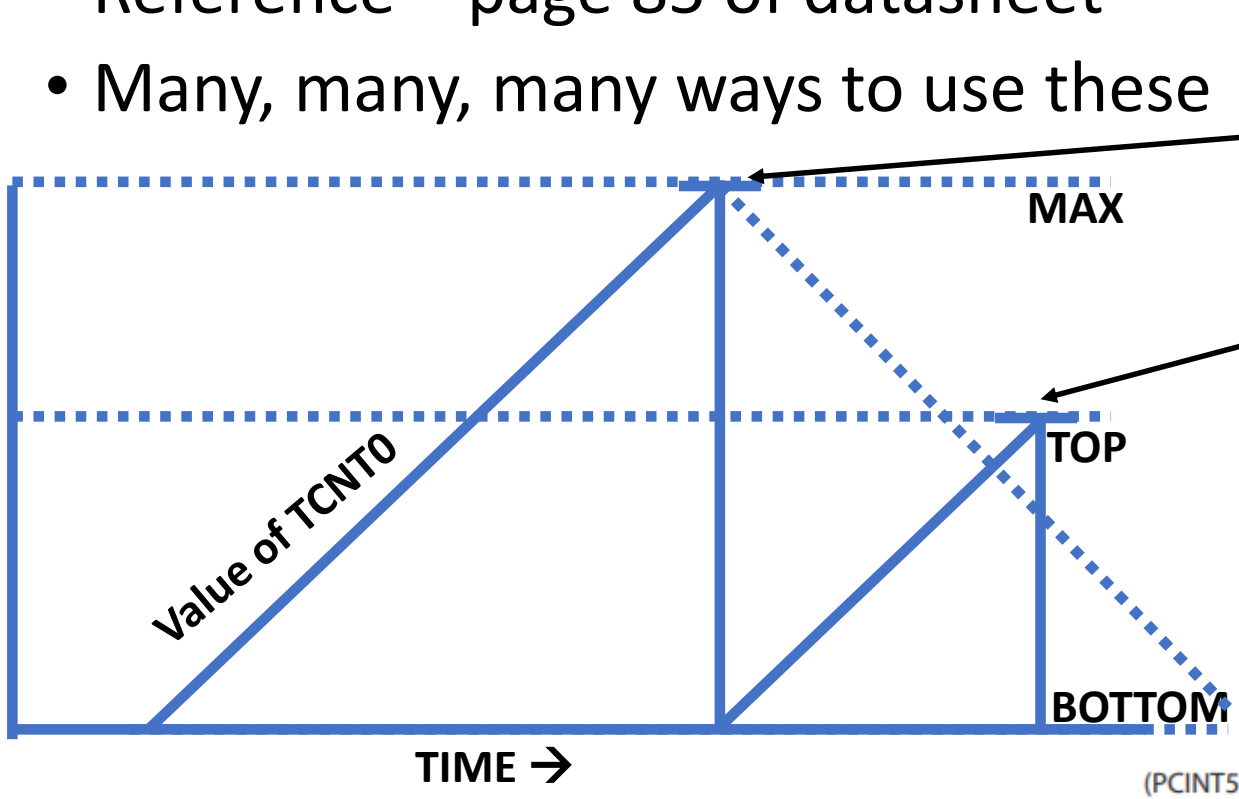
Trigger an interrupt and/or toggle a OCxx pin at MAX or…

Trigger an interrupt and/or toggle a OCxx pin at TOP



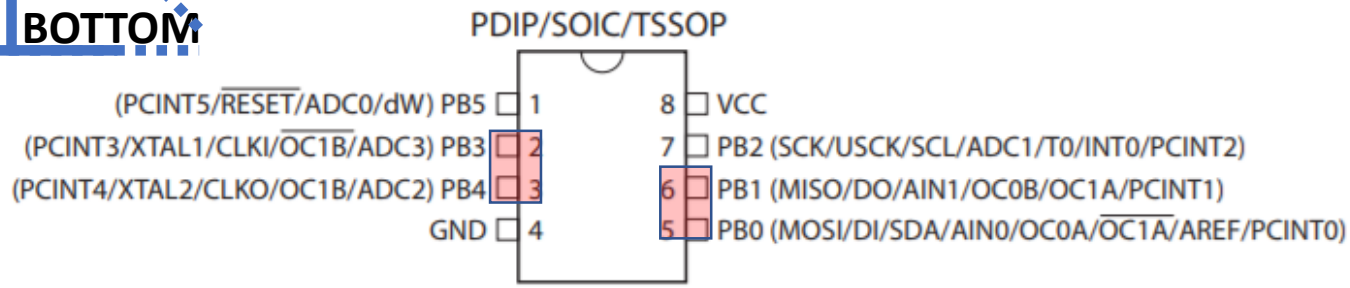
Timer/Counters

- Reference – page 85 of datasheet
- Many, many, many ways to use these



Trigger an interrupt and/or toggle a OCxx pin at MAX or…

Trigger an interrupt and/or toggle a OCxx pin at TOP



Interrupts

ATtiny85

Vector Number	Interrupt definition	Vector name
1	External Interrupt Request 0	INT0_vect
2	Pin Change Interrupt Request 0	PCINT0_vect
3	Timer/Counter1 Compare Match A	TIMER1_COMPA_vect
4	Timer/Counter1 Overflow	TIMER1_OVF_vect
5	Timer/Counter0 Overflow	TIMER0_OVF_vect
6	EEPROM Ready	EE_RDY_vect
7	Analog Comparator	ANA_COMP_vect
8	ADC Conversion Complete	ADC_vect
9	Timer/Counter1 Compare Match B	TIMER1_COMPB_vect
10	Timer/Counter0 Compare Match A	TIMER0_COMPA_vect
11	Timer/Counter0 Compare Match B	TIMER0_COMPB_vect
12	Watchdog Time-out	WDT_vect
13	USI Start Condition	USI_START_vect
14	USI Overflow	USI_OVF_vect

Example – Blinky yet again

- Same basic behavior as previous example
- However, now the delay time is adjusted as soon as the pin voltage changes (within ~32 us)
- Using normal mode (fast PWM commented out in main)

```
18 // Interrupt Service Routine - counter overflow
19 ISR(TIMER0_OVF_vect){
20
21     // change blinking speed based on state of ext_flag
22     uint16_t temp;
23     if(ext_flag)
24         temp = 1000;
25     else
26         temp = 100;
27
28     // count overflows about every 32 us
29     // 32 overflows is about 1 ms
30     count_ovf++;
31     if(count_ovf==32){
32         count_ovf = 0;
33         // count some time depending on ext_flag
34         if(count_ms<temp)
35             count_ms++;
36         else{
37             PORTB ^= 1<<PB3;
38             count_ms = 0;
39         }
40     }
41 }
```

Features to discuss in another workshop…

- PWM in depth
- ADC
- Built-in analog comparator
- Communication (UART, I2C, SPI…)
- Watchdog timer
- Brownout detection
- …

Low Power and Sleep Modes

- Reference – page 34 in the datasheet
- Basically cuts off certain features from the clock
- If not needed, disable watchdog timer and ADC to conserver power further

Table 7-1. Active Clock Domains and Wake-up Sources in the Different Sleep Modes

	Active Clock Domains					Oscillators	Wake-up Sources					
Sleep Mode	clk_{CPU}	clk_{FLASH}	clk_{IO}	clk_{ADC}	clk_{PCK}	Main Clock Source Enabled	INT0 and Pin Change	SPM/EEPROM Ready	USI Start Condition	ADC	Other I/O	Watchdog Interrupt
Idle			X	X	X	X	X	X	X	X	X	X
ADC Noise Reduction				X		X	X(1)	X	X	X		X
Power-down							X(1)		X			X

- **Active Mode:**
  - **1 MHz, 1.8V: 300 µA**
- **Power-down Mode:**
  - **0.1 µA at 1.8V**

Example – Blinky + power down

MCUCR – MCU Control Register

The MCU Control Register contains control bits for power management.

Bit	7	6	5	4	3	2	1	0	
0x35	BODS	PUD	SE	SM1	SM0	BODSE	ISC01	ISC00	MCUCR
Read/Write	R	R/W	R/W	R/W	R/W	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Table 7-2. Sleep Mode Select

SM1	SM0	Sleep Mode
0	0	Idle
0	1	ADC Noise Reduction
1	0	Power-down
1	1	Reserved

PRR – Power Reduction Register

The Power Reduction Register provides a method to reduce power consumption by allowing peripheral clock signals to be disabled.

Bit	7	6	5	4	3	2	1	0	
0x20	–	–	–	–	PRTIM1	PRTIM0	PRUSI	PRADC	PRR
Read/Write	R	R	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	