Chapter 5: Atmel's AVR 8-bit Microcontroller
Part 2 – Input/Output

Prof. Ben Lee
Oregon State University
School of Electrical Engineering and Computer Science

Chapter Goals

- Understand how to program AVR I/O:
  - AVR I/O facilities:
    - Ports, timers, interrupts, serial communications, ADC, etc.
  - Configuring I/O control registers.
- Understand what microcontrollers were designed for:
  - Programming microcontrollers is about programming I/O.

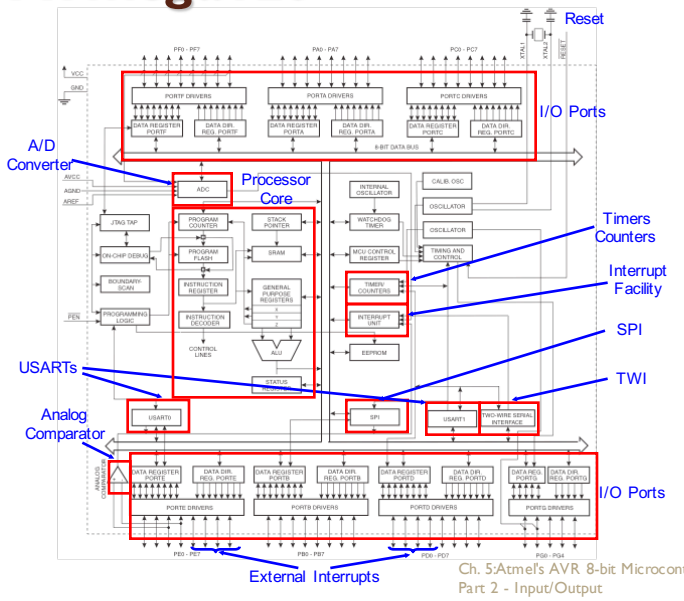
Contents

5.1 Introduction

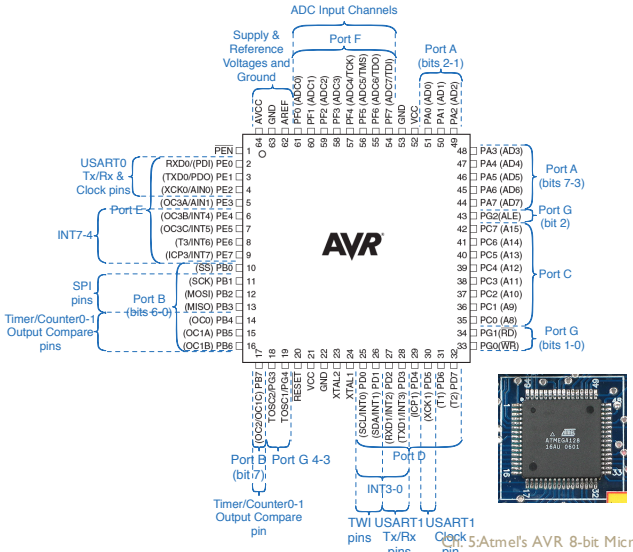
I/O

- The primary responsibility of an embedded processor is to handle I/O.
- Types of I/O handled depends on the application.
- Some I/O devices are implemented within microcontrollers, others are implemented separately and controlled by a microcontroller.
- Some common I/O devices:
  - Computer peripherals
    - Keyboard, mouse, display, printer, etc.
  - Sensors
    - Temperature, motion, pressure, light, sound, etc.
- These devices need microcontrollers!

ATmega128



I/O Pins

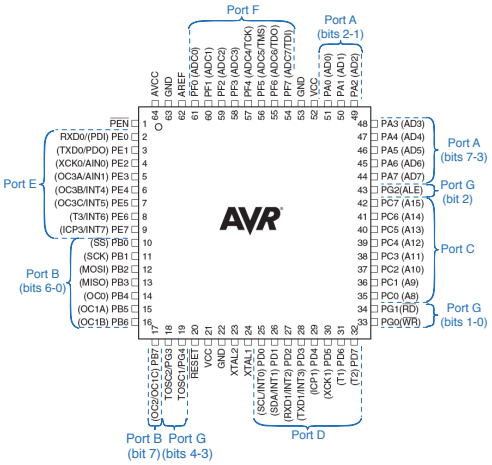


5.2 I/O Ports

I/O Ports

- Used to interface I/O devices external to the microcontroller.
- AVR Atmega128 has six 8-bit ports and one 5-bit port (53 I/O lines).
  - Port *A-G* Data Register (PORT*A-G*)
- I/O ports can be both read and write.
- How is it set to read or write?
  - Port *A-G* Data Direction Register (DDR*A-G*)
- Both PORTx and DDRx are mapped to I/O Registers address space.
  - e.g., PORTA mapped to $3B, DDRA mapped to $3A

I/O Port Pins



Setting I/O Ports

Each port has three I/O registers

"x" = A - G
"n" = 7 - 0

Port x Data Register (PORTx)

Write to

7	6	5	4	3	2	1	0	
PORTx7	PORTx6	PORTx5	PORTx4	PORTx3	PORTx2	PORTx1	PORTx0	PORTx
R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	

PORTxn = 1 and DDxn = 0 Activates Pull-Up Register => More on this later

Port x Input Pins (PINx)

Read from

7	6	5	4	3	2	1	0	
PINx7	PINx6	PINx5	PINx4	PINx3	PINx2	PINx1	PINx0	PINx
R (N/A)	R (N/A)	R (N/A)	R (N/A)	R (N/A)	R (N/A)	R (N/A)	R (N/A)	

=> Writing to PINx has no effect!

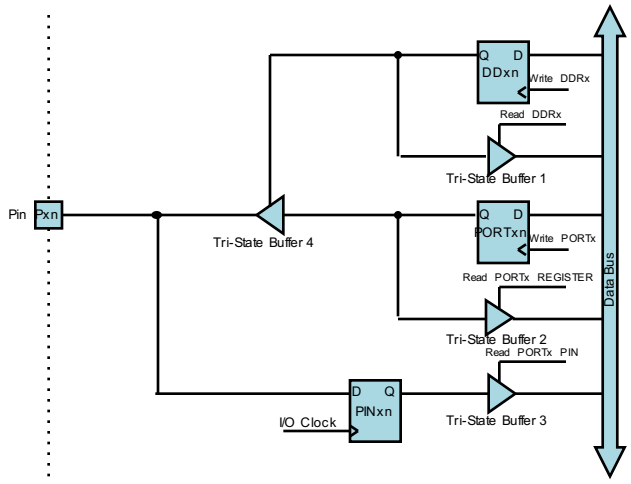
Port x Data Direction Register (DDRx)

Set direction

7	6	5	4	3	2	1	0	
DDx7	DDx6	DDx5	DDx4	DDx3	DDx2	DDx1	DDx0	DDRx
R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	

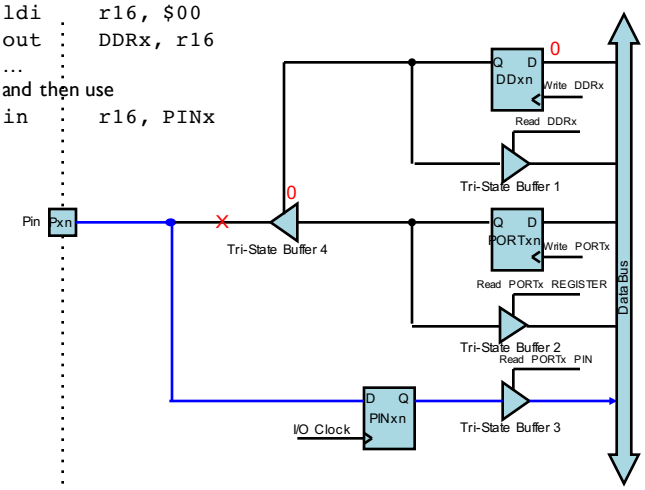
DDxn = 0 Input
DDxn = 1 Output => Initially reset to input

I/O Pins (Simplified)



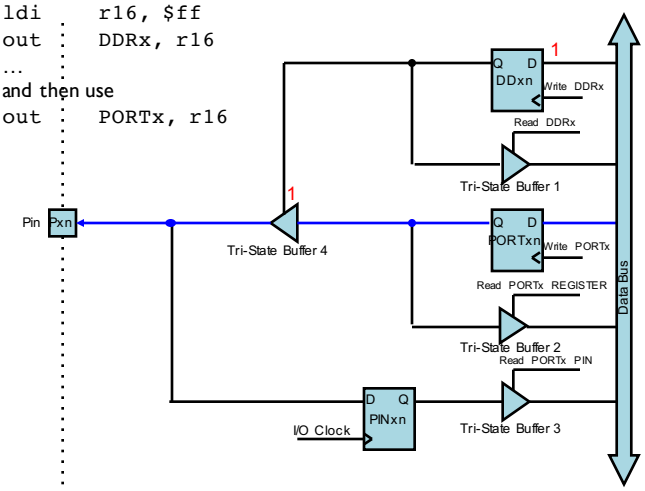
Setting for Input

```
ldi     r16, $00
out     DDRx, r16
...
and then use
in      r16, PINx
```



Setting for Output

```
ldi     r16, $ff
out     DDRx, r16
...
and then use
out     PORTx, r16
```

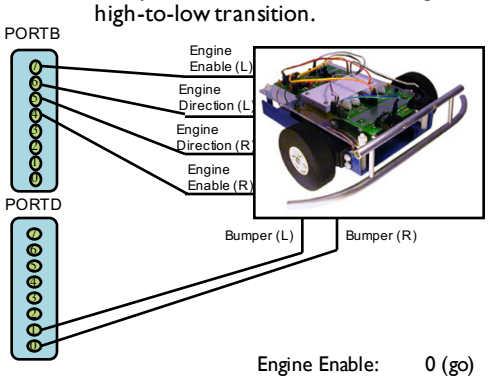


Example - Tekbot Whiskers

- Tekbot has two switches (left and right) that detect bumps.
  - Right whisker input connected to PORTE pin 4
  - Left whisker input connected to PORTE pin 5
- Detection of a bump initiates a routine to turn the Tekbot around.
  - Right/Left engine enable connected to PORTB pins 4/7
    - 0=go, 1=no go
  - Right/Left engine direction connected to PORTB pins 5/6
    - 1=forward, 0=backward

Tekbot Bumper Switch Connection

Bumper hit closes a switch causing high-to-low transition.



Engine Enable: 0 (go)
1 (stop)
Engine Direction: 0 (backward)
1 (forward)

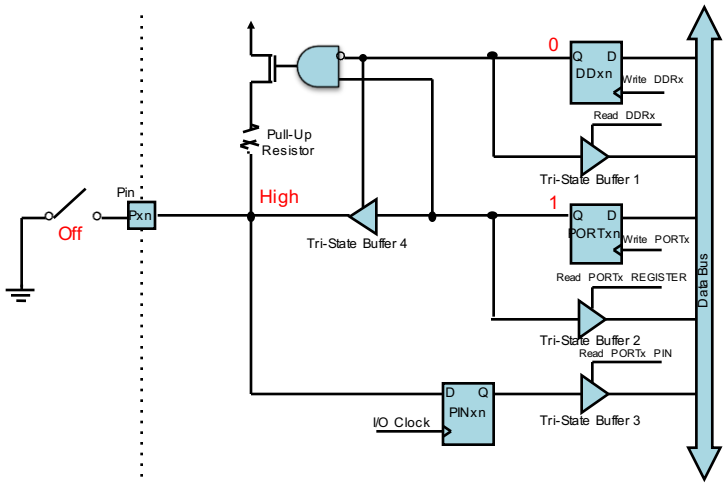
Initialization

```
; AVR Assembly code - Basic operations of Tekbot
.include "m128def.inc"
;***********************************************************
;* Internal Register Definitions and Constants
;***********************************************************
.def mpr = r16                  ; Multipurpose register
.def waitcnt = r17              ; Wait Loop Counter
.def ilcnt = r18                ; Inner Loop Counter
.def olcnt = r19                ; Outer Loop Counter
.equ WTime = 100                ; Time to wait in wait loop
.equ WskrR = 0                  ; Right Whisker Input Bit
.equ WskrL = 1                  ; Left Whisker Input Bit
.equ EngEnR = 4                 ; Right Engine Enable Bit
.equ EngEnL = 7                 ; Left Engine Enable Bit
.equ EngDirR = 5                ; Right Engine Direction Bit
.equ EngDirL = 6                ; Left Engine Direction Bit
.equ MovFwd = (1<<EngDirR|1<<EngDirL)  ; Move Forward Command
.equ MovBck = $00               ; Move Backward Command
.equ TurnR = (1<<EngDirL)       ; Turn Right Command
.equ TurnL = (1<<EngDirR)       ; Turn Left Command
.equ Halt = (1<<EngEnR|1<<EngEnL)       ; Halt Command
;***********************************************************
;* Start of Code Segment
;***********************************************************
      .cseg                     ; Beginning of code segment
```

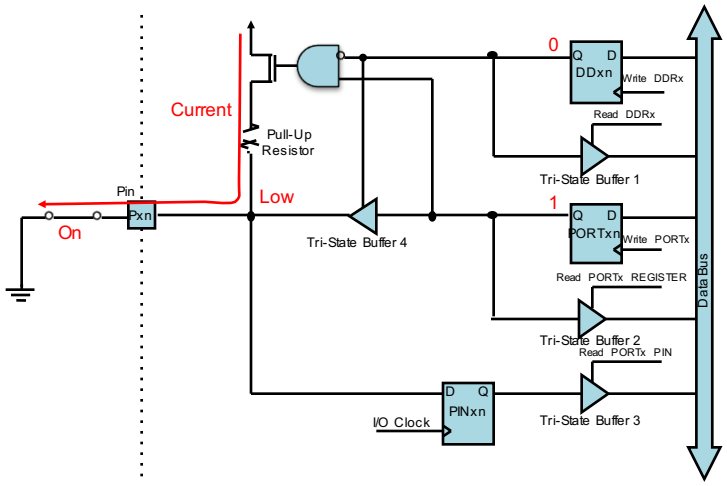

Initialization (cont.)

```
;-----------------------------------------------------------
; Interrupt Vectors
;-----------------------------------------------------------
     .org $0000                 ; Reset and Power On Interrupt
     rjmp INIT                  ; Jump to program initialization
     .org $0046                 ; End of Interrupt Vectors
;-----------------------------------------------------------
; Program Initialization
;-----------------------------------------------------------
INIT:
;; Initialize Stack ;;
; Initialize Port B for output
     ldi  mpr, (1<<EngEnL)|(1<<EngEnR)|(1<<EngDirR)|(1<<EngDirL)=> 11110000
     out  DDRB, mpr             ; Set Port B Directional Register  for output
; Initialize Port D for inputs
     ldi mpr, (0<<WskrL)|(0<<WskrR) => 00000000
     out DDRD, mpr              ; Set Port D Directional Register for input
     ldi mpr, (1<<WskrL)|(1<<WskrR) => 00000011
     out PORTD, mpr             ; Activate pull-up resistors
; Initialize TekBot Forward Movement
     ldi  r16, MovFwd           ; Load Move Forward Command => 01100000
     out  PORTB, mpr            ; Send command to motors
```

Pull-up Resistor – High Input



Pull-up Resistor – Low Input



MAIN Routine

Bumper hit causes high to low transition

```
;-----------------------------------------------------------
; Main Program
;-----------------------------------------------------------
MAIN:
     in       mpr, PIND      ; Get whisker input from Port D
     com      mpr            ; Complement since bumpers are active low
     andi     mpr, (1<<WskrL)|(1<<WskrR)     ; Mask out other bits => 00000011
     cpi      mpr, (1<<WskrR); Check for Right Whisker input => 00000001
     brne     NEXT           ; Continue with next check
     rcall    HitRight       ; Call the subroutine HitRight
     rjmp     MAIN           ; Continue with program
NEXT:
     cpi      mpr, (1<<WskrL); Check for Left Whisker input => 00000010
     brne     MAIN           ; No Whisker input, continue
     rcall    HitLeft        ; Call subroutine HitLeft
     rjmp     MAIN           ; Continue through main
```


HitRight Subroutine

```
;-----------------------------------------------------------
; Sub:    HitRight
; Desc:   Handles functionality of the TekBot when the right whisker
;         is triggered.
;-----------------------------------------------------------
HitRight:
; Move Backwards for a second
     ldi  mpr, MovBck       ; Load Move Backwards command => 00000000
     out  PORTB, mpr        ; Send command to port
     ldi  waitcnt, WTime    ; Wait for 1 second
     rcall    Wait          ; Call wait function
; Turn left for a second
     ldi  mpr, TurnL        ; Load Turn Left Command      => 00100000
     out  PORTB, mpr        ; Send command to port
     ldi  waitcnt, WTime    ; Wait for 1 second
     rcall    Wait          ; Call wait function
; Move Forward again
     ldi  mpr, MovFwd       ; Load Move Forward command   => 01100000
     out  PORTB, mpr        ; Send command to port
     ret                    ; Return from subroutine
```

Wait Subroutine

16 MHz clock rate => 62.5 ns per clock cycle
Why triple-nested loop? Only 8-bit registers!

```
;-----------------------------------------------------------
; Sub:    Wait
; Desc:   A wait loop that is 16 + 159975*waitcnt cycles or roughly
;         waitcnt*10ms.  Just initialize wait for the specific amount
;         of time in 10ms intervals. Here is the general equation
;         for the number of clock cycles in the wait loop:
;              ((3 * ilcnt + 3) * olcnt + 3) * waitcnt + 13 + call
;-----------------------------------------------------------
Wait:
OLoop:
      ldi  olcnt, 224    (1)  ; Load middle-loop count
MLoop:
      ldi  ilcnt, 237    (1)  ; Load inner-loop count
ILoop:
      dec  ilcnt         (1)  ; Decrement inner-loop count
      brne Iloop         (2/1); Continue inner-loop
      dec  olcnt         (1)  ; Decrement middle-loop
      brne Mloop         (2/1); Continue middle-loop
      dec  waitcnt       (1)  ; Decrement outer-loop count
      brne OLoop         (2/1); Continue outer-loop
      ret                     ; Return from subroutine
```


Timing of Wait Subroutine

$$(((((3\times ilcnt)-1+4)\times olcnt)-1+4)\times waitcnt)-1+16$$

Inner loop

Middle loop

Outer loop

- 3 push (2)
- 3 pop (2)
- RET (4)

(((((3*237)+3)*224)+ 3)*100)-1+16=15,993,915

15,993,915 cycles * 62.5 nsec = 0.99962 sec

5.3 Interrupts

Interrupts

- A loop that constantly checks for an input is called busy waiting.
- In contrast, interrupt is an unexpected event (in terms of timing) causing a change in the normal flow of the instruction execution:
  - Related to I/O
- Examples:
  - I/O device ready to accept more output
    - USART (Universal Synchronous/Asynchronous Receiver/Transmitter) transmit buffer empty.
  - I/O device has input ready
    - Key pressed on keyboard.
    - Tekbot switches triggered.
  - Regular interrupt
    - Timer generates interrupt, e.g., 20 ms.

Interrupts versus Traps

- Traps are software interrupts, i.e., caused by event in software rather hardware.
- Examples:
  - Overflow
  - Divide by zero
  - Undefined opcode
  - SWI (software interrupt)
- Synchronous respect to instruction execution.
- Trap handlers (service routines) do not always return to the original program.
- AVR does not provide traps.

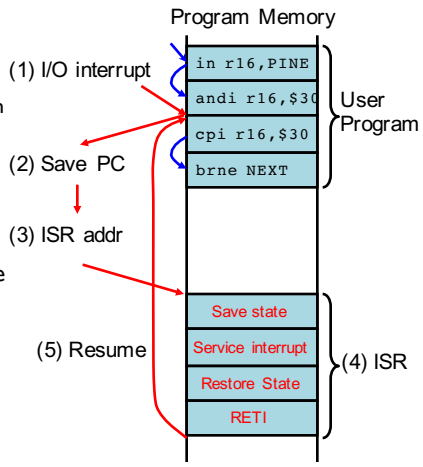
Interrupts

- An I/O interrupt is just like a trap except:
  - An I/O interrupt is "asynchronous".
  - More information needs to be conveyed.
- An I/O interrupt is asynchronous with respect to instruction execution:
  - I/O interrupt is not associated with any instruction.
  - I/O interrupt does not prevent any instruction from completion.
    - Pick convenient point to take an interrupt

Interrupt Handling

Steps are:

1. Recognize the interrupt.
   - At the end of each instruction execution, processor checks for an interrupt.
2. Push return address onto stack.
3. Determine source of interrupt.
   - Polling vs. vector
4. Execute Interrupt Service Routine (ISR).
   - Save the state of CPU.
   - Service the Interrupt
   - Restore the state of CPU.
5. Resume program execution.



Some Issues

- How is the state of the processor saved?
- Which I/O device caused interrupt?
  - Needs to convey the identity of the device that generated the interrupt.
- Can avoid interrupts during the interrupt routine?
  - What if more important interrupt occurs while servicing this interrupt?
  - Allow interrupt routine to be entered again?
- Who keeps track of status of all the devices, handle errors, know where to put/supply the I/O data?

Interrupt Facility in AVR ATmega128

- 35 interrupt sources!
  - 8 external interrupt requests
  - 4 Timer/Counters with compare and overflow
  - ADC complete
  - USART Tx/Rx complete
  - Reset
  - … and many more

Detecting Interrupts

- Triggered by pins INT0 - INT7:
  - INT7-4 connected to PORTE pins 7-4.
  - INT3-0 connected to PORTD pins 3-0.
- Latched onto a register, External Interrupt Flag Register (EIFR).
- Triggered by a falling or rising edge or low level
  - Set up by External Interrupt Control Registers (EICR)

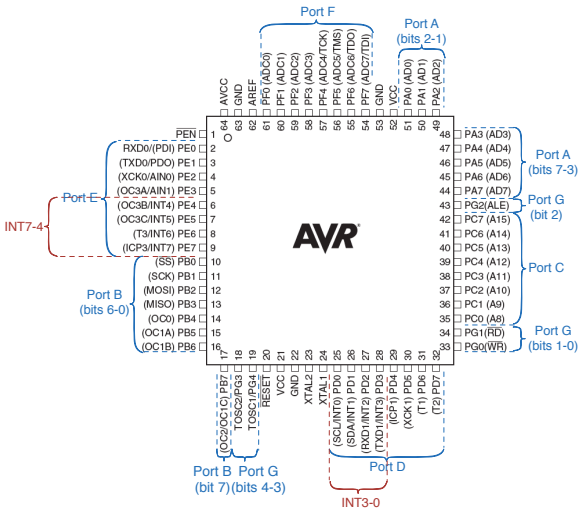
Saving/Restoring the State of the Processor

- The state of the processor is defined by
  - PC
  - SR
  - GPRs
- PC must be saved
  - CPU pushes PC onto the stack and clears I-bit in SREG
  - RETI (Return from interrupt) automatically restores PC from stack and sets I-bit in SREG.
- GPRs and SREG may need to be saved if ISR uses them.
  - Up to the programmer (using PUSH/POP).

Controlling Interrupts

- AVR has many registers that control how, when, and whether interrupts occur.
  - Global Interrupt Enable in SREG
  - External Interrupt Mask Register (EIMSK)
    - Enables/disable individual interrupts
  - External Interrupt Flag Register (EIFR)
    - Indicates which source interrupted
  - External Interrupt Control Registers (EICRA & EICRB)
    - Determines how interrupts should be detected

External Interrupt Pins



Global Interrupt Enable

Located in $35 I/O register space

Status Register (SREG)

7	6	5	4	3	2	1	0
I	T	H	S	V	N	Z	C
R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)

Bit 7 - Global Interrupt Enable
Bit 6 - Bit- Copy Storage
Bit 5 - Half Carry Flag
Bit 4 - Sign Bit
Bit 3 - Two's Complement Overflow Flag
Bit 2 - Negative Flag
Bit 1 - Zero Flag
Bit 0 - Carry Flag

Global Interrupt Enable
I = 1 enables interrupt
I = 0 disables interrupt

Can be set/cleared using SEI/CLI instructions

I-bit cleared after an interrupt
I-bit set by RETI

Interrupt Control Registers

External Interrupt Flag Register (EIFR)

7	6	5	4	3	2	1	0	
INTF7	INTF6	INTF5	INTF4	INTF3	INTF2	INTF1	INTF0	EIFR
R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	

INTFn = 1 Triggers interrupt request

External Interrupt Mask register (EIMSk)

7	6	5	4	3	2	1	0	
INT7	INT6	INT5	INT4	INT3	INT2	INT1	INT0	EIMSK
R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	

INTn = 1 Enables interrupt

External Interrupt Control Register A (EICRA)

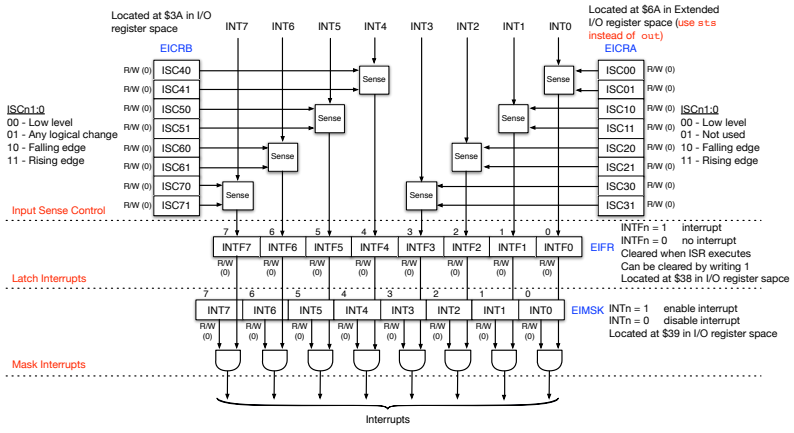
7	6	5	4	3	2	1	0	
ISC31	ISC30	ISC21	ISC20	ISC11	ISC10	ISC01	ISC00	EICRA
R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	

External Interrupt Control Register B (EICRA)

7	6	5	4	3	2	1	0	
ISC71	ISC70	ISC61	ISC60	ISC51	ISC50	ISC41	ISC40	EICRB
R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	

ISCn1:0 External Interrupt n Sense Control Bits
00 - Low level generates an interrupt request.
01 - Reserved (for ISC3-0)
Any logical change on generates an interrupt request (ISC7-4)
10 - Falling edge generates an interrupt request.
11 - Rising edge generates an interrupt request.

Setting External Interrupts



Determining Source of Interrupt

- When an interrupt occurs each source of interrupt is mapped to a vector.

Vector #	Program Address	Source
1	$0000	RESET
2	$0002	INT0
3	$0004	INT1
…	…	…
9	$0010	INT7
10	$0012	TIMER2 COMP
11	$0014	TIMER2 OVF
…	…	…

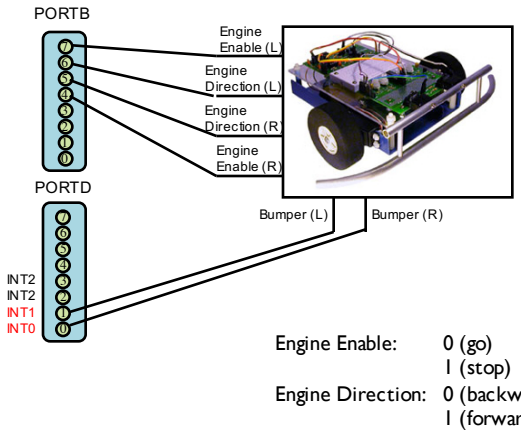
Priority
High
Low

There are 35 vectors!

Program Setup for Interrupt

Address	Label	Code
$0000		`RJMP RESET`
$0002		`RJMP EXT_INT0 ; IRQ0`
$0004		`RJMP EXT_INT1 ; IRQ1`
…		…
$0010		`RJMP EXT_INT7 ; IRQ7`
$0012		`RJMP TIME2_COMP`
$0014		`RJMP TIME2_OVF`
…	…	…
…	…	…
$????	`EXT_INT0:`	`{Your ISR for Ext. Interrupt 0}`

Tekbot Bumper



Engine Enable: 0 (go)
1 (stop)
Engine Direction: 0 (backward)
1 (forward)

Example - Tekbot Whiskers

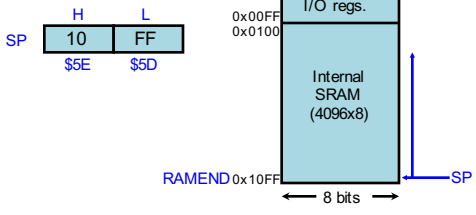
```
;*********************************************************
;*    Start of Code Segment
;*********************************************************
     .cseg                ; Beginning of code segment

;---------------------------------------------------------
; Interrupt Vectors
;---------------------------------------------------------
     .org    $0000        ; Beginning of IVs
     rjmp    INIT         ; Reset interrupt

     .org    $0002 {IRQ0 => pin0, PORTD}
     rcall   HitRight     ; Call hit right function
     reti                 ; Return from interrupt
     .org    $0004 {IRQ1 => pin1, PORTD}
     rcall   HitLeft      ; Call hit left function
     reti                 ; Return from interrupt

     .org    $0046        ; End of Interrupt Vectors
```

Initialization

```
;-----------------------------------------------------
; Program Initialization
;-----------------------------------------------------
INIT: ; The initialization routine
      ; Initialize Stack Pointer
      ldi   mpr, high(RAMEND)
      out   SPH, mpr
      ldi   mpr, low(RAMEND)
      out   SPL, mpr
      ; Initialize Port B for output
      ldi   mpr, (1<<EngBnL)|(1<<EngBnR)|(1<<EngDirR)|(1<<EngDirL)
      out   DDRB, mpr             ; Set the DDR register for Port B
      ldi   mpr, $00
      out   PORTB, mpr            ; Set the default output for Port B
      ; Initialize Port D for input
      ldi   mpr, (0<<WskrL)|(0<<WskrR)
      out   DDRD, mpr             ; Set the DDR register for Port D
      ldi   mpr, (1<<WskrL)|(1<<WskrR)
      out   PORTD, mpr            ; Set the Port D to Input with Hi-Z
      ; Initialize external interrupts
      ; Set the Interrupt Sense Control to falling edge
      ldi   mpr, (1<<ISC01)|(0<<ISC00)|(1<<ISC11)|(0<<ISC10)
      sts   EICRA, mpr            ; Use sts, EICRA in extended I/O space
      ; Set the External Interrupt Mask
      ldi   mpr, (1<<INT0)|(1<<INT1)
      out   EIMSK, mpr
      ; Turn on interrupts
      sei
```


Initialization: Stack

```
;-----------------------------------------------------
; Program Initialization
;-----------------------------------------------------
INIT: ; The initialization routine
      ; Initialize Stack Pointer
      ldi   mpr, high(RAMEND)
      out   SPH, mpr
      ldi   mpr, low(RAMEND)
      out   SPL, mpr
```



Initialization: Ports

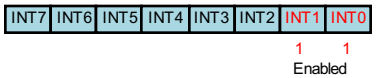
```
; Initialize Port B for output

                Bit 7       Bit 4       Bit 5       Bit 6
ldi   mpr, (1<<EngEnL)|(1<<EngEnR)|(1<<EngDirR)|(1<<EngDirL)  DDRB = 11110000 for output
out   DDRB, mpr                ; Set the DDR register for Port B

; Initialize Port D for input

                Bit 1      Bit 0
ldi   mpr, (0<<WskrL)|(0<<WskrR)    DDRE = 00000000 for input
out   DDRD, mpr                ; Set the DDR register for Port D
ldi   mpr, (1<<WskrL)|(1<<WskrR)
out   PORTD, mpr               ; Set the Port D to Input with Hi-Z
```


Initialization: Interrupts

```
; Initialize external interrupts
; Set the Interrupt Sense Control to low level
ldi   mpr, (1<<ISC01)|(0<<ISC00)|(1<<ISC11)|(0<<ISC10)
sts   EICRA, mpr  => EICRB = 00001010
; Set the External Interrupt Mask
ldi   mpr, (1<<INT0)|(1<<INT1)
out   EIMSK, mpr  => EIMSK = 00000011
; Turn on interrupts
sei
```

External Interrupt Control RegisterA (EICRA)

External Interrupt Mask Register (EIMSK)



Main Program

```
;-----------------------------------------------------------
; Main Program
;-----------------------------------------------------------
MAIN:
        ; Move Robot Forward
        ldi     mpr, MovFwd         ; Load FWD command
        out     PORTB, mpr          ; Send to motors

        rjmp    MAIN                ; Infinite loop. End of the program.
```


HitRight Subroutine

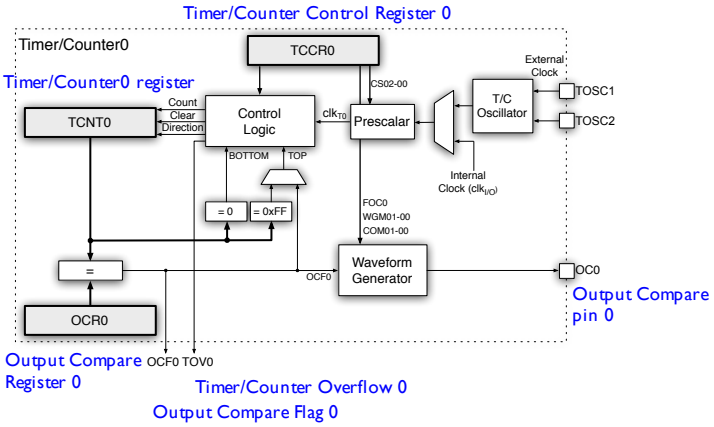
```
;-----------------------------------------------------------
; Sub:      HitRight
; Desc:     Handles functionality of the TekBot when the right whisker is triggered.
;-----------------------------------------------------------
HitRight:
          push      mpr                 ; Save mpr register
          push      waitcnt             ; Save wait register
          in        mpr, SREG           ; Save program state
          push      mpr                 ;
          ; Move Backwards for a second
          ldi       mpr, MovBck         ; Load Move Backwards command
          out       PORTB, mpr          ; Send command to port
          ldi       waitcnt, Wtime      ; Wait for 1 second
          rcall     Wait                ; Call wait function
          ; Turn left for a second
          ldi       mpr, TurnL          ; Load Turn Left Command
          out       PORTB, mpr          ; Send command to port
          ldi       waitcnt, Wtime      ; Wait for 1 second
          rcall     Wait                ; Call wait function
          pop       mpr                 ; Restore program state
          out       SREG, mpr           ;
          pop       waitcnt             ; Restore wait register
          pop       mpr                 ; Restore mpr
          ret                           ; Return from subroutine
```

5.4 Timers/Counters

Timers/Counters

- One of the most widely used features in a microcontroller.
  - Measure some elapsed time, e.g., Tekbot turning left or right for 1 sec.
  - Generate periodic outputs, e.g., pulse train.
- AVR ATmega128 has four Timers/Counters:
  - Two 8-bit Timer/Counter0 and 2 (TCNT0 & 2)
  - Two 16-bit Timer/Counter1 and 3 (TCNT1 & 3)
- Will only discuss Timer/Counter0!

Timer/Counter0

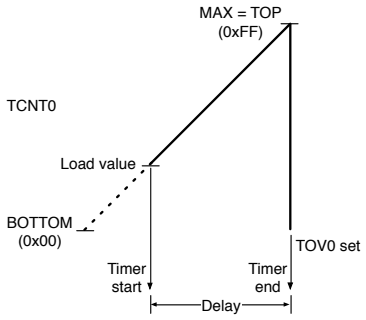


Using Timer/Counter0

- Modes of operation
  - Normal
  - Clear Timer on Compare match (CTC)
  - Fast Pulse Width Modulation (Fast PWM)
  - Phase Correct PWM
  - Phase and Frequency Correct PWM
- The first three modes are the most commonly used
  - Will not discuss the last two.

Normal Mode

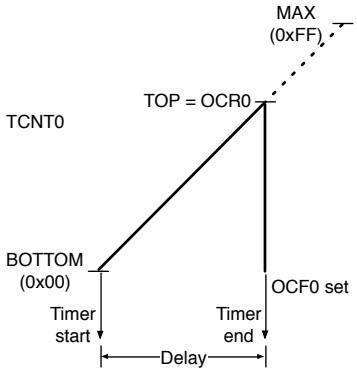
- Load a value and let it count up to MAX.
  - Timer/Counter0 & 2 => MAX = 0xFF
  - Timer/Counter1 & 3 => MAX = 0xFFFF
- TOV0 is set when TCNT0 reaches MAX.



$$Delay_{Normal} = \frac{(MAX - value) \cdot prescale}{clk_{I/O}}$$

CTC Mode

- Load OCR0 with a value.
- TCNT0 starts at 0 and counts up to OCR0 = TOP.
- OCF0 is set when TCNT0 reaches TOP.



$$Delay_{CTC} = \frac{TOP \cdot prescale}{clk_{I/O}}$$

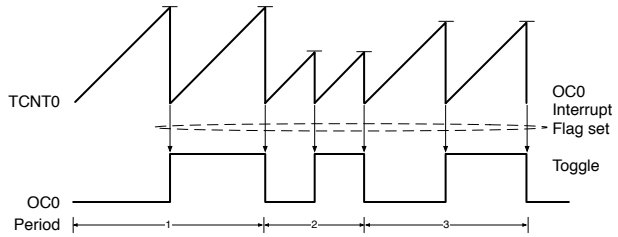
Using Timer/Counter0

- Suppose we want to generate a delay of 10 ms in Normal mode with a system clock of 16 MHz.

$$value = 255 - \frac{10ms}{prescale \cdot 62.5ns}$$

- Need to choose prescale = 1, 8, 32, 64, 128, 256, or 1024
  - Smaller the prescale, higher the resolution
- Prescale of 1024 is the only one that generates a positive value.
- value = $\lceil 255 - (10\ ms/64\ \mu s) \rceil$ = 99!

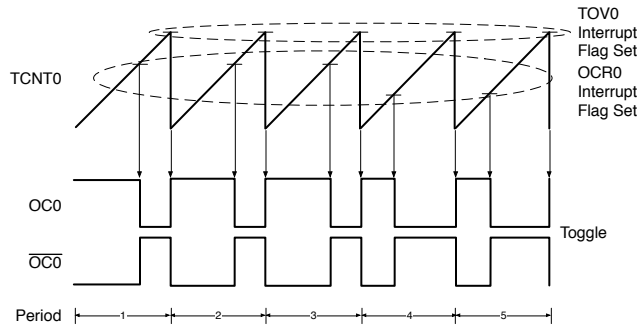
Waveform Generation using CTC Mode



- Waveform generation on OC0 pin.
- TCNT0 starts at 0 and counts up to TOP =OCR0.
- When OCF0 is set, OC0 can be toggled, set, or cleared, and TCNT0 restarts.
- Can generate waveform with fix (50%) duty cycle and varying frequency

$$f_{CTC} = \frac{clk_{I/O}}{2 \cdot prescale \cdot (1 + OCR0)}$$

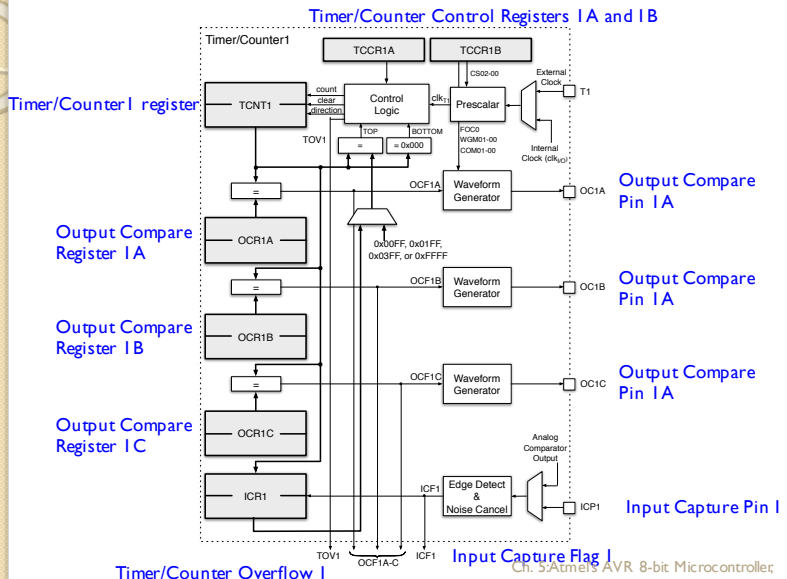
Fast PWM Mode



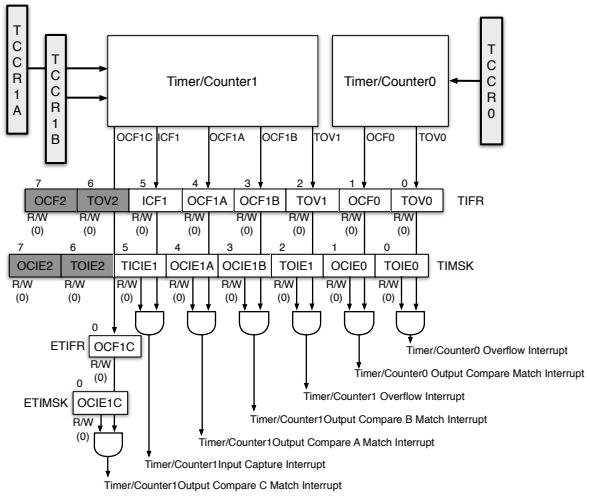
- Uses both OCF0 and TOV0
  - Clear OC0 on output compare match, set OC0 at TOP
- Can generate waveform with varying duty cycle and fixed frequency

$$f_{PWM} = \frac{clk_{I/O}}{prescale \cdot 256}$$

Timer/Counter1



TIFR and TIMSK



Generating 1 sec. delay

- Let's write a subroutine called `wait_1sec` that waits for 1 sec. before returning.

```
; AVR assembly code - Wait one second
WAIT_1sec:
   LDI   R17, 100                  ; Load count = 100
WAIT_10msec:
   LDI   R16, 99           ; Value for delay = 99
   OUT   TCNT0, R16                ; (Re)load a value for delay
LOOP:
   IN    R18, TIFR                 ; Read in TOV0
   ANDI  R18, 0b00000001   ; Check if its set
   BREQ  LOOP                      ; Loop if TOV0 not set
   LDI   R18, 0b00000001   ; Reset TOV0
   OUT   TIFR, R18                 ; Note - write 1 to reset
   DEC   R17                       ; Decrement count
   BRNE  WAIT_10msec               ; Loop if count not equal to 0
   RET
```

Controlling Timer/Counter0

Timer/Counter Control Register 0 (TCCR0)

7	6	5	4	3	2	1	0	
FOC0	WGM00	COM01	WGM01	COM00	CS02	CS01	CS00	TCCR0
W (0)	R/W (0)	R/W (0)	R/W (0)	R/0 (0)	R/W (0)	R/W (0)	R/W (0)	

Bit 7 - Force Output Compare
Bit 6, 4 - Waveform Generation Mode
Bit 5, 3 - ADC Free Running Select
Bit 2:0 - Clock Select

Clock Select bits

CS02	CS01	CS00	Description
0	0	0	No clock source
0	0	1	clkI/O
0	1	0	clkI/O/8
0	1	1	clkI/O/32
1	0	0	clkI/O/64
1	0	1	clkI/O/128
1	1	0	clkI/O/256
1	1	1	clkI/O/1024

Waveform Generation Mode bits

WGM01	WGM00	Mode
0	0	Normal
0	1	Phase Correct PWM
1	0	CTC
1	1	Fast PWM

Compare Output Mode bits

COM01	COM00	Normal	CTC	FAST PWM
0	0	Normal port operation, OC0 disconnected		
0	1	Toggle OC0 on compare match		Reserved
1	0	Clear OC0 on compare match		Clear OC0 on compare match, set OC0 at TOP
1	1	Set OC0 on compare match		Set OC0 on compare match, clear OC0 at TOP

Controlling Timer/Counter1

7	6	5	4	3	2	1	0	
COM1A1	COM1A0	COM1B1	COM1B0	COM1C1	COM1C0	WGM11	WGM10	TCCR1A
R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/0 (0)	R/W (0)	R/W (0)	R/W (0)	

7	6	5	4	3	2	1	0	
ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
R/W (0)	R/W (0)	R (0)	R/W (0)	R/0 (0)	R/W (0)	R/W (0)	R/W (0)	

Clock Select bits

CS12	CS11	CS10	Description
0	0	0	No clock source
0	0	1	clkI/O
0	1	0	clkI/O/8
0	1	1	clkI/O/32
1	0	0	clkI/O/64
1	0	1	clkI/O/128
1	1	0	clkI/O/256
1	1	1	clkI/O/1024

Waveform Generation Mode bits

WGM13-10	Mode	TOP	TOVn set on
0000	Normal	0xFFFF	MAX
0100	CTC	OCRnA	MAX
0101	Fast PWM, 8-bit	0x00FF	TOP
0110	Fast PWM, 9-bit	0x01FF	TOP
0111	Fast PWM, 10-bit	0x03FF	TOP
1100	CTC	ICRn	MAX
1110	Fast PWM	ICRn	TOP
1111	Fast PWM	OCRnA	TOP

Compare Output Mode bits

COM1A-C1	COM1A-C0	Normal	CTC	FAST PWM
0	0	Normal port operation, OC1A-C disconnected		
0	1	Toggle OC1A-C on compare match		Reserved
1	0	Clear OC1A-C on compare match		Clear OC1A-C on compare match, set OC1A-C at TOP
1	1	Set OC1A-C on compare match		Set OC1AC on compare match, clear OC1A-C at TOP

Code Examples

- Turning on and off LED connected to the OC0 pin:
  - Manually turn on OC0 for 500 ms.
  - Manually toggle OC0 for 500 ms.
  - Fast PWM mode
  - Fast PWM with adjustable duty cycle

Code Example 1 (1)

```
; AVR Assembly Code - Turn on OC0 for 500 ms
; (Normal mode, OC0 disconnected)
.INCLUDE "m128def.inc"
.DEF A = R16                    ; General purpose register A
.DEF B = R17                    ; General purpose register B

.ORG $0000                      ; Reset and Power On interrupt
       RJMP INITIALIZE          ; Jump to initialization
.ORG $0046                      ; End of interrupt vectors

INITIALIZE:
       ; Initialize stack
       LDI     A, high(RAMEN
       OUT     SPH, A
       LDI     A, low(RAMEND)
       OUT     SPL, A
       ; Initialize TCNT0
       SBI     DDRB, PB4        ; Set bit 4 of port B (OC0) for output
       LDI     A, 0b00000111    ; Activate Normal mode, OC0 disconnected,
       OUT     TCCR0, A         ; and set prescalar to 1024
```

7	6	5	4	3	2	1	0	
FOC0	WGM00	COM01	WGM01	COM00	CS02	CS01	CS00	TCCR0
0	0	0	0	0	1	1	1	

CS02:00 = 111 => prescalar = 1024
WGM01:00 = 00 => Normal
COM01:00 = 00 => Normal port operation, OC0 disconnected

Code Example 1 (2)

```
MAIN:
        SBI     PORTB, PB4          ; Turn on OC0
        RCALL   WAIT_0.5sec         ; Call WAIT_0.5sec subroutine
        CBI     PORTB, PB4          ; Turn off OC0
LOOP:
        RJMP    LOOP                ; Loop forever
; Subroutine to wait for 500 ms
WAIT_0.5msec:
        LDI     B, 50               ; Load loop count = 50
WAIT_10msec:
        LDI     A, 99               ; (Re)load value for delay
        OUT     TCNT0, A
; Wait for TCNT0 to roll over
CHECK:
        IN      A, TIFR             ; Read in TIFR
        ANDI    A, 0b00000001       ; Check if TOV0 set
        BREQ    CKECK               ; Loop if TOV0 not set
        LDI     A, 0b00000001       ; Otherwise, Reset TOV0
        OUT     TIFR, A             ; Note - write 1 to reset
        DEC     B                   ; Decrement count
        BRNE    WAIT_10msec         ; Loop if count not equal to 0
        RET
```

10 ms delay

Can also use
SBIS TIFR, TOV0
RJMP CHECK

Wait for TOV0 to be set

Can also use
CBI TIFR, TOV0

Code Example 2 (1)

```
; AVR Assembly Code - Manually toggle OC0 every 500 ms
; (CTC mode, OC0 disconnected)
.INCLUDE "m128def.inc"
.DEF A = R16                ; General purpose register A
.DEF B = R17                ; General purpose register B
.ORG $0000                  ; Reset and Power On interrupt
     RJMP INITIALIZE        ; Jump to initialization
.ORG $0046                  ; End of interrupt vectors
INITIALIZE:
; Initialize stack
...
; Initialize TCNT0
     SBI   DDRB, PB4        ; Set bit 4 of port B (OC0) for output
     LDI   A, 0b00010111    ; Activate CTC mode, OC0 disconnected,
     OUT   TCCR0, A         ; and set prescalar to 1024
     LDI   A, 157           ; Set output compare value
     OUT   OCR0, A
MAIN_LOOP:
     RCALL TOGGLE           ; Call TOGGLE subroutine
     RCALL WAIT_0.5sec      ; Call WAIT_0.5sec subroutine
     RJMP  MAIN_LOOP        ; Loop forever
```

10 ms delay

7	6	5	4	3	2	1	0	
FOC0	WGM00	COM01	WGM01	COM00	CS02	CS01	CS00	TCCR0
0	0	0	1	0	1	1	1	

CS02:00 = 111 => prescalar = 1024,WGM01:00 = 10 => CTC
COM01:00 = 00 => Normal port operation, OC0 disconnected

Code Example 2 (2)

```
; Subroutine to toggle OC0
TOGGLE:
     IN    A, PORTB          ; Get current OC0 value
     LDI   B, (1 << PB4)     ; Set bit to toggle
     EOR   A,B               ; Toggle OC0
     OUT   PORTB, A          ; Write it back
     RET
; Subroutine to wait for 500 ms
WAIT_0.5sec:
     LDI   B, 50             ; Load loop count = 50
Wait_10msec:
     LDI   A, 0              ; Initialize TCNT0 to 0
     OUT   TCNT0, A          ; Wait for TCNT0 to match with OCR0
LOOP:
     IN    A, TIFR           ; Read in OCF0 in TIFR
     ANDI  A, 0b00000010     ; Check if OCF0 set
     BREQ  LOOP              ; Loop if OCF0 not set
     LDI   A, 0b00000010     ; Otherwise, reset OCF0
     OUT   TIFR, A           ; Note - write 1 to reset
     DEC   B                 ; Decrement count
     BRNE  WAIT_10msec       ; Loop if count not equal to 0
     RET
```

0 ⊕ 1 = 1
1 ⊕ 1 = 0

Wait for OCF0 to be set

Code Example 3

```
; AVR Assembly code - Fast PWM mode
.INCLUDE "m128def.inc"
.DEF A = R16 ; General purpose register
.ORG $0000                   ; Reset and Power On interrupt
     RJMP INITIALIZE         ; Jump to initialization
.ORG $0046                   ; End of interrupt vectors
INITIALIZE:
     ; Initialize stack
     ...
     SBI   DDRB, PB4         ; Set bit 4 of port B (OC0) for output
     LDI   A, 0b01110111     ; Activate Fast PWM mode with toggle
     OUT   TCCR0, A          ; (non-inverting), and set prescalar to 1024
     LDI   A, 128            ; Set compare value
     OUT   OCR0, A
MAIN_LOOP:                   ; Do nothing loop
     RJMP  MAIN_LOOP
```

Half duty cycle

7	6	5	4	3	2	1	0	
FOC0	WGM00	COM01	WGM01	COM00	CS02	CS01	CS00	TCCR0
0	1	1	1	0	1	1	1	

CS02:00 = 111 => prescalar = 1024
WGM01:00 = 11 => Fast PWM
COM01:00 = 10 => Clear OC0 on compare match, set OC0 at TOP

Code Example 4

Intensity of LED slowly increases within a time span of 16.38 ms × 256= 4.19 sec.

```
; AVR Assembly code - Fast PWM mode with adjustable duty cycle
.INCLUDE "m128def.inc"
.DEF A = R16                ; General purpose register
.ORG $0000                  ; Reset on Power On interrupt
     RJMP  INITIALIZE       ; Jump to initialization
.ORG $001E                  ; Compare Match vector
     RJMP  TIM0_COMPA
.ORG $0046                  ; End of interrupt vectors
INITIALIZE:
     SBI   DDRB, PB4        ; Set bit 4 of port B (OC0) for output
     LDI   A, 0b01110111    ; Activate Fast PWN mode with toggle
     OUT   TCCR0, A         ; (non-inverting) & set prescalar to 1024
     LDI   A, 0b00000010    ; Enable output compare interrupt
     OUT   TIMSK, A
     SEI                    ; Enable global interrupt
MAIN_LOOP:
     RJMP  MAIN_LOOP
TIM0_COMPA:
     IN    A, OCR0
     INC   A
     OUT   OCR0, A
     RETI
```

Timer/Counter0 Output Compare Match Interrupt vector

Enable Timer/Counter0 Output Compare Match Interrupt

OCF0

7	6	5	4	3	2	1	0	
FOC0	WGM00	COM01	WGM01	COM00	CS02	CS01	CS00	TCCR0
0	1	1	1	0	1	1	1	

CS02:00 = 111 => prescalar = 1024
WGM01:00 = 11 => Fast PWM
COM01:00 = 10 => Clear OC0 on compare match, set OC0 at TOP

5.5 USART

Introduction

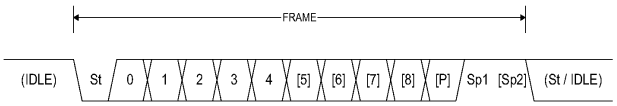
- Universal Synchronous/Asynchronous Receiver/Transmitter (USART) can be used to transmit and receive data serially to and from other devices.
  - Supported by many embedded I/O devices and sensors, e.g., Bluetooth, Infrared, RFID reader, GPS, GSM, etc.
- In older computers, devices such as mice, printers, and dial-up modems used USART to communicate via a serial port using the RS-232 protocol.
- USB has since displaced the serial port. But, still used in many test and measurement equipment, industrial machines, and networking equipment.

Serial Communications Basics

- How does the receiver know when the data being transmitted starts and ends?
  - Serial frame format
- How fast are the bits transmitted/received, and how does the sender and receiver agree on a transmission rate?
  - Synchronous vs. asynchronous
  - Baud rate

Serial Data Frame Format

Serial Data Frame Format



St	Start bit, always low.
(n)	Data bits (0 to 8).
P	Parity bit. Can be odd or even.
Sp	Stop bit, always high.
IDLE	No transfers on the communication line (RxD or TxD). An IDLE line must be high.

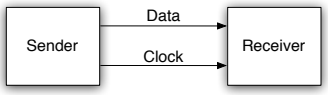
Parity (P) bit

$P_{even} = d_{n-1} \oplus \ldots \oplus d_2 \oplus d_1 \oplus d_0 \oplus 0$
$P_{odd} = d_{n-1} \oplus \ldots \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1$

Example:
00101101 => odd parity => P-bit = 1

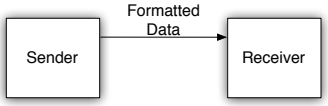
Synchronous vs. Asynchronous

Synchronous



Sender generates the clock

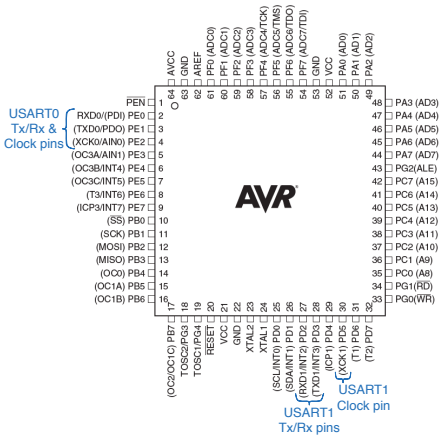
Asynchronous



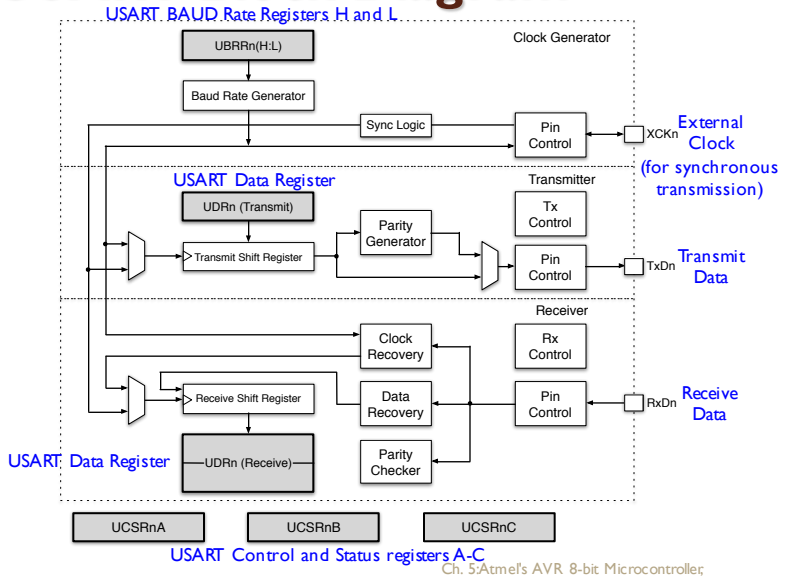
Both sender and receiver agree on the clock rate, Baud rate

AVR's USART Pins

Two identical USARTs – USART0 and USART 1



USART Block Diagram



Control and Status Registers

USART Control and Status Register A (UCSRnA)

7	6	5	4	3	2	1	0
RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn
R (0)	R/W (0)	R (1)	R (0)	R (0)	R (0)	R/W (0)	R/W (0)

Bit 7 - USART Receive Complete
Bit 6 - USART Transmit Complete
Bit 5 - USART Data Register Empty
Bit 4 - Frame error
Bit 3 - Data OverRun
Bit 2 - Parity Error
Bit 1 - Double USART Transmission Speed
Bit 0 - Multi-Processor Communication Mode

USART Control and Status Register B (UCSRnB)

7	6	5	4	3	2	1	0
RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n
R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R (0)	R/W (0)

Bit 7 - RX Complete Interrupt Enable
Bit 6 - TX Complete Interrupt Enable
Bit 5 - USART Data Register Empty Interrupt Enable
Bit 4 - Receiver Enable
Bit 3 - Transmitter Enable
Bit 2 - Character Size (combine with UCSZn1:0 in UCSRnC)
Bit 1 - Receive Data Bit 8

USART Control and Status Register C (UCSRnC)

7	6	5	4	3	2	1	0
-	UMSELn	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn
R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (1)	R/W (1)	R/W (0)

Bit 7 - Reserved Bit
Bit 6 - USART Mode Select
Bit 5:4 - Parity mode
Bit 3 - Stop bit select
Bit 2:1 - Character size UCSZn2:0
Bit 0 - Clock Polarity

USART Control and Status Registers A-C control:

- Synchronous vs. Asynchronous Mode
- Data Frame Format
- Baud Rate
- Transmitter and Receiver Enable
- Data Transmitted or Received Status
- Interrupts
- Error Reporting

Synchronous vs. Asynchronous Mode

USART Control and Status Register C (UCSRnC)

7	6	5	4	3	2	1	0
-	UMSELn	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn
R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (1)	R/W (1)	R/W (0)

Bit 6 – USARTn Mode Select
Bit 0 – USARTn Clock Polarity

Control Bits for Transmission Mode

Control bits	Bit	Meaning
UMSELn	0	Asynchronous mode
	1	Synchronous mode
UCPOLn	0	Data changes/sampled at rising/falling XCK edge
(synchronous)	1	Data changes/sampled at falling/rising XCK edge

Data Frame Format

USART Control and Status Register B (UCSRnB)

7	6	5	4	3	2	1	0
RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n
R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R (0)	R/W (0)

This is where the 8th bit of 9-bit data is held for receive and transmit

Bit 2 – USARTn Character SiZe bit 2

USART Control and Status Register C (UCSRnC)

7	6	5	4	3	2	1	0
-	UMSELn	UPMn1	UPMn0	USBSn	UCSZn1	UCSZn0	UCPOLn
R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (1)	R/W (1)	R/W (0)

Bits 5 & 4 – USARTn Parity mode bits 1 & 0
Bit 3 – USARTn Stop Bit Select
Bits 2 & 1 – USARTn Character SiZe bits 1 & 0

Control Bits for Data Frame Format

Control bits	Bits	Meaning
UCSZn2:0	000	5-bit
	001	6-bit
	010	7-bit
	011	8-bit
	100	Reserved
	101	Reserved
	110	Reserved
	111	9-bit
UPMn1:0	00	No parity
	10	Even parity
	11	Odd parity
USBSn	0	1 stop bit
	1	2 stop bits

Baud Rate (1)

- Baud rate refers to the rate at which symbols are transmitted, measured in symbols per second, and includes the synchronization bits, i.e., start bit and stop bit(s).
  - e.g., if 10-bit symbol is used per 8-bit character at 9600 baud, then this equates to 960 Bytes per second or a bit rate of 7680 bps.
- Baud rate for asynchronous mode:

$$Baud\ Rate = \frac{f_{CLK}}{16 \cdot (UBRR + 1)}$$

Baud rate divider; USART Baud Rate Register

$$UBRR = \frac{f_{CLK}}{16 \cdot (Baud\ Rate)} - 1$$

Baud Rate (2)

- Example: Required value for UBRR for a Baud rate of 2,400 baud and f_{CLK} of 16 MHz:

$$UBRR = \frac{16MHz}{16 \cdot (2400)} - 1 = 416 = 0x01A0$$

- The transfer rate can also be doubled by setting the Asynchronous Double Speed Mode (U2Xn) bit in UCSRnA - Baud rate divider becomes 8.
- Baud rate for synchronous mode:

$$Baud\ Rate = \frac{f_{CLK}}{2 \cdot (UBRR + 1)}$$

$$UBRR = \frac{f_{CLK}}{2 \cdot (Baud\ Rate)} - 1$$

Tx and Rx Enable

USART Control and Status Register B (UCSRnB)

7	6	5	4	3	2	1	0
RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n
R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R (0)	R/W (0)

Bit 4 – RX Complete Interrupt Enable
Bit 3 – TX Complete Interrupt Enable

Data Transmitted or Received Status

USART Control and Status Register A (UCSRnA)

7	6	5	4	3	2	1	0
RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn
R (0)	R/W (0)	R (1)	R (0)	R (0)	R (0)	R/W (0)	R/W (0)

Bit 7 – USARTn Receive Complete
Bit 6 – USARTn Transmit Complete
Bit 5 – USARTn Data Register Empty

Status Bits for Transmission and Reception

Status bits	Bit	Meaning
RXCn	0	Receive incomplete
	1	Receive complete
TXCn	0	Transmit incomplete
	1	Transmit complete
UDREn	0	UDRn (receive buffer) full
	1	UDRn (receive buffer) empty

Interrupts

USART Control and Status Register B (UCSRnB)

7	6	5	4	3	2	1	0
RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n
R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R/W (0)	R (0)	R/W (0)

Bit 7 – Rx Complete Interrupt Enable
Bit 6 – Tx Complete Interrupt Enable
Bit 5 – USART Data Register Interrupt Enable

Allow RXCn, TXCn, and UDREn signals to generate interrupts

Vector #	Address	
…		
19	$0024	=> UASRT0, Rx Complete
20	$0026	=> UASRT0, Data Register Empty
21	$0028	=> UASRT0, Tx Complete
…		

Error Reporting

USART Control and Status Register A (UCSRnA)

7	6	5	4	3	2	1	0
RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn
R (0)	R/W (0)	R (1)	R (0)	R (0)	R (0)	R/W (0)	R/W (0)

Bit 4 – Frame Error
Bit 3 – Data OverRun
Bit 2 – Parity Error

- Frame Error – first stop bit is low (it should be high) for the next character in UDREn.
- Data OverRun – UDRn (Receive) is full, a new character is waiting in the Receive Shift Register, and a new start bit is detected.
- Parity Error – parity error and parity checking was enabled (UPMn1:0 in UCSRnC).

Programming Model - USART Initialization (1)

```
; AVR Assembly Code - USART0 Initialization
.include "m128def.inc"              ; Include definition file
.def mpr = r16                      ; Multi-purpose register

.ORG $0000                          ; Reset and Power On interrupt
      RJMP   INITIALIZE             ; Jump to initialization
.ORG $0024                          ; USART0, Rx complete interrupt
      RCALL  USART_Receive          ; Call USART_Receive
      RETI                          ; Return from interrupt

INITIALIZE:
      ; Initialize stack
      LDI    mpr, high(RAMEND)
      OUT    SPH, mpr
      LDI    mpr, low(RAMEND)
      OUT    SPL, mpr
      ; Initialize I/O Ports
      ldi    mpr, (1<<PE1)          ; Set Port E pin 0 (RXD0) for input
      out    DDRE, mpr              ; and Port E pin 1 (TXD0) for output
```

Programming Model - USART Initialization (2)

```
; Initialize USART0
ldi     mpr, (1<<U2X0)        ; Set double data rate
out     UCSR0A, mpr
; Set baudrate at 2400
ldi     mpr, high(832)        ; Load high byte of 0x0340
sts     UBRR0H, mpr           ; UBRR0H in extended I/O space
ldi     mpr, low(832)         ; Load low byte of 0x0340
out     UBRR0L, mpr
; Set frame format: 8 data, 2 stop bits, asynchronous
ldi     mpr, (0<<UMSEL0 | 1<<USBS0 | 1<<UCSZ01 | 1<<UCSZ00)
sts     UCSR0C, mpr           ; UCSR0C in extended I/O space
; Enable both receiver and transmitter, and receive interrupt
ldi     mpr, (1<<RXEN0 | 1<<TXEN0 | 1<<RXCIE0)
out     UCSR0B, mpr           ;
```


Programming Model - Sending Data

```
; AVR Assembly Code - USART Transmit
USART_Transmit:
      sbis    UCSR0A, UDRE0       ; Loop until UDR0 is empty
      rjmp    USART_Transmit
      out     UDR0, r17           ; Move data to transmit data buffer
      ret
```

Polling on UDRE0 flag

USART Control and Status Register A (UCSRnA)

7	6	5	4	3	2	1	0
RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn
R (0)	R/W (0)	R (1)	R (0)	R (0)	R (0)	R/W (0)	R/W (0)

Set when UDR0 is empty
Cleared when UDR0 is written

Programming Model - Receiving Data

```
; AVR Assembly Code - USART Receive (interrupt driven)
USART_Receive:
        push    mpr
        in      r17, UDR0       ; Read data from Receive Data Buffer
        pop     mpr
        ret
```

USART Control and Status Register A (UCSRnA)

7	6	5	4	3	2	1	0
RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn
R (0)	R/W (0)	R (1)	R (0)	R (0)	R (0)	R/W (0)	R/W (0)

Set when there is data to be read in UDR0

Interrupt driven – jumps to this interrupt service routine when RXC0 is set.

Questions?