Jonathan Machlin

ATmega32 Memory Map

Address 16K x 16	**Program Memory** (*word* organized)
$0000	Applications Section (your program is stored here)
$3fff	Boot Loader Section

← 16-bit word →

Address 2144 x 8	**Data Memory** (*byte* organized)
$0000 $001f	32 General Purpose Registers (r0-r31)
$0020 $005f	64 I/O Registers (see back for names)
$0060 $085f	2K x 8 SRAM (temporary data, stacks, etc.)

← 8-bit byte →

Address 1K x 8	**EEPROM Memory** (*byte* organized)
$0000 $03ff	EEPROM memory

← 8-bit byte →

configuration constants

Program Memory and **EEPROM Memory** are *non-volatile*

(contents are retained when powered off)

Data Memory is *volatile*

(contents are lost when powered off)

(most I/O Registers are cleared to $00 by reset or power on)

Special **General Purpose Registers**

r0 and r1 are used specifically by some instructions

some instructions limit operands to r16-r31, r16-r23, or r(even ≥24)

For indirect addressing → X = r27:r26, Y = r29:r28, Z = r31:r30

Johnathan Muchler

ATmega32 Arithmetic Instructions

subtracted immediate data negated = ~~addition~~ add 1

modified if condition is met

Addition:	(Z,C,N,V,H modified, 1 clock)		
`add Rd,Rr`	Add	Rd ← Rd + Rr	
`adc Rd,Rr`	Add with carry	Rd ← Rd + Rr + previous carry	
Subtraction:	(Z,C,N,V,H modified, 1 clock)		
`sub Rd,Rr`	Subtract	Rd ← Rd–Rr	
`subi Rd,K`	Subtract immediate	Rd ← Rd–K	; *r16-r31 only*
`sbc Rd,Rr`	Subtract with carry	Rd ← Rd–Rr–previous carry	
`sbci Rd,K`	Subtract immediate with carry	Rd ← Rd–K–previous carry	; *r16-r31 only*

carry flag is most significant bit of a product

Multiplication:	(Z,C modified, 2 clocks)		
`mul Rd,Rr`	Multiply unsigned	r1:r0 ← Rd (unsigned) * Rr (unsigned)	
`muls Rd,Rr`	Multiply signed	r1:r0 ← Rd (2's comp) * Rr (2's comp)	; *r16-r31 only*
`mulsu Rd,Rr`	Multiply s/unsigned	r1:r0 ← Rd (2's comp) * Rr (unsigned)	; *r16-r23 only*
Other:	(Z,N,V modified, 1 clock)		
`inc Rd`	Increment	Rd ← Rd + 1	
`dec Rd`	Decrement	Rd ← Rd – 1	
`clr Rd`	Clear	Rd ← $00	
Other:	(no flags modified, 1 clock)		
`ser Rd`	Set	Rd ← $ff	; *r16-r31 only*

set register doesnt modify any flags —

Other:	(Z,C,N,V,H modified, 1 clock)		
`neg Rd`	Negate	Rd ← $00 – Rd	

invert all bits
inc

com r16
inc r16

neg r16

Key:

Rd = destination general purpose register (r0-r31)
Rr = source general purpose register (r0-r31)
K = 8-bit data ($00-$ff, or 0b00000000-0b11111111, or 0-255)

Notes:

- There is no "add immediate data" instruction, so use subtract immediate with negated data
- Addition and Subtraction instructions work on both unsigned and 2's complement data
- Separate multiply instructions support different combinations of unsigned/2's complement data
- Multiply results are 16 bits, always in r1:r0 (r1 holds the high byte, r0 holds the low byte)
- subi, sbci, muls, mulsu, and ser have restricted ranges of General Purpose Register operands

ATmega32 Logic Instructions

Or

And:	(Z,N,V modified, 1 clock)		
`and Rd,Rr`	And	Rd ← Rd & Rr	
`andi Rd,K`	And immediate	Rd ← Rd & K	; <u>*r16-r31 only*</u>
Or:	(Z,N,V modified, 1 clock)		
`or Rd,Rr`	Or	Rd ← Rd | Rr	
`ori Rd,K`	Or immediate	Rd ← Rd | K	; <u>*r16-r31 only*</u>
Exclusive Or:	(Z,N,V modified, 1 clock)		
`eor Rd,Rr`	Exclusive Or	Rd ← Rd ⊕ Rr	
sets carry flag **Complement:**	(Z,C,N,V modified, 1 clock)		
`com Rd`	Complement	Rd ← $ff – Rd	
Bit Modify:	(Z,N,V modified, 1 clock)	OrI	low half
`sbr Rd,K`	COM r16 Set bits in general purpose register	Rd ← Rd | K	; <u>*r16-r31 only*</u>
`cbr Rd,K`	ANDI r16 Clear bits in general purpose register	Rd ← Rd&($ff–K)	; <u>*r16-r31 only*</u>

orI   bit number

overflow is cleared always
→ write equivalent

A

ANI

```
clr r16        andI r16,$fE
com r16        cbr r16,$01
Andi r16
```

Key:

Rd = destination general purpose register (r0-r31)

Rr = source general purpose register (r0-r31)

K = 8-bit data ($00-$ff, or 0b00000000-0b11111111, or 0-255)

Notes:

- No interaction between bit columns occurs here, operations occur only bitwise (within a column of bits)
- andi, ori, sbr, and cbr have restricted ranges of General Purpose Register operands
- There is no exclusive or with immediate data
- And'ing is a great way to force specific bits to be zero
- Or'ing is a great way to force specific bits to be one
- Exclusive Or'ing is a great way to complement specific bits

cp GMC Jonathan Mackle

cp Rd,Rr only modifies flags

20 CIC

ATmega32 Branch Instructions

Unconditional: (no flags modified, 2 clocks, 3 clocks for jmp, -2048 to +2047 words for rjmp)

rjmp Label	1 word	Relative jump	PC ← Label
ijmp	1 word	Indirect jump to (Z)	PC ← Z (pointer register)
jmp Label	3 cycles 2 words	Direct jump	PC ← Label

flags are affected

Compare: (Z,N,V,C,H modified, 1 clock) only to modify flags here

Compare = subtraction but throws away ~~result~~ difference flags

sub	cp Rd,Rr	Compare Rd to Rr	Rd – Rr, result discarded	
subc	cpc Rd,Rr	Compare Rd to Rr with carry	Rd – Rr – carry, result discarded	
subi	cpi Rd, K	Compare Rd to immediate data	Rd – K, result discarded	; *r16-r31 only*

Skip: (no flags modified, 1/2/3 clocks)

IF no skip = 1 cycle unless instruction ~~jump~~ takes two words long
IF SKIP = 2 cycles

cpse Rd,Rr	Compare Rd to Rr, skip if equal	if Rd =Rr, PC ← PC+2 or 3	
sbrc Rr,b	Skip if bit in Rr is cleared	if Rr(b)=0, PC ← PC+2 or 3	
sbrs Rr,b	Skip if bit in Rr is set	if Rr(b)=1, PC ← PC+2 or 3	
sbic P,b	Skip if bit in P is cleared	if P(b)=0, PC ← PC+2 or 3	; *P0-P31 only*
sbis P,b	Skip if bit in P is set	if P(b)=1, PC ← PC+2 or 3	; *P0-P31 only*

— low half —

Conditional branch: (no flags modified, 1/2 clocks, -64 to +63 words)

brbs s,Label	Branch if SREG(s)=**1**	if SREG(s) = **1**, PC ← Label
brbc s,Label	Branch if SREG(s)=**0**	if SREG(s) = **0**, PC ← Label
breq Label	Branch if equal	if Z=**1**, PC ← Label
brne Label	Branch if not equal	if Z=**0**, PC ← Label
brcs Label	Branch if carry set	if C=**1**, PC ← Label
brcc Label	Branch if carry cleared	if C=**0**, PC ← Label
brsh Label	Branch if same or higher	if C=**0**, PC ← Label
brlo Label	Branch if lower	if C=**1**, PC ← Label
brmi Label	Branch if minus	if N=**1**, PC ← Label
brpl Label	Branch if plus	if N=**0**, PC ← Label
brge Label	Branch if greater or equal	if S=**0**, PC ← Label
brlt Label	Branch if less than	if S=**1**, PC ← Label
brhs Label	Branch if half carry set	if H=**1**, PC ← Label
brhc Label	Branch if half carry cleared	if H=**0**, PC ← Label
brts Label	Branch if T flag set	if T=**1**, PC ← Label
brtc Label	Branch if T flag cleared	if T=**0**, PC ← Label
brvs Label	Branch if overflow set	if V=**1**, PC ← Label
brvc Label	Branch if overflow cleared	if V=**0**, PC ← Label
brie Label	Branch if interrupts enabled	if I=**1**, PC ← Label
brid Label	Branch if interrupts disabled	if I=**0**, PC ← Label

unsigned

signed

2's complement overflow

Z flag } most used
C flag

cpi r16, 5
breq label

cpi r16, 5
brne label

ldi r16, $FF
cp r6, 5
brsh label
unsigned

ldi r16, $FF
cp r6, 5
brge label
will not branch

for Interrupts

__Key:__

Rd, Rr, P, K, b, s as before. I, T, H, S, V, N, Z, C are flag bits in SREG (except in ijmp).

PC is Program Counter. Many of these instructions are redundant, just provided for clarity in your program

Johnathan Mackle

ATmega32 Flags

?

I	T	H	S	V	N	Z	C

SREG, I/O address $3f
Flags, or Status Register

I – Interrupt enable (**1**) or disable (**0**)
T – Temporary flag for moving bits around
H – Half Carry flag, carry or borrow from low nibble
S – Signed Test flag, always the exclusive OR of N and V
V – Overflow flag, =**1** if result exceeds 2's complement range
N – Negative flag, equal to the most significant bit of result
Z – Zero flag, =**1** if result is all zero's, =**0** if result is not all zero's
C – Carry flag, usually the carry or borrow from most significant bit

should understand

Flag Maniuplation: (only the specified flag is modified, 1 clock)

`bset s`	Flag set	`SREG(s)` ← **1**
`bclr s`	Flag clear	`SREG(s)` ← **0**
`bst Rr,b`	Bit store from `Rr` to `T`	`T` ← `Rr(b)`
`bld Rd,b`	Bit load from `T` to `Rd`	`Rd(b)` ← `T`
`sec`	Set Carry flag	`C` ← **1**
`clc`	Clear Carry flag	`C` ← **0**
`sen`	Set Negative flag	`N` ← **1**
`cln`	Clear Negative flag	`N` ← **0**
`sez`	Set Zero flag	`Z` ← **1**
`clz`	Clear Zero flag	`Z` ← **0**
`sei`	Enable interrupts	`I` ← **1**
`cli`	Disable interrupts	`I` ← **0**
`ses`	Set Signed Test flag	`S` ← **1**
`cls`	Clear Signed Test flag	`S` ← **0**
`sev`	Set Overflow flag	`V` ← **1**
`clv`	Clear Overflow flag	`V` ← **0**
`set`	Set Tempory flag	`T` ← **1**
`clt`	Clear Tempory flag	`T` ← **0**
`seh`	Set Half Carry flag	`H` ← **1**
`clh`	Clear Half Carry flag	`H` ← **0**

Key:

`Rd` = destination general purpose register (`r0-r31`)
`Rr` = source general purpose register (`r0-r31`)
`s`, `b` = bit number (0-7)

Notes:

- Many of these instructions are redundant, just provided for clarity in your program

Johnathan [illegible]

ATmega32 Bit Manipulation Instructions

I/O Bits: (no flags modified, 2 clocks)

`sbi P,b` Set bit in I/O register `P(b)` ← **1** ; *P0-P31 only*

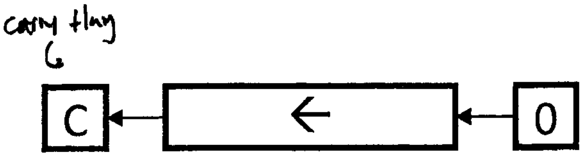
`cbi P,b` Clear bit in I/O register `P(b)` ← **0** ; *P0-P31 only*

Shifts, rotates: Z,C,N,V modified, 1 clock)

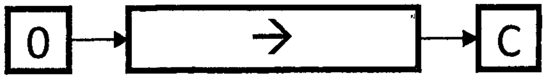
`lsl Rd` Logical shift left

Great for multiplication and multiples of 2

msb bit changes shifted bit flag is set

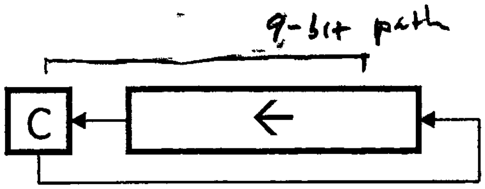


`lsr Rd` Logical shift right

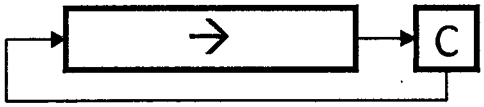


`rol Rd` Rotate left through carry

must know what's in the carry flag

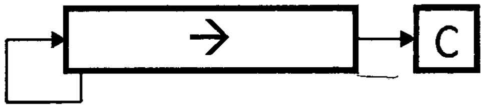


`ror Rd` Rotate right through carry



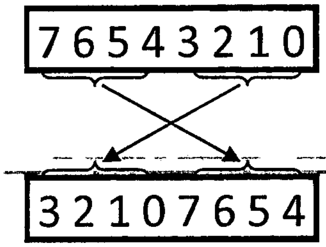
`asr Rd` Arithmetic shift right

2's complement numbers
fl sign extension"



Swap: (no flags modified, 1 clock)

`swap Rd` Swap nibbles



LDI r16, $1E
swap r16

r16 $E1

Key:

`Rd` = destination general purpose register (`r0`-`r31`)

`P` = I/O register (name, or `$00`-`$3f`, or `0b00000000`-`0b00111111`, or `0`-`63`)

`b` = bit number (`0`-`7`)

Notes:

- `sbi`, `cbi`, and `out` are the *only* instructions that modify I/O registers directly

movw r16, r18 { mov r16, r18
mov r17, r19

word 16 bits or bytes

ATmega32 Data Transfer Instructions

Register:	(no flags modified, 1 clock)		
mov Rd,Rr	copy Rr to Rd	Rd ← Rr	
movw Rd,Rr	copy Rr+1:Rr to Rd+1:Rd	Rd+1:Rd ← Rr+1:Rr	; *Rd, Rr even*
ldi Rd,K	Load Rd with immediate data	Rd ← K	; *r16-r31 only*
Load Data Mem:	(no flags modified, 2 clocks)		
ld Rd,X	Load indirect	Rd ← (X)	r18 r17 r16
ld Rd,X+	Load indirect, post increment	Rd ← (X), then X ← X+1	
ld Rd,-X	Load indirect, pre decrement	X ← X-1, then Rd ← (X)	
ld Rd,Y	Load indirect	Rd ← (Y)	
ld Rd,Y+	Load indirect, post increment	Rd ← (Y), then Y ← Y+1	
ld Rd,-Y	Load indirect, pre decrement	Y ← Y-1, then Rd ← (Y)	
ldd Rd,Y+offset	Load indirect with displacement	Rd ← (Y+offset)	
ld Rd,Z	Load indirect	Rd ← (Z)	
ld Rd,Z+	Load indirect, post increment	Rd ← (Z), then Z ← Z+1	
ld Rd,-Z	Load indirect, pre decrement	Z ← Z-1, then Rd ← (Z)	
ldd Rd,Z+offset	Load indirect with displacement	Rd ← (Z+offset)	
lds Rd,address	Load direct	Rd ← (address)	r0 → r31
Store Data Mem:	(no flags modified, 2 clocks)	Rd	0
st X,Rr	Store indirect	(X) ← Rr	
st X+,Rr	Store indirect, post increment	(X) ← Rr, then X ← X+1	
st -X,Rr	Store indirect, pre decrement	X ← X-1, then (X) ← Rr	
st Y,Rr	Store indirect	(Y) ← Rr	data memory
st Y+,Rr	Store indirect, post increment	(Y) ← Rr, then Y ← Y+1	~~pro~~
st -Y,Rr	Store indirect, pre decrement	Y ← Y-1, then (Y) ← Rr	
std Y+offset,Rr	Store indirect with displacement	(Y+offset) ← Rr	
st Z,Rr	Store indirect	(Z) ← Rr	
st Z+,Rr	Store indirect, post increment	(Z) ← Rr, then Z ← Z+1	
st -Z,Rr	Store indirect, pre decrement	Z ← Z-1, then (Z) ← Rr	
std Z+offset,Rr	Store indirect with displacement	(Z+offset) ← Rr	
~~sts address,Rr~~	~~Store direct~~	~~(address) ← Rr~~	
Program Memory:	(no flags modified, 3 clocks)		
lpm	Load Program Memory to r0	r0 ← (Z)	lpm r0, Z = lpM
lpm Rd,Z	Load Program Memory	Rd ← (Z)	
lpm Rd,Z+	Load Program Mem with post inc	Rd ← (Z), then Z ← Z+1	
spm	Store Program Memory (NOT AVAILABLE TO NORMAL USERS)		
I/O registers:	(no flags modified, 1 clock)		
in Rd,P	Copy I/O register P to Rd	Rd ← P	
out P,Rr	Copy Rr to I/O register P	P ← Rr	

6-bit offset available

unsigned

r30:r31 Z reg

download before program to the board

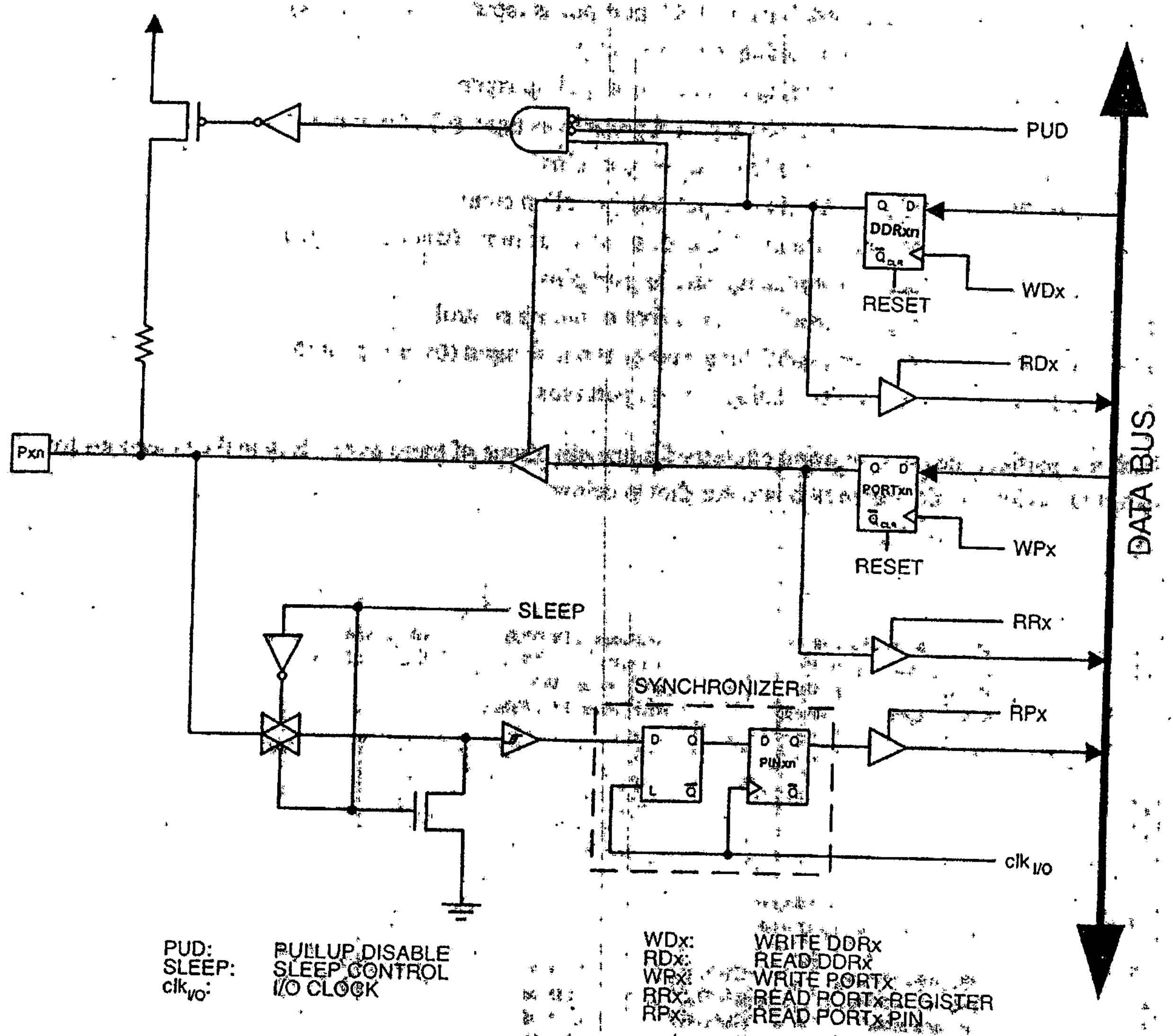
ATmega32 Parallel Ports

There are four 8-bit parallel ports on the **ATmega32** microcontroller, named **PORTA**, **PORTB**, **PORTC**, and **PORTD**. Three I/O Registers control each of the ports. `PORTx` and `DDRx` are cleared to `$00` by reset.

I/O Address	Name	Description
`$1b`	`PORTA`	register holding output data or pullup control
`$1a`	`DDRA`	register establishing each port pin as input (**0**) or output (**1**)
`$19`	`PINA`	access for reading data on port pins
`$18`	`PORTB`	register holding output data or pullup control
`$17`	`DDRB`	register establishing each port pin as input (**0**) or output (**1**)
`$16`	`PINB`	access for reading data on port pins
`$15`	`PORTC`	register holding output data or pullup control
`$14`	`DDRC`	register establishing each port pin as input (**0**) or output (**1**)
`$13`	`PINC`	access for reading data on port pins
`$12`	`PORTD`	register holding output data or pullup control
`$11`	`DDRD`	register establishing each port pin as input (**0**) or output (**1**)
`$10`	`PIND`	access for reading data on port pins

On the back is a partial schematic showing circuitry for *one pin* of *one* of these four ports. External connections for each port to I/O devices on the **EasyAVR** board are shown below.

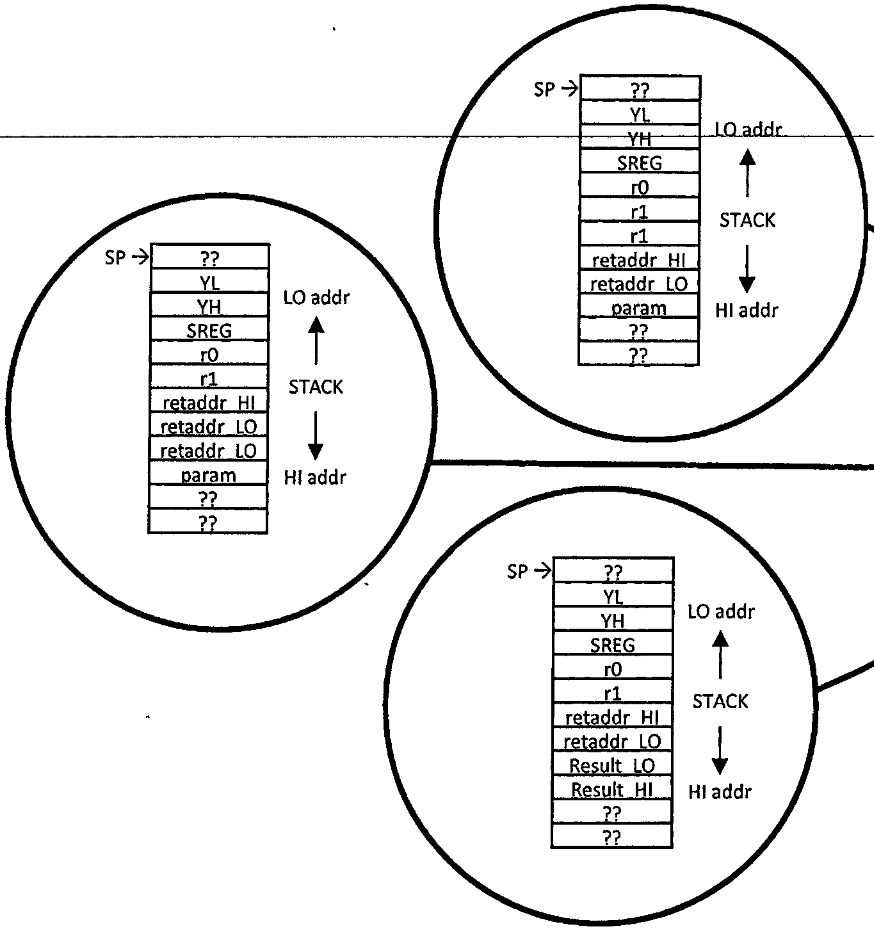
PORTA	7	-			
	6	-	analog from lower potentiometer		
	5	-	analog from upper potentiometer		
	4	-			
	3	-	Left 7-segment digit cathode	Touch screen DRIVEB	GLCD R/W
	2	-	2nd 7-segment digit cathode	Touch screen DRIVEA	LCD/GLCD RS
	1	-	3rd 7-segment digit cathode	Touch screen LEFT	
	0	-	Right 7-segment digit cathode	Touch screen BOTTOM	
PORTB	7	-			
	6	-	TIMER1 Input Capture		
	5	-			
	4	-			
	3	-		Analog comparator AIN1	TIMER0 controlled output
	2	-	external interrupt 2	Analog comparatorAIN0	
	1	-	enable left half of GLCD display	TIMER1 clock input	
	0	-	enable right half of GLCD display	TIMER0 clock input	
PORTC	7	-	7-segment anode dp	GLCD D7	LCD D7
	6	-	7-segment anode G	GLCD D6	LCD D6
	5	-	7-segment anode F	GLCD D5	LCD D5
	4	-	7-segment anode E	GLCD D4	LCD D4
	3	-	7-segment anode D	GLCD D3	
	2	-	7-segment anode C	GLCD D2	
	1	-	7-segment anode B	GLCD D1	
	0	-	7-segment anode A	GLCD D0	
PORTD	7	-	GLCD reset		TIMER2 controlled output
	6	-	LCD/GLCD data strobe		
	5	-	LCD backlight		TIMER1 controlled output A
output	4	-	speaker * Lab 2		TIMER1 controlled output B
	3	-	external interrupt 1		
	2	-	external interrupt 0		
	1	-			
	0	-			



1. WPx, WDx, RRx, RPx, and RDx are common to all pins within the same port. $\text{clk}_{I/O}$, SLEEP, and PUD are common to all ports.

Example 2: Square one 8-bit number, result is one 16-bit number

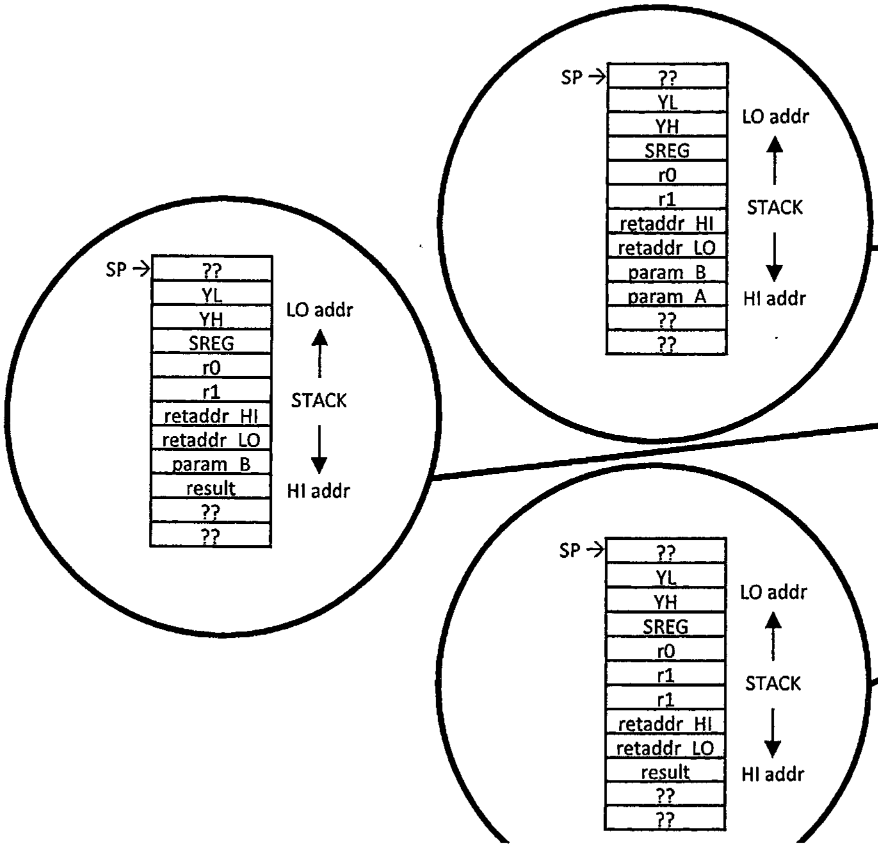
Note that here we have more bytes of result than parameters, so we have to create space on the stack



```
Sqr:    push r1
        push r1
        push r0
        in r0,SREG
        push r0
        push YH
        push YL
        in YH,SPH
        in YL,SPL
        ldd r0,Y+7
        std Y+6,r0
        ldd r0,Y+8
        std Y+7,r0
        ldd r0,Y+9
        mul r0,r0
        std Y+9,r1
        std Y+8,r0
        pop YL
        pop YH
        pop r0
        out SREG,r0
        pop r0
        pop r1
        ret
```

Example 3: Add two 8-bit numbers, result is one 8-bit number (no carry)

Note that here we have more bytes of parameters than result, so we have to eliminate space on the stack



```
Addem:  push r1
        push r0
        in r0,SREG
        push r0
        push YH
        push YL
        in YH,SPH
        in YL,SPL
        ldd r0,Y+8
        ldd r1,Y+9
        add r0,r1
        std Y+9,r0
        ldd r1,Y+7
        std Y+8,r1
        ldd r1,Y+6
        std Y+7,r1
        ldd r1,Y+5
        std Y+6,r1
        pop YL
        pop YH
        pop r0
        out SREG,r0
        pop r0
        pop r1
        pop r1
        ret
```

ATmega32 Procedure Parameters/Results on Stack

Example 1: Multiply two 8-bit numbers, result is one 16-bit number

1. Baseline solution

   Advantage: it's short
   Disadvantage: parameters/result at fixed locations
   Disadvantage: flags modified (side effect)

```
Mult:   mul r16,r17
        ret
```

2. Avoid side effects

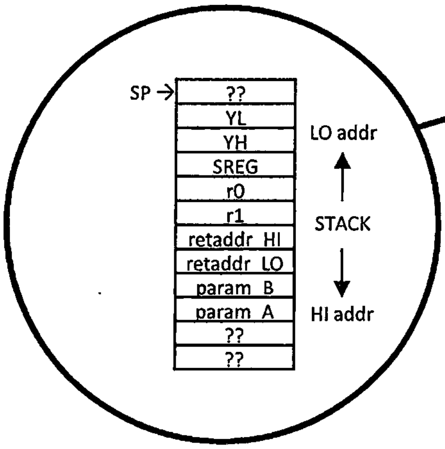
   Advantage: no side effects, result replaces parameters
   Disadvantage: overhead makes it longer
   Disadvantage: still fixed locations for parameters/result

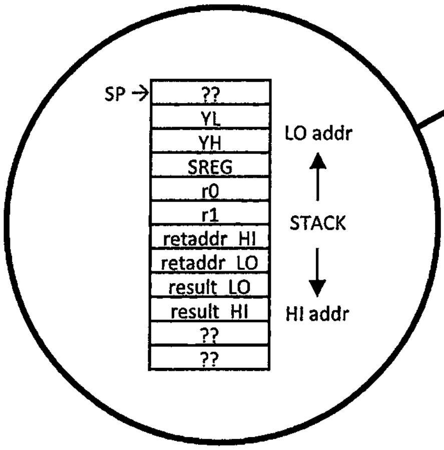
```
Mult:   push r1
        push r0
        in r0,SREG
        push r0
        mul r16,r17
        movw r16,r0
        pop r0
        out SREG,r0
        pop r0
        pop r1
        ret
```

3. Pass Parameters/Result on stack (*Elegant* solution)

   Advantage: flexible parameter/result location
   Advantage: no side effects, result replaces parameters
   Disadvantage: more overhead makes it still longer

```
Mult:   push r1
        push r0
        in r0,SREG
        push r0
        push YH
        push YL
        in YH,SPH
        in YL,SPL
        ldd r0,Y+8
        ldd r1,Y+9
        mul r0,r1
        std Y+8,r0
        std Y+9,r1
        pop YL
        pop YH
        pop r0
        out SREG,r0
        pop r0
        pop r1
        ret
```



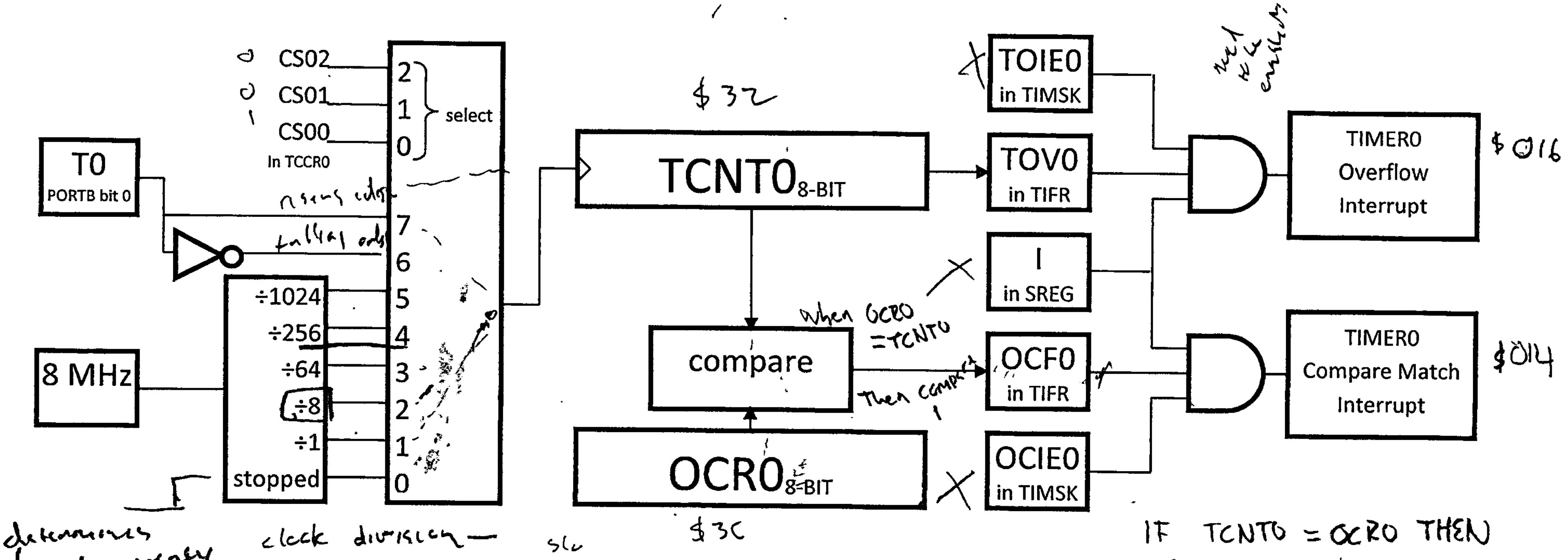


Note that here we have the same number of bytes of results as bytes of parameters, so the results are simply written over the parameters before returning. That is not always the case, as shown on the back...

CS02	CS01	CS00
0	1	2
0	0	1

ATmega32 TIMER0

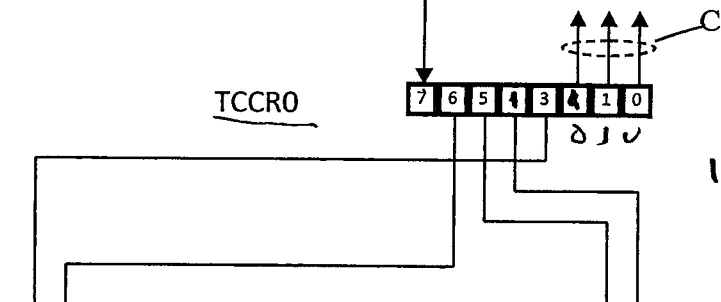
CS02	CS01	CS00	
0	1	0	÷ 8
0	1	1	÷ 64



determines the frequency

clock division —

IF TCNT0 = OCR0 THEN
COMPARE is set to 1


Write 1 to force programmed effect on bit 3 of PORTB shown below in normal or CTC modes
Clock select (above)
TCCR0
7
6
5
4
3
2
1
0
0 1 0

100

Intr every 1/256 sec

3906.25 μsec

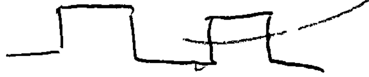
÷64 (32 μsec micro tick!)
122 times ticks

÷	freq	tick	overflow rate
1024	7812.5H	128 μsec	30.5 Hz
256	312K	32 μsec	122 Hz
64	125 KH	8 μsec	488 Hz
8	1 MHz	1 μsec	3906 Hz
1	2 MHz	1/8 μsec	31250 Hz

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

Effect on bit 3 of PORTB

	Normal or CTC	Fast PWM	PWM Phase Correct
0 0	normal PORT	normal PORT	normal PORT
0 1	toggle on match	reserved	reserved
1 0	clear on match	clear on match, set on `$ff`	clear on match while ↑, set on match while ↓
1 1	set on match	set on match, clear on `$ff`	set on match while ↑, clear on match while ↓

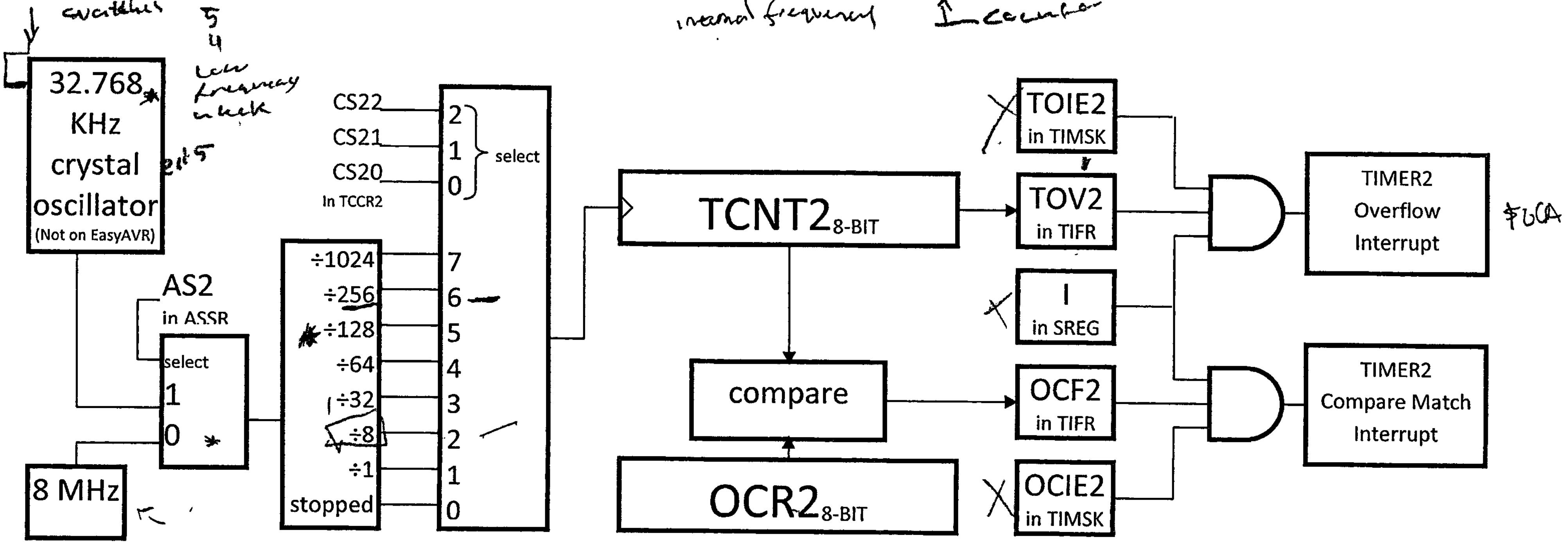


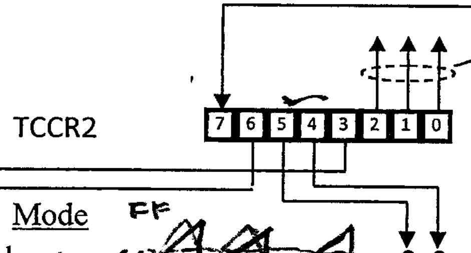
• more flexibility for clock divisor
• uses internal clock
unlike the timer0

ASSR -
Asynchronous Status Register

ATmega32 TIMER2

internal frequency ↑ counter interrupt





Write **1** to force programmed effect on bit 7 of PORTD shown below in normal or CTC modes

Clock select (above)

TCCR2 0000 0010

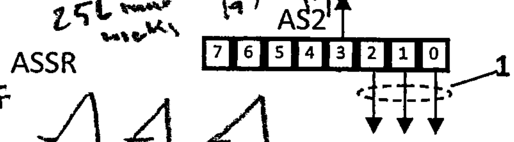
	Mode
0 0	Normal
0 1	PWM Phase Correct
1 0	CTC
1 1	Fast PWM

FF 00 OCR2 256 timer ticks

Effect on bit 7 of PORTD

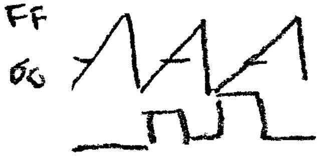
	Normal or CTC	Fast PWM	PWM Phase Correct
0 0	normal PORT	normal PORT	normal PORT
0 1	toggle on match	reserved	reserved
1 0	clear on match	clear on match, set on $ff	clear on match while ↑, set on match while ↓
1 1	set on match	set on match, clear on $ff	set on match while ↑, clear on match while ↓

period "510 timer ticks"



1 indicates asynchronous updating in progress of TCNT2, OCR2, or TCCR2, respectively

FAST PWM



"10 timer ticks" IF OCR2 = 9

```
; 1 Hz square wave on PORTD bit 5 using TIMER1 normal mode

         jmp Reset
         .org $00e
         jmp MtchA                 ; vector TIMER1 MatchA

Reset:   ldi r16,High(RAMEND)      ; stack
         out SPH,r16
         ldi r16,Low(RAMEND)
         out SPL,r16

         ldi r16,$40               ; toggle, mode 0, /256
         out TCCR1A,r16
         ldi r16,$04
         out TCCR1B,r16
         sbi DDRD,5                ; output to backlight

         ldi r16,$10               ; TIMER1 MatchA enable
         out TIMSK,r16
         sei                       ; global enable

Idle:    rjmp Idle

MtchA:   push r16
         in r16,SREG
         push r16
         push r17
         push r18
         push r19
         in r16,OCR1AL
         in r17,OCR1AH
         ldi r19,High(15625)       ; next int 15625 ticks
         ldi r18,Low(15625)
         add r16,r18
         adc r17,r19
         out OCR1AH,r17
         out OCR1AL,r16
         pop r19
         pop r18
         pop r17
         pop r16
         out SREG,r16
         pop r16
         reti
```

32 [illegible]

# of timer ticks

must be done in this order

must be done in this order

```
; 1 Hz square wave on PORTD bit 5 using TIMER1 CTC mode

     ldi r16,$40             ; toggle, mode 12, /256
     out TCCR1A,r16
     ldi r16,$1c
     out TCCR1B,r16
     sbi DDRD,5              ; output to backlight

     ldi r16,High(15624)     ; Match = 15624
     out ICR1H,r16
     ldi r16,Low(15624)
     out ICR1L,r16
Idle: rjmp Idle
```

```
; 1 Hz square wave on PORTD bit 5, TIMER1 Fast PWM mode

     ldi r16,$82             ; high pulse, mode 14, /256
     out TCCR1A,r16
     ldi r16,$1c
     out TCCR1B,r16
     sbi DDRD,5              ; output to backlight

     ldi r16,high(31249)     ; TOP = 31249
     out ICR1H,r16
     ldi r16,low(31249)
     out ICR1L,r16
     ldi r16,high(15624)     ; Match = 15624
     out OCR1AH,r16
     ldi r16,low(15624)
     out OCR1AL,r16
Idle: rjmp Idle
```

```
; 1 Hz square wave on PORTD bit 5, TIMER1 Ph.Cor.PWM mode

     ldi r16,$82             ; high pulse, mode 10, /64
     out TCCR1A,r16
     ldi r16,$13
     out TCCR1B,r16
     sbi DDRD,5              ; output to backlight

     ldi r16,high(62500)     ; TOP = 62500
     out ICR1H,r16
     ldi r16,low(62500)
     out ICR1L,r16
     ldi r16,high(31250)     ; Match = 31250
     out OCR1AH,r16
     ldi r16,low(31250)
     out OCR1AL,r16
Idle: rjmp Idle
```

Jonathan did all.

Johnathan Madra

ATmega32 Interrupts

There are 21_{10} sources of interrupts in the ATmega32 processor. Each interrupt source has a unique interrupt vector location assigned to it, as shown below. Most interrupt sources have separate enable bits for each interrupt, which, along with the Global Interrupt Enable bit (`I`) in `SREG`, must be **1** to enable that interrupt to be recognized.

When an interrupt occurs, the ATmega32 first finishes the current instruction. Then the Program Counter (`PC`) is pushed onto the stack, and the Global Interrupt Enable (`I`, in `SREG`) is cleared to disable other interrupts. Unless the `I` bit is set to **1** by the interrupt service routine, the service routine thus cannot be interrupted by another source. The `I` bit is automatically set to **1** by the Return From Interrupt (`reti`) instruction.

Global

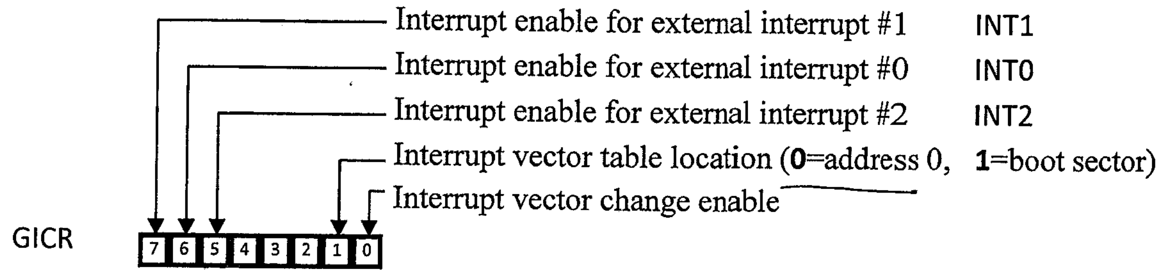
[illegible]

Interrupt Vector Table

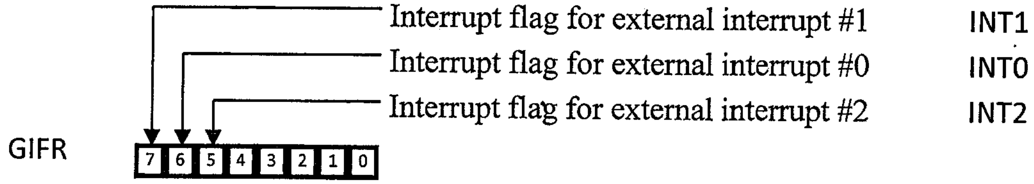
Address	Source	Definition
$000	RESET	External Pin, Power-on, Brown-out, Watchdog, JTAG
$002	INT0	External Interrupt Request 0
$004	INT1	External Interrupt Request 1
$006	INT2	External Interrupt Request 2
$008	TIMER2 COMP	Timer/Counter2 Compare Match
$00a	TIMER2 OVF	Timer/Counter2 Overflow
$00c	TIMER1 CAPT	Timer/Counter1 Capture Event
$00e	TIMER1 COMPA	Timer/Counter1 Compare Match A
$010	TIMER1 COMPB	Timer/Counter1 Compare Match B
$012	TIMER1 OVF	Timer/Counter1 Overflow
$014	TIMER0 COMP	Timer/Counter0 Compare Match
$016	TIMER0 OVF	Timer/Counter0 Overflow
$018	SPI, STC	Serial Transfer Complete
$01a	USART, RXC	USART Receive Complete
$01c	USART, UDRE	USART Data Register Empty
$01e	USART, TXC	USART Transmit Complete
$020	ADC	Analog Conversion Complete
$022	EE_RDY	EEPROM Ready
$024	ANA_COMP	Analog Comparator
$026	TWI	Two-wire Serial Interface
$028	SPM_RDY	Store Program Memory Ready

Lowest ← ……… Interrupt Priority ……… → Highest

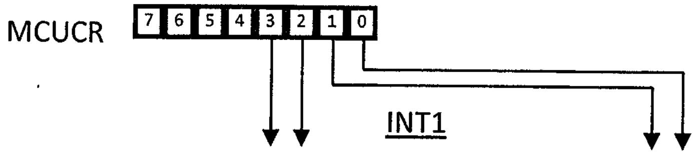
ATmega32 External Interrupts



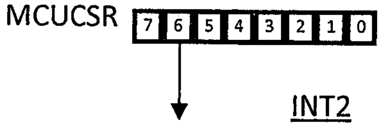
Interrupt enable for external interrupt #1 INT1
Interrupt enable for external interrupt #0 INT0
Interrupt enable for external interrupt #2 INT2
Interrupt vector table location (**0**=address 0, **1**=boot sector)
Interrupt vector change enable



Interrupt flag for external interrupt #1 INT1
Interrupt flag for external interrupt #0 INT0
Interrupt flag for external interrupt #2 INT2



	INT1		INT0
0 0	low level	**0 0**	low level
0 1	any change	**0 1**	any change
1 0	falling edge	**1 0**	falling edge
1 1	rising edge	**1 1**	rising edge



	INT2
0	falling edge
1	rising edge

INT0 is on PORTD bit 2
INT1 is on PORTD bit 3
INT2 is on PORTB bit 2

Johnathan Uechler

Done

External Interrupts

Timer Stuff

new

I/O Register Summary

enable bits and flag bits

Pull-up disable

disables all the pull-ups

Watch dog timer control register

PaLLEL PORT

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
$3F ($5F)	SREG	I	T	H	S	V	N	Z	C	10
$3E ($5E)	SPH	–	–	–	–	SP11	SP10	SP9	SP8	12
$3D ($5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	12
$3C ($5C)	OCR0	Timer/Counter0 Output Compare Register								82
$3B ($5B)	GICR	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	47, 67
$3A ($5A)	GIFR	INTF1	INTF0	INTF2	–	–	–	–	–	68
$39 ($59)	TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	82, 112, 130
$38 ($58)	TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	83, 112, 130
$37 ($57)	SPMCR	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	248
$36 ($56)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	177
$35 ($55)	MCUCR	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	32, 66
$34 ($54)	MCUCSR	JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF	40, 67, 228
$33 ($53)	TCCR0	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	80
$32 ($52)	TCNT0	Timer/Counter0 (8 Bits)								82
$31[1] ($51)[1]	OSCCAL	Oscillator Calibration Register								30
	OCDR	On-Chip Debug Register								224
$30 ($50)	SFIOR	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	56,85,131,198,218
$2F ($4F)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	107
$2E ($4E)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	110
$2D ($4D)	TCNT1H	Timer/Counter1 – Counter Register High Byte								111
$2C ($4C)	TCNT1L	Timer/Counter1 – Counter Register Low Byte								111
$2B ($4B)	OCR1AH	Timer/Counter1 – Output Compare Register A High Byte								111
$2A ($4A)	OCR1AL	Timer/Counter1 – Output Compare Register A Low Byte								111
$29 ($49)	OCR1BH	Timer/Counter1 – Output Compare Register B High Byte								111
$28 ($48)	OCR1BL	Timer/Counter1 – Output Compare Register B Low Byte								111
$27 ($47)	ICR1H	Timer/Counter1 – Input Capture Register High Byte								111
$26 ($46)	ICR1L	Timer/Counter1 – Input Capture Register Low Byte								111
$25 ($45)	TCCR2	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20	125
$24 ($44)	TCNT2	Timer/Counter2 (8 Bits)								127
$23 ($43)	OCR2	Timer/Counter2 Output Compare Register								127
$22 ($42)	ASSR	–	–	–	–	AS2	TCN2UB	OCR2UB	TCR2UB	128
$21 ($41)	WDTCR	–	–	–	WDTOE	WDE	WDP2	WDP1	WDP0	42
$20[2] ($40)[2]	UBRRH	URSEL	–	–	–	UBRR[11:8]				164
	UCSRC	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	162
$1F ($3F)	EEARH	–	–	–	–	–	–	EEAR9	EEAR8	19
$1E ($3E)	EEARL	EEPROM Address Register Low Byte								19
$1D ($3D)	EEDR	EEPROM Data Register								19
$1C ($3C)	EECR	–	–	–	–	EERIE	EEMWE	EEWE	EERE	19
$1B ($3B)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	64
$1A ($3A)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	64
$19 ($39)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	64
$18 ($38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	64
$17 ($37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	64
$16 ($36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	65
$15 ($35)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	65
$14 ($34)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	65
$13 ($33)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	65
$12 ($32)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	65
$11 ($31)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	65
$10 ($30)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	65
$0F ($2F)	SPDR	SPI Data Register								138
$0E ($2E)	SPSR	SPIF	WCOL	–	–	–	–	–	SPI2X	138
$0D ($2D)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	136
$0C ($2C)	UDR	USART I/O Data Register								159
$0B ($2B)	UCSRA	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	160
$0A ($2A)	UCSRB	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	161
$09 ($29)	UBRRL	USART Baud Rate Register Low Byte								164
$08 ($28)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	199
$07 ($27)	ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	214
$06 ($26)	ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	216
$05 ($25)	ADCH	ADC Data Register High Byte								217
$04 ($24)	ADCL	ADC Data Register Low Byte								217
$03 ($23)	TWDR	Two-wire Serial Interface Data Register								179
$02 ($22)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	179
$01 ($21)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	178
$00 ($20)	TWBR	Two-wire Serial Interface Bit Rate Register								177

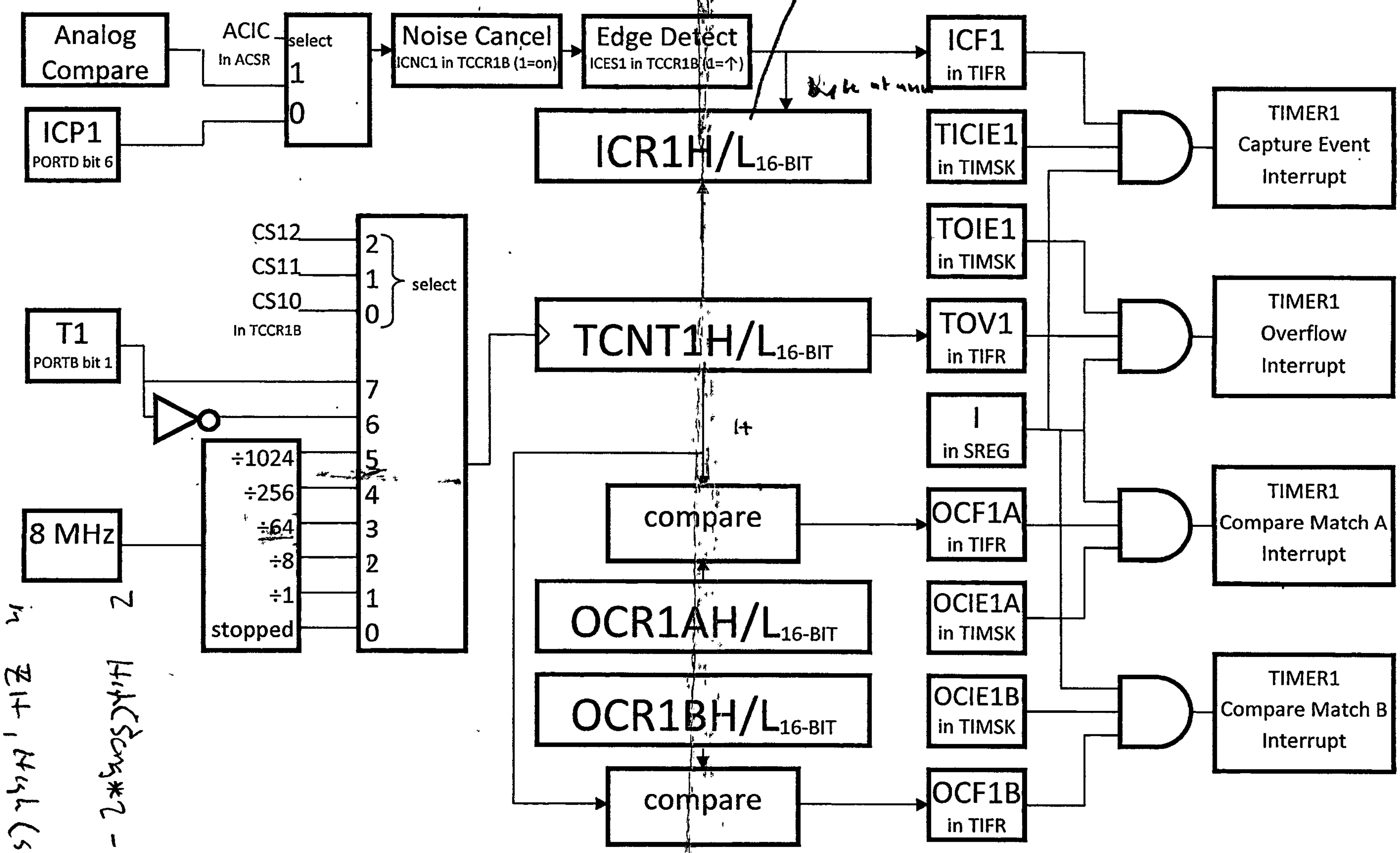
Notes:
1. When the OCDEN Fuse is unprogrammed, the OSCCAL Register is always accessed on this address. Refer to the debugger specific documentation for details on how to use the OCDR Register.
2. Refer to the USART description for details on how to access UBRRH and UCSRC.
3. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
4. Some of the Status Flags are cleared by writing a logical one to them. Note that the CBI and SBI instructions will operate on all bits in the I/O Register, writing a one back into any flag read as set, thus clearing the flag. The CBI and SBI instructions work with registers $00 to $1F only.

Schaathun-

4/1/12

ATmega32 TIMER1

must input the low byte first then input high byte

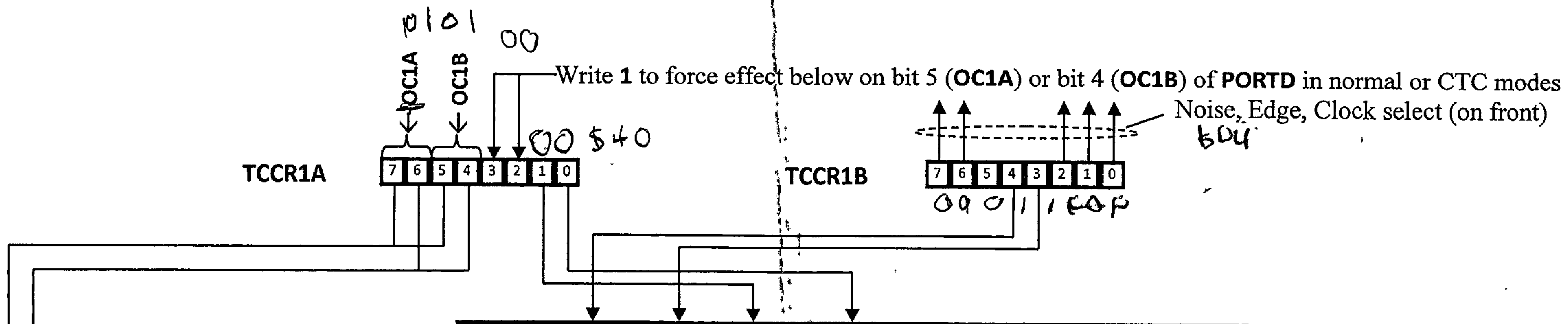


ZH, High(score*2-1)

Z High(Sscore*2 - 1)

output high byte First

input Low byte First the high byte



Write **1** to force effect below on bit 5 (**OC1A**) or bit 4 (**OC1B**) of **PORTD** in normal or CTC modes

Noise, Edge, Clock select (on front)

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	Reserved	–	–	–
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Effect on bit 5 (OC1A) or bit 4 (OC1B) of PORTD

	Normal or CTC	Fast PWM	PWM Phase Correct OR Phase & Frequency Correct
0 0	normal PORT	normal PORT	normal PORT
0 1	toggle on match	mode 15: toggle **OC1A** on match, **OC1B** not used	mode 9, 11: toggle **OC1A** on match, **OC1B** not used
1 0	clear on match	clear on match, set on TOP	clear on match while ↑, set on match while ↓
1 1	set on match	set on match, clear on TOP	set on match while ↑, clear on match while ↓

4/4/12

Touch Screen

The Touch Screen is built from two layers of transparent material with a resistive coating on each, separated by small spacers so they normally do not touch. One layer has contacts across the top and bottom, and the other layer has contacts across the left and right sides. Although the Touch Screen is mounted on top of the Graphic Liquid Crystal Display (**GLCD**), it has no electrical relation to it. Connections to **PORTA** are as follows:

	PORTA bit 3	**PORTA** bit 2			**PORTA** bit 1	**PORTA** bit 0
	0	**0**	**-**	**+5**	**-**	**-**
horizontal	**0**	**1**	**+5**	**-**	**0**	**-**
vertical →	**1**	**0**	**-**	**+5**	**-**	**0**
	1	**1**	**+5**	**-**	**0**	**0**
		Contact connection →	right	top	left	bottom

The first and last lines in the table above are not useful. The 2nd line (**PORTA** = %xxxx01xx) is used to sense horizontal position of the touch. The 3rd line (**PORTA** = %xxxx10xx) is used to sense vertical position of the touch. See the diagrams on the back for a good description of operation. (Right and left are reversed there.)

To measure horizontal position of the touch, +5 volts is connected to the right side of one layer, while the left side of that layer is at 0 volts. When a touch occurs, the other layer is connected to the first at the point of contact, and the voltage at that point is determined by the resistive divider formed by the resistance from the point of contact to the right and left sides. Similarly, to measure vertical position of the touch, +5 volts is connected to the top of the second layer, while the bottom of that layer is at 0 volts. When a touch occurs, the first layer is connected to the second at the point of contact, and the voltage at that point is determined by the resistive divider formed by the resistance from the point of contact to the top and bottom edges. Note that the resistance of the layer being used to sense the point of contact voltage is unimportant, because the current flowing into the sensor is negligible, and thus the voltage drop across the resistance of that layer is negligible.

To use the Touch Screen, you must separately determine the horizontal and vertical position of the touch by setting up values on **PORTA** bits 3 and 2 and then digitizing the resulting voltages on **PORTA** bits 1 or 0.

Johnathan Mader

Single-Ended Analog to Digital Converter

"Single-Ended" means individual voltages measured with respect to ground

10-bit resolution, successive approximation, 8 analog inputs (bits of **PORTA**)

ADMUX

7	6	5	4	3	2	1	0

Bit 5 — **0**: result is right-aligned, **1**: result is left aligned in **ADCH**:**ADCL**

Bits 4–0:

	<u>**Input to be digitized**</u>
0 0 0 0 0	**ADC0** = **PORTA** bit 0
0 0 0 0 1	**ADC1** = **PORTA** bit 1
0 0 0 1 0	**ADC2** = **PORTA** bit 2
0 0 0 1 1	**ADC3** = **PORTA** bit 3
0 0 1 0 0	**ADC4** = **PORTA** bit 4
0 0 1 0 1	**ADC5** = **PORTA** bit 5
0 0 1 1 0	**ADC6** = **PORTA** bit 6
0 0 1 1 1	**ADC7** = **PORTA** bit 7
1 1 1 1 0	1.22 volts (bandgap reference)
1 1 1 1 1	Ground, 0 volts

[illegible]

112 · 7 = 784

784 mod 255

↳ 19

3 r19

Bits 7–6:

5 volt supply voltage →

0 0	External reference voltage on **AREF** pin, internal reference turned off
0 1	Analog power, **AVCC**, is the reference voltage
1 0	Reserved
1 1	Internal 2.56 volt is the reference voltage

3(255) + 19

→ high low

ADCSRA

7	6	5	4	3	2	1	0

- Bit 7: ADC Enable
- Bit 6: Start Conversion
- Bit 5: Auto Trigger
- Bit 4: ADC Interrupt Flag
- Bit 3: ADC Interrupt Enable

Bits 2–0:

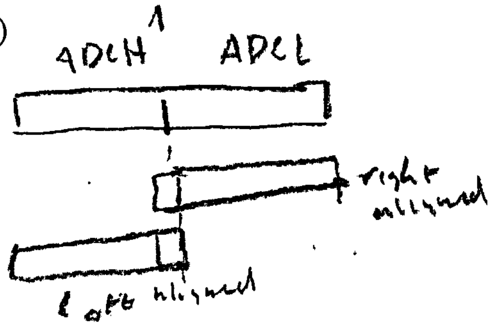
	<u>**8 MHz ÷ this to get 50–200 KHz**</u>	
0 0 0	÷ 2	
0 0 1	÷ 2	
0 1 0	÷ 4	
0 1 1	÷ 8	
1 0 0	÷ 16	
1 0 1	÷ 32	
1 1 0	÷ 64	← use this
1 1 1	÷ 128	

SFIOR

7	6	5	4	3	2	1	0

Bits 7–5: <u>**ADC Trigger Source**</u> (of Auto Trigger)

0 0 0	ADC Conversion Complete flag (i.e. free running samples)
0 0 1	Analog Comparator flag
0 1 0	External Interrupt 0 flag
0 1 1	Timer/Counter 0 Compare Match flag
1 0 0	Timer/Counter 0 Overflow flag
1 0 1	Timer/Counter 1 Compare Match B flag
1 1 0	Timer/Counter 1 Overflow flag
1 1 1	Timer/Counter 1 Input Capture flag



Differential Analog to Digital Converter

“Differential” means the value digitized is the difference between two voltages

10-bit resolution, successive approximation, 8 analog inputs (bits of **PORTA**), *2’s complement*

ADMUX | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | ← (other cases on “Single-Ended” sheet)

	Value to be digitized	**Gain**
0 1 0 0 0	**ADC0 – ADC0**	10x
0 1 0 0 1	**ADC1 – ADC0**	10x
0 1 0 1 0	**ADC0 – ADC0**	200x
0 1 0 1 1	**ADC1 – ADC0**	200x
0 1 1 0 0	**ADC2 – ADC2**	10x
0 1 1 0 1	**ADC3 – ADC2**	10x
0 1 1 1 0	**ADC2 – ADC2**	200x
0 1 1 1 1	**ADC3 – ADC2**	200x
1 0 0 0 0	**ADC0 – ADC1**	1x
1 0 0 0 1	**ADC1 – ADC1**	1x
1 0 0 1 0	**ADC2 – ADC1**	1x
1 0 0 1 1	**ADC3 – ADC1**	1x
1 0 1 0 0	**ADC4– ADC1**	1x
1 0 1 0 1	**ADC5 – ADC1**	1x
1 0 1 1 0	**ADC6 – ADC1**	1x
1 0 1 1 1	**ADC7 – ADC1**	1x
1 1 0 0 0	**ADC0 – ADC2**	1x
1 1 0 0 1	**ADC1 – ADC2**	1x
1 1 0 1 0	**ADC2 – ADC2**	1x
1 1 0 1 1	**ADC3 – ADC2**	1x
1 1 1 0 0	**ADC4 – ADC2**	1x
1 1 1 0 1	**ADC5 – ADC2**	1x

check if ADC0 - ADC0 == 0

Analog to Digital conversion times

First conversion	25 ADC clock cycles
Normal single-ended	13 ADC clock cycles
Auto-Triggered	13.5 ADC clock cycles
Normal differential	13-14 ADC clock cycles

rate at which conversions happen for this mode

15KHz

30kHz

↳ [illegible]

10 K times per second

5 kHz

+ 4 KH → telephone [illegible]

instruction wdr watchdog reset

Watchdog Timer Control Register – WDTCR

Bit	7	6	5	4	3	2	1	0	
	–	–	–	WDTOE	WDE	WDP2	WDP1	WDP0	WDTCR
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bits [7:5] – Reserved Bits**

Turn on by putting a bit in WDE

These bits are reserved bits in the ATmega32 and will always read as zero.

- **Bit 4 – WDTOE: Watchdog Turn-off Enable**

This bit must be set when the WDE bit is written to logic zero. Otherwise, the Watchdog will not be disabled. Once written to one, hardware will clear this bit after four clock cycles. Refer to the description of the WDE bit for a Watchdog disable procedure.

- **Bit 3 – WDE: Watchdog Enable**

When the WDE is written to logic one, the Watchdog Timer is enabled, and if the WDE is written to logic zero, the Watchdog Timer function is disabled. WDE can only be cleared if the WDTOE bit has logic level one. To disable an enabled Watchdog Timer, the following procedure must be followed:

1. In the same operation, write a logic one to WDTOE and WDE. A logic one must be written to WDE even though it is set to one before the disable operation starts.
2. Within the next four clock cycles, write a logic 0 to WDE. This disables the Watchdog.

- **Bits [2:0] – WDP2, WDP1, WDP0: Watchdog Timer Prescaler 2, 1, and 0**

The WDP2, WDP1, and WDP0 bits determine the Watchdog Timer prescaling when the Watchdog Timer is enabled. The different prescaling values and their corresponding Timeout Periods are shown in Table 17.

Table 17. Watchdog Timer Prescale Select

WDP2	WDP1	WDP0	Number of WDT Oscillator Cycles	Typical Time-out at V_{CC} = 3.0V	Typical Time-out at V_{CC} = 5.0V
0	0	0	16K (16,384)	17.1ms	16.3ms
0	0	1	32K (32,768)	34.3ms	32.5ms
0	1	0	64K (65,536)	68.5ms	65ms
0	1	1	128K (131,072)	0.14s	0.13 s
1	0	0	256K (262,144)	0.27s	0.26s
1	0	1	512K (524,288)	0.55s	0.52s
1	1	0	1,024K (1,048,576)	1.1s	1.0s
1	1	1	2,048K (2,097,152)	2.2s	2.1s

will reset quickly if it fails

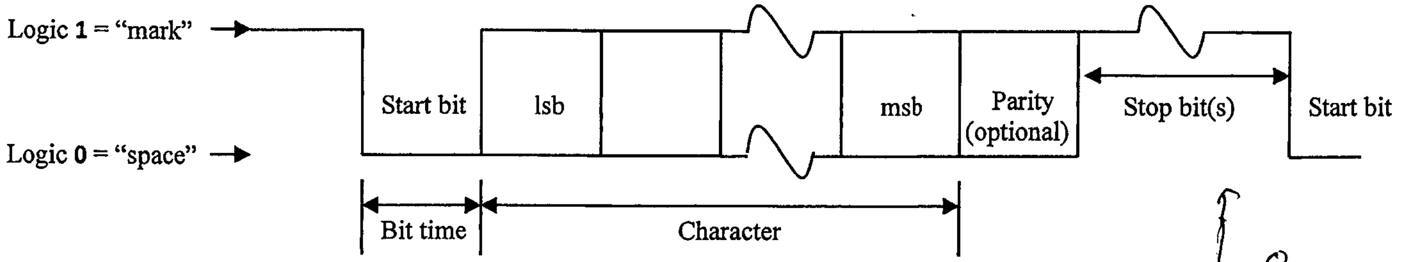
The Watchdog Timer is clocked from a separate On-chip Oscillator which runs at 1MHz. This is the typical value at V_{CC} = 5V. See characterization data for typical values at other V_{CC} levels. By controlling the Watchdog Timer prescaler, the Watchdog Reset interval can be adjusted as shown in Table 17 on page 42. The WDR – Watchdog Reset – instruction resets the Watchdog Timer. The Watchdog Timer is also reset when it is disabled and when a Chip Reset occurs. Eight different clock cycle periods can be selected to determine the reset period. If the reset period expires without another Watchdog Reset, the ATmega32 resets and executes from the Reset Vector. For timing details on the Watchdog Reset, refer to page 40.

To prevent unintentional disabling of the Watchdog, a special turn-off sequence must be followed when the Watchdog is disabled. Refer to the description of the Watchdog Timer Control Register for details.

Jchnathan Macler
4/10/13

Universal Synchronous/Asynchronous Receiver and Transmitter (USART)

Asynchronous RS-232 standard protocol:

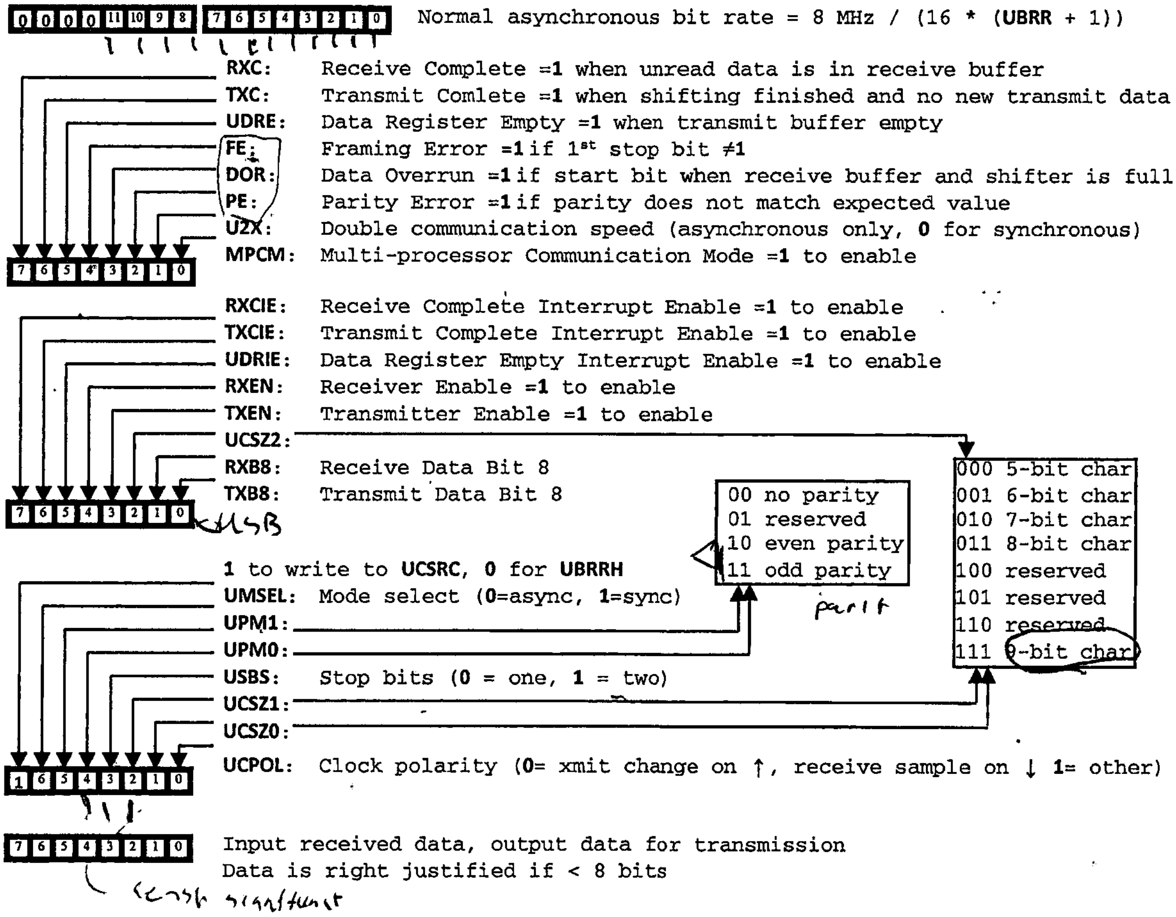


0
1
8/ICC1

For communications to work, you must establish…

1. Bit time (= 1/bit rate) (Note: bit rate ≠ baud rate, in general)
2. # bits per character (5-8 for RS-232, 5-9 for ATmega32 **USART**)
3. Minimum # stop bits (1,1½, or 2 for RS-232, 1 or 2 for ATmega32 **USART**)
4. Parity? Even/odd?
5. Voltages (or ?) for "space" & "mark" (**0** = +3 to +25 volts, **1** = -3 to -25 volts, for RS-232, i.e. *negative* logic)

bits per second (all 1s)



UBRRH : UBRRL
Bit Rate

Normal asynchronous bit rate = 8 MHz / (16 * (**UBRR** + 1))

UCSRA
Control/Status A

bits / char → must
9 / char

RXC: Receive Complete =**1** when unread data is in receive buffer
TXC: Transmit Comlete =**1** when shifting finished and no new transmit data
UDRE: Data Register Empty =**1** when transmit buffer empty
FE: Framing Error =**1** if 1st stop bit ≠1
DOR: Data Overrun =**1** if start bit when receive buffer and shifter is full
PE: Parity Error =**1** if parity does not match expected value
U2X: Double communication speed (asynchronous only, **0** for synchronous)
MPCM: Multi-processor Communication Mode =**1** to enable

UCSRB
Control/Status B

RXCIE: Receive Complete Interrupt Enable =**1** to enable
TXCIE: Transmit Complete Interrupt Enable =**1** to enable
UDRIE: Data Register Empty Interrupt Enable =**1** to enable
RXEN: Receiver Enable =**1** to enable
TXEN: Transmitter Enable =**1** to enable
UCSZ2:
RXB8: Receive Data Bit 8
TXB8: Transmit Data Bit 8

LSB

UCSRC
Control/Status C

1 to write to **UCSRC**, **0** for **UBRRH**
UMSEL: Mode select (**0**=async, **1**=sync)
UPM1:
UPM0:
USBS: Stop bits (**0** = one, **1** = two)
UCSZ1:
UCSZ0:
UCPOL: Clock polarity (**0**= xmit change on ↑, receive sample on ↓ **1**= other)

00 no parity
01 reserved
10 even parity
11 odd parity

parity

000 5-bit char
001 6-bit char
010 7-bit char
011 8-bit char
100 reserved
101 reserved
110 reserved
111 9-bit char

UDR
Data Register

Input received data, output data for transmission
Data is right justified if < 8 bits

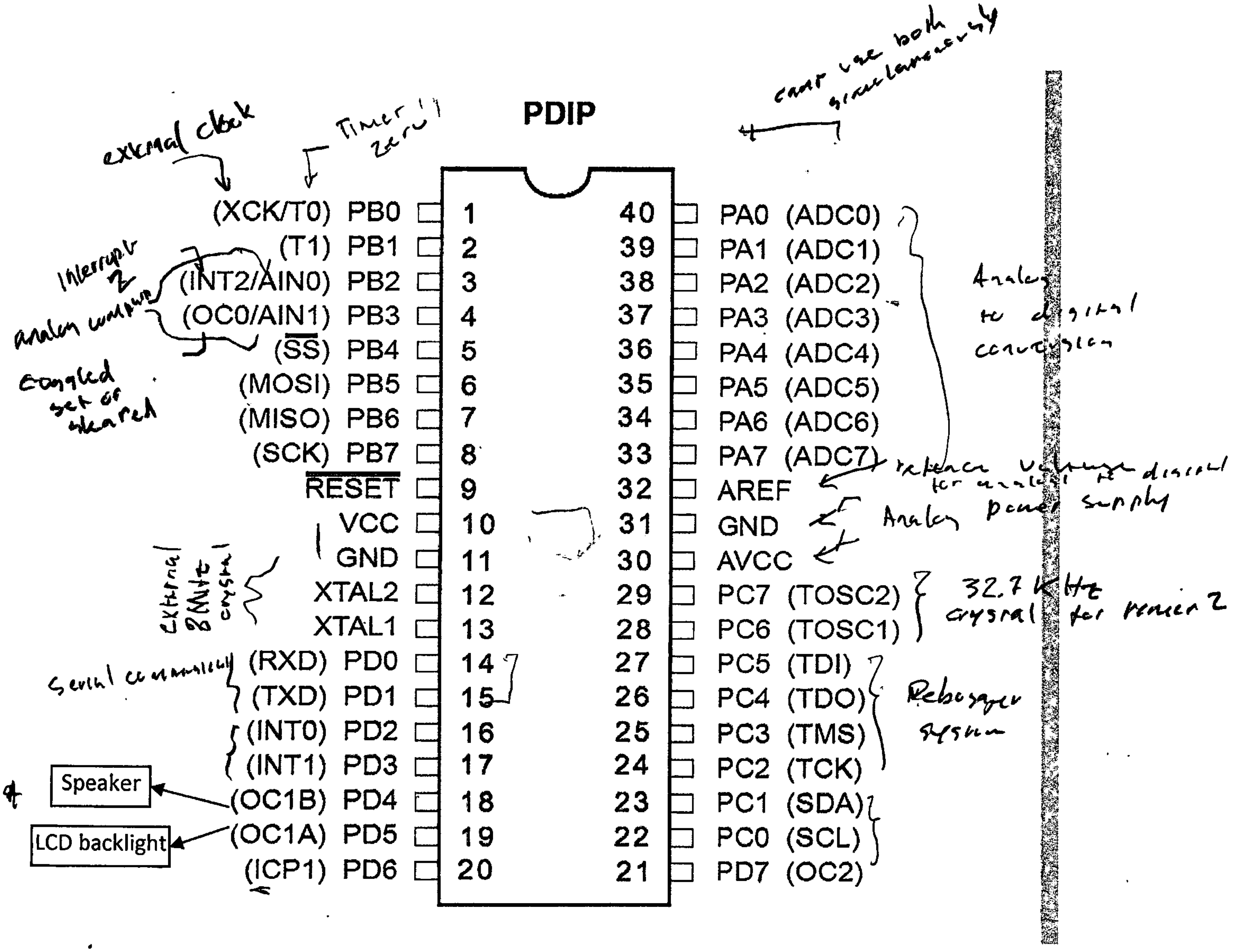
least significant

```
Normal asynchronous bit rate = 8 MHz / (16 * (UBRR + 1))
Double speed asynchronous bit rate = 8 MHz / (8 * (UBRR + 1))
Synchronous bit rate = 8 MHz / (2 * (UBRR + 1))
```

Received bits are sampled three times near the middle of each bit time, majority voting determines final value

In synchronous mode, the **DDR** bit for the clock signal determines master (clock out) or slave (clock in) mode

ATmega32 Pinout



* Try in lab.

PIN 14 and 15 need to be a available

set up to transmit at the slowest possible rate on PORTD to see the LEDs flash

4/13/18

I/O Register Summary

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
$3F ($5F)	SREG	I	T	H	S	V	N	Z	C	10
$3E ($5E)	SPH	–	–	–	–	SP11	SP10	SP9	SP8	12
$3D ($5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	12
$3C ($5C)	OCR0	Timer/Counter0 Output Compare Register								82
$3B ($5B)	GICR	INT1	INT0	INT2	–	–	–	IVSEL	IVCE	47, 67
$3A ($5A)	GIFR	INTF1	INTF0	INTF2	–	–	–	–	–	68
$39 ($59)	TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0	82, 112, 130
$38 ($58)	TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0	83, 112, 130
$37 ($57)	SPMCR	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN	248
$36 ($56)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE	177
$35 ($55)	MCUCR	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00	32, 66
$34 ($54)	MCUCSR	JTD	ISC2	–	JTRF	WDRF	BORF	EXTRF	PORF	40, 67, 228
$33 ($53)	TCCR0	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00	80
$32 ($52)	TCNT0	Timer/Counter0 (8 Bits)								82
$31[1] ($51)[1]	OSCCAL	Oscillator Calibration Register								30
	OCDR	On-Chip Debug Register								224
$30 ($50)	SFIOR	ADTS2	ADTS1	ADTS0	–	ACME	PUD	PSR2	PSR10	56,85,131,198,218
$2F ($4F)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10	107
$2E ($4E)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10	110
$2D ($4D)	TCNT1H	Timer/Counter1 – Counter Register High Byte								111
$2C ($4C)	TCNT1L	Timer/Counter1 – Counter Register Low Byte								111
$2B ($4B)	OCR1AH	Timer/Counter1 – Output Compare Register A High Byte								111
$2A ($4A)	OCR1AL	Timer/Counter1 – Output Compare Register A Low Byte								111
$29 ($49)	OCR1BH	Timer/Counter1 – Output Compare Register B High Byte								111
$28 ($48)	OCR1BL	Timer/Counter1 – Output Compare Register B Low Byte								111
$27 ($47)	ICR1H	Timer/Counter1 – Input Capture Register High Byte								111
$26 ($46)	ICR1L	Timer/Counter1 – Input Capture Register Low Byte								111
$25 ($45)	TCCR2	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20	125
$24 ($44)	TCNT2	Timer/Counter2 (8 Bits)								127
$23 ($43)	OCR2	Timer/Counter2 Output Compare Register								127
$22 ($42)	ASSR	–	–	–	–	AS2	TCN2UB	OCR2UB	TCR2UB	128
$21 ($41)	WDTCR	–	–	–	WDTOE	WDE	WDP2	WDP1	WDP0	42
$20[2] ($40)[2]	UBRRH	URSEL	–	–	–	UBRR[11:8]				164
	UCSRC	URSEL	UMSEL	UPM1	UPM0	USBS	UCSZ1	UCSZ0	UCPOL	162
$1F ($3F)	EEARH	–	–	–	–	–	–	EEAR9	EEAR8	19
$1E ($3E)	EEARL	EEPROM Address Register Low Byte								19
$1D ($3D)	EEDR	EEPROM Data Register								19
$1C ($3C)	EECR	–	–	–	–	EERIE	EEMWE	EEWE	EERE	19
$1B ($3B)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	64
$1A ($3A)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	64
$19 ($39)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	64
$18 ($38)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	64
$17 ($37)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	64
$16 ($36)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	65
$15 ($35)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	65
$14 ($34)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	65
$13 ($33)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	65
$12 ($32)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	65
$11 ($31)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	65
$10 ($30)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	65
$0F ($2F)	SPDR	SPI Data Register								138
$0E ($2E)	SPSR	SPIF	WCOL	–	–	–	–	–	SPI2X	138
$0D ($2D)	SPCR	SPIE	SPE	DORD	MSTR	CPOL	CPHA	SPR1	SPR0	136
$0C ($2C)	UDR	USART I/O Data Register								159
$0B ($2B)	UCSRA	RXC	TXC	UDRE	FE	DOR	PE	U2X	MPCM	160
$0A ($2A)	UCSRB	RXCIE	TXCIE	UDRIE	RXEN	TXEN	UCSZ2	RXB8	TXB8	161
$09 ($29)	UBRRL	USART Baud Rate Register Low Byte								164
$08 ($28)	ACSR	ACD	ACBG	ACO	ACI	ACIE	ACIC	ACIS1	ACIS0	199
$07 ($27)	ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	214
$06 ($26)	ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	216
$05 ($25)	ADCH	ADC Data Register High Byte								217
$04 ($24)	ADCL	ADC Data Register Low Byte								217
$03 ($23)	TWDR	Two-wire Serial Interface Data Register								179
$02 ($22)	TWAR	TWA6	TWA5	TWA4	TWA3	TWA2	TWA1	TWA0	TWGCE	179
$01 ($21)	TWSR	TWS7	TWS6	TWS5	TWS4	TWS3	–	TWPS1	TWPS0	178
$00 ($20)	TWBR	Two-wire Serial Interface Bit Rate Register								177

Notes:
1. When the OCDEN Fuse is unprogrammed, the OSCCAL Register is always accessed on this address. Refer to the debugger specific documentation for details on how to use the OCDR Register.
2. Refer to the USART description for details on how to access UBRRH and UCSRC.
3. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
4. Some of the Status Flags are cleared by writing a logical one to them. Note that the CBI and SBI instructions will operate on all bits in the I/O Register, writing a one back into any flag read as set, thus clearing the flag. The CBI and SBI instructions work with registers $00 to $1F only.