IO subprocessor

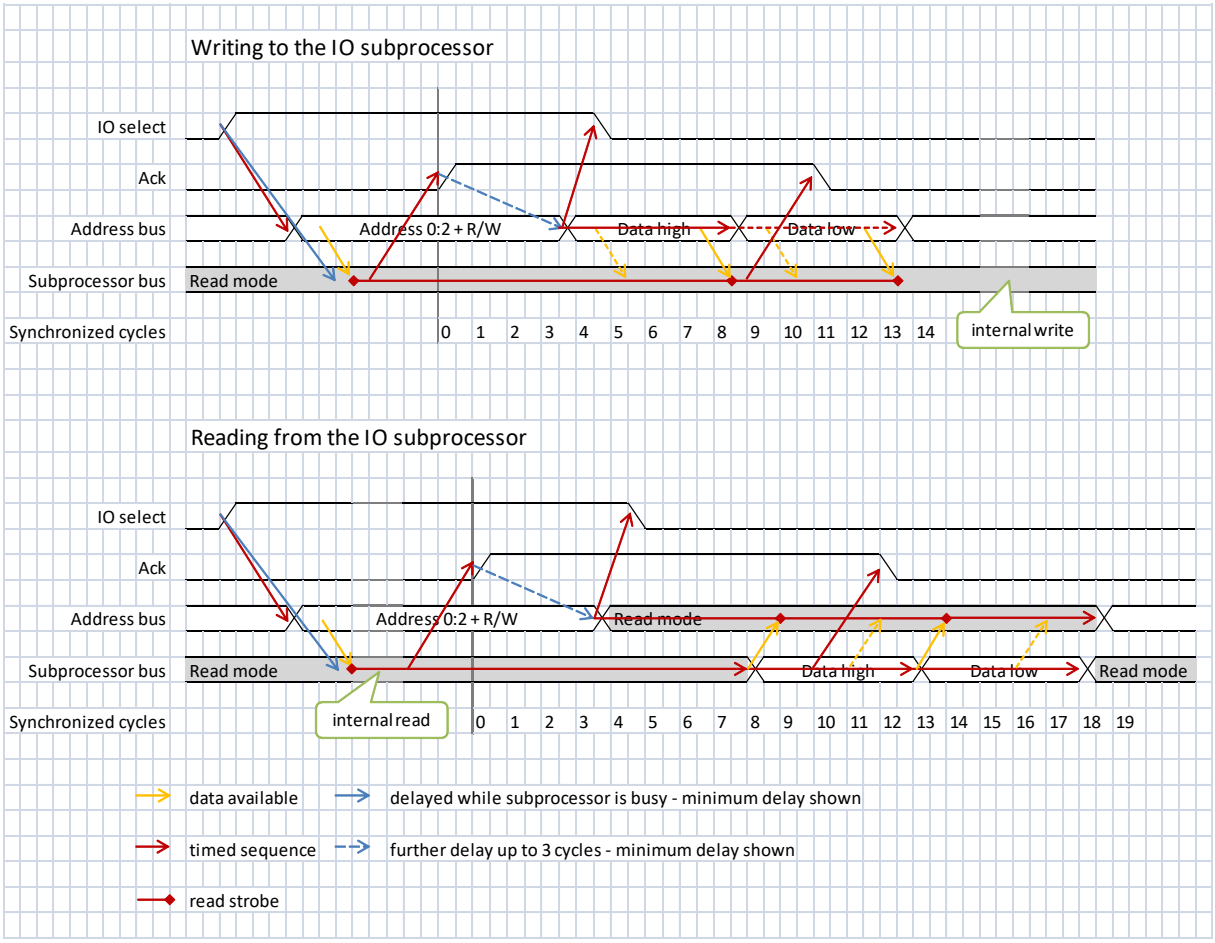
Table of content

IO subprocessor theory of operation

The IO subprocessor connects to the 6502 emulator master through a 4 bit bus. 2 extra port pins are dedicated to a master select / slave acknowledge handshake. 1 more pin acts as an open drain output to drive the master's -IRQ input.

In order to communicate with the IO subprocessor the emulator raises IO select and puts the 3-bit address (bit 0:2) and the read/-write control (bit 3) on the bus. The emulator then waits in a tight loop (3 cycles) for the IO subprocessor to respond with acknowledge.

The subprocessor interrogates the read/-write bit. If 1 (read) the data for the addressed function is fetched. Acknowledge is raised and the timed IO sequence starts.



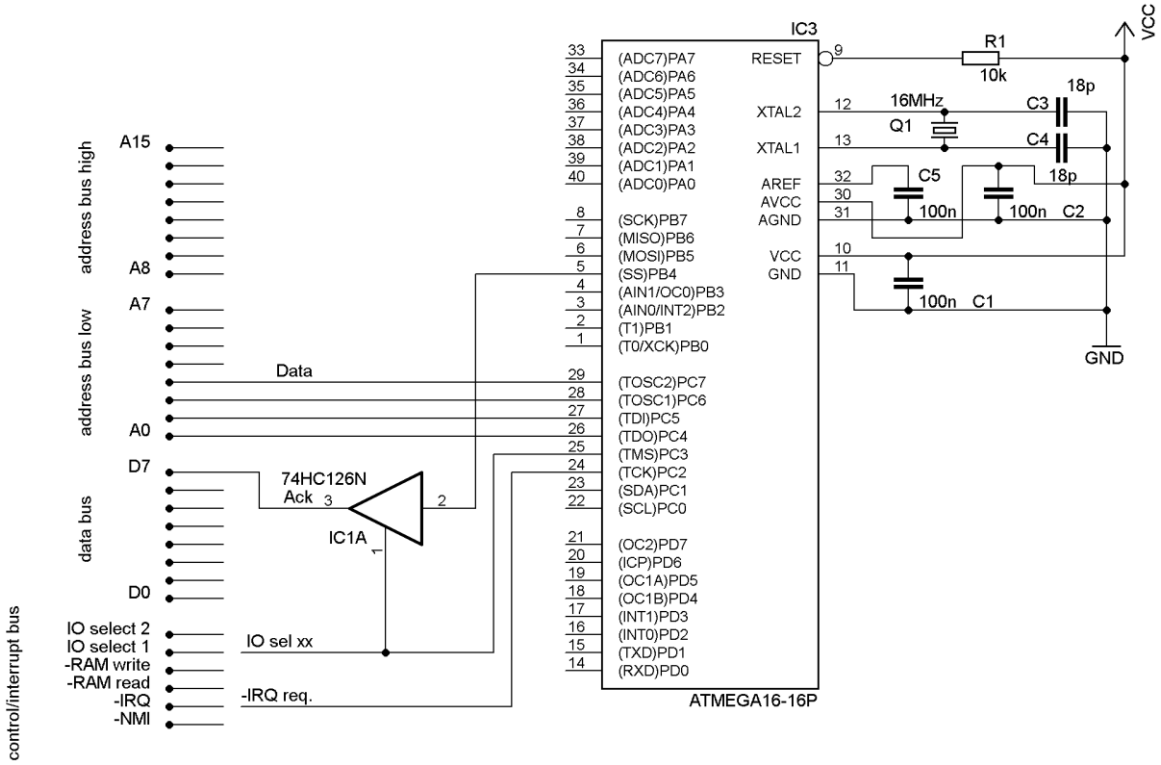
For a write the high data nibble is placed on the bus 3.5 - 6.5 cycles after acknowledge. Due to propagation delays the data is seen by the subprocessor approximately 1 cycle later (ack -> data -> bus). The bus remains valid for 5 cycles and is sampled by the subprocessor at 8.5 cycles after it sent acknowledge. Then the low data nibble is placed on the bus for 5 cycles and is sampled at 13.5 cycles after acknowledge. The data is then sent to the addressed function and the subprocessor remains busy and will not respond to IO select during that time.

For a read the address bus direction must be switched. To allow the emulator to switch the bus to input the subprocessor allows a grace period of 8 clock cycles after acknowledge before its bus is switched to output. The high data nibble is then placed on the bus for 5 cycles followed by the low data nibble. At the end the bus direction is switched back to normal at both ends.

Configuring the IO-subprocessor

The same clock speed must be used on both the master processor and the IO-subprocessor. Either use an identical crystal or supply a clock from the same canned oscillator.

The default pin configuration is shown in the circuit diagram below:



The configuration of the bus to the master processor can be modified to free specific IO pins and their associated functions. Bus control signals can be reassigned to individual pins while the 4-bit bus can only be configured to the upper or lower 4 pins of a port.

```
;
; Definition of Hardware
;
; 4 bit slave bus
.equ   sbus_bits    = 1         ;0 = bits 0:3, 1 = bits 4:7
.equ   sbus_port    = portc     ;ASSR AS2 = 0, JTAGEN fuse unprogrammed (1)!
.equ   sbus_ddr     = ddrc
.equ   sbus_in      = pinc
; chip select input - high active
.equ   cs_pin       = 3
.equ   cs_in        = pinc
; acknowledge output - high active
.equ   ack_pin      = 4         ;SPCR MSTR = 1!
.equ   ack_port     = portb
.equ   ack_ddr      = ddrb
; interrupt request (open drain, low active)
.equ   irq_pin      = 2
.equ   irq_out      = ddrc      ;set to one to force int req low
```

Pins already in use for the interface to the master must be disabled. This must also include secondary pin functions through built in IO modules of the ATMega.

```
; protected IO register bitmap, 1 = protected, 0 = writable
;
; adjust according to above IO ports usage and don't forget to
; disable the secondary functions of the IO pins used
.macro      protio
   .db 0,0,0,0                              ;0x00 TWBR TWSR TWAR TWDR
   .db 0,0,0,0                              ;0x04 ADCL ADCH ADCSRA ADMUX
   .db 0,0,0,0                              ;0x08 ACSR UBRRL UCSRB UCSRA
   .db 0,0x10,0,0                           ;0x0C UDR SPCR SPSR SPDR
   .db 0,0,0,0xfc                           ;0x10 PIND DDRD PORTD PINC
   .db 0xfc,0xfc,0x10,0x10                  ;0x14 DDRC PORTC PINB DDRB
   .db 0x10,0,0,0                           ;0x18 PORTB PINA DDRA PORTA
   .db 0,0,0,0                              ;0x1C EECR EEDR EEARL EEARH
   .db 0,0xff,0x8,0                         ;0x20 UBRRH/UCSRC WDTCR ASSR OCR2
   .db 0,0,0,0                              ;0x24 TCNT2 TCCR2 ICR1L ICR1H
   .db 0,0,0,0                              ;0x28 OCR1BL OCR1BH OCR1AL OCR1AH
   .db 0,0,0,0                              ;0x2C TCNT1L TCNT1H TCCR1B TCCR1A
   .db 0,0,0,0                              ;0x30 SFIOR OSCCAL TCNT0 TCCR0
   .db 0,0,0,0xff                           ;0x34 MCUCSR MCUCR TWCR SPMCR
   .db 0,0,0,3                              ;0x38 TIFR TIMSK GIFR GICR
   .db 0,0xff,0xff,0xff                     ;0x3C OCR0 SPL SPH SREG
.endmacro
```

Please note, that any protected bits in a register force a read prior to each write.

Hardware IO units with special support may need extra disable settings. Since pin 4 of port B is also the slave select pin of the SPI module, slave mode must be disabled by forcing the master bit of the SPI control register.

```
; required init for non default protected IO register bits
.macro      ioreg_init
   ldi   a,(1<<mstr)          ;prohibit SPI slave mode
   out   spcr,a
.endmacro
```

Always refer to the official ATMega16 manual for IO register and unit details!

During power on reset, button reset or when using the R or RC command you should see the IO subprocessor identify itself.

```
IO subprocessor @ BF70 V0.70  built Apr 14 2016 11:15:41
```

When there are problems with the bus configuration or bus hardware you will not be able to run the emulator.

```
IO subprocessor @ BF70 failed, cannot continue!
```

Using the IO-subprocessor

The IO-subprocessor has 2 basic operation modes and 8 addressable registers. The 2 operation modes are normal and vectored. The vectored mode is used, when the master processor works on a vectored interrupt.

Address	Write	Read
0	IO register	
1	IO register+1	
2	IO register+2	
3	index to IO register	pending IRQ vector
4	alternate IO register	
5	index to alt. IO register	data register
6	data register with function	
7	command register	index to data register

To access a physical AVR IO register you must first write its address to the respective index register 3 or 5. 3 consecutive AVR IO registers can be accessed with the lower index while the upper index points to an alternate AVR IO register.

Alternate IO register

The way the alternate IO register is accessed can be modified by adding a constant to the register index. When an output (PORTx) or data direction register (DDRx) is selected a read is redirected to the input register (PINx) of the same port.

Index	alternate IO register function
+0	normal register read/write
+0x40	access word register (least significant bit is ignored) * write high byte first, then low byte read low byte first, then high byte
+0x80	clear bits: logical and complemented bits with register forces a read modify write cycle next function: set bits **
+0xc0	set bits: logical or bits with register forces a read modify write cycle next function: clear bits **

* A word access must be atomic for timer 1 registers as the AVR provides only a single buffer register (see "Accessing 16-bit Registers" in the ATMega16 manual)!

Command and data register

The data register with function (address 6) allows access to some AVR hardware units automatically starting a serial or EEPROM IO operation and ending vectored mode. The read only data register (address 5) allows access to the same data register without the extra functionality.

Which unit's data and other registers are connected is determined by a connect unit command (address 7) or by the automatically loaded unit during IRQ vector processing. In addition the command register can be used to set or clear individual bits in IO port or DDR registers and for some selected IO register bits.

Command	Description
0-12	connect a hardware unit
60	DR = diagnostic read/write full address range (GPRs, IO, RAM) with post increment - precede by sreg -> register 5 (safety lock) Warning: diagnostic write bypasses the protection logic!
61	DR = write diagnostic address index high, low, then same as 60
62	end vectored mode
63	reset hardware of IO subprocessor (uses watchdog reset) precede by wdtcr -> register 5 (safety lock)
0x40-0x7F 10abbbbb	clear/set bits from list a: 0=clear, 1=set bbbbb: bitmap position 0-31
0x80-0xFF 1abbcddd	clear/set bit port/DDR a: 0=port, 1=ddr bb: port# a-d c: 0=clear, 1=set ddd: pin# 0-7

Definitions and macros with the prefix `isc_` are provided in m16def.i65.
Clear/set bits causes a rewrite of other bits in the same register and may cause unexpected results as some bits are cleared by writing a 1 to it.

List of hardware units:

	IO address	0	1	2	4	6
#	**Unit**	**Reg+0**	**Reg+1**	**Reg+2**	**AltReg**	**Dreg**
0	SPI	spcr	spsr	spdr	spcr	spdr (1)
1	TWI	twbr	twsr	twar	twcr	twdr (2)
2	USART	ubrrl	ucsrb	ucsra	ucsrc	udr
3	Ext. Interrupt	mcucsr	mcucr	~~twcr~~	gifr	gicr
4	ADC	adch	adcsra	admux	sfior	**adc:2**
5	Anacomp	~~adcsra~~	~~admux~~	acsr	sfior	**icr1:2**
6	Timer0	tcnt0	tccr0	~~mcucsr~~	tifr	ocr0
7	Timer1:ocra	tccr1b	tccr1a	sfior	**tcnt1:2**	**ocr1a:2**
8	Timer1:ocrb	tccr1b	tccr1a	sfior	**tcnt1:2**	**ocr1b:2**
9	Timer1:icr	tccr1b	tccr1a	sfior	**tcnt1:2**	**icr1:2**
10	Timer2	ocr2	tcnt2	tccr2	tifr	timsk
11	EEPROM	eedr	eearl	eearh	eecr	eedr (3)
12	EEP address	eedr	eearl	eearh	eecr	eearh (4)

~~Reg~~ = off unit **Reg:2** = word

(1) start spi by rewriting the last written pattern on read
non vectored: access spcr before spdr
(2) start twi (1 -> twcr.twint, twcr.twen, (twcr.twea on read))
(3) load address from shadow eear, increment shadow
on write: 1 -> eecr.eemwe, 1 -> eecr.eewe
(4) load shadow eearh, eearl, then same as (3)

List of IO register bits, add 0x40 to clear, 0x60 to set.

```
   regbit   adcsra,adie       ; 00: analog to digital converter interrupt enable
   regbit   acsr,acie         ; 01: analog comparator interrupt enable
   regbit   ucsrb,rxcie       ; 02: USART receive complete interrupt enable
   regbit   ucsrb,txcie       ; 03: USART transmit complete interrupt enable
   regbit   ucsrb,udrie       ; 04: USART data register empty interrupt enable
   regbit   spcr,spie         ; 05: SPI interrupt enable
   regbit   eecr,eerie        ; 06: EEPROM ready interrupt enable
   regbit   twcr,twie         ; 07: TWI/I2C interrupt enable
   regbit   timsk,ocie2       ; 08: timer2 output compare match interrupt enable
   regbit   timsk,toie2       ; 09: timer2 overflow interrupt enable
   regbit   timsk,ticie1      ; 10: timer1 input capture interrupt enable
   regbit   timsk,ocie1a      ; 11: timer1 output compare match a interrupt enable
   regbit   timsk,ocie1b      ; 12: timer1 output compare match b interrupt enable
   regbit   timsk,toie1       ; 13: timer1 overflow interrupt enable
   regbit   timsk,ocie2       ; 14: timer0 output compare match interrupt enable
   regbit   timsk,toie2       ; 15: timer0 overflow interrupt enable
   regbit   gicr,int1         ; 16: external interrupt request 1 enable
   regbit   gicr,int0         ; 17: external interrupt request 0 enable
   regbit   gicr,int2         ; 18: external interrupt request 2 enable
```

Interrupt handling

When the master processor receives an interrupt request, it can interrogate whether the source of the interrupt is the IO-subprocessor and which AVR internal vector is pending by reading the pending IRQ vector at address 3. If the subprocessor returns a non-zero vector, it automatically switches to vectored mode and loads a vector specific set of indexes for the corresponding hardware unit.

List of vectors:

			IO address	0	1	2	4	6
#	**Source**	**#**	**Unit**	**Reg+0**	**Reg+1**	**Reg+2**	**AltReg**	**Dreg**
2	int0	3	Ext. Interrupt	mcucsr	mcucr	~~twcr~~	gifr	gicr
4	int1	3	Ext. Interrupt	mcucsr	mcucr	~~twcr~~	gifr	gicr
6	t2 comp	10	Timer2	ocr2	tcnt2	tccr2	tifr	timsk
8	t2 ovf	10	Timer2	ocr2	tcnt2	tccr2	tifr	timsk
10	t1 capt	9	Timer1:icr	tccr1b	tccr1a	sfior	**tcnt1:2**	**icr1:2**
12	t1 compa	7	Timer1:ocra	tccr1b	tccr1a	sfior	**tcnt1:2**	**ocr1a:2**
14	t1 compb	8	Timer1:ocrb	tccr1b	tccr1a	sfior	**tcnt1:2**	**ocr1b:2**
16	t1 ovf	7	Timer1:ocra	tccr1b	tccr1a	sfior	**tcnt1:2**	**ocr1a:2**
18	t0 ovf	6	Timer0	tcnt0	tccr0	~~mcucsr~~	tifr	ocr0
20	spi stc	0	SPI	spcr	spsr	spdr	spcr	spdr (1) *
22	usart rxc	2	USART	ubrrl	ucsrb	ucsra	ucsrc	udr *
24	usart udre	2	USART	ubrrl	ucsrb	ucsra	ucsrc	udr *
26	usart txc	2	USART	ubrrl	ucsrb	ucsra	ucsrc	udr *
28	adc	4	ADC	adch	adcsra	admux	sfior	**adc:2**
30	eep rdy	11	EEPROM	eedr	eearl	eearh	eecr	eedr (3) *
32	ana comp	5	Anacomp	~~adcsra~~	~~admux~~	acsr	sfior	**icr1:2**
34	twi	1	TWI	twbr	twsr	twar	twcr	twdr (2)*
36	int2	3	Ext. Interrupt	mcucsr	mcucr	~~twcr~~	gifr	gicr
38	t0 comp	6	Timer0	tcnt0	tccr0	~~mcucsr~~	tifr	ocr0
40	spm rdy	--	disabled	---------- no change ----------				

~~Reg~~ = off unit **Reg:2** = word * = access ends vectored mode

(1) start spi by rewriting the last written pattern on read
(2) start twi (1 -> twcr.twint, twcr.twen, (twcr.twea on read))
(3) load address from shadow eear, increment shadow
on write: 1 -> eecr.eemwe, 1 -> eecr.eewe

Definitions of vector numbers are provided in m16def.i65 with the prefix `isv_`.

You can modify the index registers as needed during interrupt processing. When interrupt handling is complete for the current vector and the last data register access did not automatically end the vectored mode, you must read the pending IRQ vector register again to end vectored mode. The normal mode index registers are restored automatically.

Usage examples

All usage examples begin with the definition of the IO subprocessor addresses and the internal IO registers of the subprocessor.

```
iomap   equ $bf00       ;I/O page (A15:8)
; IO subprocessor (ATMega16)
ios_    equ iomap+$70   ;base address of the IO subprocessor
ios_r0  equ ios_        ;IO register @ ios_adr, read/write
ios_r1  equ ios_+1      ;IO register @ ios_adr+1, read/write
ios_r2  equ ios_+2      ;IO register @ ios_adr+2, read/write
ios_adr equ ios_+3      ;IO register address, write only
ios_vec equ ios_+3      ;IRQ vector register, read only
ios_ar  equ ios_+4      ;alternate IO register @ ios_aad, read/write
ios_aad equ ios_+5      ;alternate IO register address, write only
ios_xdr equ ios_+5      ;unit data register without special function, read only
ios_dr  equ ios_+6      ;unit data register with extra functions, read/write
ios_cr  equ ios_+7      ;command register, write only
ios_drx equ ios_+7      ;current unit data register index, read only

        nolist
        include "m16def.i65"
        list
```

See Appendix A for the definitions in "m16def.inc" and refer to the original ATMega16 manual.

Bit banging IO ports

For direct IO to input and output registers of the IO subprocessor you have 3 choices.

1. Select a PINx register by writing its address to the IO register index at address 3 allows access to the input register PINx at address 0, the data direction register DDRx at address 1 and the output register PORTx at address 2 of the IO subprocessor.
2. Select an output or data direction register (PORTx or DDRx) by writing its address to the alternate IO register index at address 5. The addressed register is available at address 4 of the IO subprocessor. You may choose to only set bits (logical or) by adding 0xC0 or clear bits (logical and with complement) by adding 0x80. You can toggle bits by writing the same bit pattern twice to address 4 as the register alternates between set and clear. A read is always redirected to the input register PINx of the same port.
3. Use the command register at address 7 to set or clear a single bit of an output or data direction register. Example: SPI is selected as the current unit and you want to clear or set an output pin as slave select without taking the focus away from the SPI registers.
   Command bit positions:

```
;1abbcddd  set or clear a port-bit
;     ddd- pin# 0-7
;    c---- 0=clear, 1=set
;  bb----- port# a-d
; a------- 0=port, 1=ddr
```

```
method1_initialize
        lda #is_pina    ;point index to lowest port a register
        sta ios_adr     ;r0 = input, r1 = ddr, r2 = output
        lda #$f         ;pin 0-3 = output, pin 4-7 = input
        sta ios_r1
method1_write
        lda #$aa        ;output = %xxxx1010, xxxx is ignored / input
        sta ios_r2
method1_read
        lda ios_r0      ;input = %xxxxyyyy, xxxx = input, yyyy = previous output

method2_initialize
        lda #is_ddrd    ;point alternate index to port d ddr register
        sta ios_aad
        lda #$c0        ;pin 6-7 = output, pin 0-5 = input
        sta ios_ar
        lda #is_portd   ;point alternate index to port d output register
        sta ios_aad
method2_write
        lda #$40        ;set bit 6, clear bit 7
        sta ios_ar
method2_read
        lda ios_ar      ;a read always selects the input register of a port!
method2_toggle
        lda #is_portd+$c0   ;set (logical or) bits to register
        sta ios_aad         ;+$80 = clear (logical and with complement)
        lda #$80        ;bit 7
        sta ios_ar      ;set bit 7
        sta ios_ar      ;clear bit 7
        sta ios_ar      ;set bit 7
        ;...

method3_initialize
        isc_portbit 1,1,1,3 ;set ddr port b pin 3 (b3 = output)
method3_write
        isc_portbit 1,0,1,3 ;set port b pin 3 (b3 = 1)
        isc_portbit 0,0,1,3 ;clear port b pin 3 (b3 = 0)
        ; there is no read for this method!
```

Using a timer interrupt as a jiffy clock

This example involves 2 steps:

1. Setting up a timer with a constant duration of 20 ms / 50 Hz. To calculate the timer register values (and other ATMega parameters) I recommend AVRCalc.
2. Dispatching the resulting IRQ to the jiffy clock handler. Unsolicited subprocessor interrupts should be trapped as errors and non-subprocessor IRQs should be forwarded to their respective handler.

```
jiffy_initialize
        lda #9          ;load unit index: timer 1 with icr1
        sta ios_cr      ;r0=tccr1b r1=tccr1a r2=sfior ar=tcnt1:2 dr=icr1:2
        lda #0          ;clear tcnt1
        sta ios_ar
        sta ios_ar
        lda #$9c        ;set ceiling for 20ms @ 16Mhz/8
        sta ios_dr      ;store high byte 1st
        lda #$3f        ;then low byte
        sta ios_dr
        lda #(1<<is_wgm13)|(1<<is_wgm12)|(1<<is_cs10)
        sta ios_r0      ;T1 CTC top=ICR, clock/8
        lda #$60+10     ;set timsk,ticie1 (enable ICR interrupt)
        sta ios_cr
        cli
main
        jmp main        ;do nothing in main

irq_dispatch            ;interrogate source of irq
        pha             ;save registers
        txa
        pha
        ldx ios_vec     ;read vector, if <> 0 then subprocessor = vectored mode
        cpx #10         ;lower 6 vectors only, expand irq_table for more
        bcc irq_unsol
        lda irq_table+1,x ;load vector dispatch address
        pha
        lda irq_table,x
        pha
        rts
irq_table
        dw  irq_other-1,irq_unsol-1,irq_unsol-1,irq_unsol-1,irq_unsol-1
        dw  irq_jiffy-1
;on a 65c02 the code can be shorter
;        ...
;        bcc irq_unsol
;        jmp (irq_table,x)
;irq_table
;        dw  irq_other,irq_unsol,irq_unsol,irq_unsol,irq_unsol,irq_jiffy

jiffy_count             ;20ms tick counter - should reside in RAM
        ds  2
irq_jiffy               ;irq source is icr1
        inc jiffy_count ;count jiffies
        bne irq_exit2
        inc jiffy_count+1
irq_exit2
        bit ios_vec     ;dummy read, subprocessor = normal mode
irq_exit
        pla             ;restore registers
        tax
        pla
        rti
irq_other               ;scan irq sources other than the subprocessor
        jmp irq_exit
irq_unsol               ;unsolicited subprocessor irq - add error handler
        jmp *           ;loop until reset

        org $fffe       ;set global IRQ vector
        dw  irq_dispatch
```

TWI/ I²C Commands

Code	Name	Description
0x11	Start	Send Start and connect DR & SR to TWI, next write DR is I²C device address
0x12	Nak	Set acknowledge to 0 before reading the last 2 bytes
0x13	Stop	Send Stop and disconnect SR & DR

TWI/I²C IO Sequence

I²C Op	Application	Interface
Start	Write Command(Start) -> XR	if state = 0 wait TWCR:TWSTP=0 (from idle) if state = 0x11 - 0x12 wait TWCR:TWINT=1 (rep. start) TWCR:TWSTA=1 * state = 0x10
	Write Data (Address) -> DR	if state = 0x10 (slave address) wait TWCR:TWINT=1 TWDR=address TWCR= * control=TWINT,TWEA,TWEN (Ack on read) if address:0 = 0 then state = 0x11 (write enable) if address:0 = 1 then state = 0x12 (read enable)
	if only reading one byte Write Command(Nak) -> XR	control=TWINT,TWEN (send NAK on read)
	Read Status <- SR device responding?	if state = 0x10 - 0x12 (TWI active) wait TWCR:TWINT=1 get TWSR if TWSR=0x40** then TWCR:TWEA=control
Write	Write Data -> DR	if state = 0x11 (write enable) wait TWCR:TWINT=1 TWDR=data TWCR= *
	optional Read Status <- SR next write acknowledged?	if state = 0x10 - 0x12 wait TWCR:TWINT=1 get TWSR
Read	if last 2 bytes to be read Write Command(Nak) -> XR	control=TWINT,TWEN (send NAK on read)
	Read Data <- DR	if state = 0x12 (read enable) wait TWCR:TWINT=1 get TWDR If control:TWSTP=0 then TWCR:TWEA=control If control:TWEA=0 then control:TWSTP=1
Stop	Write Command(Stop) -> XR	if state = 0x10 - 0x12 (TWI active) wait TWCR:TWINT=1 TWCR:TWSTP=1 * state = 0 (idle)

* TWCR:TWEN & TWINT bits are written to 1 enabling the TWI and starting the TWI operation

** TWSR 0x40 = read address acknowledge

Writing to an I²C device

1. Write 0x11 (TWI Start) to XR
2. Write I²C device address for write to DR
3. Read SR - verify ack received 0x18, device responding
4. Write data byte to DR
5. optionally Read SR - verify ack received 0x28, device accepts more data
6. Repeat from 4. for desired number of bytes
7. Write 0x13 (TWI Stop) to XR

If the device does not respond in step 3, the program may have to wait for devices to become ready again (for example an EEPROM busy executing its internal write cycle).

Instead of the stop command another start can be issued (repeated start condition).

Reading from an I²C device

1. Write 0x11 (TWI Start) to XR
2. Write I²C device address for read (+1) to DR
3. if only one byte to be read write 0x12 (Nak) to XR
4. Read SR - verify ack received 0x40, device responding
5. if there is only two remaining bytes to be read write 0x12 (Nak) to XR
6. Read data byte from DR
7. Repeat from 5. for desired number of bytes
8. Write 0x13 (stop) to XR

If the device does not respond in step 4, the program may have to wait for devices to become ready again (for example an EEPROM busy executing its internal write cycle).

Instead of the stop command another start can be issued (repeated start condition).

SPI Commands

Code	Name	Description
0x31	Start	Activate SPI, chip select and connect DR & SR to SPI
0x32	Stop	Deactivate chip select after last SPI IO is complete and disconnect DR & SR
0x33	SetCS	Set chip select pin with next DR write DR bits: 00ttlnnn - t = port#, l = polarity (0=low/1=high active), n = pin#

SPI IO Sequence

SPI Op	Application	Interface
Start with new CS	Write Command(SetCS) -> XR	if state = 0 CS = disabled state = 0x01 (set SPI CS)
	Write Data (CS address) -> DR	if state = 0x01 (set SPI CS) if data = valid CS address & not masked then CS = data (port A-D, polarity, pin) state = 0x00 (idle)
Start or Restart	Write Command(Start) -> XR	if state = 0 (idle) or state 0x30 - 0x31 (SPI active) if state = 0x31 wait SPSR:SPIF=1 if state = 0 then SPCR:SPEN,MSTR = 1; CS = active else pulse CS inactive, active state = 0x30 (SPI active)
Write	Write Data -> DR	if state 0x30 - 0x31 (SPI active) if state = 0x31 wait SPSR:SPIF=1 SPDR = data spipat = data (save pattern to clock read) state = 0x31 (SPI wait SPIF)
Read	Read Data <- DR	if state 0x30 - 0x31 (SPI active) if state = 0x31 wait SPSR:SPIF=1 get SPDR SPDR = spipat (clock next read) state = 0x31 (SPI wait SPIF)
Read last or duplex	Read Data <- SR	if state 0x30 - 0x31 (SPI active) if state = 0x31 wait SPSR:SPIF=1 get SPDR state = 0x30 (SPI active)
Stop	Write Command(Stop) -> XR	if state 0x30 - 0x31 (SPI active) if state = 0x31 wait SPSR:SPIF=1 CS = inactive state = 0 (idle)

Writing to an SPI device

1. Write 0x33 (SPI SetCS) to XR
2. Write 0b00ttlnnn (port, polarity & pin) to DR
3. Write 0x31 (SPI Start) to XR
4. Write data byte to DR
5. Repeat 4. for desired number of bytes
6. Write 0x32 (SPI Stop) to XR

Steps 1. and 2. (SPI SetCS) can be omitted, if communicating with the same device as before.
Step 6. (SPI Stop) can be omitted, if communicating with the same device afterwards.

Reading from an SPI device

1. Write 0x33 (SPI SetCS) to XR
2. Write 0b00ttlnnn (port, polarity & pin) to DR
3. Write 0x31 (SPI Start) to XR
4. Write a pattern byte to DR to clock the first read
5. Read data byte from DR - the next read is clocked with the pattern from step 4.
6. Repeat 5. for desired number of bytes -1
7. Read the last data byte from SR - no further read is clocked
8. Write 0x32 (SPI Stop) to XR

Steps 1. and 2. (SPI SetCS) can be omitted, if communicating with the same device as before.
Step 8. (SPI Stop) can be omitted, if communicating with the same device afterwards.

Full Duplex IO with an SPI device

1. Write 0x33 (SPI SetCS) to XR
2. Write 0b00ttlnnn (port, polarity & pin) to DR
3. Write 0x31 (SPI Start) to XR
4. Write a data byte to DR
5. Read data byte from SR
6. Repeat from 4. for desired number of bytes
7. Write 0x32 (SPI Stop) to XR

Steps 1. and 2. (SPI SetCS) can be omitted, if communicating with the same device as before.
Step 7. (SPI Stop) can be omitted, if communicating with the same device afterwards.

USART Commands

Code	Name	Description
0x21	Start	Connect DR & SR to USART
0x?1		Any other Start command disconnects DR &SR from USART

USART IO Sequence

USART Op	Application	Interface
Start	Write Command(Start) -> XR	UCSRB:TXEN,RXEN = 1 state = 0x20
Write	Write Data -> DR	if state = 0x20 (USART enable) wait UCSRA:UDRE=1 TWDR=data
Read	Read Status <- SR	if state = 0x20 (USART enable) get UCSRA
	Read Data <- DR	if state = 0x20 (USART enable) if UCSRA:RXC = 1 then get UDR else get 0x00 (null character)

Writing to an USART device

1. Write 0x21 (USART start) to XR
2. Write data byte to DR
3. Repeat 3. for desired number of bytes
4. Write 0x?1 (any Start) to XR

Reading from an USART device

1. Write 0x21 (USART start) to XR
2. Read data byte from DR
3. Repeat 3. for desired number of bytes
4. Write 0x?1 (any Start) to XR

IO subprocessor internal EEPROM Commands

Code	Name	Description
0x41	Start	Connect DR to internal EEPROM
0x?1		Any other Start command disconnects DR &SR from internal EEPROM

Internal EEPROM Sequence

EEP Op	Application	Interface
Start	Write Command(Start) -> XR	state = 0x40 (set internal EEPROM address low)
	Write EEP adr low -> DR	if state = 0x40 shadow EEARL = data state = 0x41 (set internal EEPROM address high)
	Write EEP adr high -> DR	if state = 0x41 shadow EEARH = data state = 0x42 (internal EEPROM enable)
Write	Write Data -> DR	if state = 0x42 (EEP enable) wait EECR:EEWE = 0 EEAR = shadow EEAR EEDR = data EECR:EEMWE = 1 EECR:EEWE = 1 shadow EEAR ++
Read	Read Data <- DR	if state = 0x42 (EEP enable) wait EECR:EEWE = 0 EEAR = shadow EEAR EECR:EERE = 1 get EEDR shadow EEAR ++

Writing to the internal EEPROM

1. Write 0x41 (internal EEPROM Start) to XR
2. Write EEP address low to DR
3. Write EEP address high to DR
4. Write data byte to DR *
5. Repeat 4. for desired number of bytes
6. Write 0x?1 (any Start) to XR

* The address is automatically incremented after every byte and wraps around if the size of the EEPROM is exceeded.

Reading from the internal EEPROM

1. Write 0x41 (internal EEPROM Start) to XR
2. Write EEP address low to DR
3. Write EEP address high to DR
4. Read data byte from DR
5. Repeat 4. for desired number of bytes
6. Write 0x?1 (any Start) to XR

* The address is automatically incremented after every byte and wraps around if the size of the EEPROM is exceeded.

Accessing the subprocessor's EEPROM

It would not be possible to write to the subprocessor's internal EEPROM directly using the subprocessor's IO registers, because the write requires a master write enable and the write pulse must follow within 4 subprocessor clock cycles.

When the EEPROM registers are connected through a connect unit command, accessing the data register loads the EEPROM address from a shadow copy and increments the shadow for the next access. A write to the data register automatically triggers the internal write process, a read triggers the read process. After a write the program must wait for the write process to complete either by waiting for the write enable bit to return to 0 or by enabling the EEPROM ready interrupt.

```
        org 0
eep_buf ds  2           ;eeprom buffer pointer
eep_y   ds  1           ;index to eeprom write buffer
        org $200
text    db  "This is only a test",0
eep_write
        lda lo(text)    ;initialize buffer pointer
        sta eep_buf
        lda hi(text)
        sta eep_buf+1
        lda #0          ;index
        sta eep_y
        lda #12         ;connect eeprom and set eeprom address
        sta ios_cr
        lda #1          ;address = $100
        sta ios_dr
        lda #0
        sta ios_dr      ;automatic connect to unit 11 (EEPROM data)
        lda #(1<<is_eewe) ;wait for previous write to complete
eepw_wait
        bit ios_ar      ;test eewe
        bne eepw_wait
        lda #(1<<is_eerie) ;enable EEPROM ready interrupt
        sta ios_ar      ;actual write is done in irw_eep_write
eepw_end
        bit ios_ar      ;wait until irq handler clears eerie
        bne eepw_end    ;  (last byte being written)

eep_read
        lda lo(readbuf) ;initialize buffer pointer
        sta eep_buf
        lda hi(readbuf)
        sta eep_buf+1
        ldy #-1         ;pre increment index
        lda #12         ;connect eeprom address
        sta ios_cr
        lda #1          ;address = $100
        sta ios_dr
        lda #0
        sta ios_dr      ;automatic connect to unit 11 (EEPROM data)
        lda #(1<<is_eewe) ;wait for last previous write to complete
eepr_wait
        bit ios_ar      ;test eewe
        bne eepr_wait
eepr_loop
        iny
        lda ios_dr      ;read EEPROM with automatic address increment
        sta (eep_buf),y
        bne eepr_loop   ;delimiter (0)?

        jmp *           ;stop

readbuf ds  256         ;must reside in RAM
```

```
irq_dispatch            ;interrogate source of irq
        pha             ;save accu
        lda ios_vec     ;read vector, if <> 0 then subprocessor = vectored mode
        beq irq_other
        cmp #isv_erdy   ;EEPROM ready?
        bne irq_unsol
irq_eep_write
        tya             ;save y
        pha
; if the eeprom is written from the main program instead you must wait for
; the eewe bit to become 0
;        lda #11         ;select EEPROM if not already selected
;        sta ios_cr
;        lda #(1<<is_eewe)
;eep_wr_wait
;        bit ios_ar      ;test eewe
;        bne eep_wr_wait
        ldy #eep_y
        lda (eep_buf),y
        bne irq_eep_cont ;buffer delimiter (0)?
        sta ios_ar      ;clear eeprom ready interrupt enable (0->eerie)
irq_eep_cont
        sta ios_dr      ;write eep, increment address & exit vectored mode
        inc eep_y       ;next write index
        pla             ;restore registers
        tay
        pla
        rti
irq_other               ;add scan irq sources other than the subprocessor
        pla
        rti
irq_unsol               ;unsolicited subprocessor irq - add error handler
        jmp *           ;loop until reset

        org $fffe       ;set global IRQ vector
        dw  irq_dispatch
```

Appendix A - 6502 assembler include file

Standard assembler definitions for the IO subprocessor (m16def.i65):

```
;
; ATMega16 IO subprocessor for the 6502_Emu project
;
; ***** I/O REGISTER DEFINITIONS ********************************************
;
is_sreg     equ 0x3f  ;write protected
is_spl      equ 0x3d  ;write protected
is_sph      equ 0x3e  ;write protected
is_ocr0     equ 0x3c
is_gicr     equ 0x3b
is_gifr     equ 0x3a
is_timsk    equ 0x39
is_tifr     equ 0x38
is_spmcsr   equ 0x37  ;self programming disabled
is_twcr     equ 0x36
is_mcucr    equ 0x35  ;no sleep modes
is_mcucsr   equ 0x34
is_tccr0    equ 0x33
is_tcnt0    equ 0x32
is_osccal   equ 0x31  ;not used
is_ocdr     equ 0x31  ;not used
is_sfior    equ 0x30
is_tccr1a   equ 0x2f
is_tccr1b   equ 0x2e
is_tcnt1l   equ 0x2c
is_tcnt1h   equ 0x2d
is_ocr1al   equ 0x2a
is_ocr1ah   equ 0x2b
is_ocr1bl   equ 0x28
is_ocr1bh   equ 0x29
is_icr1l    equ 0x26
is_icr1h    equ 0x27
is_tccr2    equ 0x25
is_tcnt2    equ 0x24
is_ocr2     equ 0x23
is_assr     equ 0x22
is_wdtcr    equ 0x21  ;watchdog disabled
is_ubrrh    equ 0x20
is_ucsrc    equ 0x20
is_eearl    equ 0x1e
is_eearh    equ 0x1f
is_eedr     equ 0x1d
is_eecr     equ 0x1c
is_porta    equ 0x1b
is_ddra     equ 0x1a
is_pina     equ 0x19
is_portb    equ 0x18
is_ddrb     equ 0x17
is_pinb     equ 0x16
is_portc    equ 0x15
is_ddrc     equ 0x14
is_pinc     equ 0x13
is_portd    equ 0x12
is_ddrd     equ 0x11
is_pind     equ 0x10
is_spdr     equ 0x0f
is_spsr     equ 0x0e
is_spcr     equ 0x0d  ;slave mode disabled
is_udr      equ 0x0c
is_ucsra    equ 0x0b
is_ucsrb    equ 0x0a
is_ubrrl    equ 0x09
is_acsr     equ 0x08
is_admux    equ 0x07
is_adcsra   equ 0x06
is_adch     equ 0x05
```

```
is_adcl     equ 0x04
is_twdr     equ 0x03
is_twar     equ 0x02
is_twsr     equ 0x01
is_twbr     equ 0x00


; ***** BIT DEFINITIONS ************************************************

; ***** TIMER_COUNTER_0 **************
; TCCR0 - Timer/Counter Control Register
is_cs00     equ 0 ; Clock Select 1
is_cs01     equ 1 ; Clock Select 1
is_cs02     equ 2 ; Clock Select 2
is_wgm01    equ 3 ; Waveform Generation Mode 1
is_com00    equ 4 ; Compare match Output Mode 0
is_com01    equ 5 ; Compare Match Output Mode 1
is_wgm00    equ 6 ; Waveform Generation Mode 0
is_foc0     equ 7 ; Force Output Compare

; TIMSK - Timer/Counter Interrupt Mask Register
is_toie0    equ 0 ; Timer/Counter0 Overflow Interrupt Enable
is_ocie0    equ 1 ; Timer/Counter0 Output Compare Match Interrupt register

; TIFR - Timer/Counter Interrupt Flag register
is_tov0     equ 0 ; Timer/Counter0 Overflow Flag
is_ocf0     equ 1 ; Output Compare Flag 0

; SFIOR - Special Function IO Register
is_psr10    equ 0 ; Prescaler Reset Timer/Counter1 and Timer/Counter0


; ***** TIMER_COUNTER_1 **************
; TIMSK - Timer/Counter Interrupt Mask Register
is_toie1    equ 2 ; Timer/Counter1 Overflow Interrupt Enable
is_ocie1b   equ 3 ; Timer/Counter1 Output CompareB Match Interrupt Enable
is_ocie1a   equ 4 ; Timer/Counter1 Output CompareA Match Interrupt Enable
is_ticie1   equ 5 ; Timer/Counter1 Input Capture Interrupt enable

; TIFR - Timer/Counter Interrupt Flag register
is_tov1     equ 2 ; Timer/Counter1 Overflow Flag
is_ocf1b    equ 3 ; Output Compare Flag 1B
is_ocf1a    equ 4 ; Output Compare Flag 1A
is_icf1     equ 5 ; Input Capture Flag 1

; TCCR1A - Timer/Counter1 Control Register A
is_wgm10    equ 0 ; Waveform Generation Mode
is_wgm11    equ 1 ; Waveform Generation Mode
is_foc1b    equ 2 ; Force Output Compare 1B
is_foc1a    equ 3 ; Force Output Compare 1A
is_com1b0   equ 4 ; Compare Output Mode 1B, bit 0
is_com1b1   equ 5 ; Compare Output Mode 1B, bit 1
is_com1a0   equ 6 ; Compare Ouput Mode 1A, bit 0
is_com1a1   equ 7 ; Compare Output Mode 1A, bit 1

; TCCR1B - Timer/Counter1 Control Register B
is_cs10     equ 0 ; Prescaler source of Timer/Counter 1
is_cs11     equ 1 ; Prescaler source of Timer/Counter 1
is_cs12     equ 2 ; Prescaler source of Timer/Counter 1
is_wgm12    equ 3 ; Waveform Generation Mode
is_wgm13    equ 4 ; Waveform Generation Mode
is_ices1    equ 6 ; Input Capture 1 Edge Select
is_icnc1    equ 7 ; Input Capture 1 Noise Canceler


; ***** EXTERNAL_INTERRUPT ***********
; GICR - General Interrupt Control Register
is_ivce     equ 0 ; Interrupt Vector Change Enable
is_ivsel    equ 1 ; Interrupt Vector Select
```

```
is_int2     equ 5 ; External Interrupt Request 2 Enable
is_int0     equ 6 ; External Interrupt Request 0 Enable
is_int1     equ 7 ; External Interrupt Request 1 Enable

; GIFR - General Interrupt Flag Register
is_intf2    equ 5 ; External Interrupt Flag 2
is_intf0    equ 6 ; External Interrupt Flag 0
is_intf1    equ 7 ; External Interrupt Flag 1

; MCUCR - General Interrupt Control Register
is_isc00    equ 0 ; Interrupt Sense Control 0 Bit 0
is_isc01    equ 1 ; Interrupt Sense Control 0 Bit 1
is_isc10    equ 2 ; Interrupt Sense Control 1 Bit 0
is_isc11    equ 3 ; Interrupt Sense Control 1 Bit 1

; MCUCSR - MCU Control And Status Register
is_isc2     equ 6 ; Interrupt Sense Control 2


; ***** EEPROM ***********************
; EECR - EEPROM Control Register
is_eere     equ 0 ; EEPROM Read Enable
is_eewe     equ 1 ; EEPROM Write Enable
is_eemwe    equ 2 ; EEPROM Master Write Enable
is_eerie    equ 3 ; EEPROM Ready Interrupt Enable


; ***** CPU **************************
; MCUCSR - MCU Control And Status Register
is_porf     equ 0 ; Power-on reset flag
is_extrf    equ 1 ; External Reset Flag
is_borf     equ 2 ; Brown-out Reset Flag
is_wdrf     equ 3 ; Watchdog Reset Flag
is_jtrf     equ 4 ; JTAG Reset Flag
is_jtd      equ 7 ; JTAG Interface Disable

; SFIOR - Special function I/O register
;    equPSR10    equ 0 ; Prescaler reset
is_psr2     equ 1 ; Prescaler reset
is_pud      equ 2 ; Pull-up Disable


; ***** TIMER_COUNTER_2 **************
; TIMSK - Timer/Counter Interrupt Mask register
is_toie2    equ 6 ; Timer/Counter2 Overflow Interrupt Enable
is_ocie2    equ 7 ; Timer/Counter2 Output Compare Match Interrupt Enable

; TIFR - Timer/Counter Interrupt Flag Register
is_tov2     equ 6 ; Timer/Counter2 Overflow Flag
is_ocf2     equ 7 ; Output Compare Flag 2

; TCCR2 - Timer/Counter2 Control Register
is_cs20     equ 0 ; Clock Select bit 0
is_cs21     equ 1 ; Clock Select bit 1
is_cs22     equ 2 ; Clock Select bit 2
is_wgm21    equ 3 ; Waveform Generation Mode
is_com20    equ 4 ; Compare Output Mode bit 0
is_com21    equ 5 ; Compare Output Mode bit 1
is_wgm20    equ 6 ; Waveform Genration Mode
is_foc2     equ 7 ; Force Output Compare

; ASSR - Asynchronous Status Register
is_tcr2ub   equ 0 ; Timer/counter Control Register2 Update Busy
is_ocr2ub   equ 1 ; Output Compare Register2 Update Busy
is_tcn2ub   equ 2 ; Timer/Counter2 Update Busy
is_as2      equ 3 ; Asynchronous Timer/counter2


; ***** SPI **************************
```

```
; SPSR - SPI Status Register
is_spi2x    equ 0 ; Double SPI Speed Bit
is_wcol     equ 6 ; Write Collision Flag
is_spif     equ 7 ; SPI Interrupt Flag

; SPCR - SPI Control Register
is_spr0     equ 0 ; SPI Clock Rate Select 0
is_spr1     equ 1 ; SPI Clock Rate Select 1
is_cpha     equ 2 ; Clock Phase
is_cpol     equ 3 ; Clock polarity
is_mstr     equ 4 ; Master/Slave Select
is_dord     equ 5 ; Data Order
is_spe      equ 6 ; SPI Enable
is_spie     equ 7 ; SPI Interrupt Enable


; ***** USART ************************
; UCSRA - USART Control and Status Register A
is_mpcm     equ 0 ; Multi-processor Communication Mode
is_u2x      equ 1 ; Double the USART transmission speed
is_upe      equ 2 ; Parity Error
is_dor      equ 3 ; Data overRun
is_fe       equ 4 ; Framing Error
is_udre     equ 5 ; USART Data Register Empty
is_txc      equ 6 ; USART Transmitt Complete
is_rxc      equ 7 ; USART Receive Complete

; UCSRB - USART Control and Status Register B
is_txb8     equ 0 ; Transmit Data Bit 8
is_rxb8     equ 1 ; Receive Data Bit 8
is_ucsz2    equ 2 ; Character Size
is_txen     equ 3 ; Transmitter Enable
is_rxen     equ 4 ; Receiver Enable
is_udrie    equ 5 ; USART Data register Empty Interrupt Enable
is_txcie    equ 6 ; TX Complete Interrupt Enable
is_rxcie    equ 7 ; RX Complete Interrupt Enable

; UCSRC - USART Control and Status Register C
is_ucpol    equ 0 ; Clock Polarity
is_ucsz0    equ 1 ; Character Size
is_ucsz1    equ 2 ; Character Size
is_usbs     equ 3 ; Stop Bit Select
is_upm0     equ 4 ; Parity Mode Bit 0
is_upm1     equ 5 ; Parity Mode Bit 1
is_umsel    equ 6 ; USART Mode Select
is_ursel    equ 7 ; Register Select


; ***** TWI *************************
; TWCR - TWI Control Register
is_twie     equ 0 ; TWI Interrupt Enable
is_twen     equ 2 ; TWI Enable Bit
is_twwc     equ 3 ; TWI Write Collition Flag
is_twsto    equ 4 ; TWI Stop Condition Bit
is_twsta    equ 5 ; TWI Start Condition Bit
is_twea     equ 6 ; TWI Enable Acknowledge Bit
is_twint    equ 7 ; TWI Interrupt Flag

; TWSR - TWI Status Register
is_twps0    equ 0 ; TWI Prescaler
is_twps1    equ 1 ; TWI Prescaler
is_tws3     equ 3 ; TWI Status
is_tws4     equ 4 ; TWI Status
is_tws5     equ 5 ; TWI Status
is_tws6     equ 6 ; TWI Status
is_tws7     equ 7 ; TWI Status

; TWAR - TWI (Slave) Address register
is_twgce    equ 0 ; TWI General Call Recognition Enable Bit
```

```
is_twa0     equ 1 ; TWI (Slave) Address register Bit 0
is_twa1     equ 2 ; TWI (Slave) Address register Bit 1
is_twa2     equ 3 ; TWI (Slave) Address register Bit 2
is_twa3     equ 4 ; TWI (Slave) Address register Bit 3
is_twa4     equ 5 ; TWI (Slave) Address register Bit 4
is_twa5     equ 6 ; TWI (Slave) Address register Bit 5
is_twa6     equ 7 ; TWI (Slave) Address register Bit 6


; ***** ANALOG_COMPARATOR ************
; SFIOR - Special Function IO Register
is_acme     equ 3 ; Analog Comparator Multiplexer Enable

; ACSR - Analog Comparator Control And Status Register
is_acis0    equ 0 ; Analog Comparator Interrupt Mode Select bit 0
is_acis1    equ 1 ; Analog Comparator Interrupt Mode Select bit 1
is_acic     equ 2 ; Analog Comparator Input Capture Enable
is_acie     equ 3 ; Analog Comparator Interrupt Enable
is_aci      equ 4 ; Analog Comparator Interrupt Flag
is_aco      equ 5 ; Analog Compare Output
is_acbg     equ 6 ; Analog Comparator Bandgap Select
is_acd      equ 7 ; Analog Comparator Disable


; ***** AD_CONVERTER *****************
; ADMUX - The ADC multiplexer Selection Register
is_mux0     equ 0 ; Analog Channel and Gain Selection Bits
is_mux1     equ 1 ; Analog Channel and Gain Selection Bits
is_mux2     equ 2 ; Analog Channel and Gain Selection Bits
is_mux3     equ 3 ; Analog Channel and Gain Selection Bits
is_mux4     equ 4 ; Analog Channel and Gain Selection Bits
is_adlar    equ 5 ; Left Adjust Result
is_refs0    equ 6 ; Reference Selection Bit 0
is_refs1    equ 7 ; Reference Selection Bit 1

; ADCSRA - The ADC Control and Status register
is_adps0    equ 0 ; ADC  Prescaler Select Bits
is_adps1    equ 1 ; ADC  Prescaler Select Bits
is_adps2    equ 2 ; ADC  Prescaler Select Bits
is_adie     equ 3 ; ADC Interrupt Enable
is_adif     equ 4 ; ADC Interrupt Flag
is_adate    equ 5 ; ADC Auto Trigger enable
is_adsc     equ 6 ; ADC Start Conversion
is_aden     equ 7 ; ADC Enable

; SFIOR - Special Function IO Register
is_adts0    equ 5 ; ADC Auto Trigger Source 0
is_adts1    equ 6 ; ADC Auto Trigger Source 1
is_adts2    equ 7 ; ADC Auto Trigger Source 2


; ***** INTERRUPT VECTORS ************************************************
isv_int0    equ 2   ; External Interrupt Request 0
isv_int1    equ 4   ; External Interrupt Request 1
isv_oc2     equ 6   ; Timer/Counter2 Compare Match
isv_ovf2    equ 8   ; Timer/Counter2 Overflow
isv_icp1    equ 10  ; Timer/Counter1 Capture Event
isv_oc1a    equ 12  ; Timer/Counter1 Compare Match A
isv_oc1b    equ 14  ; Timer/Counter1 Compare Match B
isv_ovf1    equ 16  ; Timer/Counter1 Overflow
isv_ovf0    equ 18  ; Timer/Counter0 Overflow
isv_spi     equ 20  ; Serial Transfer Complete
isv_urxc    equ 22  ; USART, Rx Complete
isv_udre    equ 24  ; USART Data Register Empty
isv_utxc    equ 26  ; USART, Tx Complete
isv_adcc    equ 28  ; ADC Conversion Complete
isv_erdy    equ 30  ; EEPROM Ready
isv_aci     equ 32  ; Analog Comparator
isv_twi     equ 34  ; 2-wire Serial Interface
```

```
isv_int2    equ 36  ; External Interrupt Request 2
isv_oc0     equ 38  ; Timer/Counter0 Compare Match
isv_spmr    equ 40  ; Store Program Memory Ready


; ***** Command register (ios_cr) ****************************************
isc_spi     equ 0   ;connect unit SPI
isc_twi     equ 1   ;connect unit TWI
isc_usart   equ 2   ;connect unit USART
isc_xint    equ 3   ;connect unit external interrupt
isc_adc     equ 4   ;connect unit ADC
isc_acmp    equ 5   ;connect unit analog comparator
isc_t0      equ 6   ;connect unit timer 0
isc_ocra1   equ 7   ;connect unit timer 1 with ocra
isc_ocrb1   equ 8   ;connect unit timer 1 with ocrb
isc_icr1    equ 9   ;connect unit timer 1 with icr
isc_t2      equ 10  ;connect unit timer 2
isc_eep     equ 11  ;connect unit EEPROM with eedr
isc_eear    equ 12  ;connect unit EEPROM with shadow eear
isc_diagd   equ 60  ;connect data register to diag data
isc_diaga   equ 61  ;connect data register to diag address
isc_endv    equ 62  ;end vectored mode
isc_reset   equ 63  ;reset
                    ; clear/set a register bit
isc_regbit  macro   ;\1 0=clear, 1=set, \2 0-31 = bitmap #
            lda #$40|(\1<<5)|\2
            sta ios_cr
            endm
isc_xregbit macro   ;\1 0=clear, 1=set, \2 0-31 = bitmap #
            ldx #$40|(\1<<5)|\2
            stx ios_cr
            endm
isc_yregbit macro   ;\1 0=clear, 1=set, \2 0-31 = bitmap #
            ldy #$40|(\1<<5)|\2
            sty ios_cr
            endm
                    ; clear/set a ddr or port bit
isc_portbit macro   ;\1 0=clear, 1=set, \2 0=port, 1=ddr
                    ;\3 0-3 = port a-d, \4 0-7 = bit
            lda #$80|(\2<<6)|(\3<<4)|(\1<<3)|\4
            sta ios_cr
            endm
isc_xportbit macro   ;\1 0=clear, 1=set, \2 0=port, 1=ddr
                     ;\3 0-3 = port a-d, \4 0-7 = bit
            ldx #$80|(\2<<6)|(\3<<4)|(\1<<3)|\4
            stx ios_cr
            endm
isc_yportbit macro   ;\1 0=clear, 1=set, \2 0=port, 1=ddr
                     ;\3 0-3 = port a-d, \4 0-7 = bit
            ldy #$80|(\2<<6)|(\3<<4)|(\1<<3)|\4
            sty ios_cr
            endm
```