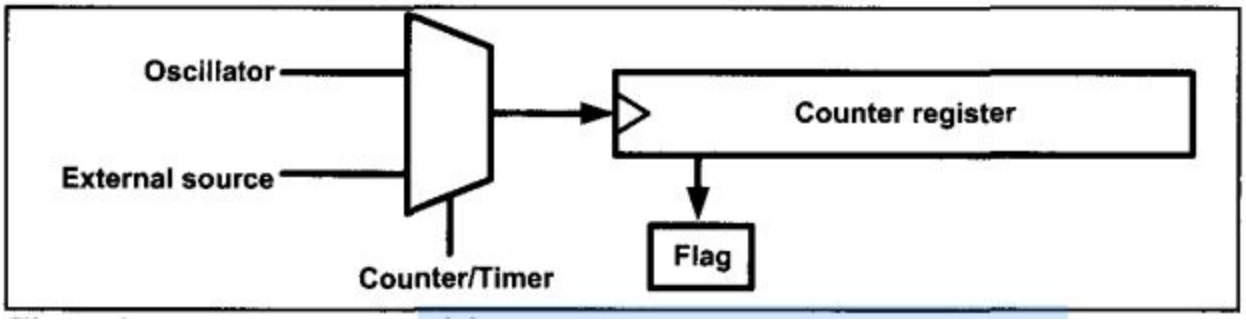
Module 3
Timer, Interrupt, Programming

SYLLABUS

Programming Timers 0, 1, and 2 - Counter Programming - AVR interrupts – Programming Timer Interrupts –Programming External Hardware interrupts– Interrupt priority in the AVR .

Counters and Timers in Microcontrollers

- Many applications need to count an event or generate time delays. So,here are counter registers in microcontrollers for this purpose.
- When we want to count an event, we connect the external event source to the clock pin of the counter register. Then, when an event occurs externally, the content of the counter is incremented; in this way, the content of the counter represents how many times an event has occurred.
- When we want to generate time delays, we connect the oscillator to the clock pin of the counter. So, when the oscillator ticks, the content of the counter is incremented. As a result, the content of the counter register represents how many ticks have occurred from the time we have cleared the counter.



- In the microcontrollers, there is a flag for each of the counters. The flag is set when the counter overflows, and it is cleared by software.
- The second method to generate a time delay is to load the counter register and wait until the counter overflows and the flag is set.
- The AYR has one to six timers depending on the family member.
- They are referred to as Timers 0, 1, 2, 3, 4, and 5. They can be used as timers to generate a time delay or as counters to count events happening outside the microcontroller.

PROGRAMMING TIMERS 0, 1, AND 2

- Every timer needs a clock pulse to tick.
- The clock source can be internal or external.
- If we use the internal clock source, then the frequency of the crystal oscillator is fed into the timer. Therefore, it is used for time delay generation and consequently is called a timer.
- By choosing the external clock option, we feed pulses through one of the AYR's pins. This is called a counter.

Basic registers of timers

TCNTn

- ❖ In AYR, for each of the timers, there is a TCNTn (timer/counter) register. That means in ATmega32 we have TCNTO, TCNTl, andTCNT2.
- ❖ The TCNTn register is a counter. Upon reset, the TCNTn contains zero. It counts up with each pulse.
- ❖ The contents of the timers/counters can be accessed using the TCNTn. You can load a value into the TCNTn register or read its value.

TOYn

- ❖ Each timer has a TOYn (Timer Overflow) flag, as well. When a timer overflows, its TOY n flag will be set.

3. TCCRn.

- ❖ Each timer also has the TCCRn (Timer/counter control register) register for setting modes of operation.
- ❖ For example, you can specify TimerO to work as a timer or a counter by loading proper values into the TCCRO.

4. OCRn

- ❖ Each timer also has an OCRn (Output Compare Register) register.
- ❖ Each timer also has an OCRn (Output Compare Register) register. The content of the OCRn is compared with the content of the TCNTn. When they are equal the OCFn (Output Compare Flag) flag will be set.

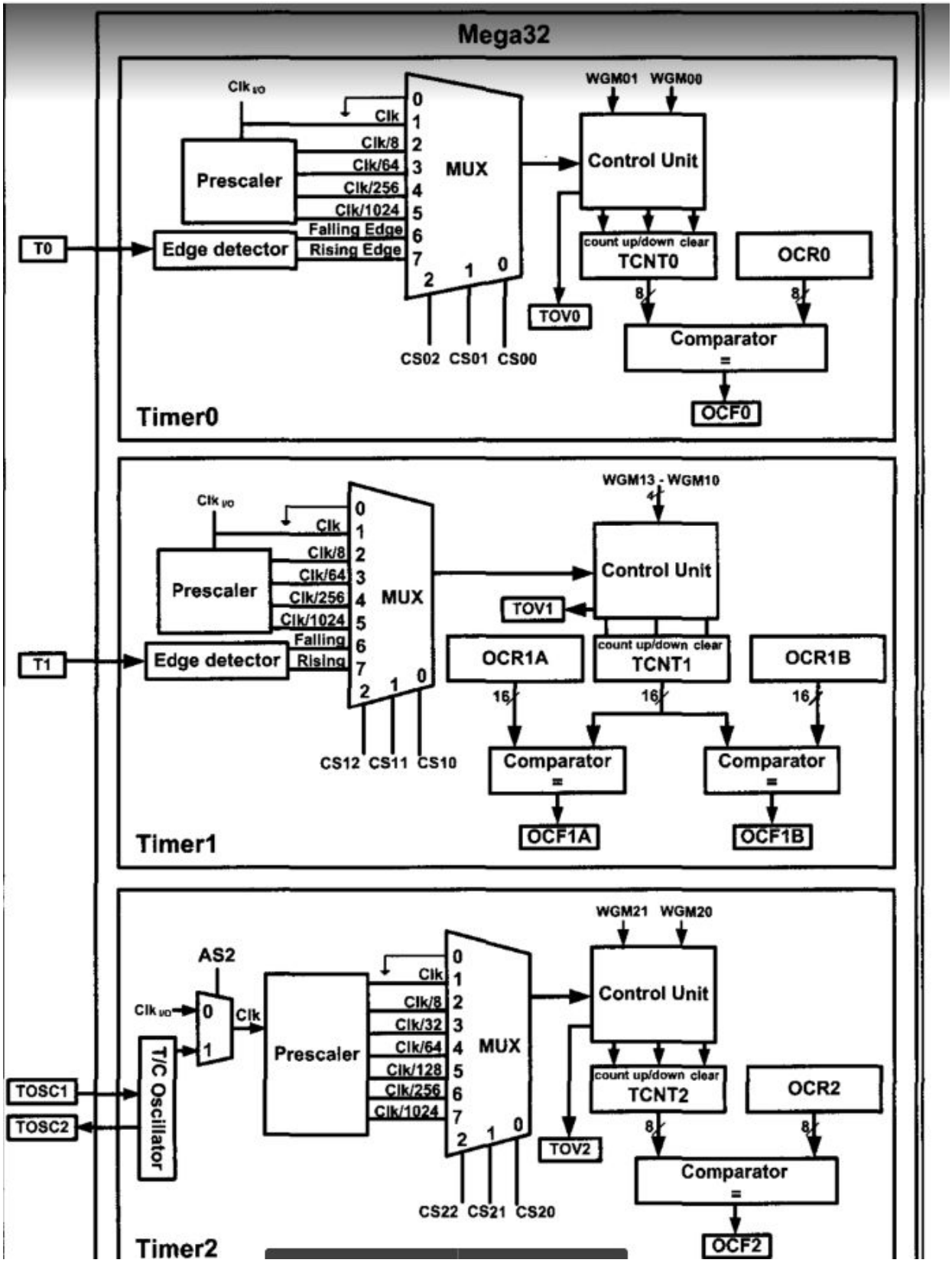
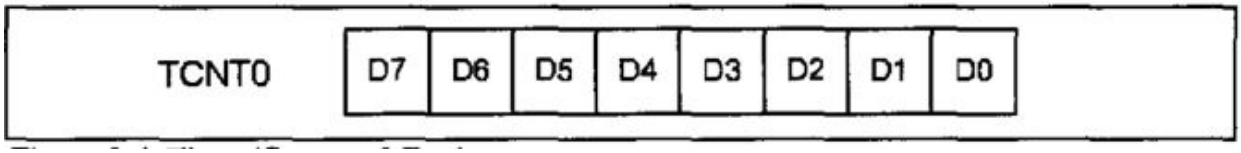


figure: Timers in ATmega32

TimerO programming

TimerO is 8-bit in ATmega32; thus, TCNTO is 8-bit as shown in Figure.

TCNTn



Timer/Counter 0 Register

TCCRO (Timer/Counter Control Register) register

TCCOR is an 8-bit register used for control of TimerO. The bits for TCCRO are shown in Figure.

- **CS02:CSOO (TimerO clock source)**

❏ These bits in the TCCR register are used to choose the clock source.
❏ If CS02-CSOO has values between 001 and IOI, the oscillator is used as a clock source and the timer/counter acts as a timer.
❏ If CS02-CSOO are 110 or 111, the external clock source is used and it acts as a counter.

- **WGM 01:00**

❏ TimerO can work in four different modes.
❏ Normal, phase correct PWM,CTC, and Fast PWM.
❏ The WGMOl and WGMOO bits are used to choose one of them.

Bit	7	6	5	4	3	2	1	0
	FOC0	WGM00	COM01	COM00	WGM01	CS02	CS01	CS00
Read/Write	W	RW	RW	RW	RW	RW	RW	RW
Initial Value	0	0	0	0	0	0	0	0

FOC0 D7 Force compare match: This is a write-only bit, which can be used while generating a wave. Writing 1 to it causes the wave generator to act as if a compare match had occurred.

WGM00, WGM01

D6	D3	Timer0 mode selector bits
0	0	Normal
0	1	CTC (Clear Timer on Compare Match)
1	0	PWM, phase correct
1	1	Fast PWM

COM01:00 D5 D4 Compare Output Mode:
These bits control the waveform generator (see Chapter 15).

CS02:00

D2	D1	D0	Timer0 clock selector
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	clk (No Prescaling)
0	1	0	clk / 8
0	1	1	clk / 64
1	0	0	clk / 256
1	0	1	clk / 1024
1	1	0	External clock source on T0 pin. Clock on falling edge.
1	1	1	External clock source on T0 pin. Clock on rising edge.

Figure: TCCRO (Timer/Counter Control Register) Register

TIFR (Timer/counter Interrupt Flag Register) register

The TIFR register contains the flags of different timers, as shown in Figure
Next, we discuss the TOYO flag, which is related to TimerO.

Bit	7	6	5	4	3	2	1	0
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

TOV0 D0 Timer0 overflow flag bit
0 = Timer0 did not overflow.
1 = Timer0 has overflowed (going from $FF to $00).
OCF0 D1 Timer0 output compare flag bit
0 = compare match did not occur.
1 = compare match occurred.
TOV1 D2 Timer1 overflow flag bit
OCF1B D3 Timer1 output compare B match flag
OCF1A D4 Timer1 output compare A match flag
ICF1 D5 Input Capture flag
TOV2 D6 Timer2 overflow flag
OCF2 D7 Timer2 output compare match flag

Figure: TIFR (Timer/Counter Interrupt Flag Register)

Timer2 programming

Timer2 is an 8-bit timer. Therefore it works the same way as TimerO. But there are two differences between TimerO and Timer2.

1. Timer2 can be used as a real time counter. To do so, we should connect a crystal of 32.768 kHz to the TOSCl and TOSC2 pins of AYR and set the AS2 bit. For more information about this feature, see the AVR datasheet.

2. In TimerO, when CS02-CSOO have values 110 or 111, TimerO counts the external events. But in Timer2, the multiplexer selects between the different scales of the clock. In other words, the same values of the CS bits can have different meanings for TimerO and Timer2.

Bit	7	6	5	4	3	2	1	0
	FOC2	WGM20	COM21	COM20	WGM21	CS22	CS21	CS20
Read/Write	W	RW	RW	RW	RW	RW	RW	RW
Initial Value	0	0	0	0	0	0	0	0

FOC2 D7 Force compare match: a write-only bit, which can be used while generating a wave. Writing 1 to it causes the wave generator to act as if a compare match had occurred.

WGM20, WGM21

D6	D3	Timer2 mode selector bits
0	0	Normal
0	1	CTC (Clear Timer on Compare Match)
1	0	PWM, phase correct
1	1	Fast PWM

COM21:20 D5 D4 Compare Output Mode:
These bits control the waveform generator (see Chapter 15).

CS22:20

D2	D1	D0	Timer2 clock selector
0	0	0	No clock source (Timer/Counter stopped)
0	0	1	clk (No Prescaling)
0	1	0	clk / 8
0	1	1	clk / 32
1	0	0	clk / 64
1	0	1	clk / 128
1	1	0	clk / 256
1	1	1	clk / 1024

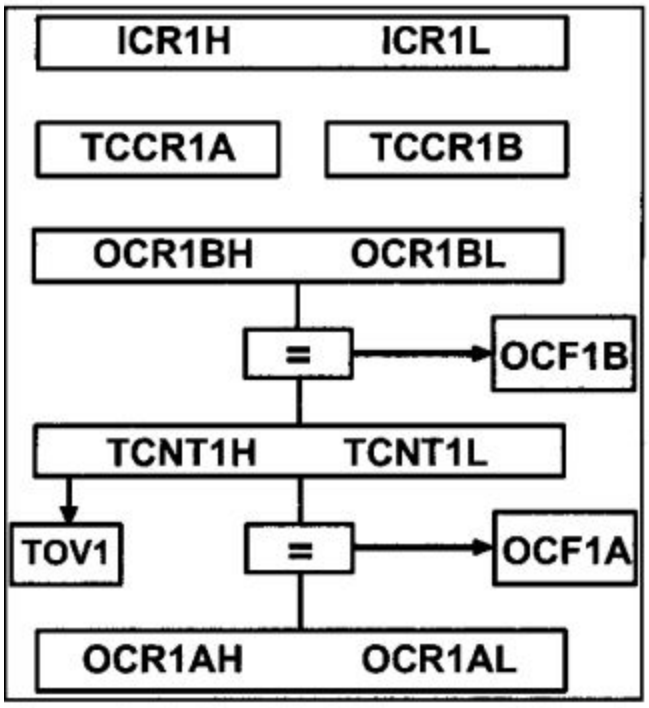
figure:TCCR2 (Timer/Counter Control Register) Register

(Block diagram is same as of timer 0.please refer)

Timer1 programming

- Timer1 is a 16-bit timer and has lots of capabilities.
- Since Timer1 is a 16-bit timer its 16-bit register is split into two bytes.
- These are referred to as **TCNTIL** (Timerl low byte) and **TCNTIH** (Timerl high byte).
- Timer1 also has two control registers named **TCCRIA** (Timer/counter 1 control register) and **TCCRIB**. The TOVl (timer overflow) flag bit goes HIGH.
- when overflow occurs. Timer 1 also has the prescaler options of 1:1, 1:8, 1:64,1:256, and 1:1024.
- There are two **OCR** registers in Timerl: **OCRIA** and **OCRIB**. There
  There are two separate flags for each of the OCR registers.

- Whenever TCNT 1 equals OCR I A, the OCF I A flag will be set on the next timer clock. When TCNT equals OCRlB, the OCFlB flag will be set on the next clock.



Simplified Diagram of Timerl.

- As Timer 1 is a 16-bit timer, the OCR registers are 16-bit registers as well and they are made of two 8-bit registers.

The TIFR register contains the TOVl, OCFlA, and OCFIB flags.

Bit	7	6	5	4	3	2	1	0
	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

TOV0 D0 Timer0 overflow flag bit
0 = Timer0 did not overflow.
1 = Timer0 has overflowed (going from $FF to $00).

OCF0 D1 Timer0 output compare flag bit
0 = compare match did not occur.
1 = compare match occurred.

TOV1 D2 Timer1 overflow flag bit

OCF1B D3 Timer1 output compare B match flag

OCF1A D4 Timer1 output compare A match flag

ICF1 D5 Input Capture flag

TOV2 D6 Timer2 overflow flag

OCF2 D7 Timer2 output compare match flag

TIFR (Timer/Counter Interrupt Flag Register)

TCCRIA (Timer 1 Control) Register

Bit	7	6	5	4	3	2	1	0
	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

COM1A1:COM1A0	D7 D6	Compare Output Mode for Channel A (discussed in Section 9-3)
COM1B1:COM1B0	D5 D4	Compare Output Mode for Channel B (discussed in Section 9-3)
FOC1A	D3	Force Output Compare for Channel A (discussed in Section 9-3)
FOC1B	D2	Force Output Compare for Channel B (discussed in Section 9-3)
WGM11:10	D1 D0	Timer1 mode (discussed in Figure 9-18)

TCCRlB (Timer 1 Control) Register

Bit	7	6	5	4	3	2	1	0	
	ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10	TCCR1B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

ICNC1 D7 Input Capture Noise Canceler
0 = Input Capture is disabled.
1 = Input Capture is enabled.

ICES1 D6 Input Capture Edge Select
0 = Capture on the falling (negative) edge
1 = Capture on the rising (positive) edge

D5 Not used

WGM13:WGM12 D4 D3 Timer1 mode

Mode	WGM13	WGM12	WGM11	WGM10	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	TOP	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	TOP	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	TOP	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	TOP	TOP
15	1	1	1	1	Fast PWM	OCR1A	TOP	TOP

CS12:CS10 D2D1D0 Timer1 clock selector

0 0 0	No clock source (Timer/Counter stopped)
0 0 1	clk (no prescaling)
0 1 0	clk / 8
0 1 1	clk / 64
1 0 0	clk / 256
1 0 1	clk / 1024
1 1 0	External clock source on T1 pin. Clock on falling edge.
1 1 1	External clock source on T1 pin. Clock on rising edge.

COUNTER PROGRAMMING

- The AVR timer can also be used to count, detect, and measure the time of events happening outside the AVR. The use of the timer as an event counter is covered in this section.
- When the timer is used as a timer, the AVR's crystal is used as the source of the frequency.
- When it is used as a counter, however, it is a pulse outside the AVR that increments the TCNTx register.
- Notice that, in counter mode,registers such as TCCR, OCRO, and TCNT are the same as for the timer.

CSOO, C501, and C502 bits in the TCCR0 register

- The CS bits (clock selector) in the TCCR register decide the source of the clock for the timer.
- If CS02:00 is between I and 5, the timer gets pulses from the crystal oscillator.
- In contrast, when CS02:00 is 6 or 7, the timer is used as a counter and gets its pulses from a source outside the AVR chip.
- counter counts up as pulses are fed from pin TO (Timer/Counter 0 External Clock input).
- In the case of TimerO, when CS02:00 is 6 or 7, pin TO provides the clock pulse and the counter counts up after each clock pulse coming from that pin.
- Timer1, when CS12:10 is 6 or 7, the clock pulse coming in from pin Tl.(Timer/Counter 1 External Clock input) makes the TCNTl counter count up.
- When CS 12: 10 is 6, the counter counts up on the negative (falling) edge. When CS12:10 is 7, the counter counts up on the positive (rising) edge.

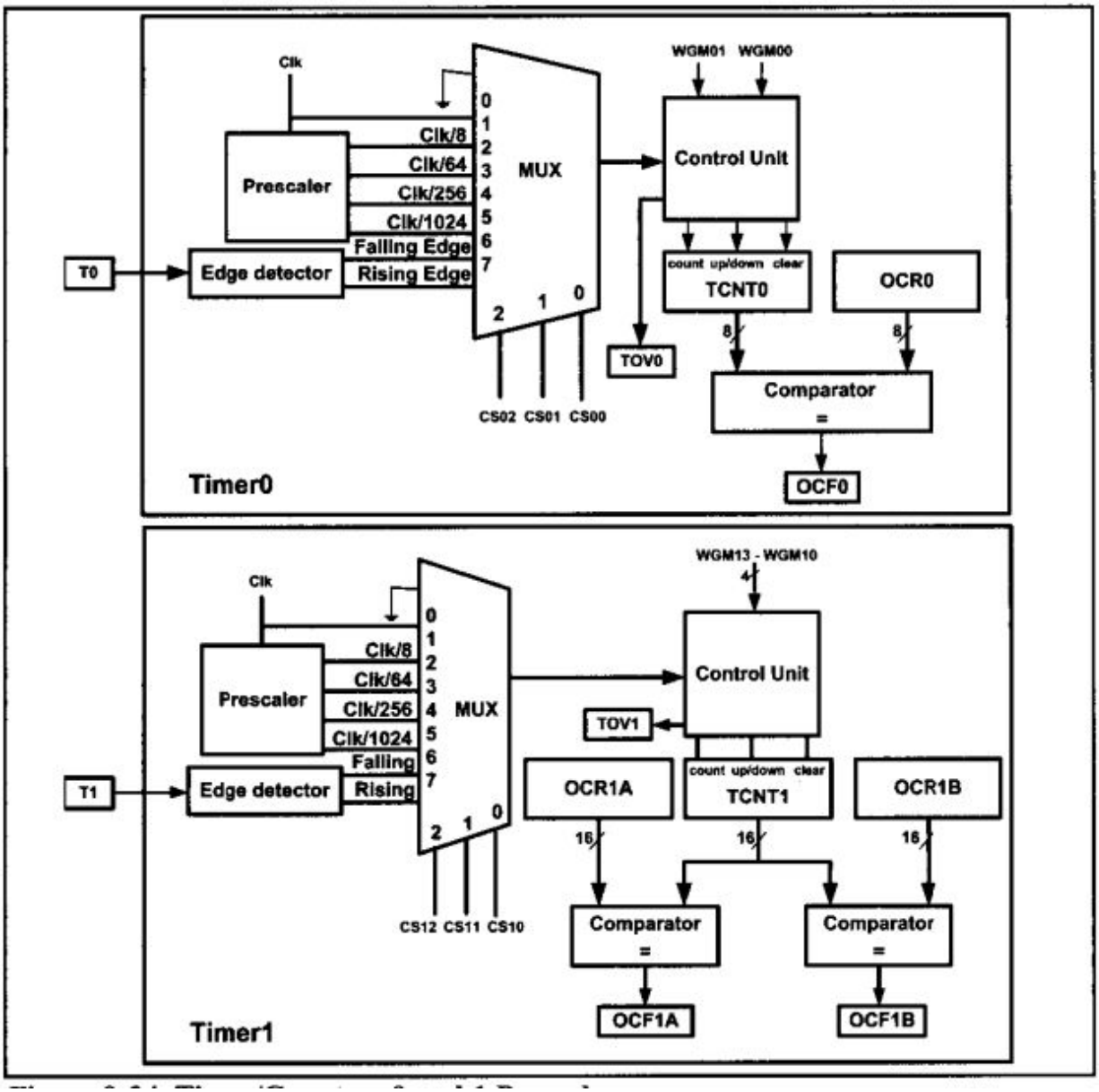


figure:Timer/Counters 0 and 1 Prescalers.

AVR INTERRUPTS

Interrupts vs. polling

- A single microcontroller can serve several devices.
- There are two methods by which devices receive service from the microcontroller: **interrupts or polling**.
- In the ***interrupt method***, whenever any device needs the microcontroller's service, the device notifies it by sending an interrupt signal.
- Upon receiving an interrupt signal, the microcontroller stops whatever it is doing and serves the device.
- The program associated with the interrupt is called the interrupt service routine (ISR) or interrupt handler.
- In ***polling***, the microcontroller continuously monitors the status of a given device; when the status condition is met, it performs the service.
- After that, it moves on to monitor the next device until each one is serviced.
- Although polling can monitor the status of several devices and serve each of them as certain conditions are met, it is not an efficient use of microcontroller.
- The advantage of interrupts is that the microcontroller can serve many devices (not all at the same time, of course); each device can get the attention of the microcontroller based on the priority assigned to it.
- The polling method cannot assign priority because it checks all devices in a round-robin fashion. More importantly, in the interrupt method the microcontroller can also ignore (mask) a device request for service.
- This also is not possible with the polling method.
- The most important reason that the interrupt method is preferable is that the polling method wastes much of the microcontroller's time by polling devices that do not need service.
- So interrupts are used to avoid tying down the microcontroller.

Interrupt service routine

- For every interrupt, there must be an interrupt service routine (ISR), or interrupt handler.
- When an interrupt is invoked, the microcontroller runs the **interrupt service routine**.
- Generally, in most microprocessors, for every interrupt there is a fixed location in memory that holds the address of its ISR.
- The group of memory locations set aside to hold the addresses of ISRs is called the **interrupt vector table.**

Steps in executing an interrupt

Upon activation of an interrupt, the microcontroller goes through the following steps:

I. It finishes the instruction it is currently executing and saves the address of the next instruction (program counter) on the stack.

2. It jumps to a fixed location in memory called the interrupt vector table. The interrupt vector table directs the microcontroller to the address of the interrupt service routine (ISR).

3. The microcontroller starts to execute the interrupt service subroutine until it reaches the last instruction of the subroutine, which is RETI (return from interrupt).

4. Upon executing the RETI instruction, the microcontroller returns to the place where it was interrupted. First, it gets the program counter (PC) address from the stack by popping the top bytes of the stack into the PC. Then it starts to execute from that address.

Table 10-1: Interrupt Vector Table for the ATmega32 AVR

Interrupt	ROM Location (Hex)
Reset	0000
External Interrupt request 0	0002
External Interrupt request 1	0004
External Interrupt request 2	0006
Time/Counter2 Compare Match	0008
Time/Counter2 Overflow	000A
Time/Counter1 Capture Event	000C
Time/Counter1 Compare Match A	000E
Time/Counter1 Compare Match B	0010
Time/Counter1 Overflow	0012
Time/Counter0 Compare Match	0014
Time/Counter0 Overflow	0016
SPI Transfer complete	0018
USART, Receive complete	001A
USART, Data Register Empty	001C
USART, Transmit Complete	001E
ADC Conversion complete	0020
EEPROM ready	0022
Analog Comparator	0024
Two-wire Serial Interface (I2C)	0026
Store Program Memory Ready	0028

Sources of interrupts in the AVR

There are many sources of interrupts in the AYR, depending on which peripheral is incorporated into the chip. The following are some of the most widely used sources of interrupts in the AYR:

I. There are at least two interrupts set aside for each of the timers, one for overflow and another for compare match.

2. Three interrupts are set aside for external hardware interrupts. Pins PD2(PORTD.2), PD3 (PORTD.3), and PB2 (PORTB.2) are for the external hardware interrupts INTO, INTI, and INT2, respectively.

3. Serial communication's USART has three interrupts, one for receive and tw interrupts for transmit.
4. The SPI interrupts.
5. The ADC (analog-to-digital converter).

Enabling and disabling an interrupt

Upon reset, all interrupts are disabled (masked), meaning that none will be responded to by the microcontroller if they are activated. The interrupts must be enabled (unmasked) by software in order for the microcontroller to respond to them. The 07 bit of the SREG (Status Register) register is responsible for enabling and disabling the interrupts globally. Figure shows the SREG register. The I bit makes the job of disabling all the interrupts easy. With a single instruction "CLI" (Clear Interrupt), we can make I= 0 during the operation of a critical task.

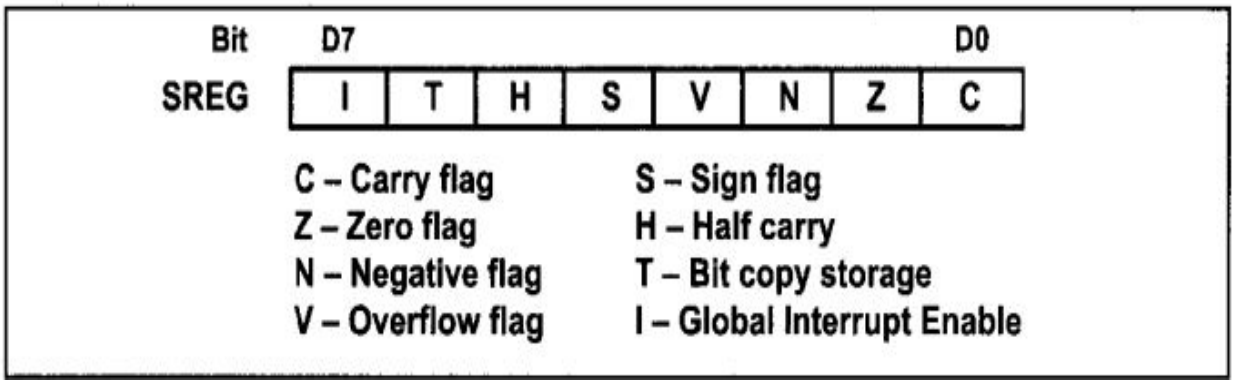


figure:Bits of Status Register (SREG)

Steps in enabling an interrupt

To enable any one of the interrupts, we take the following steps:

1. Bit D7 (I) of the SREG register must be set to HIGH to allow the interrupts to happen. This is done with the "SEI" (Set Interrupt) instruction.

2. If I = 1, each interrupt is enabled by setting to HIGH the interrupt enable (IE) flag bit for that interrupt. There are some I/O registers holding the interrupt

enable bits. Figure above below that the TIMSK register has interrupt enable bits for TimerO, Timer I, and Timer2.

It must be noted that if I = 0, no interrupt will be responded to, even if the corresponding interrupt enable bit is high.

D7							D0
OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	OCIE0	TOIE0

TOIE0 Timer0 overflow interrupt enable
= 0 Disables Timer0 overflow interrupt
= 1 Enables Timer0 overflow interrupt

OCIE0 Timer0 output compare match interrupt enable
= 0 Disables Timer0 compare match interrupt
= 1 Enables Timer0 compare match interrupt

TOIE1 Timer1 overflow interrupt enable
= 0 Disables Timer1 overflow interrupt
= 1 Enables Timer1 overflow interrupt

OCIE1B Timer1 output compare B match interrupt enable
= 0 Disables Timer1 compare B match interrupt
= 1 Enables Timer1 compare B match interrupt

OCIE1A Timer1 output compare A match interrupt enable
= 0 Disables Timer1 compare A match interrupt
= 1 Enables Timer1 compare A match interrupt

TICIE1 Timer1 input capture interrupt enable
= 0 Disables Timer1 input capture interrupt
= 1 Enables Timer1 input capture interrupt

TOIE2 Timer2 overflow interrupt enable
= 0 Disables Timer2 overflow interrupt
= 1 Enables Timer2 overflow interrupt

OCIE2 Timer2 output compare match interrupt enable
= 0 Disables Timer2 compare match interrupt
= 1 Enables Timer2 compare match interrupt

These bits, along with the I bit, must be set high for an interrupt to be responded to. Upon activation of the interrupt, the I bit is cleared by the AVR itself to make sure another interrupt cannot interrupt the microcontroller while it is servicing the current one. At the end of the ISR, the RETI instruction will make I = 1 to allow another interrupt to come in.

TIMSK (Timer Interrupt Mask) Register

PROGRAMMING TIMER INTERRUPTS

Rollover timer flag and interrupt

- The timer overflow flag is raised when the timer rolls over.
- In polling TOVO, we have to wait until TOVO is raised.
- The problem with this method is that the microcontroller is tied down waiting for TOVO to be raised, and cannot do anything else.
- Using interrupts avoids tying down the controller.
- If the timer interrupt in the interrupt register is enabled,TOVO is raised whenever the timer rolls over and the microcontroller jumps to the interrupt vector table to service the ISR.
- In this way, the microcontroller can do other things until it is notified that the timer has rolled over.
- To use an interrupt in place of polling, first we must enable the interrupt because all the interrupts are masked upon reset.
- The TOIEx bit enables the interrupt for a given timer. TOIEx bits are held by the TIMSK register.

Timer Interrupt Flag Bits and Associated Registers

Interrupt	Overflow Flag Bit	Register	Enable Bit	Register
Timer0	TOV0	TIFR	TOIE0	TIMSK
Timer1	TOV1	TIFR	TOIE1	TIMSK
Timer2	TOV2	TIFR	TOIE2	TIMSK

PROGRAMMING EXTERNAL HARDWARE INTERRUPTS

- The number of external hardware interrupt interrupts varies in different AYRs.
- The ATmega32 has three external hardware interrupts: pins PD2 (PORTD.2), PD3 (PORTD.3), and PB2 (PORTB.2), designated as INTO, INTI, and INT2, respectively.
- Upon activation of these pins, the AYR is interrupted in whatever it is doing and jumps to the vector table to perform the interrupt service routine.

External interrupts INTO, INT1, and INT2

- There are three external hardware interrupts in the ATmega32: INTO, INTI, and INT2. They are located on pins PD2, PD3, and PB2, respectively.
- As the interrupt vector table locations $2, $4, and $6 are set aside for INTO,INTI, and INT2, respectively.
- The hardware interrupts must be enabled before they can take effect. This is done using the INTx bit located in the GICR register.

D7							D0
INT1	INT0	INT2	-	-	-	IVSEL	IVCE

INT0 External Interrupt Request 0 Enable
= 0 Disables external interrupt 0
= 1 Enables external interrupt 0

INT1 External Interrupt Request 1 Enable
= 0 Disables external interrupt 1
= 1 Enables external interrupt 1

INT2 External Interrupt Request 2 Enable
= 0 Disables external interrupt 2
= 1 Enables external interrupt 2

These bits, along with the I bit, must be set high for an interrupt to be responded to.

GICR (General Interrupt Control Register) Register

INTERRUPT PRIORITY IN THE AVR

We must deal with what happens when two interrupts are activated at the same time. Which of these two interrupts is responded to first.

Interrupt priority

- If two interrupts are activated at the same time, the interrupt with the higher priority is served first.
- The priority of each interrupt is related to the address of that interrupt in the interrupt vector.
- The interrupt that has a lower address, has a higher priority.
- For example, the address of external interrupt 0 is 2, while the address of external interrupt 2 is 6; thus, external interrupt 0 has a higher priority, and if both of these interrupts are activated at the same time, external interrupt 0 is served first.

Interrupt inside an interrupt

- What happens if the AYR is executing an ISR belonging to an interrupt and another interrupt is activated?
- When the AYR begins to execute an ISR, it disables the I bit of the SREG register, causing all the interrupts to be disabled, and no other interrupt occurs while serving the interrupt.
- When the RETI instruction is executed, the AYR enables the I bit, causing the other interrupts to be served.
- If you want another interrupt (with any priority) to be served while the current interrupt is being served you can set the I bit using the SEI instruction.
- For example, in a low-level-triggered external interrupt, enabling the I bit while the pin is still active will cause the ISR to be reentered infinitely, causing the stack to overflow with unpredictable consequences.