MODULE – III Timer, Interrupt, Programming

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

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.
- There for, 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 AVR's pins. This is called a *counter*

Basic registers of timers

1. TCNTn (timer/Counter)
2. TOVn(Timer Overflow)
3. TCCRn (timer/counter Control register)
4. OCRn(Output Compare Register)
5. TIFR(Timer/counter Interrupt Flag Register) register

TCNTn (timer/Counter)

- InATmega32we have TCNTO, TCNTI, and TCNT2.
- The TCNTn register isa counter.
- Upon reset, the TCNTn contains zero.
- It counts up with each pulse. You can load a value into the TCNTn register or read its value.

TOVn(Timer Overflow)

- It is a flag register.
- When a time rover flows, its TOVn flag will be set.

TCCRn (timer/counter Control register)

- This register is used for setting modes of operation.

OCRn(Output Compare 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.
- The timer registers are located in the I/O register memory. Therefore, you can read or write from timer registers using IN and OUT instructions, like the other I/O registers.

TIFR(Timer/counter Interrupt Flag Register) register

- The TIFR register contains the flags of different timers.

Timer0 programming

- Timer0 is 8-bit in ATmega32.

TCCR0(Timer/Counter Control Register) register

- TCCR0 is an 8-bit register used for control of Timer0..

- The bits for TCCR0 are shown below:

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

CS02:CS00 (Timer0 clock source)

- These bits in the TCCR0 register are used to choose the clock source.
- If CS02:CS00=000, then the counter is stopped.
- If CS02-CS00 have values between001 and 101, the oscillator is used as clock source and the timer/counter acts as a timer. In this case, the timers are often used for time delay generation.
- If CS02-CS00 are 110 or 111, the external clock source is used and it acts as a counter.

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.

WGM01 :WGM00

- Timer0 can work in four different modes:

1. Normal mode
2. Fast PWM
3. CTC
4. Phase correct PWM

- The WGM01 and WGM00 bits are used to choose one of them.

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

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.

COM01:00	D5 D4	Compare Output Mode: These bits control the waveform generator

TIFR(Timer/counter Interrupt Flag Register) register

- The bits for TIFR are shown below:

7	6	5	4	3	2	1	0
OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

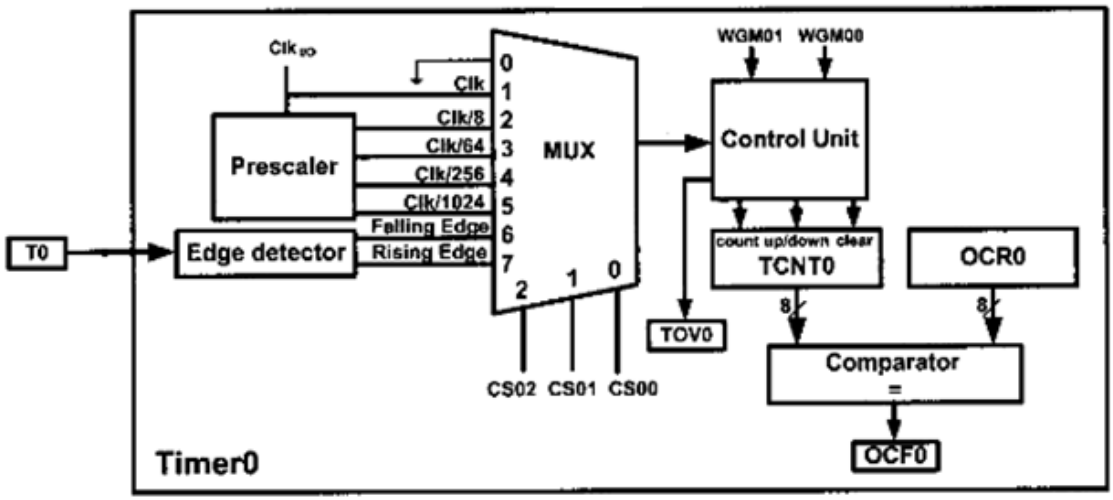
TOV0	D0	Timer0 overflow flag bit 0 = Timer0 did not overflow. 1 = Timer0 has overflowed (going from $FF to $00).
OCF0	D1	Timer0 output compare flag bit 0 = compare match did not occur. 1 = compare match occurred.
TOV1	D2	Timer1 overflow flag bit
OCF1B	D3	Timer1 output compare B match flag
OCF1A	D4	Timer1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

TOV0(Timer0Overflow)

- The flag is set when the counter overflows, going from $FF to $00.
- When the timer rolls over from $FF to 00, the TOV0 flag is set to 1 and it remains set until the software clears it.
- For example, the following program clears TOV0:

```
LDI R20,0x01
OUT  TIFR,R20     ;TIFR  0b00000001
```

Block diagram Timer0 in ATmega32



Modes of Operation in Timer0

Normal mode of operation

- In this mode, the content of the timer/counter increments with each clock.

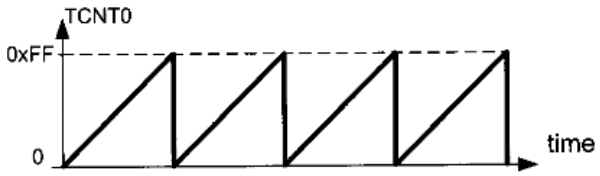
- It counts up until it reaches its max of OxFF.
- When it rolls over from 0xFF to 0x00, it sets high a flag bit called TOV0(Timer Over flow).

Steps to program Timer 0 in Normal mode

- To generate at time delay using Timer0 in Normal mode, the following steps are taken:

1. Load the TCNTO register with the initial count value.
2. Load the value into the TCCR0.
3. Keep monitoring the timer over flow flag(TOV0) to see if it is raised.
4. Stop the timer by disconnecting the clock source.
5. Clear the TOVO flag for the next round.
6. Go back to Step 1 to load TCNTO again.

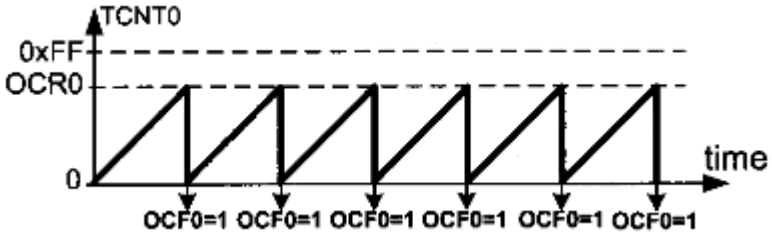
Timer/counter0 normal mode :



ClearTimer0 on compare match (CTC) mode programming

- The OCR0 register is used with CTC mode.
- In the CTC mode, the timer is incremented with a clock.
- But it counts up until the content of the TCNT0 register becomes equal to the content of OCR0 (compare match occurs); then, the timer will be cleared and the OCF0 flag will be set when the next clock occurs.
- The OCF0 flag is located in theTIFR register.

Timer/counter0 CTC mode :

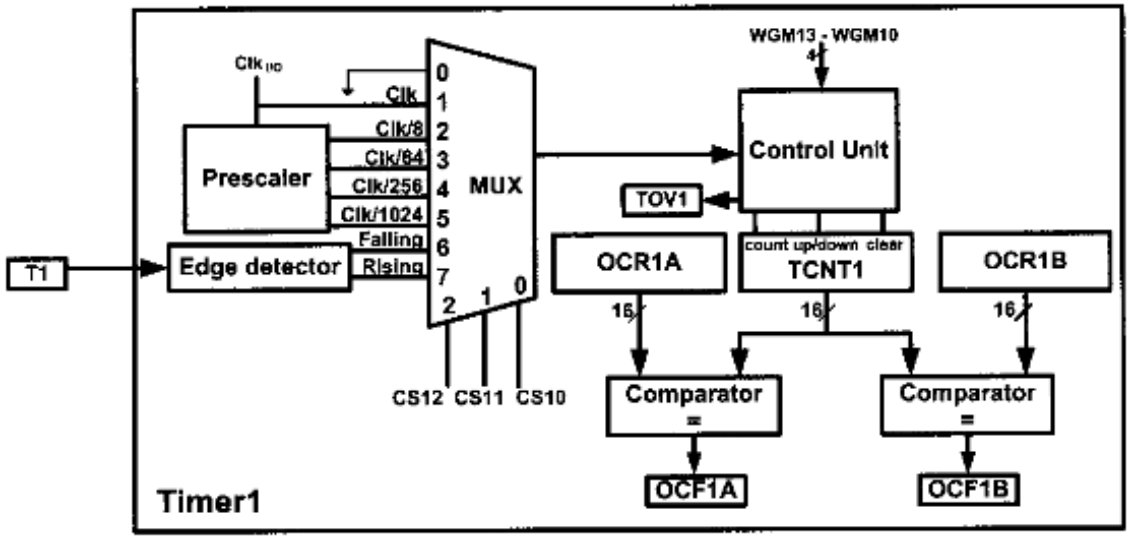


Timer1 programming

- Timer1 is a16-bit timer.
- Its 16-bit register is split in to two bytes.
- The TCNT1 is referred as TCNT1L (Timer1 low byte) and TCNT1H (Timer1 high byte).
- Timer1 also has two control registers named TCCR1A(Timer/counter1 control register and TCCR1B.

- TheTOV1 (timer over flow)flag bit goes HIGH when over flow occurs.
- There are two OCR registers in Timer1:OCR1A andOCR1B.
- There are two separate flags for each of the OCR registers.
- Whenever TCNT1 equals OCR1A, the OCF1A flag will be set on the next clock.
- When TCNT1 equals OCR1B, the OCF1B flag will be set on the next clock.
- As Timer1 is a 16-bit timer, the OCR registers are16-bit registers as well and they are made of two 8-bit registers.
- For example, OCR1A is made of OCR1AH (OCR1A high byte) and OCR1AL (OCR1A low byte).

Block diagram Timer1 in ATmega32



- TIFR register is shown below:

7	6	5	4	3	2	1	0
OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	OCF0	TOV0
R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
0	0	0	0	0	0	0	0

- TCCR1A register is shown below:

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

COM1B1:COM1B0 D5 D4 Compare Output Mode for Channel B

FOC1A D3 Force Output Compare for Channel A

FOC1B D2 Force Output Compare for Channel B

WGM11:10 D1 D0 Timer1 mode

- TCCR1B register is shown below:

7	6	5	4	3	2	1	0
ICNC1	ICES1	-	WGM13	WGM12	CS12	CS11	CS10

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

WGM13:10

- The WGM13,WGM12,WGM11,and WGM10 bits define the mode of Timer1.

Timer1 operation modes

Normal mode (WGM13:10=0000)

- In this mode, the timer counts up until it reaches $FFFF(which is the maximum value) and then it rolls over from $FFFF to 0000.
- When the timer rolls over from $FFFF to 0000,theTOV1 flag will be set.

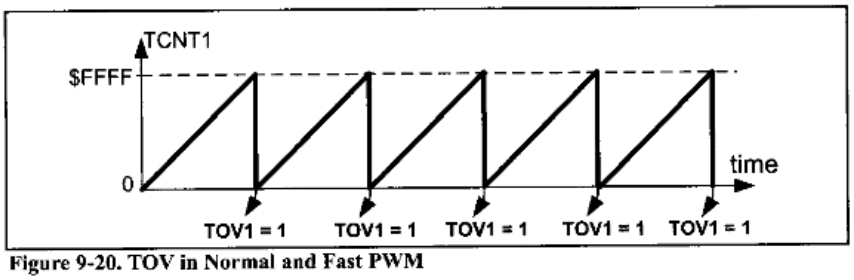


Figure 9-20. TOV in Normal and Fast PWM

CTC mode (WGM13:10=0100)

- The timer counts up until the content of theTCNT1 register be comes equal to the content of OCR1A (compare match occurs); then ,the timer will be cleared when the next clock occurs.
- The OCF1A flag will be set as a result of the compare match as well.

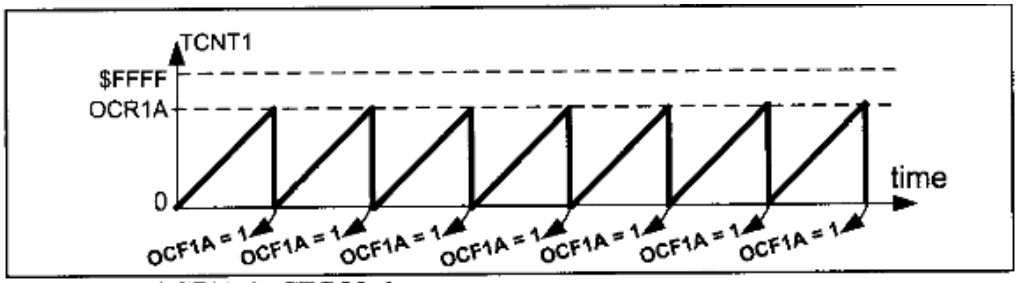


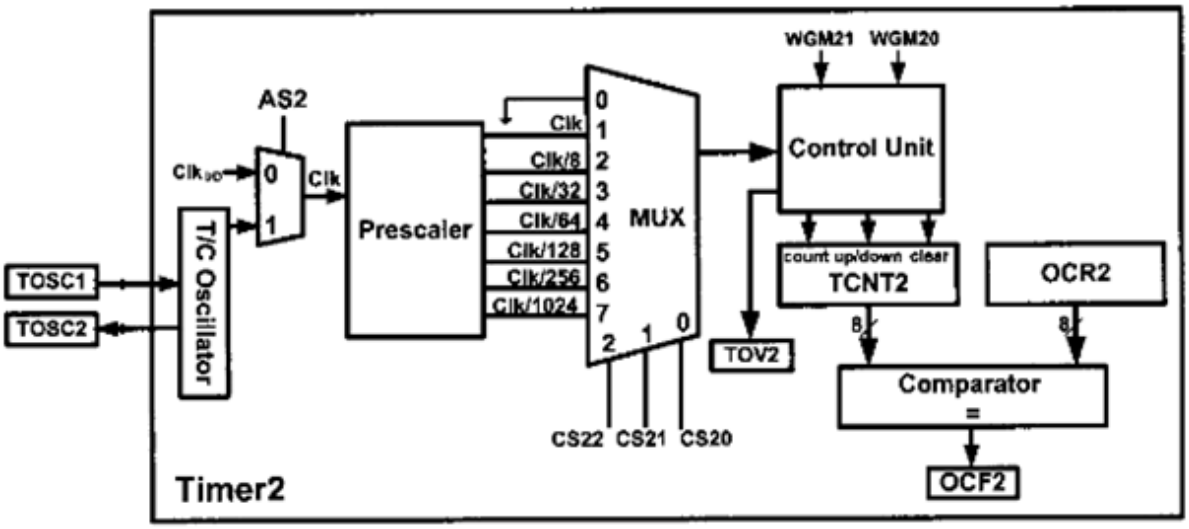
Figure 9-21. OCF1A in CTC Mode

Timer2 Programming

- Timer2 is an 8-bit timer.
- Therefore it works the same way as Timer0. But there are two differences between Timer0 and Timer2:

1. Timer2 can be used as a real time counter.
2. In Timer0, when CS02–CS00 have values 110 or 111,Timer0 counts the external events. But inTimer2 ,the multiplexer selects between the different scales of the clock.

Block diagram Timer2 in ATmega32



Mode of operation timer2

- Same as timer0

AVR INTERRUPTS

Interrupts

- There are two methods by which devices receive service from the microcontroller: interrupts or polling.
- 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*.
- 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.
- 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 ofISRs is called the *interrupt vector table***.**

Interrupts vs. polling

Interrupts	**Polling**
Whenever any device needs the microcontroller's service, the device notifies it by sending an interrupt signal.	The microcontroller continuously monitors the status of a given device; when the status condition is met, it performs the service.
Each device is serviced based on the priority as signed to it.	It checks all devices in a round-robin fashion.
It can ignore a device request.	It cannot ignore a device request.
It avoids the tying down the microcontroller.	It wastes much of the time by polling devices that do not need service.

Steps in executing an interrupt

1. 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.

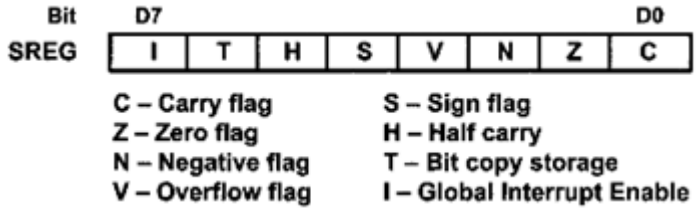
Sources of interrupts in the AVR

- Depending on which peripheral is incorporated in to the chip.
- The following are some of the most widely used sources of interrupts in the AYR:

1. There are at least two interrupts set aside for each of the timers, one for over flow 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 INT0 ,INT1, and INT2, respectively.
3. Serial communication's USART has three interrupts, one for receive and two interrupts for transmit.
4. The SPI interrupts.
5. The ADC (analog-to-digita lconverter).

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(I flag) of the SREG(Status Register)register is responsible for enabling and disabling the interrupts globally.
- Bits of status register is shown below:



Steps in enabling an interrupt

1. Bit D7(I) of the SREG register must be set too HIGH to allow the interrupts to happen. This is done with the "SEI" (Set Interrupt) instruction.
2. If I= I, each interrupt is enabled by setting to HIGH the interrupt enable (IE) flag bit for that interrupt. If I=0,no interrupt will be responded.

PROGRAMMING TIMER INTERRUPTS

- There are 2 timer interrupts.
    1. Overflow 2.Compare match

Rollovertimer flag and interrupt

- The timer overflow flag is raised when the timer rolls over.
- When the timer overflows, the TOV0 flag will set.
- If the timer interrupt in the interrupt register is enabled, TOY0 is raised whenever the timer rolls over and the microcontroller jumps to the interrupt vector table to service the ISR.

Compare match timer flag and interrupt

- We load the OCR register with the proper value and initialize the timer to the CTC mode.
- When the content of TCNT matches with OCR ,the OCF flag is set, which causes the compare match interrupt to occur.

PROGRAMMING EXTERNAL HARDWARE INTERRUPTS

- There are three external hardware interrupts in the ATmega32:

1. INTO(occur at pin PD2 ie, PORTD.2)
2. INTI (occur at pin PD3 ie, PORTD.3)
3. INT2(occur at pin PB2 ie, PORTB.2)

- **When these pins are activated, then the AVR is interrupted in whatever it is doing.**
- **Then it jumps to the vector table to perform the associated interrupt service routine.**
- The interrupt vector table locations $2,$4, and $6 are set aside for INT0, INT1, and INT2, respectively.

INT0

- ✓ The INT0 is a low-level-triggered interrupt.
- ✓ When a low signal is applied to pinPD2(PORTD.2),the controller will be interrupted and jump to location $0002 in the vector table to service the ISR.

INT1

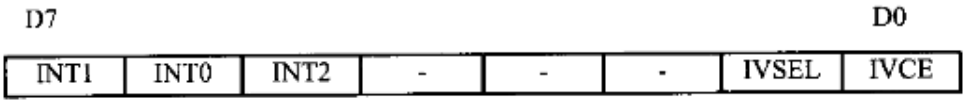
- ✓ INT1 is either low-level-triggered or edge triggered interrupt.

INT2

- ✓ INT2i s edge triggered interrupt.

- The hardware interrupts must been abled before they can take effect. This is done using the INTx bit located in the GICR register.

- The GICR register is shown below:



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

INTERRUPT PRIORITY IN THE AVR

- 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.
- When the AVR 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 AVR enables the I bit, causing the other interrupts to be served.

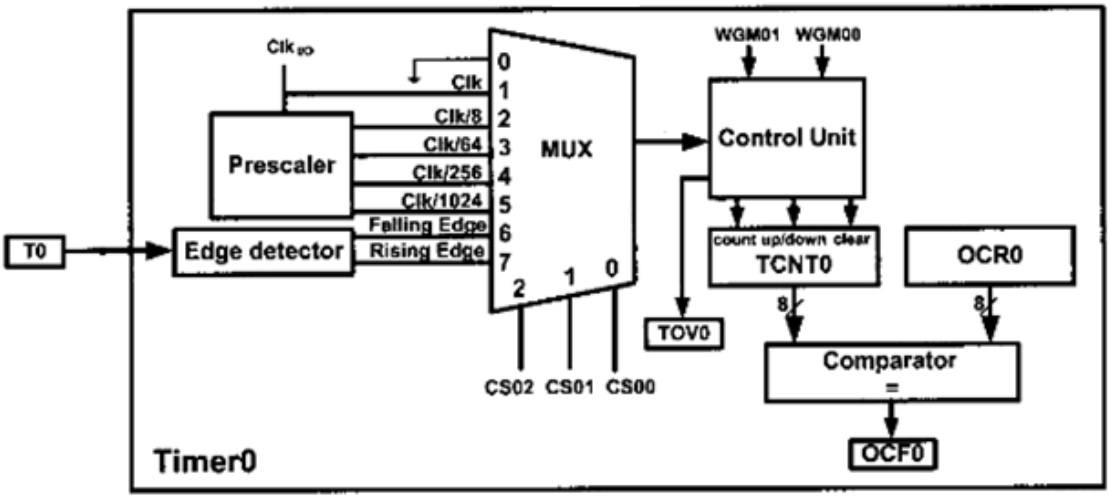
Context saving in task switching

- In multitasking systems, such as multitasking real-time operating systems (RTOS),the CPU serves one task(job or process)at a time and then moves to the next one.
- In these cases, we can save the contents of registers on the stack before execution of each task ,and reload the registers at the end of the task.
- This saving of the CPU contents before switching to a new task is called *context saving* (or *context switching)*.

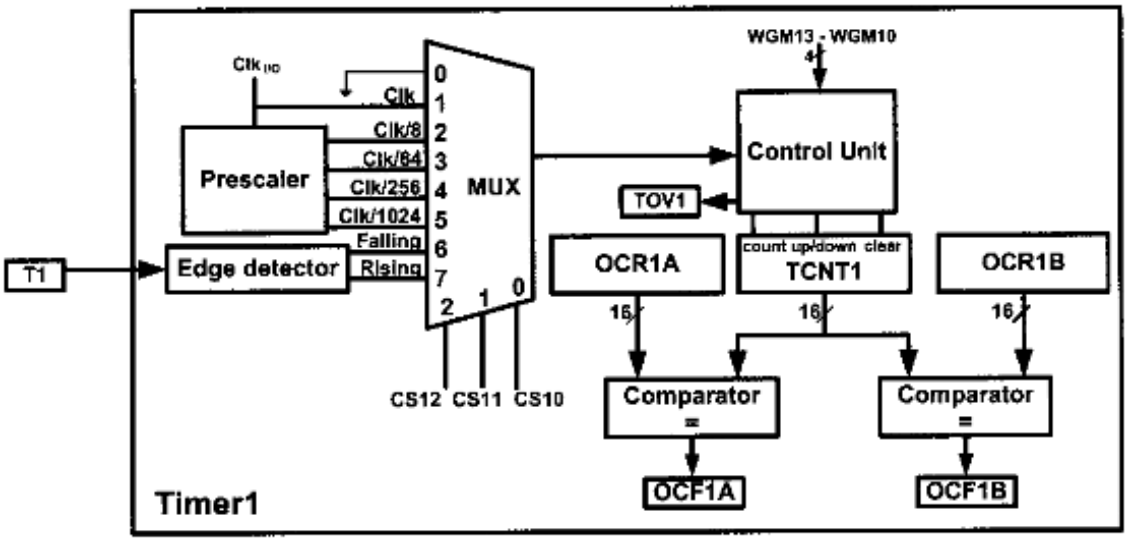
Interrupt latency

- The time from the moment an interrupt is activated to the moment the CPU starts to execute the task is called the *interrupt latency.*
- This latency is 4machine cycle times.

Block diagram Timer0 in ATmega32



Block diagram Timer1 in ATmega32



Block diagram Timer2 in ATmega32