**A**

**Mini Project Report**

**On**

**IMPLEMENTATION OF WATCH DOG TIMER IN ARDUINO UNO**

**Submitted to**

**JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY, ANANTAPUR**

***In partial fulfillment of the thirdyear 2nd semester***

# BACHELOR OF TECHNOLOGY

# IN

# ELECTRONICS & COMMUNICATION ENGINEERING

Submitted By

**V.LAVANYA 18P11A04A6**

**Under the Esteemed Guidance of**

**Dr. K. SOUNDARARAJAN M. Tech, Ph.D**

**Professor**



**CHADALAWADA RAMANAMMA ENGINEERING COLLEGE**

**(AUTONOMOUS)**

Renigunta road, Tirupati-517506, Andhra Pradesh, India

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING**

**(2018-2022)**

**CHADALAWADA RAMANAMMA ENGINEERING COLLEGE**

**(AUTONOMOUS)**

# Department of Electronics & Communication Engineering



# *Certificate*

This is to certify that the project work entitled **“**IMPLEMENTATION OF WATCH DOG TIMER IN ARDUINO UNO**”** is a bonafide work done by **V. LAVANYA (18P11A04A6)U.GOUTAMI(18P11A0418)Y.VENKATASURESH (18P11A04B4 )**

**R. DEVENDRA KUMAR (18P11A0490)** in the Department of **“ELECTRONICS & COMMUNICATION ENGINEERING”**, and submitted to ***Chadalawada Ramanamma Engineering College (Autonomous), Tirupati*** is a project work carried out by them under my guidance during the academic year 2020-2021.

### *Guide* Head

**Dr. K. SOUNDARARAJAN** **M. Tech, Dr. P. KRISHNA MURTHY M. Tech, Ph.D. Ph.D.**

Professor Associate Professor & Head

Department of ECE Department of ECE

**INTERNAL EXAMINER EXTERNAL EXAMINER**

**ACKNOWLEDGEMENT**

A Grateful thanks to Dr. CHADALAWADA KRISHNA MURTHY, chairman C.V.S Teja group of Institutions for providing education in this esteemed institution.

We extend our thanks to our principal Dr. P. SANJEEVA RAYUDU, **M. Tech** **Ph.D.** Chadalawada Ramanamma Engineering College, Tirupati, for providing all the facilities and supporting me to complete this work.

Our sincere thanks to **Dr. P. KRISHNA MURTHY**, **M. Tech, Ph.D.** Associate Professor and Head of the Department of Electronics & Communication Engineering, Chadalawada Ramanamma Engineering College, Tirupati, for providing infrastructural facilities to complete our mini project work.

We are Greatly indebted to my guide **Dr. K. SOUNDARARAJAN** **M.Tech Ph.D.** Professor, in the Department of Electronics &communication Engineering, Chadalawada Ramanamma Engineering College , Tirupati ,who encourages me in all spheres of activities, constant encouragement and keen interest enriched to complete this work.

We extend our thanks to all the Teaching staff of the Department of E.C.E for their support, encouragement and inspired me. I also thank the Non-Teaching staff of the department of E.C.E for being helpful in many ways in successful completion of my work.

The chain of our gratitude would be definitely incomplete, if we forget to thank all my friends of Chadalawada Ramanamma Engineering College, Tirupati for their constant support and encouragement. Finally, we thank all those who helped us directly or indirectly in successful completion of our project work.

**V.LAVANYA 18P11A04A6**

**Y. VENKATA SURESH 18P11A04B4**

**U.GOUTAMI 19P11A0418**

**R. DEVENDRA KUMAR 18P11A0490**

**DECLARATION**

We are here by declare that the project work on“IMPLEMENTATION OF WATCH DOG TIMER IN ARDUINO UNO” is the genuine work carried by us, in B. Tech **(Electronics &Communication Engineering**) Degree course in **Chadalawada Ramanamma Engineering college.**

We declare that this written submission represents our ideas in our own words and where others' ideas or words have been included, we have adequately cited and referenced the original sources. We also declare that, we have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea / data / fact / source in our project report submission.

**V.LAVANYA 18P11A04A6**

**Y. VENKATA SURESH 18P11A04B4**

**U. GOWTHAMI 19P15A0418**

**R. DEVENDRA KUMAR 18P11A0490**

## 

## 

## ABSTRACT

This work presents a brief explanation about the watch dog timer implementation in microcontroller using Arduino uno. Usually watch dog is used in microcontrollers, embedded systems etc. we are using in microcontroller. A microcontroller (MCU) is a compact processor for controlling electronic devices. Integrated into a wide variety of electronic devices, MCUs come pre-loaded with program software whose commands are used to control electronic devices.  
This makes safeguarding normal MCU operation essential. Should the MCU program, for some reason, run out of control or stop running altogether, the electronic device may behave erratically, which in the worst case could cause damage or any deviation from main program.A timing device such that it is set for a preset time interval and an event must occur during that interval else the device will generate the timeout signal on failure to get that event in the watched time interval. Timeout may result in processor start a service routine or start from beginning.

**TABLE OF CONTENTS**

**TITLE** **Page No**

**LIST OF FIGURES**

**LIST OF TABLES**

**CHAPTER 1: INTRODUCTION TO WATCH DOG TIMERS**

1.1 Introduction 1

1.2 Architecture and operation 2

1.3 Single Stage Watch Dog 3

1.4 Multi Stage Watch Dog 4

1.5 Time Intervals 5

1.6 Corrective Actions

### CHAPTER 2: LITERATURE REVIEW

2.1 Introduction 6

2.2 Literature review 7

2.2.1 Software watch dog function 8

2.2.2 Watch dog timer 9

2.2.3 Feed The Watch dog 9

2.2.4 Enable Watch Dog 10

2.2.5 Watch Dog reset 11

**CHAPTER 3: EXISTING METHOD**

3.1 Existing method

### CHAPTER 4: PROPOSED METHOD

4.1 Microcontroller 12

4.2 Arduino Uno Microcontroller 13

4.3 Communication 13 4.4 Arduino Uno Board 14

4.5 Introduction to Arduino Ide 22

4.6 Arduino Datatypes 23

**CHAPTER 5: RESULTS AND OBSERVATION**

5.1 Purpose 40

5.2 Observation 42

**CHAPTER 6: CONCLUSION AND FUTURE SCOPE**

6.1 Conclusion 44

6.2 Future Scope 45

**REFERENCES**

**LIST OF FIGURES**

#### Title Page no.

1.Single stage watch dog 02

1. Multistage watch dog 03
2. Two stage watch dog 04
3. Arduino UNO Board 07
4. Pin Diagram 16
5. USB Board 25

7.Observation 33

**LIST OF TABLES**

**Table.No Page No**

1.Arduino UNO Specifications 13

**CHAPTER 1**

## INTRODUCTION TO WATCH DOG TIMERS

**1.1 INTRODUCTION**

A watchdog timer is an electronic timer that is used to detect and recover from computer malfunctions. During normal operation, the computer regularly restarts the watchdog timer to prevent it from elapsing, or "timing out". If, due to a hardware fault or program error, the computer fails to restart the watchdog, the timer will elapse and generate a timeout signal. The timeout signal is used to initiate corrective action or actions. The corrective actions typically include placing the computer system in a safe state and restoring normal system operation. Watchdog timers are commonly found in embedded systems and other computer-controlled equipment where humans cannot easily access the equipment or would be unable to react to faults in a timely manner. In such systems, the computer cannot depend on a human to reboot it if it hangs; it must be self-reliant. For example, remote embedded systems such as space probes are not physically accessible to human operators; these could become permanently disabled if they were unable to autonomously recover from faults. A watchdog timer is usually employed in cases like these. Watchdog timers may also be used when running untrusted code in a sandbox, to limit the CPU time available to the code and thus prevent some types of denial-of-service attacks.

The main program typically has a loop that it constantly goes through performing various functions. The watchdog timer is loaded with an initial value greater than the worst case time delay through the main program loop. Each time it goes through the main loop the code resets the watchdog timer (sometimes called “kicking” or “feeding” the dog). If a fault occurs and the main program does not get back to reset the timer before it counts down to zero, an interrupt is generated to reset the processor. Used in this way, the watchdog timer can detect a fault on an unattended arduino program and attempt corrective action with a reset .Typically after reset, a register can also be read to determine if the watchdog timer generated the reset or if it was a normal reset. On the arduino this register is called the Watchdog Reset Flag Register (WDRF).

**1.2 ARCHITECTURE AND OPERATION**

**WATCH DOG TIMER RESTART**

The act of restarting a watchdog timer is commonly referred to as "kicking the dog" or other similar terms; this is typically done by writing to a watchdog control port. Alternatively, in microcontrollers that have an integrated watchdog timer, the watchdog is sometimes kicked by executing a special machine language instruction. An example of this is the CLRWDT (clear watchdog timer) instruction found in the instruction set of some PIC microcontrollers. In computers that are running operating systems, watchdog resets are usually invoked through a device driver. For example, in the Linux operating system, a user space program will kick the watchdog by interacting with the watchdog device driver, typically by writing a zero character to /dev/watchdog. The device driver, which serves to abstract the watchdog hardware from user space programs, is also used to configure the time-out period and start and stop the timer.

**1.3 SINGLE STAGE WATCH DOG**

Watch dog timers come in many configurations, and many allow their configurations to be altered. Microcontrollers often include an integrated, on-chip watchdog. In other computers the watchdog may reside in a nearby chip that connects directly to the CPU, or it may be located on an external expansion card in the computer's chassis. The watchdog and CPU may share a common clock signal, as shown in the block diagram below, or they may have independent clock signals sigFig:1Singlestagewatchdog

**1.4 MULTI STAGE WATCHDOG**

Two or more timers are sometimes cascaded to form a multistage watchdog timer, where each timer is referred to as a timer stage, or simply a stage. For example, the block diagram below shows a threestage watchdog. In a multistage watchdog, only the first stage is kicked by the processor. Upon first stage timeout, a corrective action is initiated and the next stage in the cascade is started. As each subsequent stage times out, it triggers a corrective action and starts the next stage. Upon final stage timeout, a corrective action is initiated, but no other stage is started because the end of the cascade has been reached. Typically, single-stage watchdog timers are used to simply restart the computer, whereas multistage watchdog timers will sequentially trigger a series of corrective actions, with the final stage triggering a computer restart.

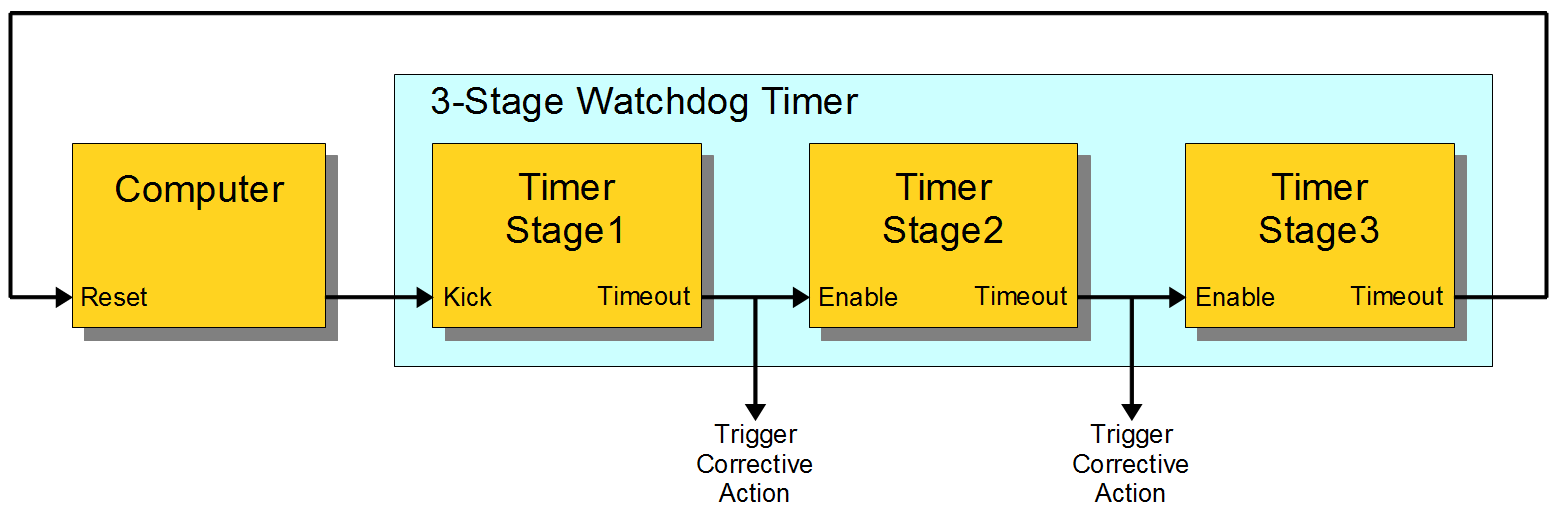


Fig 2: Multi stage Watch dog

**1.5 TIME INTERVALS**

Watchdog timers may have either fixed or programmable time intervals. Some watchdog timers allow the time interval to be programmed by selecting from among a few selectable, discrete values. In others, the interval can be programmed to arbitrary values. Typically, watchdog time intervals range from ten milliseconds to a minute or more. In a multistage watchdog, each timer it may have it’s unique time interval.

**1.6 CORRECTIVE ACTIONS**

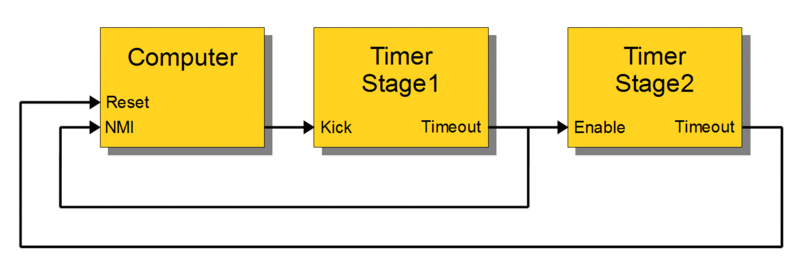
A watchdog timer may initiate any of several types of corrective action, including processor reset, non maskable interrupt, maskable interrupt, power cycling, fail-safe state activation, or combinations of these. Depending on its architecture, the type of corrective action or actions that a watchdog can trigger may be fixed or programmable. Some computers require a pulsed signal to invoke a processor reset. In such cases, the watchdog typically triggers a processor reset by activating an internal or external pulse generator, which in turn creates the required reset pulses. In embedded systems and control systems, watchdog timers are often used to activate fail-safe circuitry. When activated, the fail-safe circuitry forces all control outputs to safe states (e.g., turns off motors, heaters, and high-voltages) to prevent injuries and equipment damage while the fault persists. In a two stage watchdog, the first timer is often used to activate fail-safe outputs and start the second timer stage; the second stage will reset the computer if the fault cannot be corrected before the timer elapses. Watchdog timers are sometimes used to trigger the recording of system state information—which may be useful during fault recovery—or debug information (which may be useful for determining the cause of the fault) onto a persistent medium. In such cases, a second timer—which is started when the first timer elapses—is typically used to reset the computer later, after allowing sufficient time for data recording to complete. This allows time for the information to be saved, but ensures that the computer will be reset even if the recording process fails. They have its own,uniquetime.

Fig 3: Two stage watch dog

For example, the above diagram shows a likely configuration for a two-stage watchdog timer. During normal operation the computer regularly kicks Stage1 to prevent a timeout. If the computer fails to kick Stage1 (e.g., due to a hardware fault or programming error), Stage1 will eventually timeout. This event will start the Stage2 timer and, simultaneously, notify the computer (by means of a non-maskable interrupt) that a reset is imminent. Until Stage2 times out, the computer may attempt to record state information, debug information, or both. The computer will be reset upon Stage2 timeout.

**CHAPTER 2**

**LITERATURE REVIEW**

**2.1 INTRODUCTION**

A watchdog timer (WDT) is a timer that monitors microcontroller (MCU) programs to see if they are out of control or have stopped operating. It acts as a “watchdog” watching over MCU operation.

A microcontroller (MCU) is a compact processor for controlling electronic devices. Integrated into a wide variety of electronic devices, microcontrollers, embedded systems. The watch dog timer makes safeguarding normal MCU operation essential. Should the MCU program, for some reason, run out of control or stop running altogether, the electronic device may behave erratically, which in the worst case could cause damage or an accident.

To proactively prevent such incidents, it falls to the role of the watchdog timer toconstantly watch over the MCU to ensure it is operating normally.

Watch dog timers are commonly found in embedded systems, microcontrollers and other computer equipment where humans cannot access the equipment or would be unable to react to faults in a timely manner. In such systems, the computer cannot depend on a human to reboot it if it hangs; it must be self reliant. A watch dog timer is usually employed in cases like these. It may be also used when running untrusted.

During normal operation, the computer regularly restarts the watchdog timer to prevent it from elapsing, or "timing out". If, due to a hardware fault or program error, the computer fails to restart the watchdog, the timer will elapse and generate a timeout signal. The timeout signal is used to initiate corrective actions. The corrective actions typically include placing the computer and associated hardware in a safe state and invoking a computer reboot.

Every watchdog timer, however simple or sophisticated, must initiate two corrective actions. First, it must set the computer's control outputs to safe levels so that potentially dangerous devices such as motors and heaters will not pose threats to people or equipment. This is a high priority action that must occur as soon as a fault is detected. After setting the outputs to safe levels, the next order of business is to restore normal system operation. This can be as simple as restarting the computer, as if a human operator has pressed the computer's reset pushbutton, or it may involve a sequence of actions that ultimately ends with a computer restart.

A watchdog is a timer that, when not reset before expiring, triggers the reset of the system [1] that is monitoring. In our case, the system will be the ESP8266 microcontroller.

So, the main program needs to periodically reset the watchdog timer, to prevent the reset of the CPU and keep working normally.

The watchdog should be configured with a time greater that the worst case scenario delay in the program [1], so it only triggers in error / unpredicted problems that may make the main program to be locked and not recover on its own [2]. In those locking cases, the watchdog is not reset and, when it expires, resets the system.

This concept is of extreme importance, specially in microcontrollers, which may be affected by environmental conditions such as electrical noise, which can cause hardware malfunction that locks the execution. Additionally, it is  useful for problem in the code that may put the execution in an undesired infinite loop. The ESP8266 has two watch dogs one implemented in software and other implemented in hardware.

2.2LITERATURE REVIEW

2.2.1THE SOFTWARE WATCH DOG FUNCTIONS

In order to access the watchdog functions on the ESP8266, we have the **Esp Class**, which can be analysed in more detail [here](https://github.com/esp8266/Arduino/blob/4897e0006b5b0123a2fa31f67b14a3fff65ce561/cores/esp8266/Esp.cpp#L82). We can access the functionality of this class by using an extern variable called ESP, which is declared [here](https://github.com/esp8266/Arduino/blob/4897e0006b5b0123a2fa31f67b14a3fff65ce561/cores/esp8266/Esp.h) in the libraries.

In order to disable the software watchdog, we just need to call the **wdt Disable** method on the ESP object, as indicated bellow.

|  |  |
| --- | --- |
| 1 | ESP.wdtDisable(); |

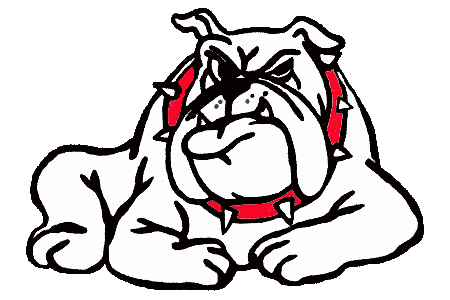
Although this disables the software watchdog, the hardware watchdog will still remain active, causing a reset after some time. As indicated in the  of the **wdtDisable** method, if we stop the software watchdog by more that 6 seconds, the hardware watchdog will trigger. This is approximately what I got when testing it.

We can re-enable the software watchdog by calling the **wdtEnable** method, as indicated bellow.

**2.2.2Watch Dog Timer**

*"*A watchdog timer (***WDT***) is a hardware timer that automatically generates a system reset if the main program neglects to periodically service it.  It is often used to automatically reset an embedded device that hangs because of a software or hardware fault.*" (*os.mbed.com/cookbook/Watchdog-Timer*).*

**2.2.3Feed the Dog !**

 If you have a dog in your home. You need to feed that dog on a regular interval. if you can't feed one day, it will bite you! Like this watchdog timer works.

We have a main part in program which runs over and over(loop). We are enabling watchdog timer is loaded with an initial value greater than the total delay in main program. Each time main program reset this timer. If any case the main program does not get back to reset the  timer before it counts down to zero, an interrupt is generated to reset the processor. Likewise watchdog timer protect micro-controller from hang case.

In Arduino UNO uses ATMEGA328P micro-controller.

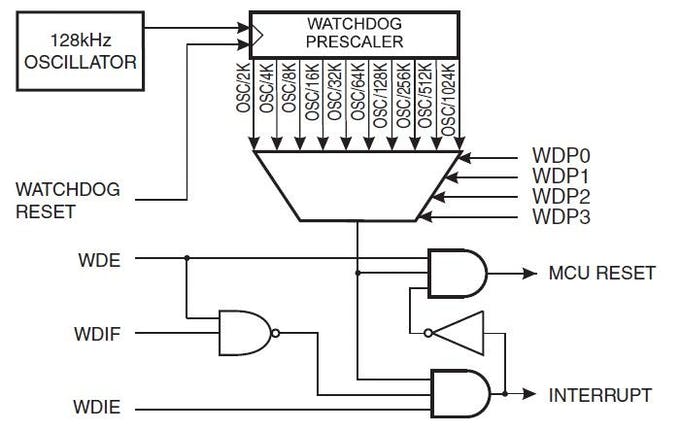


Figure4: ATMEGA328P Watch dog timer

Watchdog timer [library](https://www.nongnu.org/avr-libc/user-manual/group__avr__watchdog.html)

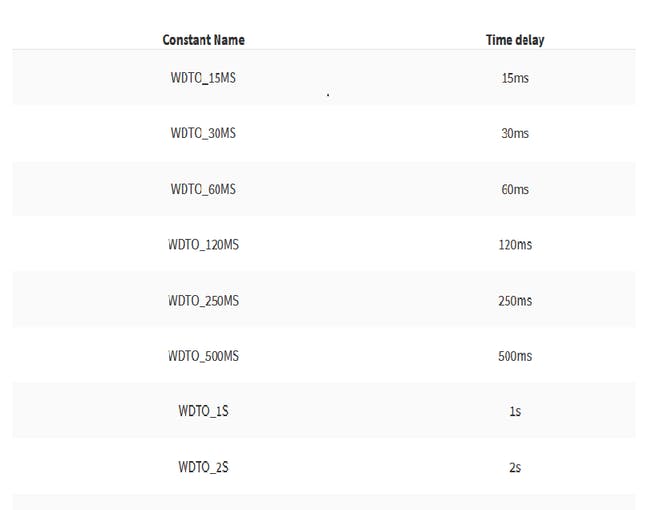
#include <avr/wdt.h>

Library is needed to use watchdog timer in Arduino

### 2.2.4Enable Watchdog timer:

wdt\_enable(WDT Reset Timer);

To enable watchdog timer, WDT RESET TIMER varies from 15ms - 8s



Eg: wdt\_enable(WDT0\_8S); --Enabled watchdog timer for 8Seconds

### 2.3.5Reset watchdog timer

wdt\_reset();

### This function is used for resetting the watchdog timer. Reset function uses inside **loop()**. If your program uses a larger **delay()** which greater than threshold delay of watchdog timer, add reset function before that delay too. Else, It will reset Micro-controller before completing the task. Disabling Watchdog timer

wdt\_disable();

**CHAPTER 3**

**3.1EXISTING METHOD**

The electronic devices and other hard ware devices like embedded systems ,microprocessors and microcontrollers are usually employed for different kinds of purposes . During their normal operation these electronic devices , processors and controllers may face the problem of struck or hanging . In the same way the computer systems may struck in an infinity loop and become unable to perform their normal operation.

In that case to make the processors ,controllers and computer devices work properly we go for using reset button and switching of the devices et. It is not possible all the time to go for using reset button. There may be some situations where humans are unable to use reset system or device.

If we do not reset the device when it enters into any infinite loop or hanging due to some other reasons then the device may become un useful permanently. To avoid these kind of damages to the systems and devices extra focusing is being employed to the high quality devices. But this kind of security also cannot help them to come out of the malfunctions and struck. This is the draw back of using reset and restart buttons . To avoid this kind of severe damages for the devices we use watch dog timers, which can prevent the systems from malfunctions and hanging without human intervention.

**CHAPTER 4**

**PROPOSED METHOD**

**4.1 MICROCONTROLLER**

**Introduction:**

Microcontroller as the name suggest, a small controller. They are like single chip computer that are often embedded into other systems to function as processing/controlling unit. For example , the control you are using probably has microcontrollers inside that do decoding andother controlling functions. They are also used in automobiles, washing machines, microwave ovens, toys….etc , where automation is needed.

**4.2Arduino Uno Microcontroller:**

The Arduino Uno is a microcontroller board based on the ATmega328 (datasheet). It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz crystal oscillator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega8U2 programmed as a USB-to-serial converter. "Uno" means "One" in Italian and is named to mark the upcoming release of Arduino 1.0. The Uno and version 1.0 will be the reference versions of Arduino, moving forward. The Uno is the latest in a series of USB Arduino boards, and the reference model for the Arduino platform; for a comparison with previous versions, see the index of Arduino boards.

The Arduino Uno can be powered via the USB connection or with an external power supply. The powersource is selected automatically.External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adaptercan be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from abattery can be inserted in the Gnd and Vin pin headers of the POWER connector.The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5Vpin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts

3.2The power pins are as follows:·

* **VIN.** The input voltage to the Arduino board when it's using an external power source (as opposed to5 volts from the USB connection or other regulated power source). You can supply voltage throughthis pin, or, if supplying voltage via the power jack, access it through this pin.·
* **5V.** The regulated power supply used to power the microcontroller and other components on the board. This can come either from VIN via an on-board regulator, or be supplied by USB or another regulated 5V supply.
* **3.3V.**A 3.3 volt supply generated by the on-board regulator. Maximum current draw is 50 mA.
* **GND.** Ground pins.

**Memory:**

The Atmega328 has 32 KB of flash memory for storing code (of which 0,5 KB is used for the bootloader); Ithas also 2 KB of SRAM and 1 KB of EEPROM (which can be read and written with the EEPROM library).

**Input and Output:**

Each of the 14 digital pins on the Uno can be used as an input or output, using pinMode(), digitalWrite(), and digitalRead() functions. They operate at 5 volts. Each pin can provide or receive a maximum of 40 mA andhas an internal pull-up resistor (disconnected by default) of 20-50 kOhms. In addition, some pins havespecialized functions:

* **Serial: 0 (RX) and 1 (TX).** Used to receive (RX) and transmit (TX) TTL serial data. These pins are connected to the corresponding pins of the ATmega8U2 USB-to-TTL Serial chip .
* **External Interrupts: 2 and 3.** These pins can be configured to trigger an interrupt on a low value, arising or falling edge, or a change in value. See the attach Interrupt() function for details.
* **PWM: 3, 5, 6, 9, 10, and 11.** Provide 8-bit PWM output with the analogWrite() function.
* **SPI: 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK).** These pins support SPI communication, which although provided by the underlying hardware, is not currently included in the Arduino language.
* **LED: 13.** There is a built-in LED connected to digital pin 13. When the pin is HIGH value, the LED ison, when the pin is LOW, it's off.

The Uno has 6 analog inputs, each of which provide 10 bits of resolution (i.e. 1024 different values). Bydefault they measure from ground to 5 volts, though is it possible to change the upper end of their rangeusing the AREF pin and the analogReference() function. Additionally, some pins have specialized

functionality:

* **I2C: 4 (SDA) and 5 (SCL).** Support I2C (TWI) communication using the Wire library.

There are a couple of other pins on the board:

* **AREF.** Reference voltage for the analog inputs. Used with analogReference().
* **Reset.** Bring this line LOW to reset the microcontroller. Typically used to add a reset button toshields which block the one on the board.

**4.3 Communication:**

The Arduino Uno has a number of facilities for communicating with a computer, another Arduino, or other microcontrollers. The ATmega328 provides UART TTL (5V) serial communication, which is available on digital pins 0 (RX) and 1 (TX). An ATmega8U2 on the board channels this serial communication over USB and appears as a virtual com port to software on the computer. The '8U2 firmware uses the standard USBCOM drivers, and no external driver is needed. However, on Windows, an \*.inf file is required. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the Arduino board. The RX and TX LEDs on the board will flash when data is being transmitted via the USB-toserial chip and USB connection to the computer (but not for serial communication on pins 0 and 1). A SoftwareSerial library allows for serial communication on any of the Uno's digital pins. The ATmega328 also support I2C (TWI) and SPI communication. The Arduino software includes a Wire library to simplify use of the I2C bus

**4.4 ARDUINO UNO BOARD:**

The Arduino Uno is a microcontroller board based on the ATmega328. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz ceramic resonator, a USB connection, a power jack, an ICSP header, and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

****

**Figure 3.1:** Arduino uno board

The Uno differs from all preceding boards in that it does not use the FTDI USB-to-serial driver chip. Instead, it features the Atmega16U2 (Atmega8U2 up to version R2) programmed as a USB-to-serial converters.

**3.5 Technical Specifications:**



**Table 1:** Arduino uno specifications

The Arduino Uno can be powered via the USB connection or with an external power supply. The power source is selected automatically. External (non-USB) power can come either from an AC-to-DC adapter (wall-wart) or battery. The adapter can be connected by plugging a 2.1mm center-positive plug into the board's power jack. Leads from a battery can be inserted in the Gnd and Vin pin headers of the POWER connector. The board can operate on an external supply of 6 to 20 volts. If supplied with less than 7V, however, the 5V pin may supply less than five volts and the board may be unstable. If using more than 12V, the voltage regulator may overheat and damage the board. The recommended range is 7 to 12 volts.

**1.USB Interface:**

Arduino board can be powered by using the USB cable from your computer. All you need to do is connect the USB cable to the USB connection

**2.External power supply:**

Arduino boards can be powered directly from the AC mains power supply by connecting it to the power supply (Barrel Jack)

**3.Voltage Regulator:** The function of the voltage regulator is to control the voltage given to the Arduino board and stabilize the DC voltages used by the processor and other elements.

**4.Crystal Oscillator:**

The crystal oscillator helps Arduino in dealing with time issues. How does Arduino calculate time? The answer is, by using the crystal oscillator. The number printed on top of the Arduino crystal is 16.000H9H. It tells us that the frequency is 16,000,000 Hertz or 16 MHz.

**Arduino Reset:**

It can reset your Arduino board, i.e., start your program from the beginning. It can reset the UNO board in two ways. First, by using the reset button (17) on the board. Second, you can connect an external reset button to the Arduino pin labelled RESET (5).

**Pins (3.3, 5, GND, Vin):**

* 3.3V (6): Supply 3.3 output volt
* 5V (7): Supply 5 output volt
* Most of the components used with Arduino board works fine with 3.3 volt

and 5 volt.

* GND (8)(Ground): There are several GND pins on the Arduino, any of which can be used to ground your circuit.
* Vin (9): This pin also can be used to power the Arduino board from an

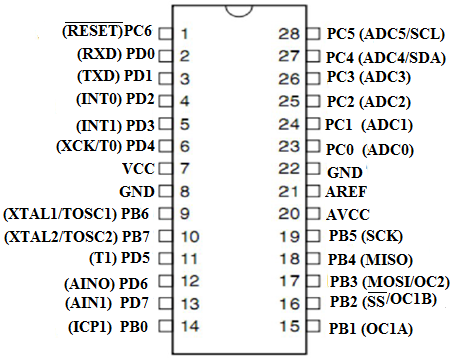
external power source, like AC mains power supply.

**10.Analog pins:**

The Arduino UNO board has five analog input pins A0 through A5. These pins can read the signal from an analog sensor like the humidity sensor or temperature sensor and convert it into a digital value that can be read by the microprocessor.

Each Arduino board has its own microcontroller (11). You can assume it as the brain of your board. The main IC (integrated circuit) on the Ardsuino is slightly different from board to board. The microcontrollers are usually of the ATMEL Company. You must know what IC your board has before loading up a new program from the Arduino IDE. This information is available on the top of the IC. For more details about the IC construction and functions, you can refer to the data sheet.

The Atmega8U2 programmed as a USB-to-serial converter. "Uno" means "One" in Italian and is named to mark the upcoming release of Arduino 1.0. The Uno and version 1.0 will be the reference versions of Arduino, moving forward. The Uno is the latest in a series of USB Arduino boards, and the reference model for the Arduino platform; for a comparison with previous versions, see the index of Arduino boards

****

**Figure 4.1:** Pin diagram

**Pin Description:**

**VCC:** Digital supply voltage.

**GND:** Ground.

**Port B (PB[7:0]) XTAL1/XTAL2/TOSC1/TOSC2:**

Port B is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port B output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port B pins that are externally pulled low will source current if the pull-up resistors are activated. The Port B pins are tri-stated when a reset condition becomes active, even if the clock is not running.

Depending on the clock selection fuse settings, PB6 can be used as input to the inverting Oscillator amplifier and input to the internal clock operating circuit.

Depending on the clock selection fuse settings, PB7 can be used as output from the inverting Oscillator amplifier.

If the Internal Calibrated RC Oscillator is used as chip clock source, PB[7:6] is used as TOSC[2:1] input for the Asynchronous Timer/Counter2 if the AS2 bit in ASSR is set.

**Port C (PC[5:0]):**

Port C is a 7-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The PC[5:0] output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs,

Port C pins that are externally pulled low will source current if the pull-up resistors are activated. The Port C pins are tri-stated when a reset condition becomes active, even if the clock is not running.

**PC6/RESET:**

If the RSTDISBL Fuse is programmed, PC6 is used as an I/O pin. Note that the electrical characteristics of PC6 differ from those of the other pins of Port C.

If the RSTDISBL Fuse is unprogrammed, PC6 is used as a Reset input. A low level on this pin for longer than the minimum pulse length will generate a Reset, even if the clock is not running. Shorter pulses are not guaranteed to generate a Reset.

**Port D (PD[7:0]):**

Port D is an 8-bit bi-directional I/O port with internal pull-up resistors (selected for each bit). The Port D output buffers have symmetrical drive characteristics with both high sink and source capability. As inputs, Port D pins that are externally pulled low will source current if the pull-up resistors are activated. The Port D pins are tri-stated when a reset condition becomes active, even if the clock is not running.

**AVCC:** AVCC is the supply voltage pin for the A/D Converter, PC[3:0], and PE[3:2]. It should be externally connected to VCC, even if the ADC is not used. If the ADC is used, it should be connected to VCC through a low-pass filter. Note that PC[6:4] use digital supply voltage, VCC.

**AREF:** AREF is the analog reference pin for the A/D Converter.

**ADC [7:6] (TQFP and VFQFN Package Only):** In the TQFP and VFQFN package, ADC[7:6] serve as analog inputs to the A/D converter. These pins are powered from the analog supply and serve as 10-bit ADC channels.

**12. ICSP pin:** Mostly, ICSP (12) is an AVR, a tiny programming header for the Arduino consisting of MOSI, MISO, SCK, RESET, VCC, and GND. It is often referred to as an SPI (Serial Peripheral Interface), which could be considered as an "expansion" of the output. Actually, you are slaving the output device to the master of the SPI bus.

**13. Power LED indicator:** This LED should light up when you plug your Arduino into a power source to indicate that your board is powered up correctly. If this light does not turn on, then there is something wrong with the connection.

**14. TX and RX LEDs:** On your board, you will find two labels: TX (transmit) and RX (receive). They appear in two places on the Arduino UNO board. First, at the digital pins 0 and 1, to indicate the pins responsible for serial communication. Second, the TX and RX led (13). The TX led flashes with different speed while sending the serial data. The speed of flashing depends on the baud rate used by the board. RX flashes during the receiving process.

**15. Digital I / O:** The Arduino UNO board has 14 digital I/O pins (15) (of which 6 provide PWM (Pulse Width Modulation) output. These pins can be configured to work as input digital pins to read logic values (0 or 1) or as digital output pins to drive different modules like LEDs, relays, etc. The pins labeled “~” can be used to generate PWM.

**16. AREF:**AREF stands for Analog Reference. It is sometimes, used to set an external reference voltage (between 0 and 5 Volts) as the upper limit for the analog input pins working.

**Introduction to Arduino IDE**

Arduino is a prototype platform (open-source) based on an easy-to-use hardware and software. It consists of a circuit board, which can be programmed (referred to as a microcontroller) and a ready-made software called Arduino IDE (Integrated Development Environment), which is used to write and upload the computer code to the physical board.

**The key features are**:

* Arduino boards are able to read analog or digital input signals from different sensors and turn it into an output such as activating a motor, turning LED on/off, connect to the cloud and many other actions.
* You can control your board functions by sending a set of instructions to the microcontroller on the board via Arduino IDE (referred to as uploading software).
* Unlike most previous programmable circuit boards, Arduino does not need an extra piece of hardware (called a programmer) in order to load a new code onto the board. You can simply use a USB cable.
* Additionally, the Arduino IDE uses a simplified version of C++, making it easier to learn to program.
* Finally, Arduino provides a standard form factor that breaks the functions of the micro-controller into a more accessible package.

After learning about the main parts of the Arduino UNO board, we are ready to learn how to set up the Arduino IDE. Once we learn this, we will be ready to upload our program on the Arduino board.

**Arduino data types:**

Data types in C refers to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in the storage and how the bit pattern stored is interpreted.

The following table provides all the data types that you will use during Arduino programming.

**Void:**

The void keyword is used only in function declarations. It indicates that the function is expected to return no information to the function from which it was called.

**Example:**

Void Loop ( )

{

// rest of the code

}

**Boolean:**

A Boolean holds one of two values, true or false. Each Boolean variable occupies one byte of memory.

**Example:**

Boolean state= false ; // declaration of variable with type boolean and initialize it with false.

Boolean state = true ; // declaration of variable with type boolean and initialize it with false.

**Char:**A data type that takes up one byte of memory that stores a character value. Character literals are written in single quotes like this: 'A' and for multiple characters, strings use double quotes: "ABC".

However, characters are stored as numbers. You can see the specific encoding in the ASCII chart. This means that it is possible to do arithmetic operations on characters, in which the ASCII value of the character is used. For example, 'A' + 1 has the value 66, since the ASCII value of the capital letter A is 65.

**Example:**

Char chr\_a = ‘a’ ;//declaration of variable with type char and initialize it with character a.

Char chr\_c = 97 ;//declaration of variable with type char and initialize it with character 97

**Unsigned char:**

**Unsigned char** is an unsigned data type that occupies one byte of memory. The unsigned char data type encodes numbers from 0 to 255.

**Example:**

Unsigned Char chr\_y = 121 ; // declaration of variable with type Unsigned char and initialize it with character y

**Byte:**

A byte stores an 8-bit unsigned number, from 0 to 255.

**Example:**

byte m = 25 ;//declaration of variable with type byte and initialize it with 25

**int:**

Integers are the primary data-type for number storage. **int** stores a 16-bit (2-byte) value. This yields a range of -32,768 to 32,767 (minimum value of -2^15 and a maximum value of (2^15) - 1).

The **int** size varies from board to board. On the Arduino Due, for example, an **int** stores a 32-bit (4-byte) value. This yields a range of -2,147,483,648 to 2,147,483,647 (minimum value of -2^31 and a maximum value of (2^31) - 1).

**Example:**

int counter = 32 ;// declaration of variable with type int and initialize it with 32.

**Unsigned int:**

Unsigned ints (unsigned integers) are the same as int in the way that they store a 2 byte value. Instead of storing negative numbers, however, they only store positive values, yielding a useful range of 0 to 65,535 (2^16) - 1). The Due stores a 4 byte (32-bit) value, ranging from 0 to 4,294,967,295 (2^32 - 1).

**Example:**

Unsigned int counter= 60 ; // declaration of variable with type unsigned int and initialize it with 60.

**Word:**

On the Uno and other ATMEGA based boards, a word stores a 16-bit unsigned number. On the Due and Zero, it stores a 32-bit unsigned number.

**Example**

word w = 1000 ;//declaration of variable with type word and initialize it with 1000.

**Long:**

Long variables are extended size variables for number storage, and store 32 bits (4 bytes), from 2,147,483,648 to 2,147,483,647.

**Example:**

Long velocity= 102346 ;//declaration of variable with type Long and initialize it with 102346

**Unsigned long:**Unsigned long variables are extended size variables for number storage and store 32 bits (4 bytes). Unlike standard longs, unsigned longs will not store negative numbers, making their range from 0 to 4,294,967,295 (2^32 - 1).

**Example:**

Unsigned Long velocity = 101006 ;// declaration of variable with type Unsigned Long and initialize it with 101006.

**Short:**

A short is a 16-bit data-type. On all Arduinos (ATMega and ARM based), a short stores a 16-bit (2-byte) value. This yields a range of -32,768 to 32,767 (minimum value of -2^15 and a maximum value of (2^15) - 1).

**Example:**

float num = 1.352;//declaration of variable with type float and initialize it with 1.352.

**Double:**

On the Uno and other ATMEGA based boards, Double precision floating-point number occupies four bytes. That is, the double implementation is exactly the same as the float, with no gain in precision. On the Arduino Due, doubles have 8-byte (64 bit) precision.

**Example:**

double num = 45.352 ;// declaration of variable with type double and initialize it with 45.352.



Figure 1:USB cable

First you must have your Arduino board (you can choose your favorite board) and a USB cable. In case you use Arduino UNO, Arduino Duemilanove, Nano, Arduino Mega2560, or Diecimal, you will need a standard USB cable (A plug to B plug), the kind you would connect to a USB printer as shown in the above image.

**CHAPTER 5**

**RESULTS AND OBSERVATION**

**5.1 PURPOSE**:

The **setup()** function is called when a sketch starts. Use it to initialize the variables, pin modes, start using libraries, etc. The setup function will only run once, after each power up or reset of the Arduino board.

**INPUT**

**OUTPUT**

**RETURN**

Void Loop ( )

{

}

**PURPOSE:**

After creating a **setup ()** function, which initializes and sets the initial values, the **loop()** function does precisely what its name suggests, and loops seductively, allowing your program to change and respond. Use it to actively control the Arduino board.

Input

Output

Return

**Program**

#include <avr/wdt.h>void setup(){

Serial.begin(9600);

Serial.println("Watchdog Timer");

Serial.println("Setup started :");

// make a delay before enable WDT

// this delay help to complete all initial tasks

delay(2000);

//wdt\_enable(WDT Reset Timer);

wdt\_enable(WDTO\_4S); //Enable Watchdog timer

}

void loop(){

Serial.println("LOOP started ! ");

for(int i=0; i<=5; i++){

Serial.print("Loop : ");

Serial.print(i);

Serial.println();

delay(1000);

wdt\_reset();//Reset watchdog timer

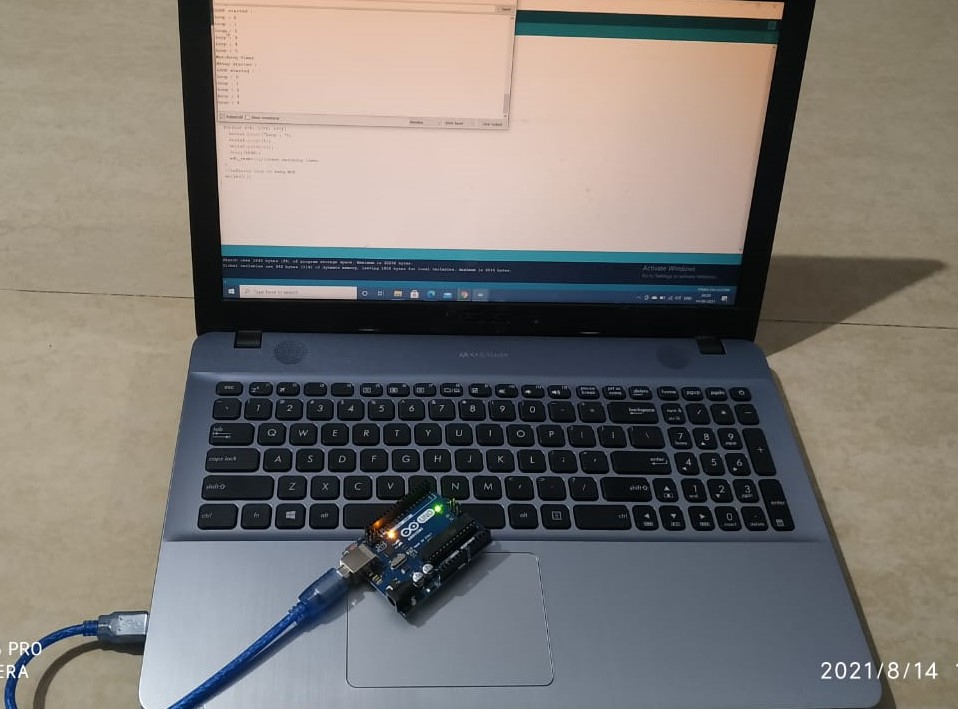
}

//infinity loop to hang MCU

while(1){}

}

**Observation:**



**Figure 5.1: Observation**

**CHAPTER 6**

**6.1 CONCLUSIONS**

*"*A watchdog timer is a hardware timer that automatically generates a system reset if the main program neglects to periodically service it.  It is often used to automatically reset an embedded device that hangs because of a software or hardware fault.*" (*os.mbed.com/cookbook/Watchdog-Timer*)* . We have a main part in program which runs over and over(loop). We are enabling watchdog timer is loaded with an initial value greater than the total delay in main program. Each time main program reset this timer. If any case the main program does not get back to reset the timer before it counts down to zero, an interrupt is generated to reset the processor. Likewise watchdog timer protect micro-controller from hang case. Here we have used watch dog timer in Arduino to prevent it from hanging, strucking and to make its work effectively. Watch dog timer is implemented in Arduino using Arduino ide program and then it id dumped into Arduino using the cable. By implementing the watch dog timer in Arduino we can reset the timer without the human intervention.

6.2 FUTURE SCOPE

Watch dog timer is implemented in Arduino to ensure that the microcontroller should work properly. Watch dog timer can be used in home automation, computers processors, when there occurs hanging, struck and whenever the system goes into any infinity loop. The modification of the project is that we implement watch dog timer in

Operating systems and embedded systems.

**REFERENCES:**

[1] <https://developer.mbed.org/cookbook/WatchDog-Timer>

[2] <http://searchmobilecomputing.techtarget.com/definition/watchdog-timer>

[3] <https://github.com/esp8266/Arduino/blob/4897e0006b5b0123a2fa31f67b14a3fff65ce561/doc/faq/a02-my-esp-crashes.md>

[4] <http://www.esp8266.com/viewtopic.php?p=2096#p2112>

["The Grenade Timer: Fortifying the Watchdog Timer Against Malicious Mobile Code"](http://www.cl.cam.ac.uk/~rja14/Papers/grenade.pdf)

 by Frank Stajano and Ross Anderson (2000).

[5] [**^**](https://en.m.wikipedia.org/wiki/Watchdog_timer#cite_ref-ESD_3-0) Murphy, Niall & Barr, Michael. ["Watchdog Timers"](https://www.embedded.com/electronics-blogs/beginner-s-corner/4023849/Introduction-to-Watchdog-Timers). Embedded Systems Programming. Retrieved 18 February 2013.

[6]^ [**a**](https://en.m.wikipedia.org/wiki/Watchdog_timer#cite_ref-SMSWT_4-0) [**b**](https://en.m.wikipedia.org/wiki/Watchdog_timer#cite_ref-SMSWT_4-1) [**c**](https://en.m.wikipedia.org/wiki/Watchdog_timer#cite_ref-SMSWT_4-2) [**d**](https://en.m.wikipedia.org/wiki/Watchdog_timer#cite_ref-SMSWT_4-3) Lamberson, Jim. ["Single and Multistage Watchdog Timers"](http://www.sensoray.com/downloads/appnote_826_watchdog_1.0.0.pdf) (PDF). Sensoray. Retrieved 10 September 2013.

[7][**^**](https://en.m.wikipedia.org/wiki/Watchdog_timer#cite_ref-6) Weingel, Christer. ["The Linux Watchdog driver API"](https://www.kernel.org/doc/html/latest/watchdog/watchdog-api.html). Retrieved 20 January 2021.

[8][**^**](https://en.m.wikipedia.org/wiki/Watchdog_timer#cite_ref-7) ["Watchdog 'man' page"](http://linux.die.net/man/8/watchdog). Retrieved 10 September 2013.

[9][**^**](https://en.m.wikipedia.org/wiki/Watchdog_timer#cite_ref-8) ["Linux Watchdog - General Tests"](http://www.sat.dundee.ac.uk/~psc/watchdog/watchdog-general-tests.html). Retrieved 10 September 2013.