**REPORT ON CAR WIPER CONTROL SYSTEM**

**Submitted By**

**Anitha Jinkala**

**CONTENTS**

**1 Abstract**

**2 Introduction**

**3 Requirements**

* **Hardware**
* **Software**
* **Embedded C Language**

**4 Design or Architecture**

* **Data Flow Diagram**
* **Proposed System**
* **Flow Chart**
* **UML Diagram**

**5 Implementation**

* **C Program (main.c)**
* **Make File**
* **Doxygen**
* **Doxygen\_log**
* **Description about component**

**6 Test-Plan**

**1 Abstract**

The traditional wiper system requires driver’s attention to switch on the wiper system during rainfall. Whereas in traffic condition, driver should not be diverted by manual adjustment of switching the wiper system which may lead to accident. Probably 80% of accidents happen due to the distraction of driver. In this scenario we need to design an automatic wiping on the wind screen during rain so as to avoid distraction of driver. The existing automatic wiper system has false wiping just after the rainfall stops which can be overcome by using proposed wiper system. Always just after the rainfall a few droplets on the existing water sensor will be sustained until it is cleaned or inherently evaporated. These water drops make a connection between two grid lines to occur false wiping. The advantage of proposed automatic wiper system is compared with the water sensor of existing automatic wiper system after rainfall. The proposed system in this paper is more accurate and economically cheap which can be implemented in all low and middle level vehicles. In order to avoid critical situation this automatic wiper system provides variable wiping speed based on precipitation level. This automatic wiper system has low cost water sensor, STM32F407 microcontroller, STM32cubeMX, QEMU Software, LCD Module, LED and power cable and Power Supply

**2 Introduction**

All automotive industries seek to provide low cost system for all the applications including automatic wiper system in vehicle. In recent time automotive industries focusing on autonomous vehicle which means self-drive system on different applications. For this scenario, this paper offers low Cost wiper system with simple and effective concept of electro mechanical concept to wipe the windscreen automatically. Automatic wiping has been done during rainfall without human interrupt. Thus an uninterrupted makes to avoid distraction of the drive and secure from accidents. Nowadays vehicles are more automated whereas the cost of the embedded system used for different critical applications are too high. Basically, increase in technology will enhance the vehicle cost. This criterion makes to develop low cost automatic wiper system. The wiper system has been implemented to forecast in all low cost vehicles. In the current scenario, only luxury vehicles employ intelligent rain sensing wind shield Wiper systems. Our system is modelled to demonstrate how useful is an automatic wiper system that adjusts speed itself based on rainfall intensity. Such a system improves the safety of a ride. An automatic, intelligent system like ours removes any manual errors. Our system adjusts wiper speed according to the intensity of rainfall and hence improves the safety.

**3 Requirements**

The Requirements which are essential to construct “Car Wiper Control System”. A windshield wiper system comprises a wiper drive and two wiper arms. The drive moves the two wiper arms at a certain angle across the windshield, providing a clear view for the driver and passenger. A specially-shaped rubber wiping lip ensures an optimal wiping result.

Visibility is further improved by the new generation of Jet Wiper arms. With the Jet Wiper, the washer fluid is applied exactly where it is needed, i.e. right in front of the wiper blade. The spray nozzles are integrated in the wiper arm, which improves cleaning performance, extends the field of vision and permits a clear view. Unlike conventional systems, no expansive spray mist is generated. This significantly improves visibility even at higher speeds.

A rear wiper system comprises a rear wiper drive, which only powers a single wiper arm. The system can be mounted to the top or bottom of the rear window.

**Hardware**

The Hardware equipment are the basic components to build Car Wiper Control System. The following are the different hardware components to develop this project and they are,

* STM32F407 Microcontroller
* LED’s (RED, GREEN, BLUE)
* Switch or Push Button
* Timer
* Power Supply

**STM32F407 Microcontroller**

The STM32 F4-series is the first group of STM32 microcontrollers based on the ARM Cortex-M4F core. The F4-series is also the first STM32 series to have DSP and floating-point instructions. The F4 is [pin-to-pin compatible](https://en.wikipedia.org/wiki/Pin-compatibility) with the STM32 F2-series and adds higher clock speed, 64 KB CCM static RAM, full-duplex I²S, improved real-time clock, and faster ADCs.

* **Core:**
* [ARM Cortex-M4F](https://en.wikipedia.org/wiki/ARM_Cortex-M4F) core at a maximum clock rate of 84 / 100 / 168 / 180 [MHz](https://en.wikipedia.org/wiki/MHz" \o "MHz).
* **Memory:**
  + - * [Static RAM](https://en.wikipedia.org/wiki/Static_RAM) consists of up to 192 KB general-purpose, 64 KB core-coupled memory (CCM), 4 KB battery-backed, 80 bytes battery-backed with tamper-detection erase.
      * Flash consists of 512 / 1024 / 2048 [KB](https://en.wikipedia.org/wiki/Kilobyte) general-purpose, 30 KB system boot, 512 bytes one-time programmable (OTP), 16 option bytes.
      * Each chip has a factory-programmed 96-bit unique device identifier number.
* **Peripherals:**
  + - * Common peripherals included in all IC packages are [USB](https://en.wikipedia.org/wiki/USB) 2.0 OTG HS and FS, two CAN 2.0B, one SPI + two SPI or full-duplex [I²S](https://en.wikipedia.org/wiki/I%C2%B2S), three [I²C](https://en.wikipedia.org/wiki/I%C2%B2C), four USART, two [UART](https://en.wikipedia.org/wiki/UART), [SDIO](https://en.wikipedia.org/wiki/Secure_Digital#SDIO) for [SD](https://en.wikipedia.org/wiki/Secure_Digital)/[MMC](https://en.wikipedia.org/wiki/MultiMediaCard) cards, twelve 16-bit [timers](https://en.wikipedia.org/wiki/Timers#Computer_timers), two 32-bit timers, two [watchdog](https://en.wikipedia.org/wiki/Watchdog_timer) timers, [temperature](https://en.wikipedia.org/wiki/Temperature) sensor, 16 or 24 channels into three ADCs, two DACs, 51 to 140 [GPIOs](https://en.wikipedia.org/wiki/General-purpose_input/output), sixteen [DMA](https://en.wikipedia.org/wiki/Direct_memory_access), improved real-time clock ([RTC](https://en.wikipedia.org/wiki/Real-time_clock)), [cyclic redundancy check](https://en.wikipedia.org/wiki/Cyclic_redundancy_check) (CRC) engine, [random number generator](https://en.wikipedia.org/wiki/Random_number_generation) (RNG) engine. Larger IC packages add 8/16-bit external [memory bus](https://en.wikipedia.org/wiki/Memory_bus) capabilities.
      * The STM32F4x7 models add [ethernet](https://en.wikipedia.org/wiki/Ethernet) [MAC](https://en.wikipedia.org/wiki/Media_Independent_Interface) and [camera interface](https://en.wikipedia.org/wiki/Camera_interface).
      * The STM32F41x/43x models add a [cryptographic processor](https://en.wikipedia.org/wiki/Cryptographic_accelerator) for [DES](https://en.wikipedia.org/wiki/Data_Encryption_Standard) / [TDES](https://en.wikipedia.org/wiki/Triple_DES) / [AES](https://en.wikipedia.org/wiki/Advanced_Encryption_Standard), and a hash processor for [SHA-1](https://en.wikipedia.org/wiki/SHA-1) and [MD5](https://en.wikipedia.org/wiki/MD5).
      * The STM32F4x9 models add a [LCD-TFT](https://en.wikipedia.org/wiki/TFT_LCD) controller.
    - [Oscillators](https://en.wikipedia.org/wiki/Electronic_oscillator) consists of internal (16 MHz, 32 kHz), optional external (4 to 26 MHz, 32.768 to 1000 kHz).
    - [IC packages](https://en.wikipedia.org/wiki/Integrated_circuit_packaging): [WLCSP](https://en.wikipedia.org/wiki/WLCSP)64, [LQFP](https://en.wikipedia.org/wiki/LQFP)64, LQFP100, LQFP144, LQFP176, [UFBGA](https://en.wikipedia.org/wiki/UFBGA)176. STM32F429/439 also offers LQFP208 and [UFBGA](https://en.wikipedia.org/wiki/UFBGA)216.
    - Operating voltage range is 1.8 to 3.6 Volts.

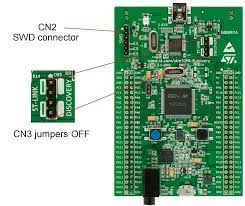
**** 

Fig:STM32F4 Microcontroller

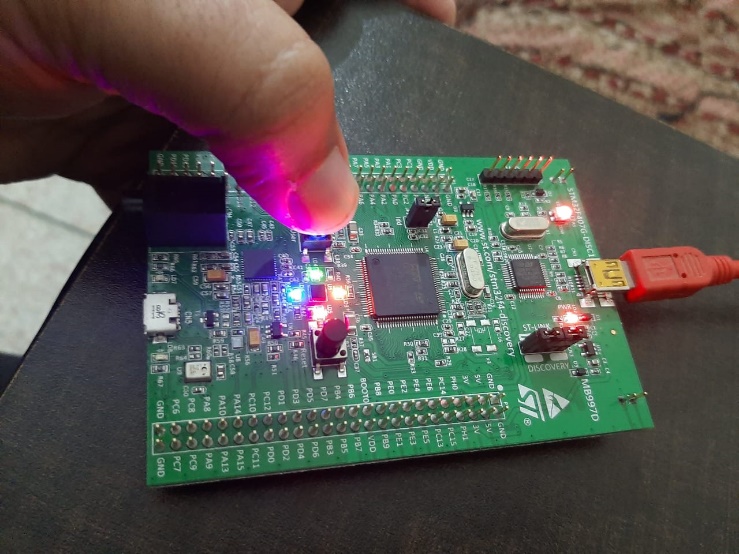


Fig:STM32F4 Microcontroller with Push Button Action

**Features**

* ART Accelerator™ enabling 0 wait state executing from internal Flash
* Up to 2x USB2.0 OTG FS/HS
* SDIO
* USART, SPI, I²C
* 16-bit and 32-bit timers
* Up to 3x 12-bit ADC
* Up to 2x 12-bit DAC
* External memory controller
* 1.7V to 3.6V low voltage

**Advantages**

The ARM® Cortex®-M4-based **STM32F4 series** of MCUs leverages STMicroelectronics' NVM technology and ST’s ART Accelerator™ to reach the industry’s highest benchmark scores for **Cortex-M-based microcontrollers** with up to 225 DMIPS / 608 CoreMark executing from Flash memory at up to 180 MHz operating frequency.

The **STM32F4 series** consists of 7 lines of digital signal controllers (**DSC**), a perfect symbiosis of the real-time control capabilities of an **MCU** and the signal processing performance of a **DSP**:

* STM32F401 – 84 MHz CPU/105 DMIPS, the smallest, cost-effective solution with outstanding power efficiency (Dynamic Efficiency Line)
* STM32F411 – 100 MHz CPU/125 DMIPS, outstanding power efficiency with large SRAM and new smart DMA optimizing power consumption for data batching (Dynamic Efficiency Line with Batch Acquisition Mode)
* [STM32F446](https://www.utmel.com/productdetail/stmicroelectronics-stm32f446ret6-4812562) – 180 MHz/225 DMIPS, up to 512 Kbytes of Flash with dual Quad SPI and SDRAM interfaces
* STM32F405/415 – 168 MHz CPU/210 DMIPS, up to 1 Mbyte of Flash with advanced connectivity and encryption
* STM32F407/417 – 168 MHz CPU/210 DMIPS, up to 1 Mbyte of Flash adding Ethernet MAC and camera interface
* STM32F427/437 – 180 MHz CPU/225 DMIPS, up to 2 Mbytes of dual-bank Flash with SDRAM interface, Chrom-ART Accelerator™, serial audio interface, more performance and lower static power consumption
* STM32F429/439 – 180 MHz CPU/225 DMIPS, up to 2 Mbytes of dual-bank Flash with SDRAM interface, Chrom-ART Accelerator™ and LCD-TFT controller
* STM32F469/479 – 180 MHz CPU/225 DMIPS, up to 2 Mbytes of dual-bank Flash with SDRAM and QSPI interface, Chrom-ART Accelerator™, LCD-TFT controller and MPI-DSI interface

**Working of STM32 To Build This Project**

The Following are the steps to develop project,

**Ignition Key Position at ACC:** The Red LED is ON, if the user button is pressed and held for 2 secs

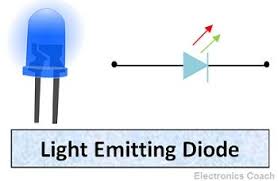
**Wiper ON:** *Wiper is OFF:* On press of the user input, Blue, Green and Orange LEDs come ON one at a time with the set frequency, The frequency changes on every alternate key press, 3 frequency levels with 1, 4 and 8 Hz

**Wiper OFF:** *Wiper is ON:* The LED glow pattern stops on the 4th press; the wiper action starts next press onwards as mentioned in step 2

**Ignition Key Position at Lock:** The Red LED is OFF, if the user button is pressed and held for 2 secs

**LEDs**

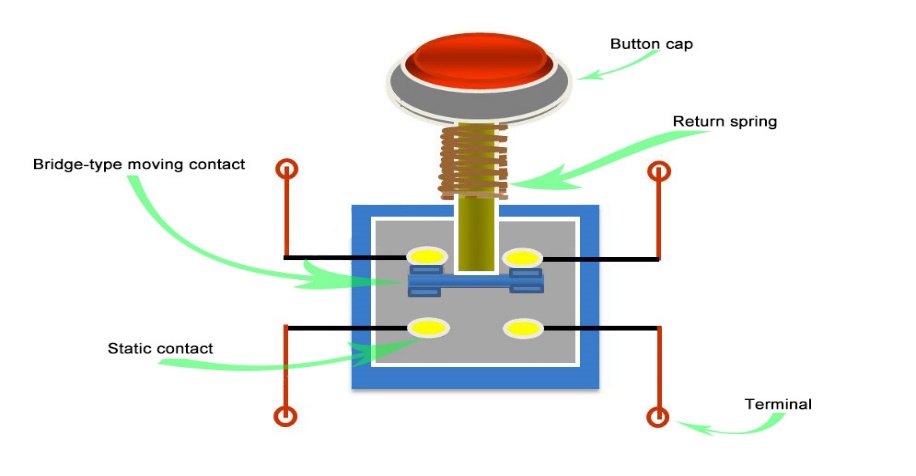
A Light Emitting Diode (LED) is a semiconductor device, which can emit light when an electric current passes through it. To do this, holes from p-type semiconductors recombine with electrons from n-type semiconductors to produce light.

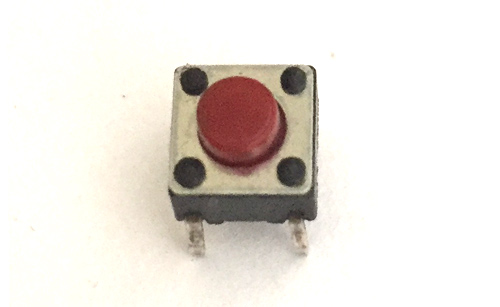


The LED is in built on STM32 Microcontroller. Thus LED is we used to develop this project.

**Push Button or Switch**

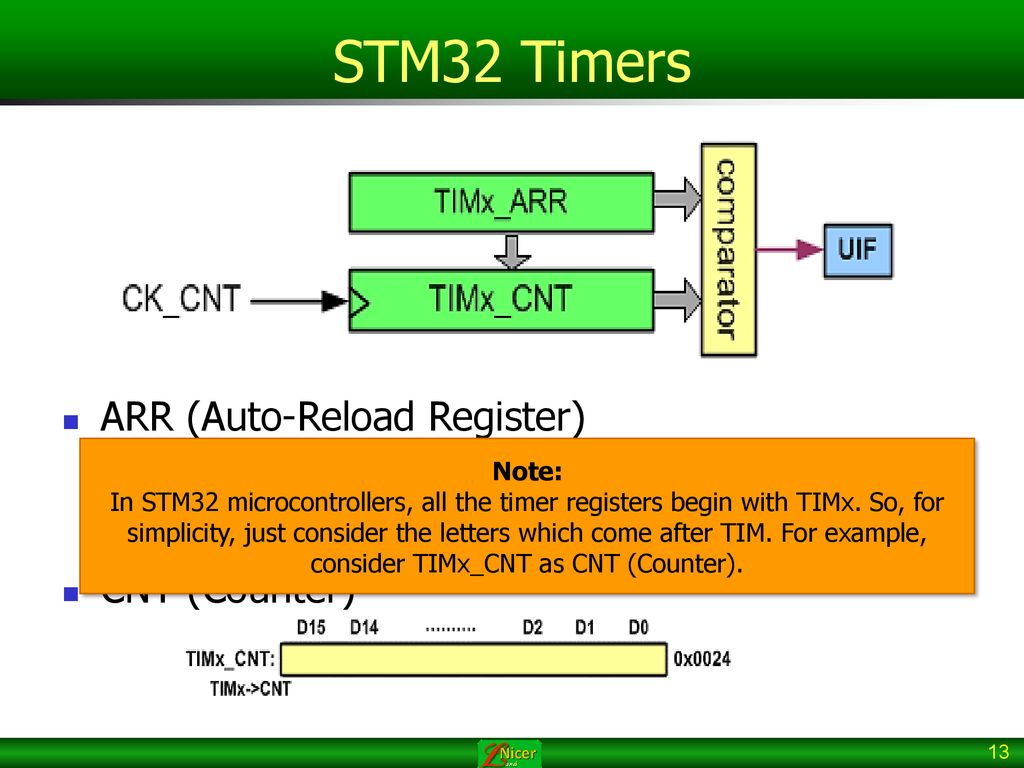
A Push Button switch is a type of switch which consists of a simple electric mechanism or air switch mechanism to turn something on or off. Depending on model they could operate with momentary or latching action function. The button itself is usually constructed of a strong durable material such as metal or plastic.





**Timer**

A timer is a specialized type of clock which is used to measure time intervals. A timer that counts from zero upwards for measuring time elapsed is often called a stopwatch. It is a device that counts down from a specified time interval and used to generate a time delay, for example, an hourglass is a timer.



**Software**

QEMU is a machine emulator that can run operating systems and programs for one machine on a different machine. Mostly it is not used as emulator but as Virtualizer in collaboration with KVM kernel components. In that case it utilizes the virtualization technology of the hardware to virtualize guests.

While Qemu has a [command line interface](https://qemu-project.gitlab.io/qemu/system/invocation.html) and a [monitor](https://qemu-project.gitlab.io/qemu/system/monitor.html) to interact with running guests those is rarely used that way for other means than development purposes. [Libvirt](https://ubuntu.com/server/docs/virtualization-qemu#libvirt) provides an abstraction from specific versions and hypervisors and encapsulates some workarounds and best practices.

**Running Qemu/KVM**

While there are much more user friendly and comfortable ways, using the command below is probably the quickest way to see some called Ubuntu moving on screen is directly running it from the net boot iso.

Warning: this is just for illustration - not generally recommended without verifying the checksums; [Multi pass](https://discourse.ubuntu.com/t/virtualization-multipass) and [UV Tool](https://discourse.ubuntu.com/t/virtualization-uvt) are much better ways to get actual guests easily.

Run:

sudo qemu-system-x86\_64 -enable-kvm -cdrom http://archive.ubuntu.com/ubuntu/dists/bionic-updates/main/installer-amd64/current/images/netboot/mini.iso

You could download the ISO for faster access at runtime and e.g. add a disk to the same by:

* creating the disk
* qemu-img create -f qcow2 disk.qcow 5G
* Using the disk by adding -drive file=disk.qcow,format=qcow2

Those tools can do much more, as you’ll find in their respective (long) man pages. There also is a vast assortment of auxiliary tools to make them more consumable for specific use-cases and needs - for example [virt-manager](https://virt-manager.org/) for UI driven use through libvirt. But in general - even the tools eventually use that - it comes down to:

qemu-system-x86\_64 options image[s]

So take a look at the man page of [qemu](http://manpages.ubuntu.com/manpages/bionic/man1/qemu-system.1.html), [qemu-img](http://manpages.ubuntu.com/manpages/bionic/man1/qemu-img.1.html) and the documentation of [qemu](https://www.qemu.org/documentation/) and see which options are the right one for your needs.

**Graphics**

Graphics for qemu/kvm always comes in two pieces.

* A front end - controlled via the -vga argument - which is provided to the guest. Usually one of cirrus, std, qxl, virtio. The default these days is qxl which strikes a good balance between guest compatibility and performance. The guest needs a driver for what is selected, which is the most common reason to switch from the default to either cirrus (e.g. very old Windows versions)
* A back end - controlled via the -display argument - which is what the host uses to actually display the graphical content. That can be an application window via gtk or a vnc.
* In addition one can enable the -spice back-end (can be done in addition to vnc) which can be faster and provides more authentication methods than vnc.
* if you want no graphical output at all you can save some memory and cpu cycles by setting -nographic

If you run with spice or vnc you can use native vnc tools or virtualization focused tools like virt-viewer. More about these in the libvirt section.

All those options above are considered basic usage of graphics. There are advanced options for further needs. Those cases usually differ in their [ease-of-use and capability](https://cpaelzer.github.io/blogs/006-mediated-device-to-pass-parts-of-your-gpu-to-a-guest/) are:

* *Need some 3D acceleration*: -vga virtio with a local display having a GL context -display gtk,gl=on; That will use [virgil3d](https://virgil3d.github.io/) on the host and needs guest drivers for [virt3d] which are common in Linux since [Kernels >=4.4](https://www.kraxel.org/blog/2016/09/using-virtio-gpu-with-libvirt-and-spice/) but hard to get by for other cases. While not as fast as the next two options, the big benefit is that it can be used without additional hardware and without a proper [IOMMU setup for device passthrough](https://www.kernel.org/doc/Documentation/vfio-mediated-device.txt).
* *Need native performance*: use PCI passthrough of additional GPUs in the system. You’ll need an IOMMU setup and unbind the cards from the host before you can pass it through like -device vfio-pci,host=05:00.0,bus=1,addr=00.0,multifunction=on,x-vga=on -device vfio-pci,host=05:00.1,bus=1,addr=00.1
* *Need native performance, but multiple guests per card*: Like PCI Passthrough, but using mediated devices to shard a card on the Host into multiple devices and pass those like -display gtk,gl=on -device vfio-pci,sysfsdev=/sys/bus/pci/devices/0000:00:02.0/4dd511f6-ec08-11e8-b839-2f163ddee3b3,display=on,rombar=0. More at [kraxel on vgpu](https://www.kraxel.org/blog/2018/04/vgpu-display-support-finally-merged-upstream/) and [Ubuntu GPU mdev evaluation](https://cpaelzer.github.io/blogs/006-mediated-device-to-pass-parts-of-your-gpu-to-a-guest/). The sharding of the cards is driver specific and therefore will differ per manufacturer like [Intel](https://github.com/intel/gvt-linux/wiki/GVTg_Setup_Guide) or [Nvidia](https://docs.nvidia.com/grid/latest/grid-vgpu-user-guide/index.html).

Especially the advanced cases can get pretty complex, therefore it is recommended to use qemu through libvirt for those cases. [Libvirt](https://discourse.ubuntu.com/t/virtualization-libvirt) will take care of all but the host kernel/bios tasks of such configurations. Below are the common basic actions needed for faster options being pass-through and mediated devices pass-through.

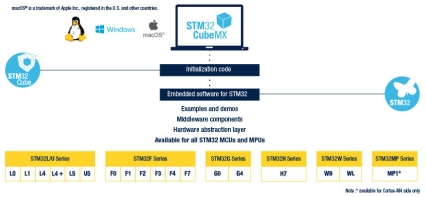
**Preparations for PCI and mediated devices pass-through - IOMMU**

The initial steps for both of these are the same, you want to ensure your system has its IOMMU enabled and the device to pass should be in a group of its own. Enablement of the VT-d and IOMMU is usually an bios action and thereby manufacturer dependent.

On the kernel side for the [IOMMU feature](https://www.kernel.org/doc/html/latest/x86/intel-iommu.html) there are various [options you can enable/configure](https://www.kernel.org/doc/html/latest/admin-guide/kernel-parameters.html?highlight=iommu). In recent Ubuntu Kernels (>=5.4 => Focal or Bionic-HWE kernels) everything usually works by default, unless your hardware setup makes you need any of those tuning options.

**STM32cubeMX**

STM32CubeMX is a graphical tool that allows a very easy configuration of STM32 microcontrollers and microprocessors, as well as the generation of the corresponding initialization C code for the Arm® Cortex®-M core or a partial Linux® Device Tree for Arm® Cortex®-A core, through a step-by-step process. The first step consists in selecting either an STMicrolectronics STM32 microcontroller, microprocessor or a development platform that matches the required set of peripherals, or an example running on a specific development platform.  
For microprocessors, the second step allows to configure the GPIOs and the clock setup for the whole system, and to interactively assign peripherals either to the Arm® Cortex®-M or to the Cortex®-A world. Specific utilities, such as DDR configuration and tuning, make it easy to get started with STM32 microprocessors. For Cortex®-M core, the configuration includes additional steps that are exactly similar to those described for microcontrollers.

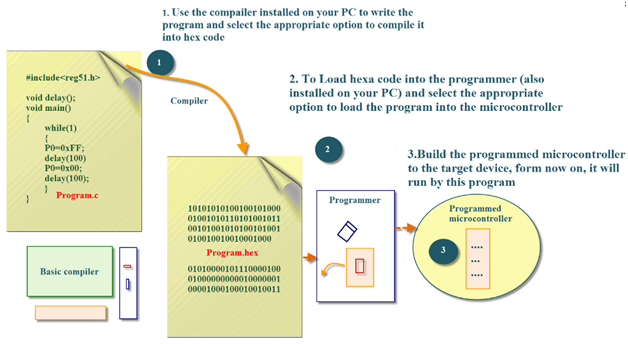


**Embedded C Language**

Embedded C is most popular programming language in software field for developing electronic gadgets. Each processor used in electronic system is associated with embedded software.

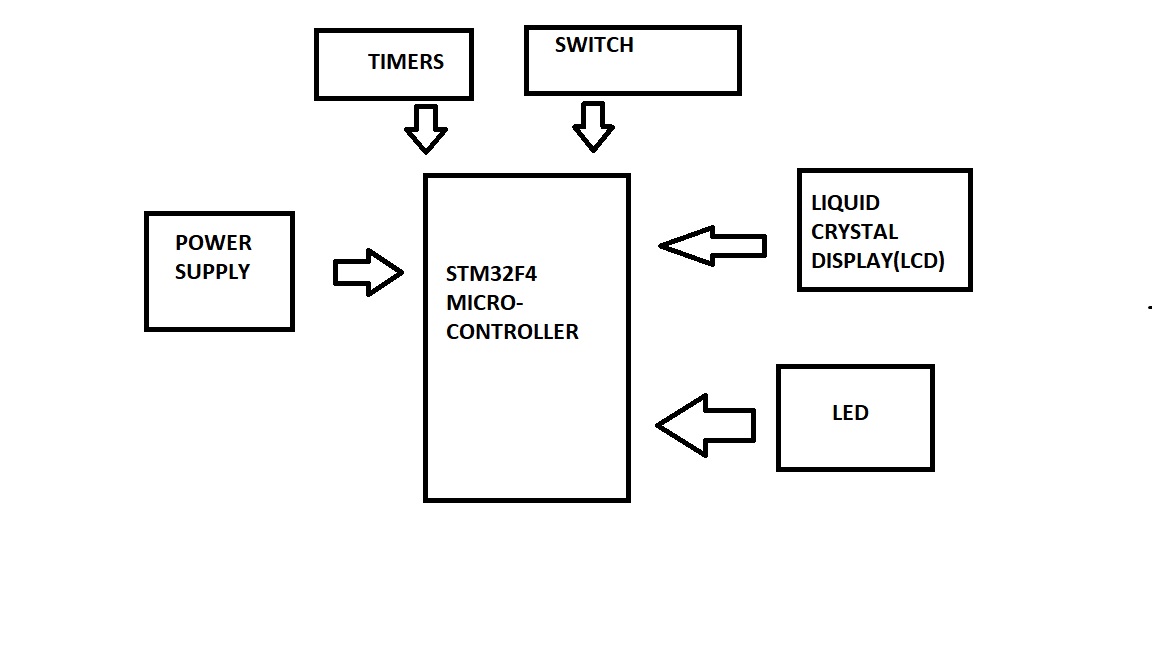
Embedded C programming plays a key role in performing specific function by the processor. In day-to-day life we used many electronic devices such as mobile phone, washing machine, digital camera, etc. These all device working is based on microcontroller that are programmed by embedded C.

Embedded C Programming with Keil Language2

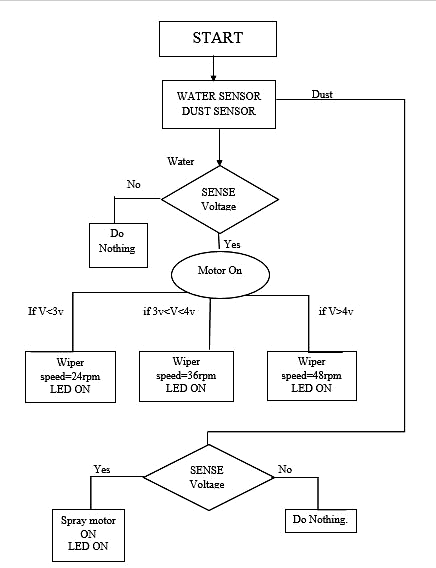


**4 Architecture**

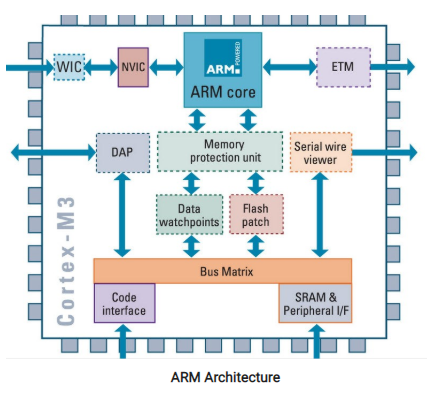
The block diagram for proposed system is given by



**Data Flow Diagram**



**ARM Architecture**



**5 Implementation**

**Source Code**

The program is developed to run the hardware kit with set of instructions,

/\*PROJECT :- WIPER CONTROL SYSTEM\*/

//DEFINING all Required pins

#define GPIOD\_BASE\_ADDR 0x40020C00 //Defining Base address for GPIO//for LESDs

#define GPIOD\_ODR\_OFFSET 0x14 //LED are outputs so we are using ODR //Defining Offset Address for LEDs

#define GPIOD\_ODR \*(volatile long \*)(GPIOD\_BASE\_ADDR + GPIOD\_ODR\_OFFSET)

//Adding Base address and offset address for LEDs

#define RCC\_BASE\_ADDR 0x40023800 //Defining RCC Address

#define RCC\_AHB1ENR\_OFFSET 0x30 //Defining Offset for RCC

#define RCC\_AHB1ENR \*(volatile long \*)(RCC\_BASE\_ADDR + RCC\_AHB1ENR\_OFFSET)

//Adding Base address and offset address for RCC

#define GPIOA\_BASE\_ADDR 0x40020000 //Defining Base address for switch

#define GPIOA\_IDR\_OFFSET 0x10 //switch is input so we are using IDR //Defining Offset of Switch

#define GPIOA\_IDR \*(volatile long \*)(GPIOA\_BASE\_ADDR + GPIOA\_IDR\_OFFSET)

//Adding Base address and offset address for SWITCH

#define GPIOD\_MODE\_OFFSET 0x00

#define GPIOD\_MODE \*(volatile long \*)(GPIOD\_BASE\_ADDR + GPIOD\_MODE\_OFFSET)

//Adding Base address and offset address To clear Previous for LEDs

#define GPIOA\_MODE\_OFFSET 0x00

#define GPIOA\_MODE \*(volatile long \*)(GPIOA\_BASE\_ADDR + GPIOA\_MODE\_OFFSET)

//Adding Base address and offset address to clear Previous Switch settings

//STARTING of Main PROGRAM

int main()

{

int i,b=0,j=0; //Declaring and Initilizing Variables

RCC\_AHB1ENR |= (1<<3)|(1<<0); //Enabling the LED and the switch

GPIOD\_MODE = 0; //For Clear LED

GPIOA\_MODE = 0; //For Clear Switch

GPIOA\_IDR |= 1<<3;

GPIOD\_MODE |= (0<<25)|(1<<24); //to assign LED 12 as output pin

GPIOD\_MODE |= (0<<27)|(1<<26); //to assign LED 13 as output pin

GPIOD\_MODE |= (0<<29)|(1<<28); //to assign LED 14 as output pin

GPIOD\_MODE |= (0<<31)|(1<<30); //to assign LED 14 as output pin as per the Datasheet

GPIOA\_MODE |= (0<<1)|(0<<0); //to assign Switch as input pin as per the Datasheet

GPIOD\_ODR = 0X00;

while(1)

{

while((GPIOA\_IDR & 0x01) == 1) // Switch Status On

{

b++; //Increment for Switch

}

if(b>30000)

{

b=0;//RED Led should be in ON state

GPIOD\_ODR |= 1<<14;

for(i=0;i<100000;i++);

}

else if(b>0 && b<30000)

{

j=1;

}

else

{

}

if(j==1)

{

//By using Fast Frequency delay

for(j=0;j<5;j++) //For LED GREEN

{

GPIOD\_ODR |= 1<<12;

for(i=0;i<20000;i++);

GPIOD\_ODR &= ~(1<<12);

for(i=0;i>=20000;i++);

}

for(j=0;j<5;j++) //For LED ORANGE

{

GPIOD\_ODR |= 1<<13;

for(i=0;i<20000;i++);

GPIOD\_ODR &= ~(1<<13);

for(i=0;i>=20000;i++);

}

for(j=0;j<5;j++) //For LED BLUE

{

GPIOD\_ODR |= 1<<15;

for(i=0;i<20000;i++);

GPIOD\_ODR &= ~(1<<15);

for(i=0;i>=20000;i++);

}

//Medium Speed Delay

for(j=0;j<5;j++) //For LED GREEN

{

GPIOD\_ODR |= 1<<12;

for(i=0;i<40000;i++);

GPIOD\_ODR &= ~(1<<12);

for(i=0;i>=40000;i++);

}

for(j=0;j<5;j++) //For LED ORANGE

{

GPIOD\_ODR |= 1<<13;

for(i=0;i<40000;i++);

GPIOD\_ODR &= ~(1<<13);

for(i=0;i>=40000;i++);

}

for(j=0;j<5;j++) //For LED BLUE

{

GPIOD\_ODR |= 1<<15;

for(i=0;i<40000;i++);

GPIOD\_ODR &= ~(1<<15);

for(i=0;i>=40000;i++);

}

//High Speed Delay

for(j=0;j<5;j++) //For LED GREEN

{

GPIOD\_ODR |= 1<<12;

for(i=0;i<80000;i++);

GPIOD\_ODR &= ~(1<<12);

for(i=0;i>=80000;i++);

}

for(j=0;j<5;j++) //For LED ORANGE

{

GPIOD\_ODR |= 1<<13;

for(i=0;i<80000;i++);

GPIOD\_ODR &= ~(1<<13);

for(i=0;i>=80000;i++);

}

for(j=0;j<5;j++) //For LED BLUE

{

GPIOD\_ODR |= 1<<15;

for(i=0;i<80000;i++);

GPIOD\_ODR &= ~(1<<15);

for(i=0;i>=80000;i++);

}

}

}

return 0;

}

**Makefile**

The make utility requires a file, Makefile or makefile , which defines set of tasks to be executed. You may have used make to compile a program from source code. Most open source projects use make to compile a final executable binary, which can then be installed using make install.

# Automatically-generated file. Do not edit!

# Toolchain: GNU Tools for STM32 (9-2020-q2-update)

################################################################################

-include ../makefile.init

RM := rm -rf

# All of the sources participating in the build are defined here

-include sources.mk

-include Startup/subdir.mk

-include Src/subdir.mk

-include subdir.mk

-include objects.mk

ifneq ($(MAKECMDGOALS),clean)

ifneq ($(strip $(S\_DEPS)),)

-include $(S\_DEPS)

endif

ifneq ($(strip $(S\_UPPER\_DEPS)),)

-include $(S\_UPPER\_DEPS)

endif

ifneq ($(strip $(C\_DEPS)),)

-include $(C\_DEPS)

endif

endif

-include ../makefile.defs

OPTIONAL\_TOOL\_DEPS := \

$(wildcard ../makefile.defs) \

$(wildcard ../makefile.init) \

$(wildcard ../makefile.targets) \

BUILD\_ARTIFACT\_NAME := NEWP

BUILD\_ARTIFACT\_EXTENSION := elf

BUILD\_ARTIFACT\_PREFIX :=

BUILD\_ARTIFACT := $(BUILD\_ARTIFACT\_PREFIX)$(BUILD\_ARTIFACT\_NAME)$(if $(BUILD\_ARTIFACT\_EXTENSION),.$(BUILD\_ARTIFACT\_EXTENSION),)

# Add inputs and outputs from these tool invocations to the build variables

EXECUTABLES += \

NEWP.elf \

SIZE\_OUTPUT += \

default.size.stdout \

OBJDUMP\_LIST += \

NEWP.list \

OBJCOPY\_BIN += \

NEWP.bin \

# All Target

all: main-build

# Main-build Target

main-build: NEWP.elf secondary-outputs

# Tool invocations

NEWP.elf: $(OBJS) $(USER\_OBJS) D:\RBEI\ Sushu\STM32CubeIDE\PROJECT\NEWP\STM32F407VGTX\_FLASH.ld makefile objects.list $(OPTIONAL\_TOOL\_DEPS)

arm-none-eabi-gcc -o "NEWP.elf" @"objects.list" $(USER\_OBJS) $(LIBS) -mcpu=cortex-m4 -T"D:\RBEI Sushu\STM32CubeIDE\PROJECT\NEWP\STM32F407VGTX\_FLASH.ld" --specs=nosys.specs -Wl,-Map="NEWP.map" -Wl,--gc-sections -static --specs=nano.specs -mfpu=fpv4-sp-d16 -mfloat-abi=hard -mthumb -Wl,--start-group -lc -lm -Wl,--end-group

@echo 'Finished building target: $@'

@echo ' '

default.size.stdout: $(EXECUTABLES) makefile objects.list $(OPTIONAL\_TOOL\_DEPS)

arm-none-eabi-size $(EXECUTABLES)

@echo 'Finished building: $@'

@echo ' '

NEWP.list: $(EXECUTABLES) makefile objects.list $(OPTIONAL\_TOOL\_DEPS)

arm-none-eabi-objdump -h -S $(EXECUTABLES) > "NEWP.list"

@echo 'Finished building: $@'

@echo ' '

NEWP.bin: $(EXECUTABLES) makefile objects.list $(OPTIONAL\_TOOL\_DEPS)

arm-none-eabi-objcopy -O binary $(EXECUTABLES) "NEWP.bin"

@echo 'Finished building: $@'

@echo ' '

# Other Targets

clean:

-$(RM) $(SIZE\_OUTPUT)$(OBJDUMP\_LIST)$(EXECUTABLES)$(OBJS)$(S\_DEPS)$(S\_UPPER\_DEPS)$(C\_DEPS)$(OBJCOPY\_BIN) NEWP.elf

-@echo ' '

secondary-outputs: $(SIZE\_OUTPUT) $(OBJDUMP\_LIST) $(OBJCOPY\_BIN)

fail-specified-linker-script-missing:

@echo 'Error: Cannot find the specified linker script. Check the linker settings in the build configuration.'

@exit 2

warn-no-linker-script-specified:

@echo 'Warning: No linker script specified. Check the linker settings in the build configuration.'

.PHONY: all clean dependents fail-specified-linker-script-missing warn-no-linker-script-specified

-include ../makefile.targets

**Doxygen**

Doxygen allows you to put your documentation blocks practically anywhere (the exception is inside the body of a function or inside a normal C style comment block).Doxygen (/ˈdɒksidʒən/ DOK-see-jən) is **a documentation generator and static analysis tool for software source trees**. When used as a documentation generator, Doxygen extracts information from specially-formatted comments within the code.

**Implementation**

**About Programming**

The complete procedure of writing, compiling and testing the user program can be done in a set of

programs called the development suite.

The following files to create the machine code for the microcontroller:

\* The user program, a text file written in “C” language.

\* The file “startup\_stm32f4xx.s”, a text file written in assembly language; this file contains

instructions for the initial set-up of the microcontroller (stack, program counter, interrupt

vector table, initial system clock …; the description of the file is given in its header).

\* The file “system\_stm32f4xx.c”, a text file written in “C”; this file contains functions for the

detailed microcontroller set-up (system clock, clock distribution …; the description of this file is

given in its header).

In our case the file may have different names; since the file contains clock configuration

commands it is best to prepare several files in advance, each for a different configuration of the

clock and then simply include in the process of compiling the file that corresponds to the

current needs. We will run the microcontroller at its maximum speed, and the corresponding

file is named “system\_stm32f4xx\_CLK168\_HSE8.c”.

\* The header file “stm32f4xx.h”; this file defines processor used, names the registers and bits

inside the STM32F4xx microcontroller, and defines some register structures. This is the only

file that must be included from the user program in the process of compilation. The file has

been modified from the original (obtained from the ST site):

\* by uncommenting the line 68 (select the microcontroller STM32F40xx by default),

\* by uncommenting the line 88 (allow the use of standard peripheral driver) and

\* by changing the HSE frequency in line 100

\* The header file “system\_stm32f4xx.h”; allows the definition of some stuff for the use of CMSIS.

\* The header file “stm32F4xx\_conf.h”; defines and includes some files mandatory for the project.

The header file defines the data structures used in

accessing the peripheral and names the registers and

constants (“stm32f4xx\_$$$$.h”).

-The source file contains the actual functions to access

the registers responsible for the behavior of

peripherals (“stm32f4xx\_$$$$.c”).

## Struct

A structure is used to store the different types of data and every data (structure member) has own independent memory that means we can access any member anytime.

Syntax of structure in C:

struct [name of structure] {member-list };

We cannot initialize the member of the structure at the time of the structure declaration because there is no memory is allocated to the members at the time of declaration.

struct MyData

{

int Age;

float fees;

char name[4];

} data;

### Example

typedef struct

{

uint32\_t GPIO\_Pin;

GPIOMode\_TypeDef GPIO\_Mode;

GPIOSpeed\_TypeDef GPIO\_Speed;

GPIOOType\_TypeDef GPIO\_OType;

GPIOPuPd\_TypeDef GPIO\_PuPd;

}GPIO\_InitTypeDef;

EX:

static struct platform\_driver leds\_platform\_driver = {

.driver = {

.name = "imx6ul-led",

.of\_match\_table = leds\_of\_match,

},

.probe = leds\_probe,

.remove = leds\_remove,

};

# enum Keyword

The enum is nothing but enemuration.In which the values of some data are often limited and can only be a very small number of integers, and it is best to take a name for each value to facilitate its use in subsequent codes.

enum week a, b, c;

Typedef enum

### Example

enum week{

Mon,

Tues,

Wed,

Thurs,

Fri,

Sat,

Sun

};

## Functions

A function is a group of statements that together perform a task.

Every C program has at least one function, which is main(), and all the most trivial programs can define additional functions.

You can divide up your code into separate functions.

\* Avoid repetition of codes.

\* Increases program readability.

\* Divide a complex problem into simpler ones.

\* Reduces chances of error.

\* Modifying a program becomes easier by using function.

## Macros

The macros are for port and pin read and write.A macro is a piece of code in a program that is replaced by the value of the macro. Macro is defined by #define directive. Whenever a macro name is encountered by the compiler, it replaces the name with the definition of the macro.

#define Mon 1

#define Tues 2

#define Wed 3

#define Thurs 4

#define Fri 5

#define Sat 6

#define Sun 7

### Example

#ifndef PINOUT\_BLUE\_PILL\_H

#define PINOUT\_BLUE\_PILL\_H

#define SIZE\_OF\_PIN (sizeof(pin\_t))

#define PIN\_BLUE\_PILL\_PA0 ((pin\_t const){GPIOA, 0U})

#define PIN\_BLUE\_PILL\_PA1 ((pin\_t const){GPIOA, 1U})

...

#define PIN\_BLUE\_PILL\_WKUP (PIN\_BLUE\_PILL\_PA0)

#define PIN\_BLUE\_PILL\_USART2\_CTS (PIN\_BLUE\_PILL\_PA0)

#define PIN\_BLUE\_PILL\_ADC12\_IN0 (PIN\_BLUE\_PILL\_PA0)

#define PIN\_BLUE\_PILL\_TIM2\_CH1\_ETR (PIN\_BLUE\_PILL\_PA1)

#define PIN\_BLUE\_PILL\_USART2\_RTS (PIN\_BLUE\_PILL\_PA1)

#define PIN\_BLUE\_PILL\_TIM2\_CH2 (PIN\_BLUE\_PILL\_PA1)

...

#endif // PINOUT\_BLUE\_PILL\_H

## Loops

Loop is used to execute the block of code several times according to the condition given in the loop.

It means it executes the same code multiple times so it saves code and also helps to traverse the elements of an array.

\* For Loop

\* While Loop

Ex: delay(10000)

# Description about Components

## STM32F4 Microcontroller

The STM32 F4-series is the first group of STM32 microcontrollers based on the ARM Cortex-M4F core. The F4-series is also the first STM32 series to have DSP and floating-point instructions. The F4 is pin-to-pin compatible with the STM32 F2-series and adds higher clock speed, 64 KB CCM static RAM, full-duplex I²S, improved real-time clock, and faster ADCs.

\* Core

\* Memory

\* Peripherals

![image](https://user-images.githubusercontent.com/101356849/168000036-a4acf1be-4eb8-447d-a5d3-76e4d6f15c29.png)

## LEDs

A Light Emitting Diode (LED) is a semiconductor device, which can emit light when an electric current passes through it. To do this, holes from p-type semiconductors recombine with electrons from n-type semiconductors to produce light.

## Push Button or Switch

A Push Button switch is a type of switch which consists of a simple electric mechanism or air switch mechanism to turn something on or off. Depending on model they could operate with momentary or latching action function. The button itself is usually constructed of a strong durable material such as metal or plastic.

![image](https://user-images.githubusercontent.com/101356849/168000228-7dd21e85-3560-4977-b40d-7d1a8e66a200.png)

![image](https://user-images.githubusercontent.com/101356849/168000259-076d8ae7-3d78-42c0-b429-fd04cbe520bf.png)

## Timer

The timer circuit is basically implemented ton introduce delay that is delay(5000).A timer is a specialized type of clock which is used to measure time intervals. A timer that counts from zero upwards for measuring time elapsed is often called a stopwatch.

It is a device that counts down from a specified time interval and used to generate a time delay, for example, an hourglass is a timer.

## Software

QEMU is a machine emulator that can run operating systems and programs for one machine on a different machine. Mostly it is not used as emulator but as Virtualizer in collaboration with KVM kernel components. In that case it utilizes the virtualization technology of the hardware to virtualize guests.

While Qemu has a command line interface and a monitor to interact with running guests those is rarely used that way for other means than development purposes. Libvirt provides an abstraction from specific versions and hypervisors and encapsulates some workarounds and best practices.

**7 Test Plan and Output**

**High Level Test Plan**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ID | Description | Expected Output | Actual Output | Type of test |
| HLTP\_01 | Wiper is moving along the windshield | PASSED | SUCCESS | Scenario |
| HLTP\_02 | Wiper is comes to rest at the end | PASSED | SUCCESS | Boundary |

**Low Level Test Plan**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **ID** | **Description** | **Expected**  **Output** | **Actual**  **Output** | **Type of test** |
| **LLTP\_01** | **Car wiper turned ON** | **PASSED** | **SUCCESS** | Scenario |
| **LLTP\_02** | **Car Wiper Turned OFF** | **PASSED** | **SUCCESS** | Boundary |