MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

MTE 2161: MICROCONTROLLER LAB

THIRD SEMESTER

DEPARTMENT OF MECHATRONICS

Prepared By	Dadi Ravikanth (Lab-in-charge) Dr. Asha C S (Assistant Professor – Sr Scale)	
Approved By	Dr. Chandrashekar Bhat (HoD - Department of Mechatronics)	

MTE 2161: MICROCONTROLLER LAB

THIRD SEMESTER

NAME: _____

REG NO: _____

ROLL NO: _____

DEPARTMENT OF MECHATRONICS

VISION OF MAHE

Global leadership in human development, excellence in education and healthcare.

MISSION OF MAHE

Be the most preferred choice of students, faculty, and industry.
Be in the top 10 in every discipline of education health sciences, engineering, andmanagement.

VISION OF MIT

Excellence in Technical Education through Innovation and Teamwork.

MISSION OF MIT

Educate students professionally to face societal challenges by providing a healthy learning environment grounded well in the principles of engineering, promoting creativity, and nurturingteamwork.

VISION OF THE MECHATRONICS DEPARTMENT

Excellence in Mechatronics Education through Innovation and Teamwork.

MISSION OF THE MECHATRONICS DEPARTMENT

Educate students professionally to face societal challenges by providing a healthy learning environment grounded well in the principles of Mechatronics engineering, promoting creativity, and nurturing teamwork.

NBA PROGRAM EDUCATIONAL OUTCOMES OF THE MECHATRONICS ENGINEERING DEPARTMENT (PEOs)

The graduates**:**

PEO1: Are expected to apply analytical skills and modelling methodologies to recognize, analyze, synthesize, and implement operational solutions to engineering problems, product design and development, and manufacturing.

PEO2: Will be able to work in national and international companies as engineers who can contribute to research and development and solve technical problems by taking an initiative to develop and execute projects and collaborate with others in a team.

PEO3: Shall be capable of pursuing higher education in globally reputed universities by conducting original research in related disciplines or interdisciplinary topics, ultimately contributing to the scientific community with novel research findings.

PEO4: Are envisioned to become technology leaders by starting high – tech companies based on

social demands and national needs.

PEO5: Shall develop flexibility to unlearn and relearn by being in pursuit of research and development, evolving technologies and changing societal needs thus keeping themselves professionally relevant.

NBA PROGRAM OUTCOMES (PO)

The POs are exemplars of the attributes expected of a graduate of an accredited programme:

PO1- Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

PO2- Identify, formulate, research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

PO3- Design solutions for complex engineering problems and design system components or processes that meet t h e specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

PO4- Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

PO5- Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.

PO6- Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues, and the consequent responsibilities relevant to the professional engineering practice.

PO7- Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

PO8- Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

PO9- Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

PO10- Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

PO11- Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

PO12- Recognize the need for and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

NBA PROGRAM SPECIFIC OUTCOMES OF THE MECHATRONICS ENGINEERING DEPARTMENT (PSO'S)

At the end of the course the student will be able to:

PSO1: Able to apply the knowledge of sensors, drives, actuators, controls, robotics and modern software tool to integrate a system to perform specified tasks.

PSO2: Able to design, model, analyze, and testing of intelligent products, systems, and controllers using appropriate technology and software tools.

PSO3: Able to interface devices and elements to a central system having the capability of real time data sharing, storage, retrieval, analysis, decision making with global connectivity features for visibility and intervention.

IET LEARNING OUTCOMES (LO)

C1. Apply knowledge of mathematics, statistics, natural science and engineering principles to the solution of complex problems. Some of the knowledge will be at the forefront of the particular subject of study.

C2. Analyse complex problems to reach substantiated conclusions using first principles of mathematics, statistics, natural science, and engineering principles.

C3. Select and apply appropriate computational and analytical techniques to model complex problems, recognising the limitations of the techniques employed.

C4. Select and evaluate technical literature and other sources of information to address complex problems.

C5. Design solutions for complex problems that meet a combination of societal, user, business and customer needs as appropriate. This will involve consideration of applicable health & safety, diversity, inclusion, cultural, societal, environmental and commercial matters, codes of practice and industry standards.

C6. Apply an integrated or systems approach to the solution of complex problems.

C7. Evaluate the environmental and societal impact of solutions to complex problems and minimise adverse impacts.

C8. Identify and analyse ethical concerns and make reasoned ethical choices informed by professional codes of conduct.

C9. Use a risk management process to identify, evaluate and mitigate risks (the effects of uncertainty) associated with a particular project or activity.

C10. Adopt a holistic and proportionate approach to the mitigation of security risks.
C11. Adopt an inclusive approach to engineering practice and recognise the responsibilities, benefits and importance of supporting equality, diversity and inclusion.

C12. Use practical laboratory and workshop skills to investigate complex problems.

C13. Select and apply appropriate materials, equipment, engineering technologies and processes, recognising their limitations.

C14. Discuss the role of quality management systems and continuous improvement in the context of complex problems.

C15. Apply knowledge of engineering management principles, commercial context, project and change management, and relevant legal matters including intellectual property rights.

C16. Function effectively as an individual, and as a member or leader of a team.

C17. Communicate effectively on complex engineering matters with technical and non-technical audiences.

C18. Plan and record self-learning and development as the foundation for lifelong learning/CPD.

COURSE LEARNING OUTCOMES

CO1: Explain the use of ARM assembly language instructions for an Advance RISC Microcontroller using Keil Micro vision software.

CO2. Write an Embedded C program for interfacing modules of Advance RISC Microcontroller (MSP432P401R) with integration of sensors and actuators.

CO3. Develop an Embedded C code for industry and robotic applications using microcontroller-based system concerning the embedded functional safety, ethics, industry standards.

CO4. Communicate the design solutions to complex engineering problems involving risk analysis & risk assessment for microcontroller-based systems through a case study report.

S. No	Course Outcome	Learning Outcomes	Program Outcomes
1	Explain the use of ARM assembly language instructions for an Advance RISC Microcontroller using Keil Micro vision software.	C1	PO1
2	Write an Embedded C program for interfacing modules of Advance RISC Microcontroller (MSP432P401R) with integration of sensors and actuators.	C6, C13	PO1, PO3
3	Develop an Embedded C code for industry and robotic applications using microcontroller-based system concerning the embedded functional safety, ethics, industry standards.	C4, C5, C6, C7, C13	PO1, PO2, PO3, PO7, PO8
4	Communicate the design solutions to complex engineering problems involving risk analysis & risk assessment for microcontroller-based systems through a case study report.	C6, C7, C12, C13, C16, C17, C18	PO1, PO2, PO3, PO5, PO6, PO7, PO8, PO9, PO10, PO12

DO's and DON'Ts

Electronics:

1. Connect the microcontroller before opening the software.
2. Do not eject or remove the microcontroller while in debugging mode.
3. Do not connect any jumper pins while the power supply is connected to microcontroller.
4. While performing any external connection please call the faculty for cross verificationsbefore dumping the code into the microcontroller.
5. Do not make any loose connections as the microcontroller to be used is a low powerchip.
6. Do not give any external voltage greater than 3.3 Volts.
7. Before dumping the code, please check the clock frequency for simulation in options for target.
8. Dump the code only after you reach zero errors in debugging window.
9. Do not use mobile phones.
10. Eating inside lab is strictly prohibited.

Human Error:

11. Do not place water bottles or wet umbrellas over the desk.
12. Please leave at least 20 cm gap between keyboard and microcontroller.
13. Do not force shut down the PC, in case of the emergency please call the lab technician.
14. Please hold the microcontroller development board at the corner, do not touch themicrocontroller directly with your hand.

DO's and DON'Ts - COVID 19 Rules:

1. Maintain Social Distance.
2. Wear mask throughout the lab.
3. Sanitize your hands before entering the lab and while exiting the lab.
4. Show the ID card and COVID card to the faculty before entering the lab.
5. Sit near to the computer which is marked with yellow sticker.

Safety standards in ARM eco-Systems

Medical applications:

The medical equipment industry has been adopting electronics-based safety technologies with increasingly complex basic designs. Infusion pumps and pacemakers that keep patients alive are increasingly using safety related semiconductor devices at the heart of their designs. As systems become connected — either to share data or for remote operation in 'telehealth' applications — the safety and security elements of the designs become even more important. This is resulting in a wide range of safety standards that have to be considered across the range of medical equipment development, alongside stringent approvals processes from organizations such as the FDA in the US, or the notified bodies operating within the framework of the European Medical Device Directive.

The IEC 60601 medical electrical equipment standard covers equipment such as EEG monitors, IV pumps, imaging systems, ECG devices, vital signs monitor, and other devices that connect directly to a patient. Devices and systems not directly connected to the patient are covered by IEC 61010, including measurement, control and laboratory systems. For medical devices and systems, comprehensive risk management is an integral part of ensuring patient safety. ISO 14971 requires identification of hazards for all operating modes and fault scenarios. The hazard identification process needs to be comprehensive, including electronics and software within the device or system. Analysis of identified hazards is typically done using a risk matrix, which provides a mapping from expected severity of harm and probability of occurrence to the overall risk associated with the hazard. The resulting risk for each hazard is then used to determine required countermeasures, which can be design-based, operational, or documentation related. Software in medical devices and systems is also regulated with two different testing approaches. IEC 60601-1 Annex H treats software as a black box component of the system, with the functionality qualified and tested as part of the overall system. For devices with more complex software IEC 62304 also applies. The standard classifies software in three categories, ranging from the less critical Class A to Class C where a failure could result in death or serious injury. The standard outlines requirements at each stage of the software development lifecycle and defines the minimum activities and tasks that need to be performed to provide confidence that the software has been developed in a way that reduces the risks from potential malfunctions caused by software errors to a tolerable level. Medical devices also often utilize legacy software, often referred to as software of unknown pedigree (SOUP) and rely on third party software and drivers for function such as internet connectivity, USB communication, or file management. These software elements also have to be tested and integrated as part of an IEC 62304 compliant process which can be complex and time consuming.

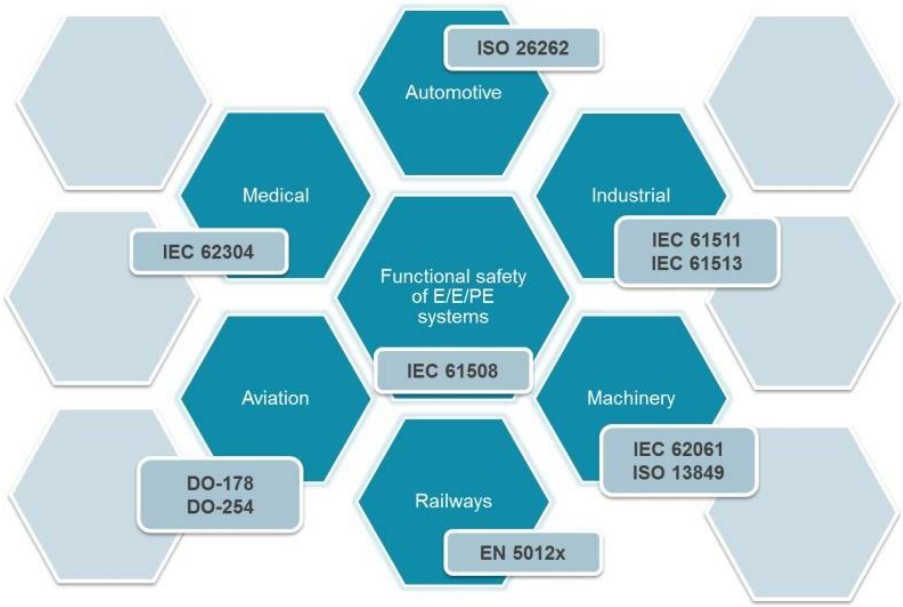
Industrial applications:

Areas of industrial equipment design have had to deal with safety concerns for many years. Process equipment controlling critical operations that can fail, causing explosions, fires, environmental disasters and even fatalities have to be designed with safety and security in mind. One of the ways to address these risks is to use a risk based approach, which drove the development of the

functional safety standards such as IEC 61508 and IEC 61511. However, as more and more industrial equipment becomes connected to networks and to the Internet, there is more focus on the combined safety and security of these systems. Water pumps or electrical systems failing, either by accident or by malicious attack, can cause widespread disruption that is both costly and potentially disastrous. Ensuring that industrial systems are safe and secure is an increasingly vital consideration. Fortunately, the safety standards that have evolved over the last couple of decades have taken a broad approach to safety and so are equally applicable in today's networked world.

Automotive applications:

Automotive design has been the driving force for safety in integrated silicon devices over the last20 years. Standards such as ISO 26262 and AUTOSAR have been developed to ensure that systems, software and hardware can be developed and operated in a way which allows risks relatedto malfunctions to be mitigated and controlled. Progress in automotive driver assistance capabilities and onward to semi-autonomous driving, whether by controlling brakes, steering or other safety-related parts of the car, is limited by the ability to demonstrate the safety of the electronics systems. Vehicle manufacturers and their suppliers are focusing their efforts to develop and validate these systems such that appropriate safety, quality, and reliability attributes can be demonstrated. Recalling a range of vehicles can be incredibly costly and severely damaging to thebrand of the car maker. This is driving an understanding that component cost is less important than ensuring the safety systems are correctly architected and developed, giving semiconductor vendors the opportunity to innovate. This experience and innovation can then be transferred to other applications in medical and industrial markets



Main functional safety and related standards

IEC 61508 – Functional safety for electrical, electronic, and programmable electronic devices:

IEC 61508 is designated as a basic safety publication in the IEC standardization framework, meaning that it defines generic requirements for functional safety across applications and industries. The standard was originally published in 2000, with the current second edition being from 2010. The standard covers the complete safety life cycle through 16 phases in three groups covering the analysis, implementation and operation of a system. At the heart of the standard are the concepts of risk and safety function. Risk is defined as a function of the likelihood of a hazardous event and the severity of its consequences. The risk associated with the hazardous eventis then reduced to a tolerable level by applying safety functions either in hardware and software orin other technologies, although only the electronic, electrical, and programmable electronic elements are covered by the standard. The risk management therefore relies on the functionality of such safety functions, hence 'functional safety'. While IEC 61508 accepts that zero risk can never be reached, intolerable risks must be reduced. To do this, systems and operations must be designed for safety from the beginning. Engineering safety as an afterthought generally results in less effective and more complex solutions with an increased cost, compared to systems where appropriate safety engineering principles have been considered from the beginning.

Importance of hardware support for functional safety:

The development of safety systems requires consideration of both hardware and software. Recent processors have additional features to assist in the implementation of safety-related designs. The ARMv7-R architecture includes features in the architecture such as user and privileged software operating modes with a Memory Protection Unit (MPU) and advanced error management, which help safety-related processor designers. In addition, implementations of the ARMv7-R architecture support lock-step processor configuration, which allows a high diagnostic coverage to be achieved for random hardware faults. These features were a key part of the original ARM® Cortex® -R processor family, starting with the Cortex-R4 in 2005 for automotive systems, industrial control, wireless baseband, hard disk drive controllers and other real-time applications. ARM's Cortex-R processors implementing the ARMv7-R architecture support high clock frequencies with a deeplypipelined micro-architecture and hardware SIMD instructions for very high-performance DSP functions. The processors were designed for fast, bounded, and deterministic interrupt response with Tightly Coupled Memories (TCM) local to each core for fast-responding code and data and a Low Latency Interrupt Mode (LLIM) to accelerate interrupt entry. Bounded and deterministic response times are essential for most safety-related applications, as these typically also have strict real-time requirements. Following on from the success of ARMv7-R architecture for real-time safety applications, the ARMv8-R architecture was announced in 2013. The ARMv8-R architecture profile is still targeted for 32-bit embedded applications, especially in automotive andindustrial applications, while bringing a number of improvements over the ARMv7-R architecture. The most important addition in the ARMv8-R architecture is an enhanced hardware-assisted hypervisor mode which provides an additional privilege level within the processor hardware. Thisprivilege level determines what the virtualized software can and cannot do, with basic user tasks having the lowest level of privilege, zero or PL0. An operating system typically runs with higher

privilege at PL1, managing the interrupt controller and configuring the attributes for regions of memory and peripheral address map to police user task accesses. Such policing is performed in a processor's hardware by the MPU.

This supports the development of safety applications and the separation of operating systems and tasks. In addition it provides a separate mode for virtualization and the handling of critical safety and security events in the system. Real-time critical interrupts can be handled directly by the hypervisor, saving the overhead of switching to a guest OS.

The safety ecosystem:

Designers of safety-related systems are faced with several challenges throughout the development process. In addition to the underlying hardware design, the designers need to consider the safety-related software that is being developed, and any supporting software tools that are required for the development of the system. Therefore, the decision to use processor architecture for safety-related designs needs to be done with the overall safety requirements in mind. ARM has been working with several partners to ensure that there is a robust ecosystem to support safety related development activities on designs based on the ARM architecture. This includes work with semiconductor vendors, compiler and support tool vendors, and design consultancies.

Several ARM partners have developed microcontroller products targeted for safety-related markets. Typical products are currently based on the ARMv7-R architecture and include either the ARM Cortex-R4 or ARM Cortex-R5 real-time processor. These products are available from Texas Instruments with its Hercules range of multicore safety controllers, as well as Toshiba, Spansion, ScaleoChip, Renesas, STMicroelectronics, Infineon, Broadcom, LSI, Fujitsu and many others. Software development tools are a vital element in the ecosystem as without the appropriate development tools developing applications for safety-related systems is challenging. When developing software for safety-related applications, the developers need to ensure that the compiler, for example, does not introduce errors into the generated object code which will be executed on the safety-related system. Safety standards have different ways to describe requirements for software tool qualification. ISO 26262, for example, contains requirements for establishing a level of confidence for the use of software tools. The process of establishing sufficient confidence in the tools, such as compilers, typically requires additional software tool qualification activities to be conducted by the system developers. Compiler and analysis tool vendors supporting the ARM architecture have adopted a diverse range of approaches to facilitate such compiler qualification activities. To address the requirements of tool qualification for various safety-related markets, ARM has developed the ARM Compiler Qualification Kit, which contains essential additional information about the ARM C/C++ compiler that can be used as part of tool qualification activities. The ARM Compiler Qualification Kit includes a Safety Manual for the compiler, describing the details of the compilation flow, together with information about possible failure modes of the compiler tool chain. Information about constraints of use is also included, with recommendations on applicable configuration options. Further, the ARM Compiler Qualification Kit includes detailed Defect Report, Test Report, and Development Process documents. The ARM Compiler toolchain is further certified by TÜV SÜD, a recognized safety industry expert. The TÜV

Certificate and the accompanying report confirm that the ARM Compiler version 5.04 fulfils the requirements for development tools classified T3 according to IEC 61508-3. This enables customers to use the ARM Compiler 5.04 for safety-related development up to SIL 3 (IEC 61508) or ASIL D (ISO 26262) without further qualification activities when following the recommendations and conditions documented in the Qualification Kit. The ARM ecosystem for compilers is diverse, as in addition to C and C++ compilers from vendors such as Green Hills Software and IAR, AdaCore has an Ada compiler targeting the ARM architecture. Ada is a lesser-known language that is well-suited for applications having stringent requirements for safety and security, such as avionics, industrial control, and railway control, as it has support for strong typing, concurrency, low-level programming, and mechanisms to support development of large-scale programs. In addition to compilers, safety-related software development typically requires the use of debugging, testing and analysis tools. Examples include static analysis tools for demonstrating language subset conformance, such as MISRA C which is widely used within the automotive industry. Conformance to programming language subsets is a requirement in multiple functional safety standards, including IEC 61508 and ISO 26262.

Other program analysis tools that can be used to analyze worst case execution times (WCET) andmaximum stack space usage of programs are also available for the ARM architecture. ARM is alsoworking closely with software vendors such as Green Hills Software, Mentor Graphics, eSOL, SYSGO and ETAS to develop various hypervisor implementations for the ARMv8-R architecture. For automotive safety system developers, these hardware and software solutions will permit consolidation of electronic control units via virtualization to save manufacturing cost and allow re-use of software between projects. Each software system can run in its own virtual machine with the hypervisor managing processing resources by time slicing, event triggering, or even scheduling between multi-processor cores.

References regulatory:

- ISO 14971:2007. Medical devices. Application of risk management to medical devices.
- ISO/IEC 15408 (series). Information technology – Security techniques – Evaluation criteria for IT security.
- ISO 26262:2011-2012. Road vehicles - Functional safety.
- IEC 60601 (series). Medical electrical equipment.
- IEC 61010 (series): Safety requirements for electrical equipment for measurement, control, and laboratory use.
- IEC 61508:2010. Functional safety of electrical / electronic / programmable electronic safety-related systems.
- IEC 61511:2003. Functional safety – Safety instrumented systems for the process industry sector.
- IEC 62304:2006. Medical device software. Software lifecycle processes.

List of Experiments

S.No	Experiments	Learning Outcome	Program Outcomes
1	Introduction to MSP432P401R using simulation software Keil Micro vision	C1	PO1
2	Introduction to Instruction Sets in ARM Cortex M4	C1	PO1
3	Introduction to Embedded C and Developing Algorithms for GPIO & NVIC	C6, C13	PO1, PO2
4	Developing Algorithms for Timers and Counters (SysTick and Timer 32).	C6, C13	PO1, PO2
5	Use of Timer A and Generation of PWM signals using Timer A.	C6, C13	PO1, PO2
6	Performing Serial Communication using UART.	C6, C13	PO1, PO2
7	Case Study 1: Developing algorithms for autonomous transportation system and address the societal functional safety.	C4, C5, C6, C7, C13	PO3, PO2, PO6, PO7, PO8
8	Case Study 2: Developing algorithms for industrial automation and address the industrial functional safety.	C4, C5, C6, C7, C13	PO3, PO2, PO6, PO7, PO8
9	Case Study 3: Developing algorithms for collecting data for medical applications and address the medical functional safety.	C4, C5, C6, C7, C13	PO3, PO2, PO6, PO7, PO8
10	Design a case study file with algorithm for given application with the inclusion of address risk analysis, and risk assessment.	C6, C7, C12, C13, C16, C17, C18	PO1, PO2, PO5 PO6, PO7, PO9, PO10, PO12

EXPERIMENT NO.1: INTRODUCTION TO MSP432P401R USING SIMULATION SOFTWARE KEIL MICROVISION

AIM: To be familiarized with ARM Cortex M4, MSP432P401R simulation software and basic instruction set.

INTRODUCTION:

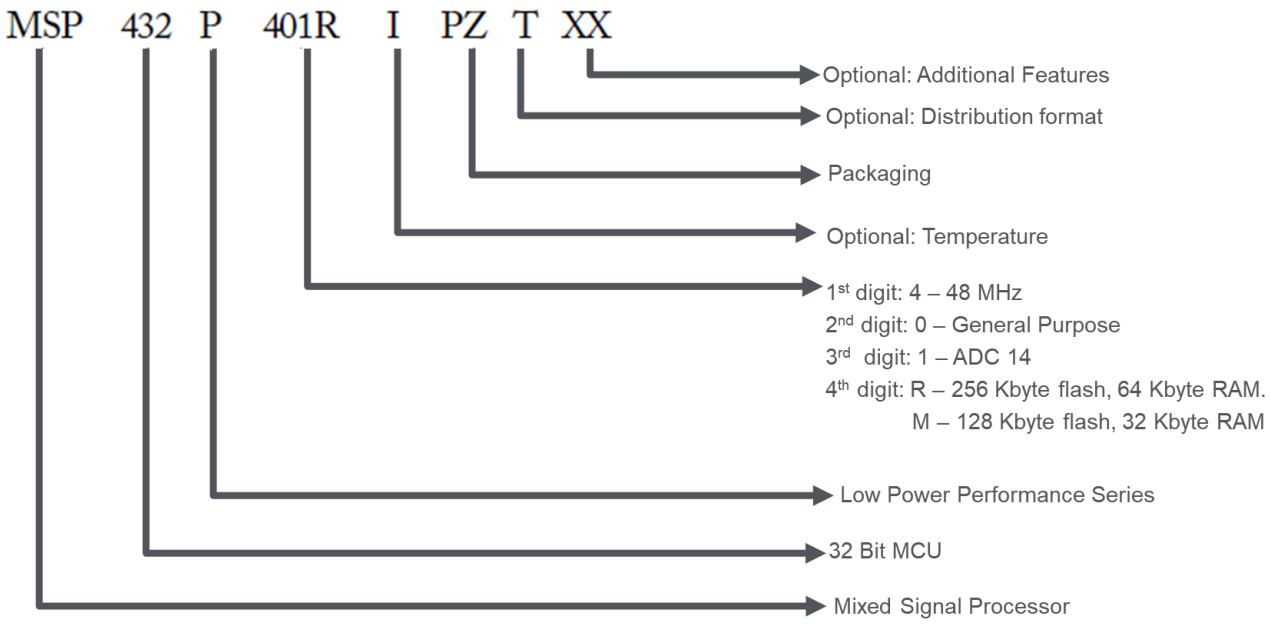
The MSP432P401R microcontroller have more than 10 development platforms and they are provided by different vendors. However, the following platforms and tools are relatively popular:

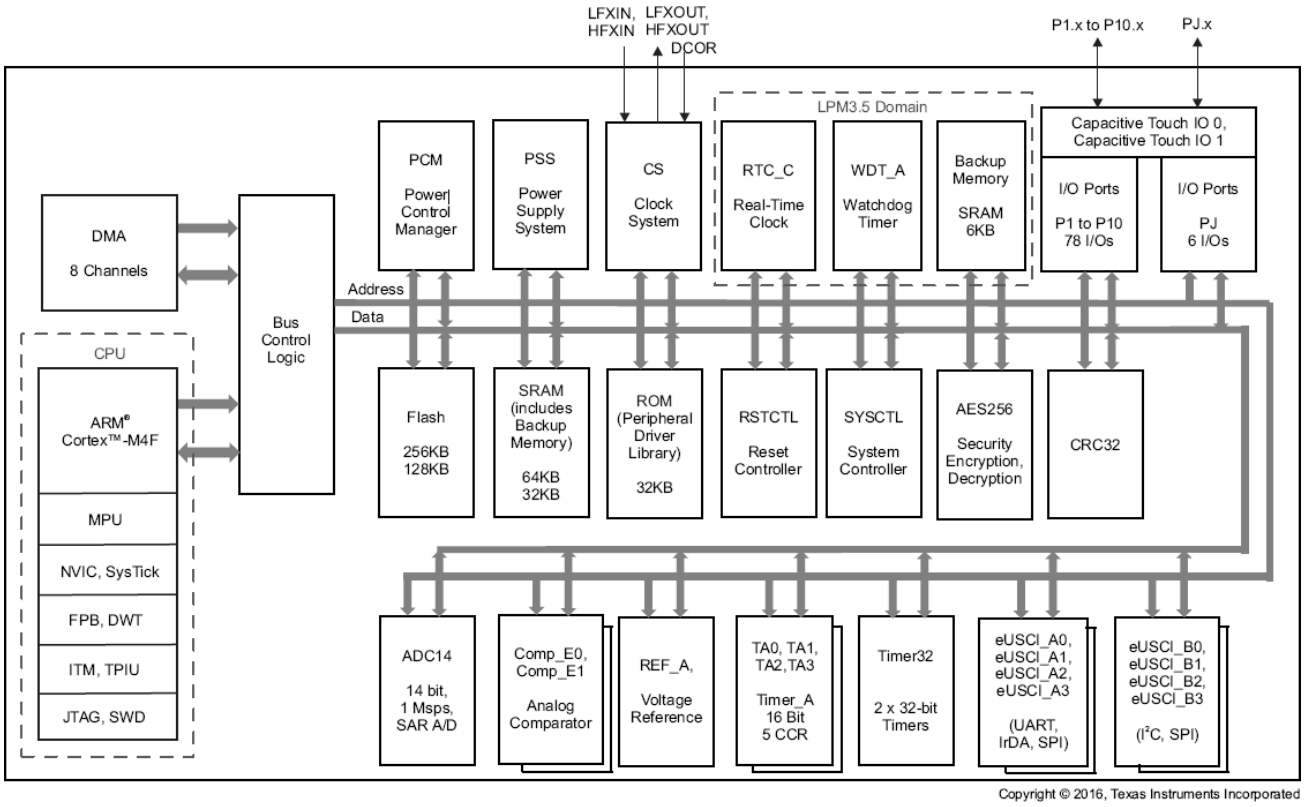
1. Keil MDK-ARM Microcontroller Development Kit (MDK) IDE.
2. Texas Instruments' Code Composer Studio™ (CCS) IDE.
3. IAR Embedded Workbench for ARM.
4. Mentor Graphics Source Code Bench.
5. GNU Compiler Collection (GCC).

Keil MDK-ARM and Code Composer Studio are the two most widely used in academia for student's exposure over architecture-based programming.

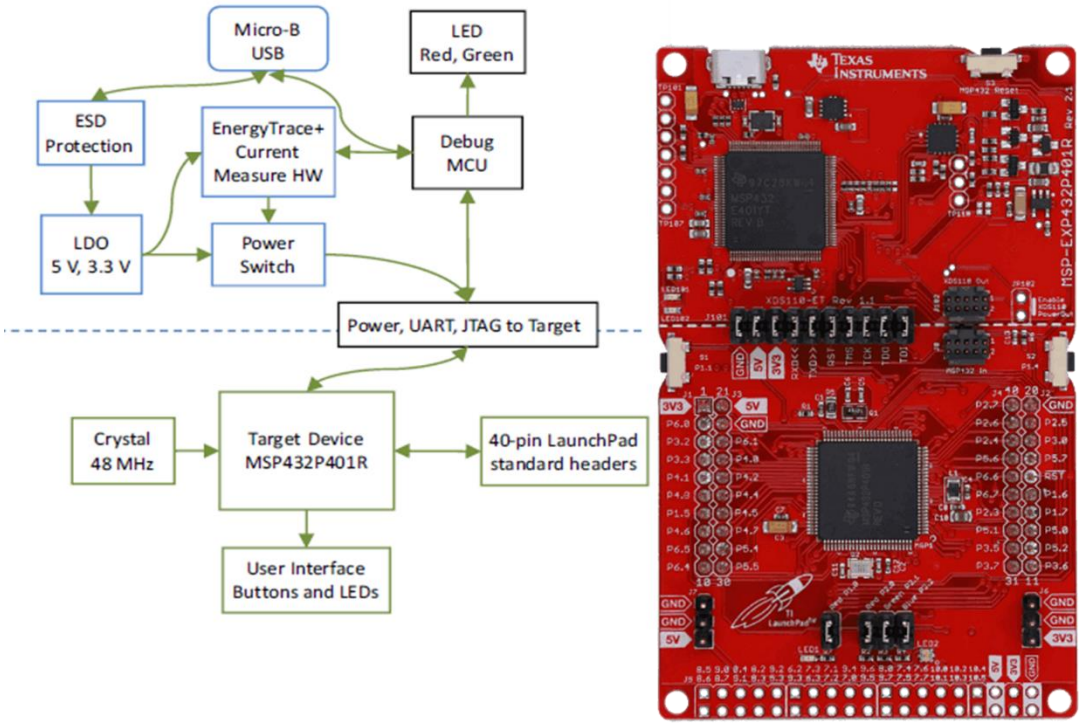
The ARM MSP432P401R data sheet is the requirement for programming the internal/external modes.

Nomenclature of MSP432P401R:





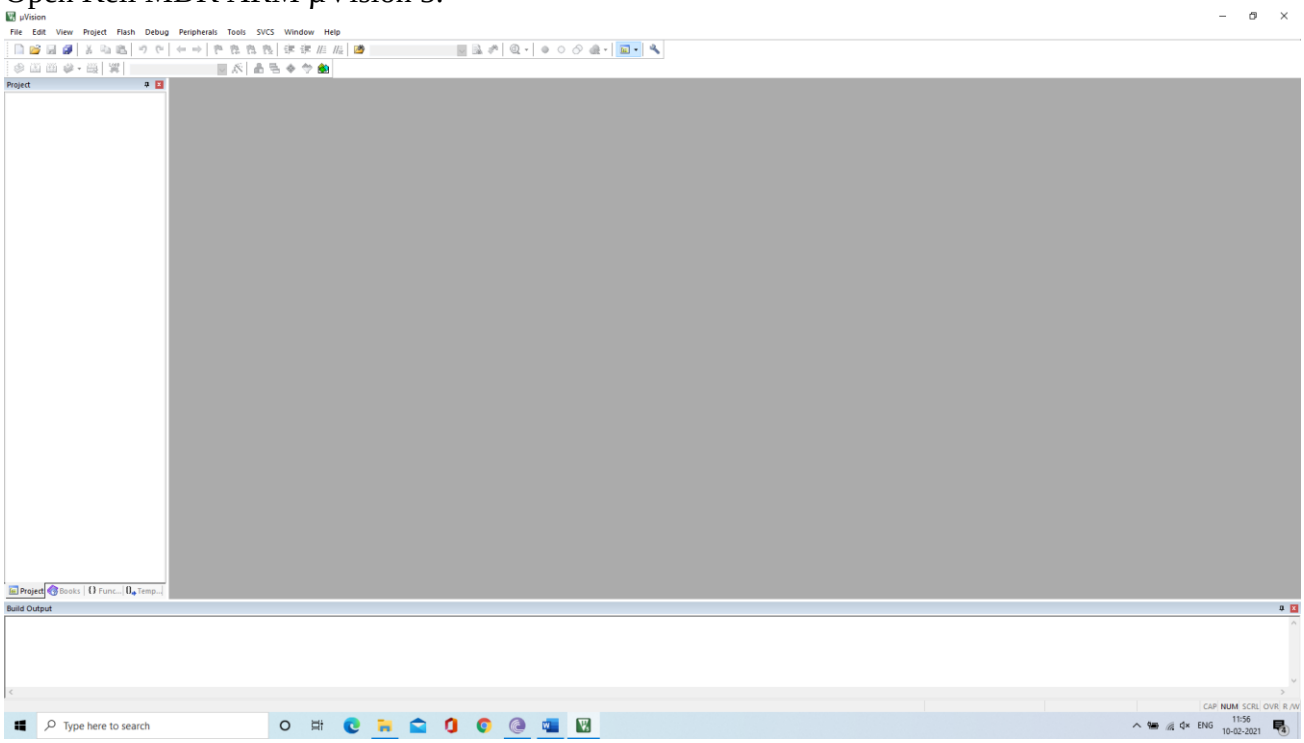
MSP432P401R Architecture



MSP432P401R Launchpad schematic

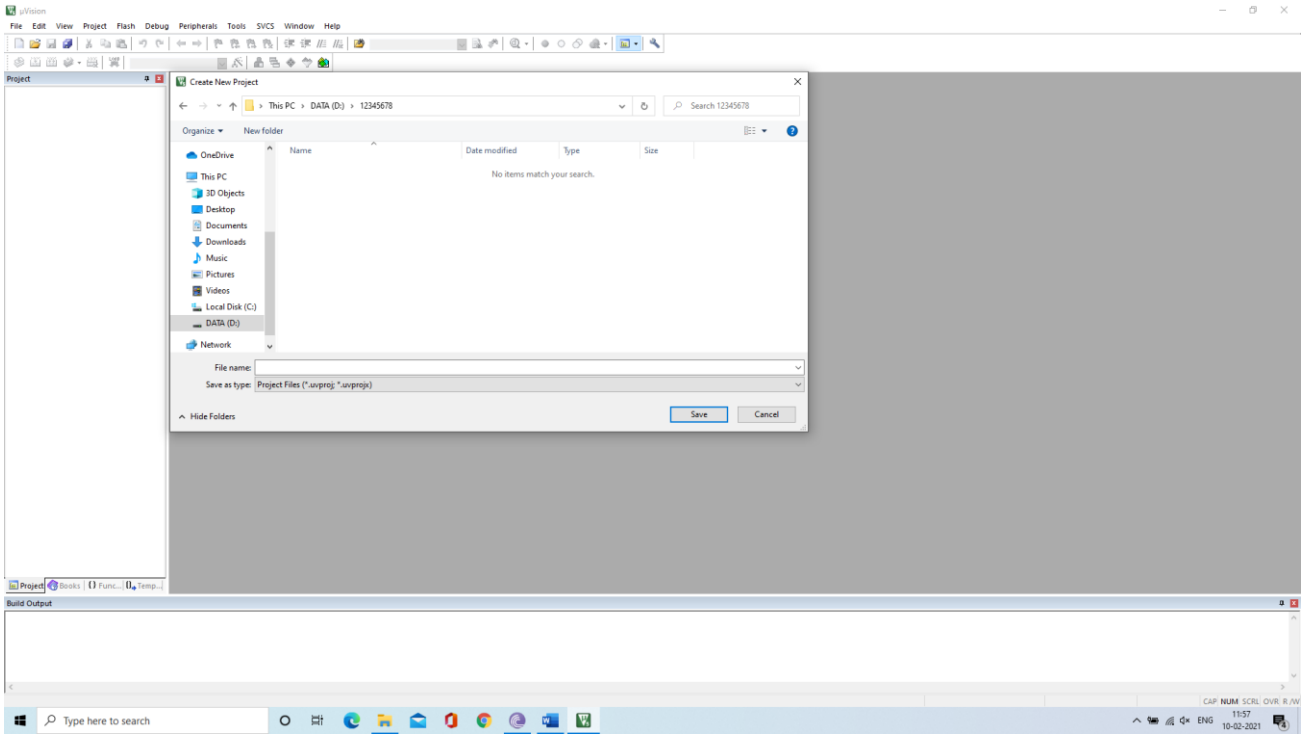
STEPS OF INSTRUCTIONS:

1. Open Keil MDK ARM µVision 5.



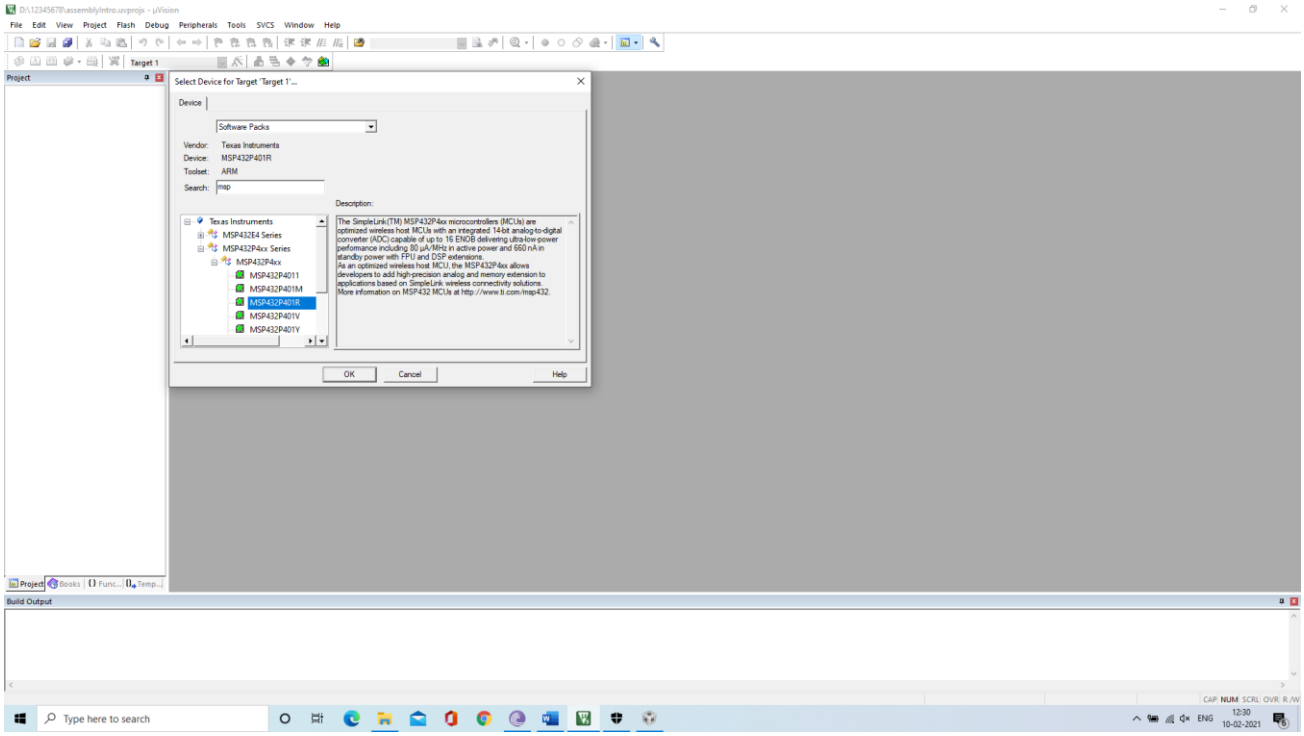
2. Create a Folder with your Registration Number.

3. Click on the new µVision Project.

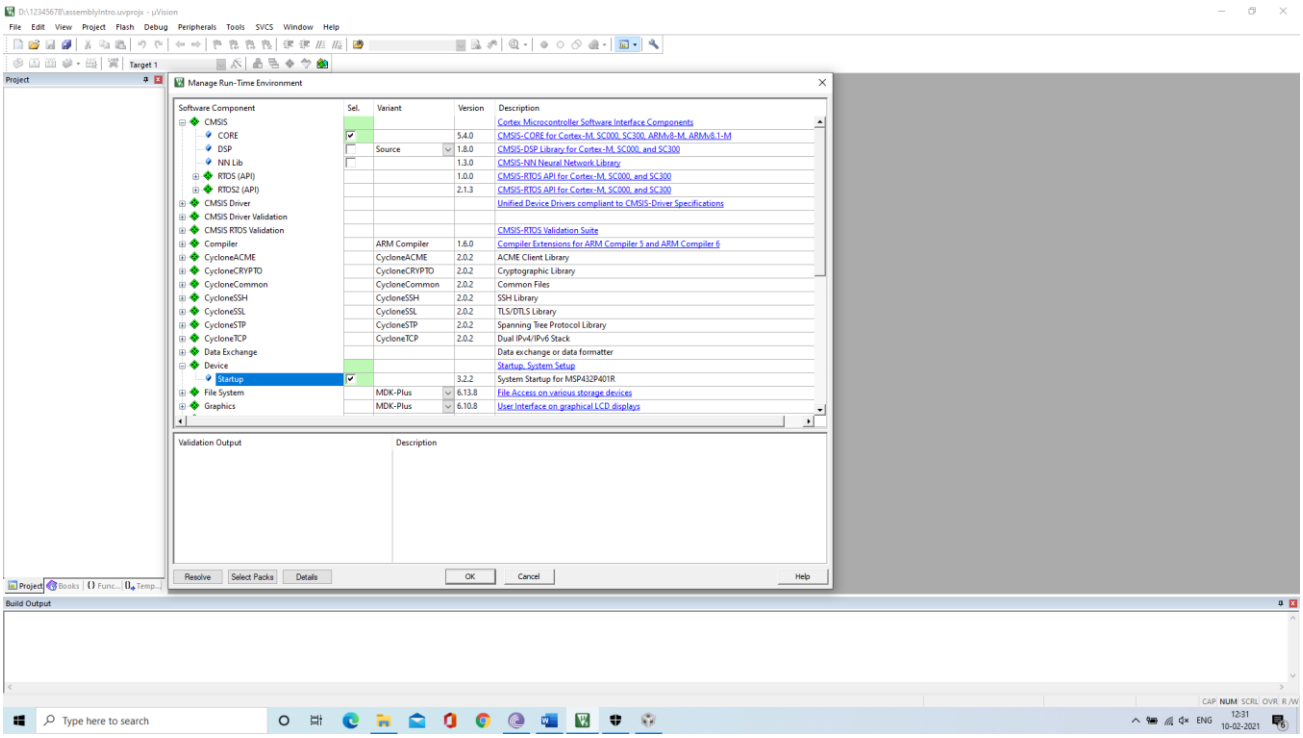


4. Enter the file name (Ex: assemblyIntro)→Click Save
5. It opens a new selection window and type msp in the search space. Select MSP432P401R under Texas

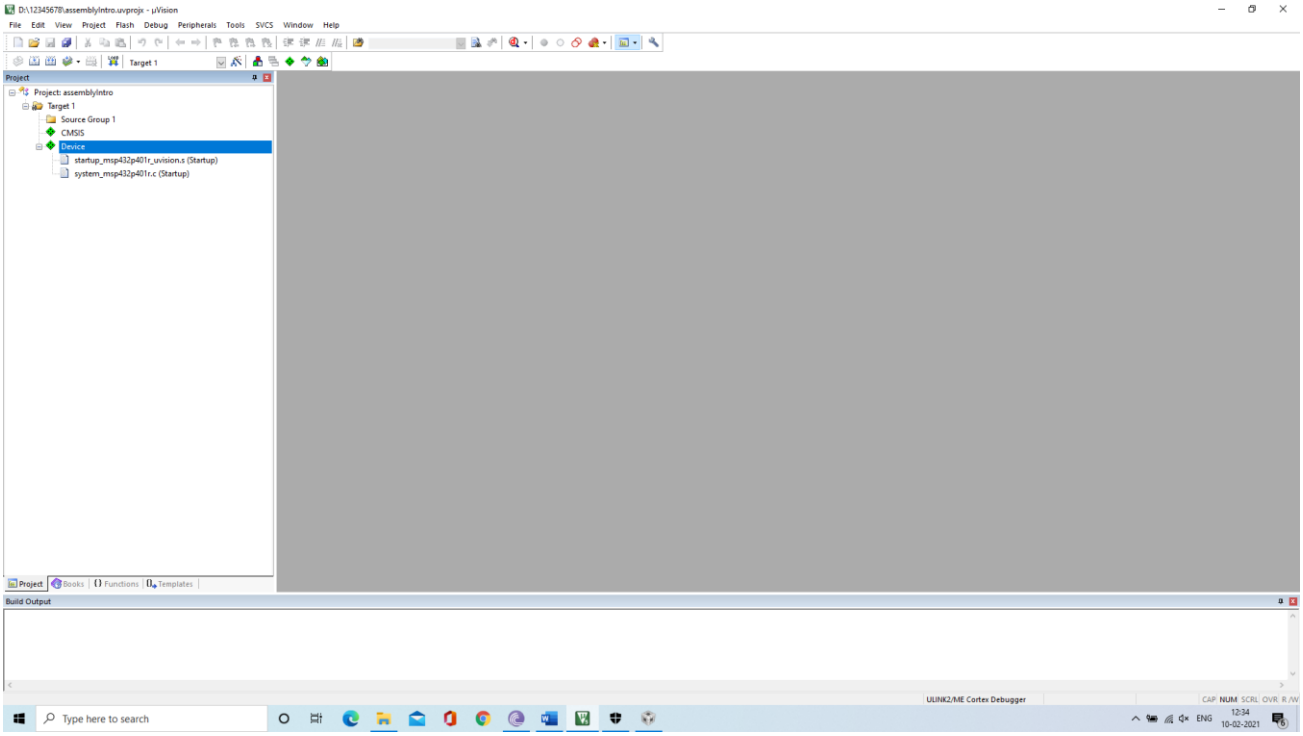
Instruments.



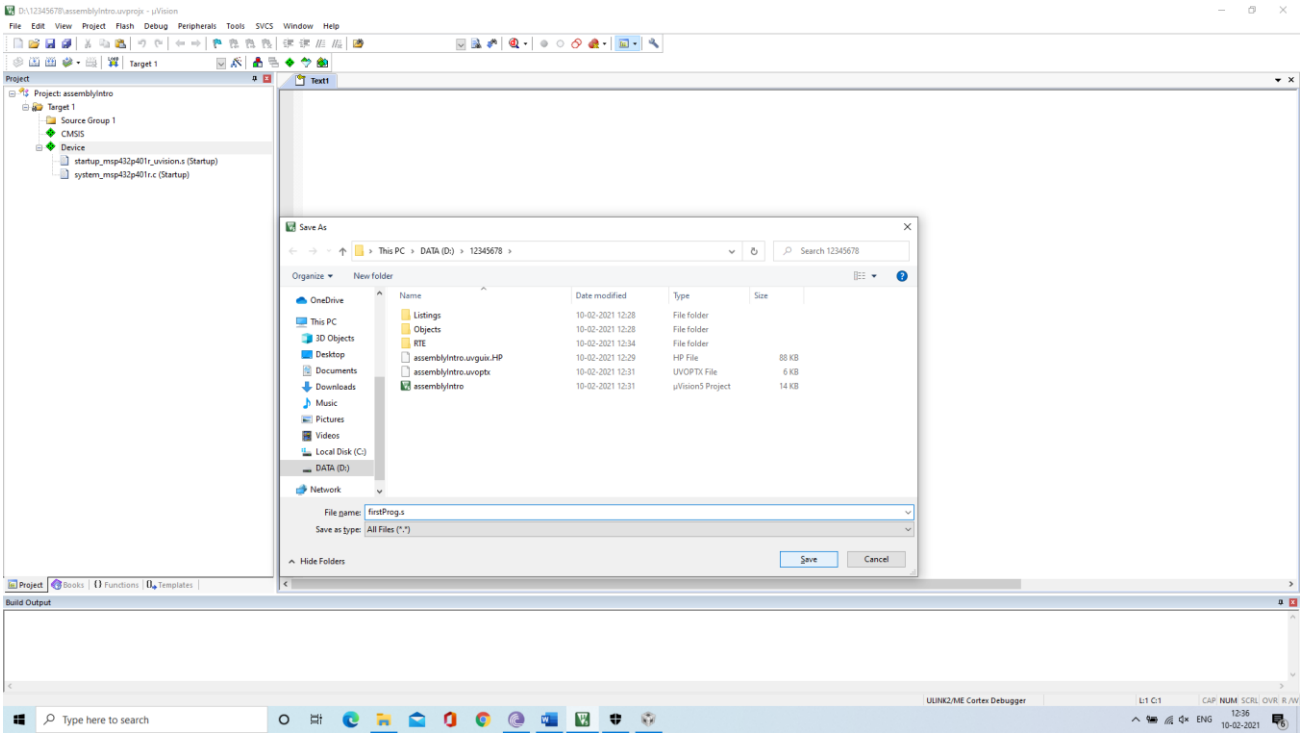
6. Select the Core from CMSIS and check right side description box.



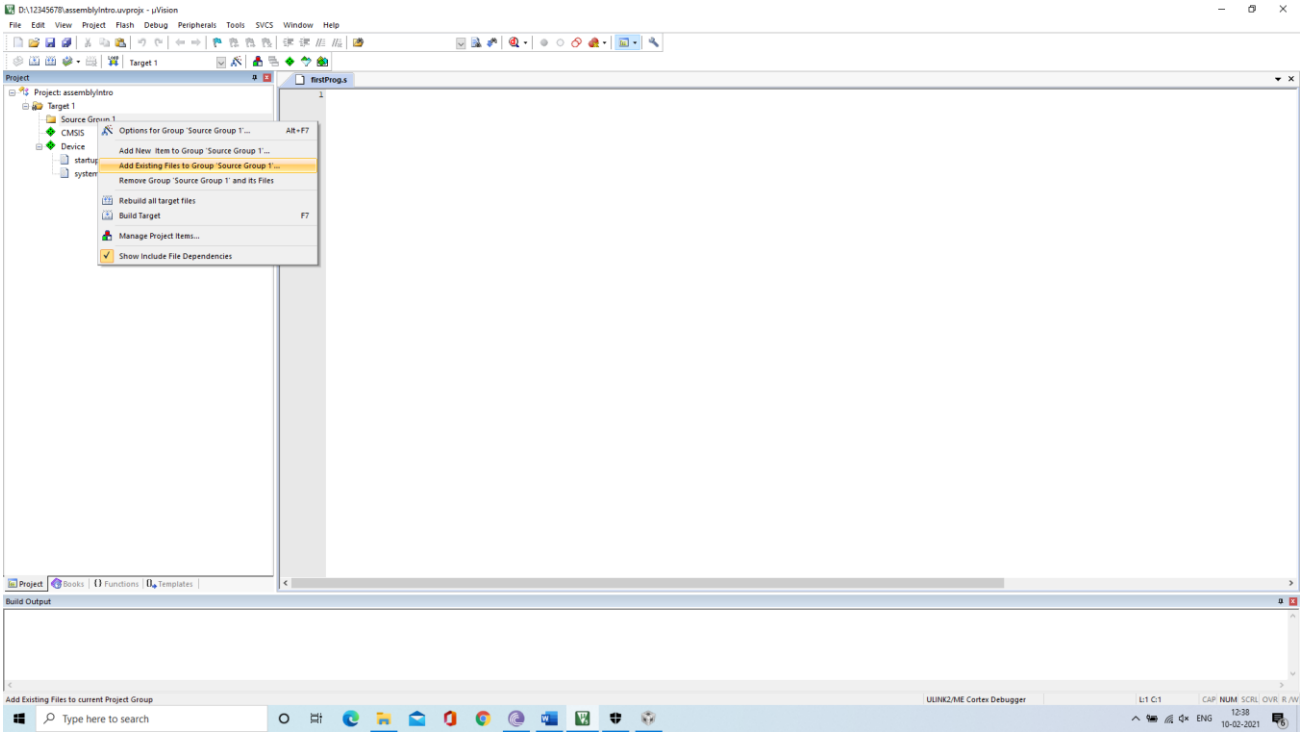
7. Select Startup from Device and check the right side description box.
8. Click OK.



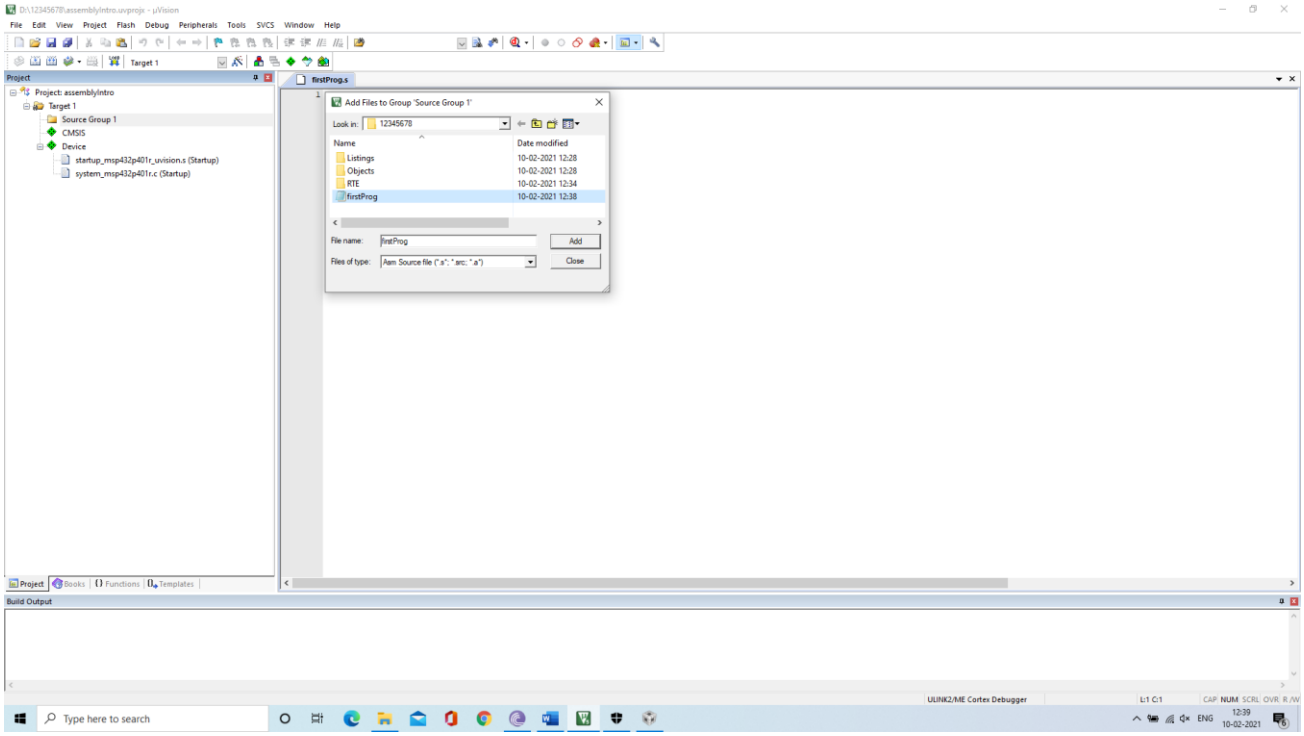
9. File→New



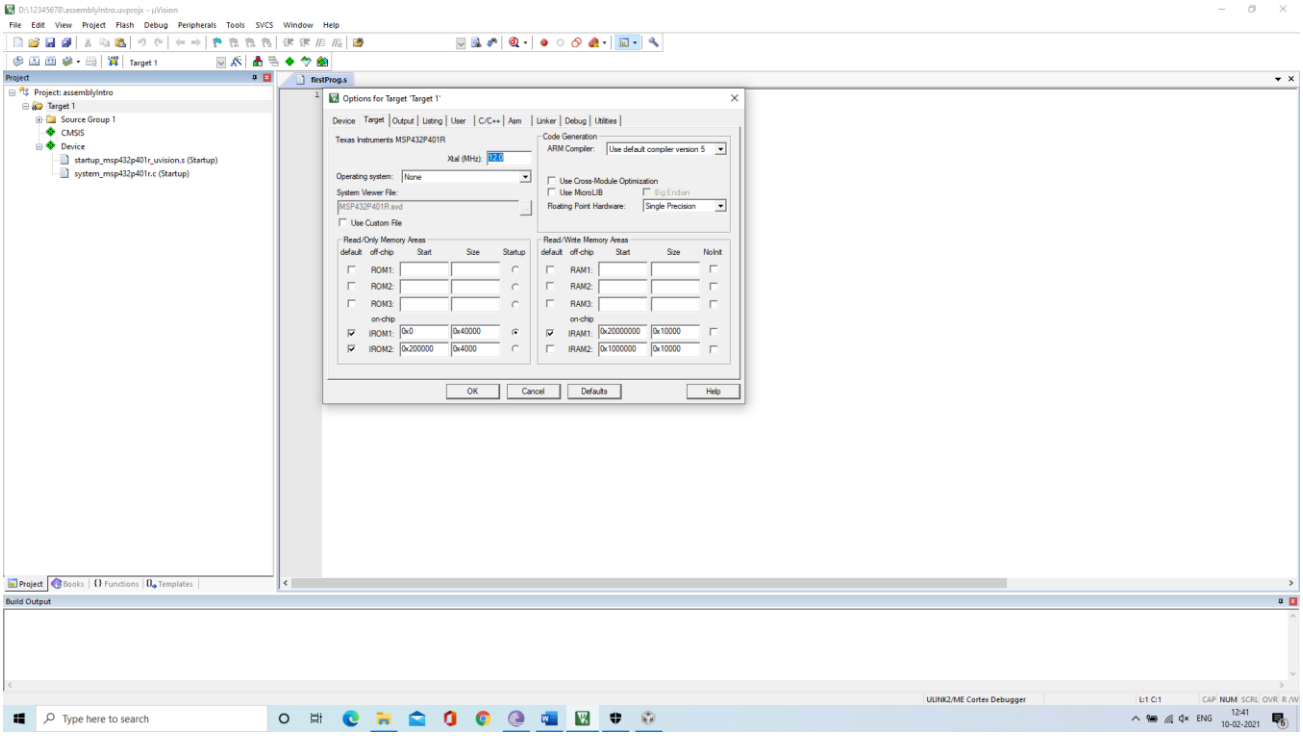
10. Save the file as filename with .s extension (For ex: firstProg.s), Click Save.
11. Right click on Source Group 1 and click on add existing files.

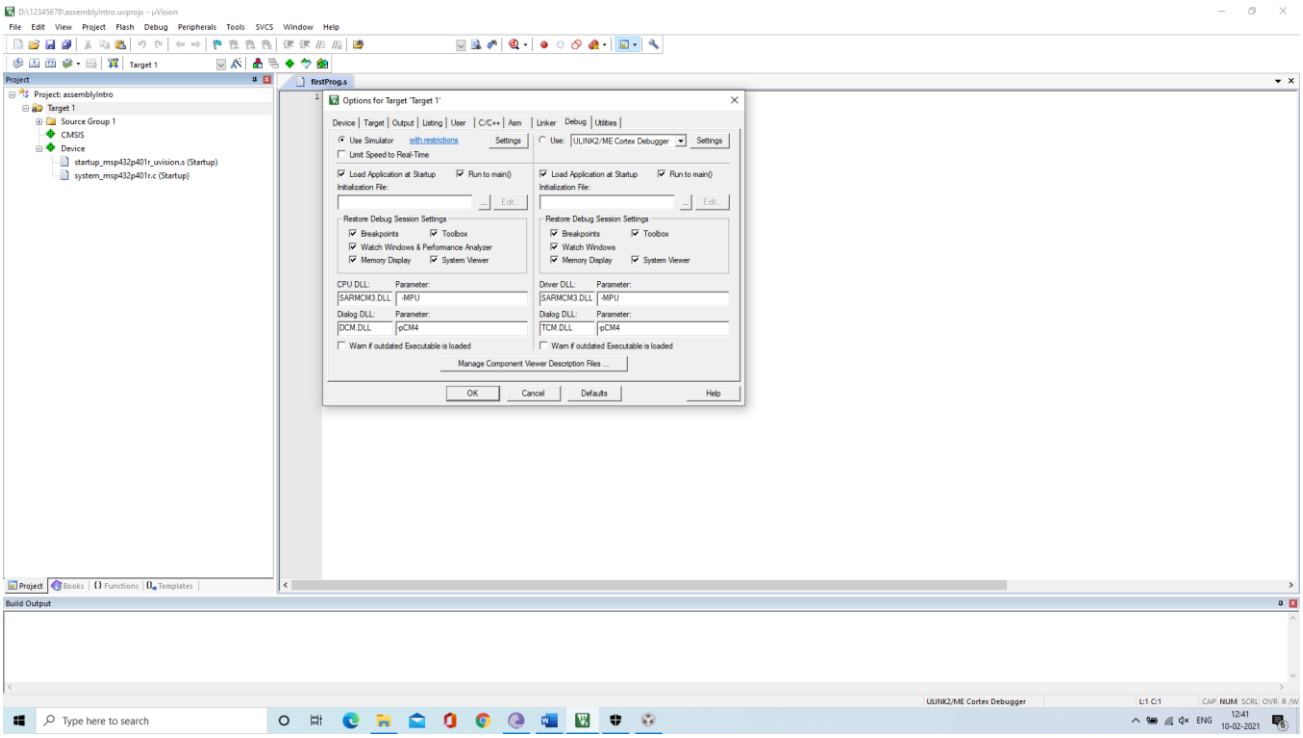


12. Select the .s file saved earlier, click Add, and Close button.



13. Right Click on Target 1 and select options for target, Select compiler version 5 in Target and use simulator in Debug option.

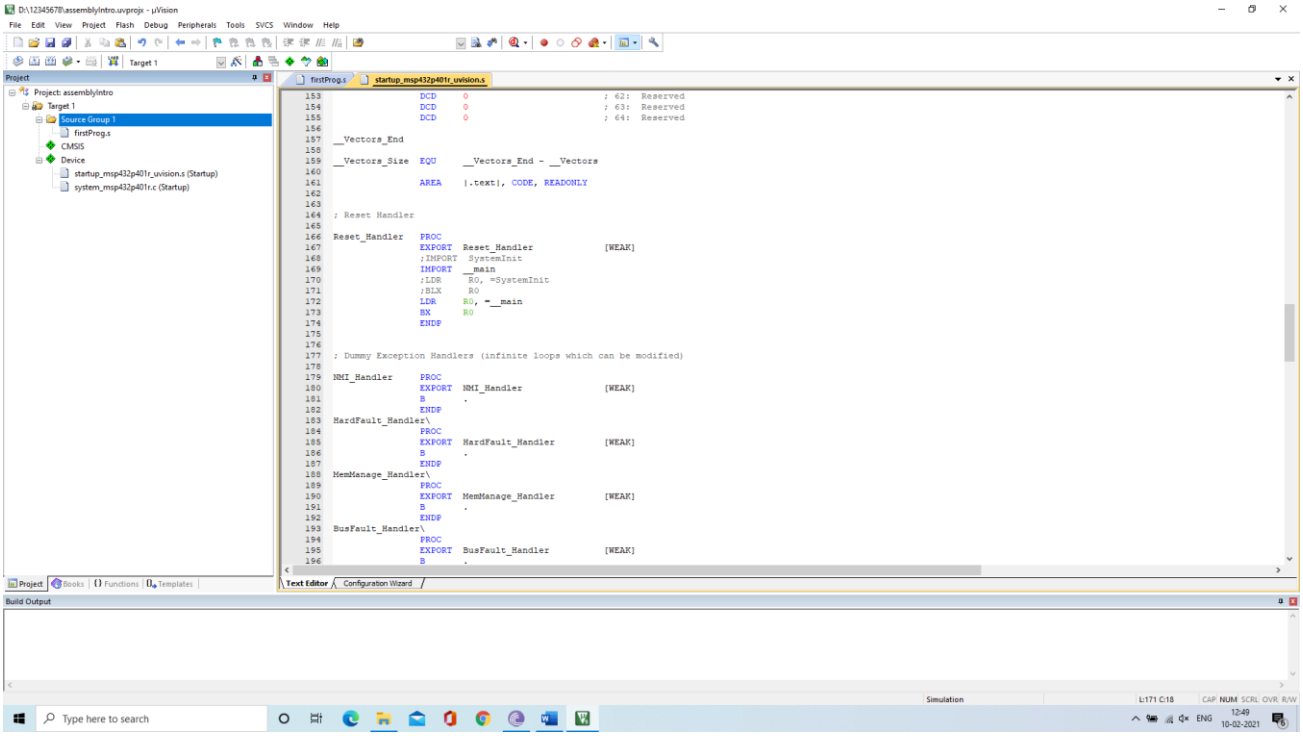

Options for Target 'Target 1'
Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities
Texas Instruments MSP432P401R
Xtal (MHz): 12.0
Code Generation
ARM Compiler: Use default compiler version 5
Operating system: None
System Viewer File:
MSP432P401R.svd
Use Custom File
Use Cross-Module Optimization
Use MicroLIB
Big Endian
Floating Point Hardware: Single Precision
Read/Only Memory Areas
default off-chip Start Size Startup
ROM1:
ROM2:
ROM3:
on-chip
IROM1: 0x0 0x40000
IROM2: 0x200000 0x4000
Read/Write Memory Areas
default off-chip Start Size NoInit
RAM1:
RAM2:
RAM3:
on-chip
IRAM1: 0x20000000 0x10000
IRAM2: 0x1000000 0x10000
OK
Cancel
Defaults
Help


Options for Target 'Target 1'
Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities
Use Simulator with restrictions
Settings
Use: ULINK2/ME Cortex Debugger
Limit Speed to Real-Time
Load Application at Startup
Run to main()
Initialization File:
Restore Debug Session Settings
Breakpoints
Toolbox
Watch Windows & Performance Analyzer
Watch Windows
Memory Display
System Viewer
CPU DLL:
Parameter:
SARMCM3.DLL
-MPU
Dialog DLL:
DCM.DLL
pCM4
Driver DLL:
TCM.DLL
Warn if outdated Executable is loaded
Manage Component Viewer Description Files ...
OK
Cancel
Defaults
Help

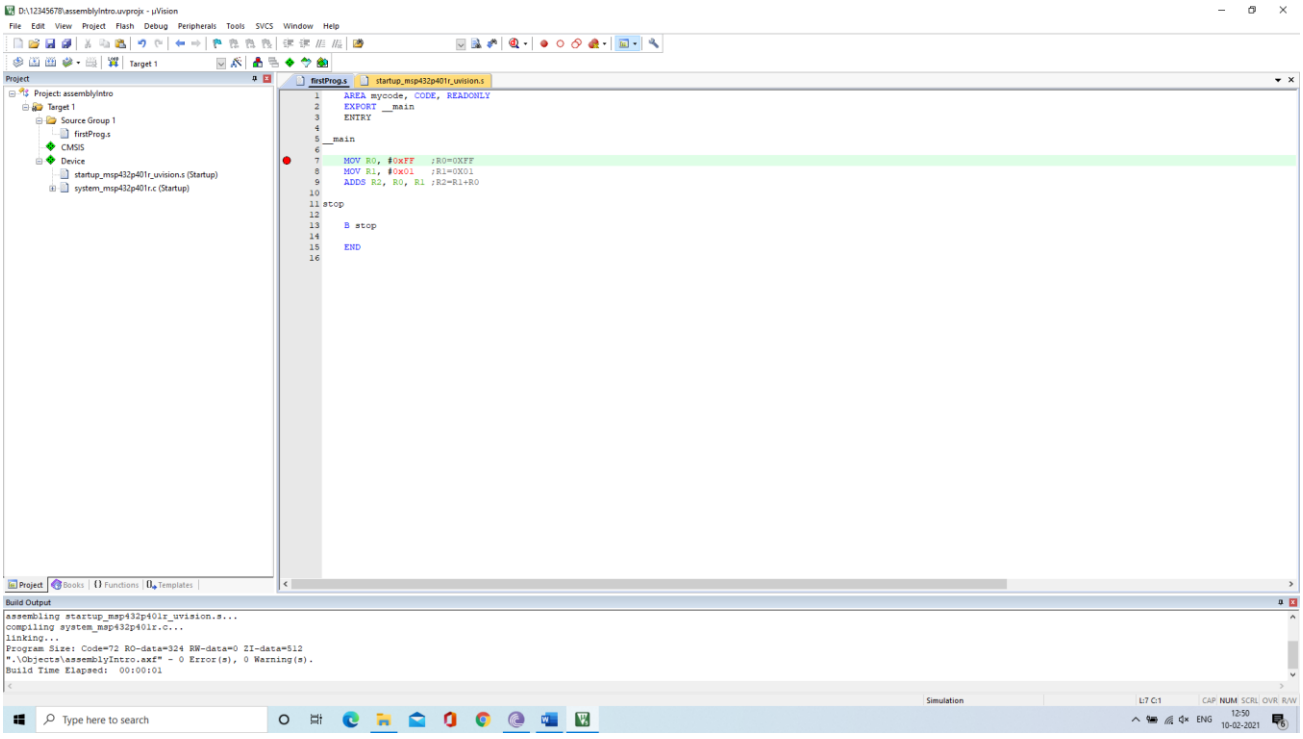
14. Write Code (As shown below)

```
;Author Name
;Date when created
;Name of the program (Aim)
;Equipment used for the experiments and connection procedure

AREA mycode, CODE, READONLY
EXPORT __main
ENTRY

__main

MOV R0, #0xFF          ;R0=0XFF
MOV R1, #0x01          ;R1=0X01
ADDS R2, R0, R1        ;R2=R1+R0

stop

B stop

END
```

15. Open startup.s file and comment three lines as shown. (optional)

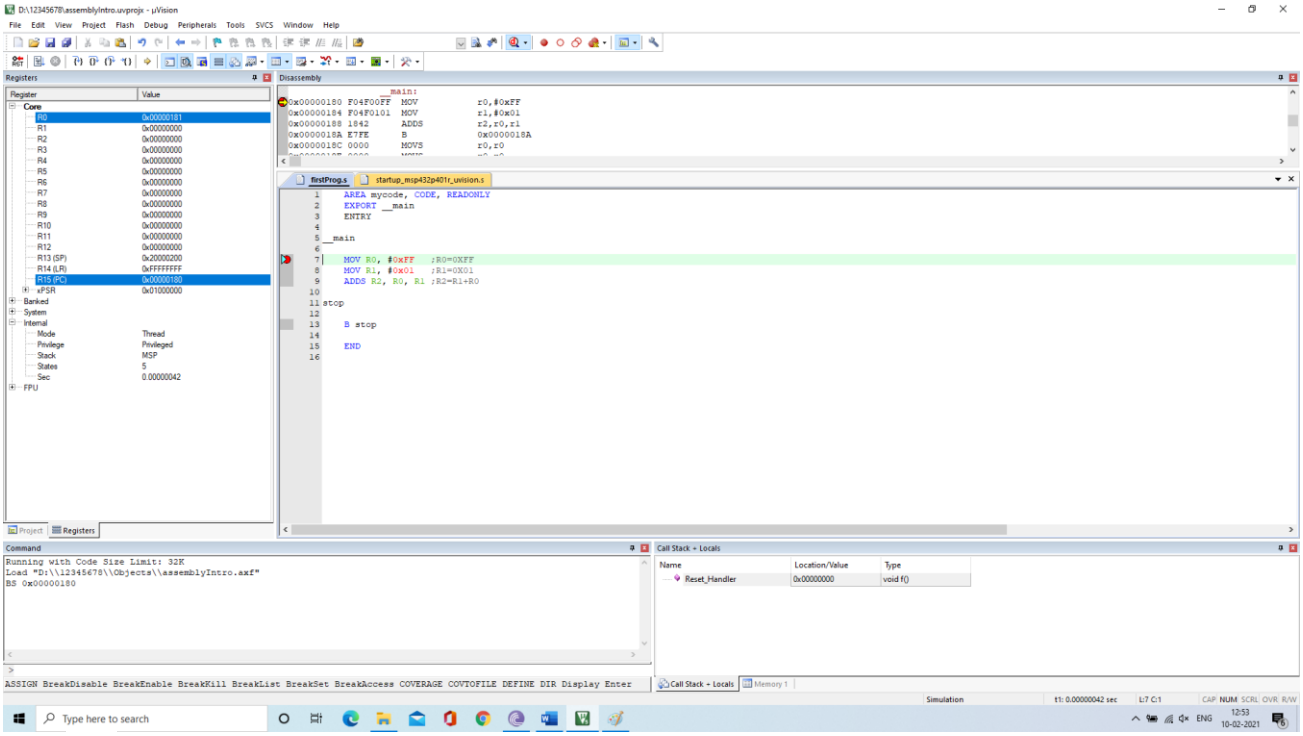


16. Click on  to build the program.

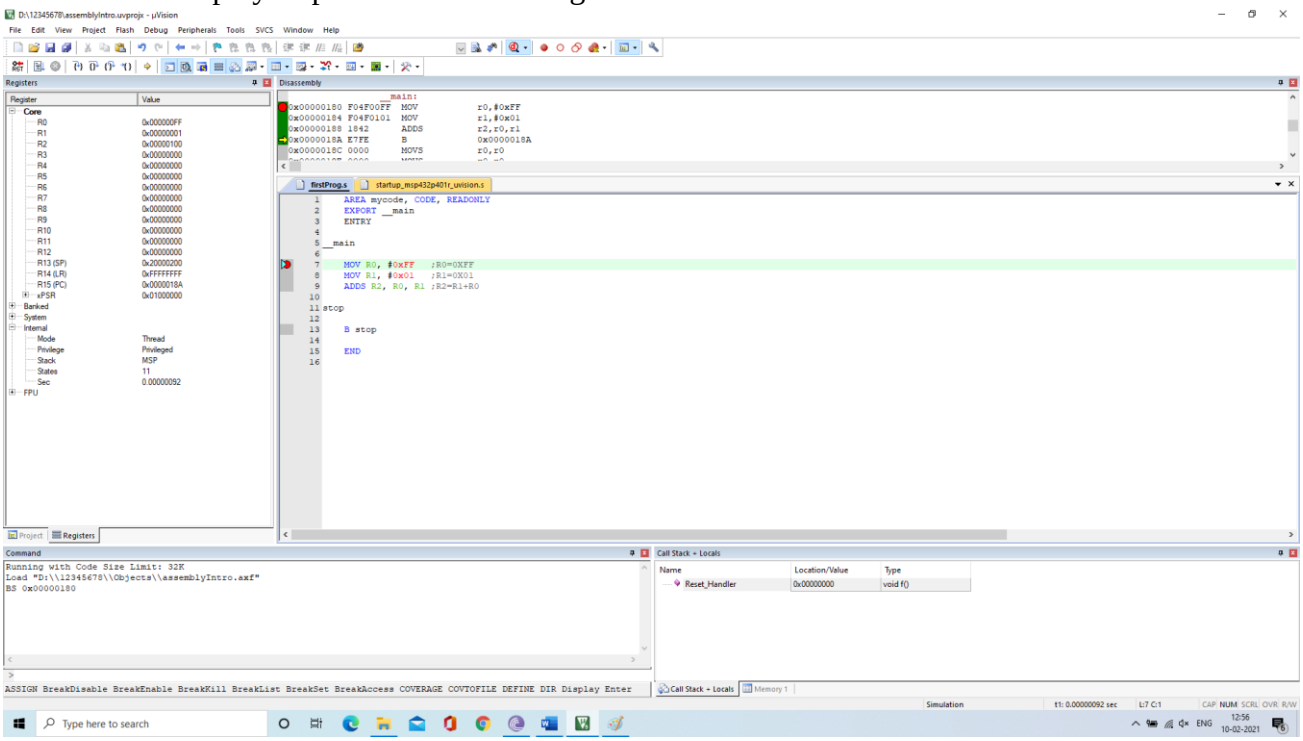


17. Press  debug button to start debugging

18. Select Breakpoint at the first line of the code. Select  run button to run the program till the first breakpoint.



19. Select {} to run step by step and observe the registers.




```
;Author Name
;Date when created
;Name of the program (Aim)
```

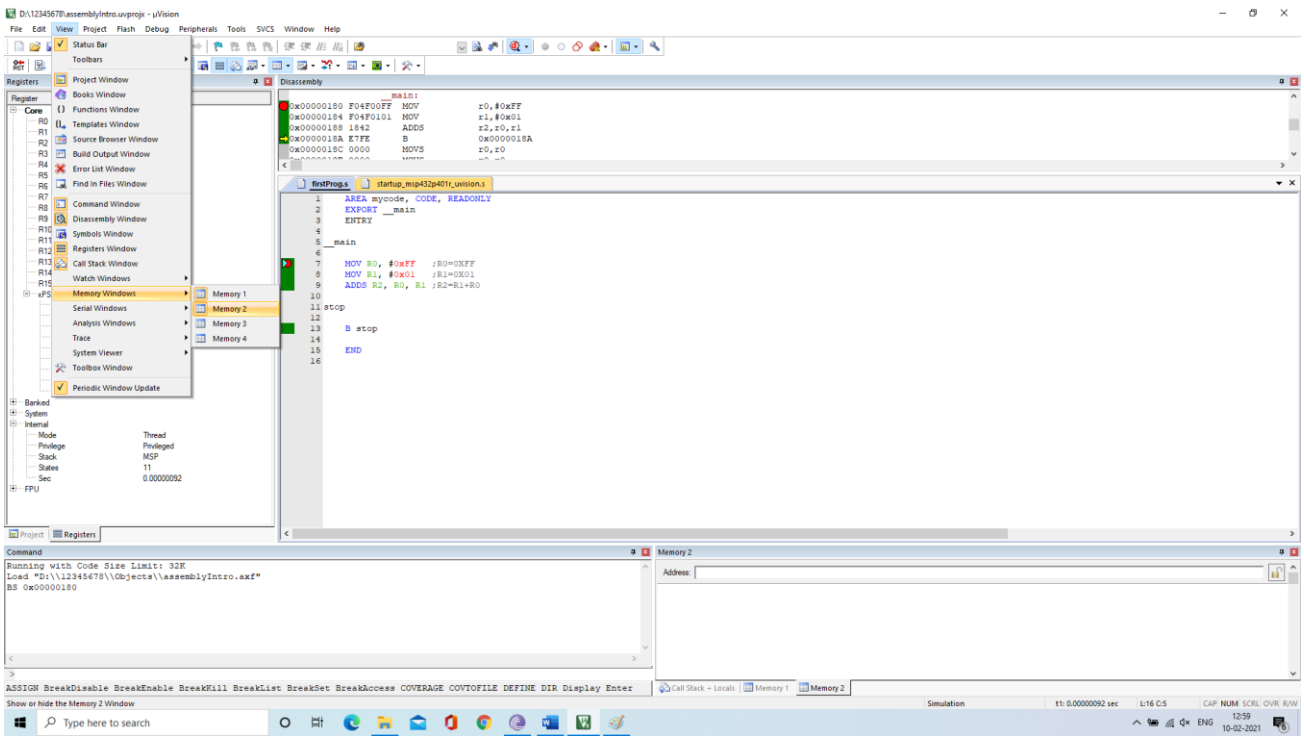
;Equipment used for the experiments and connection procedure

Program (code with comments)

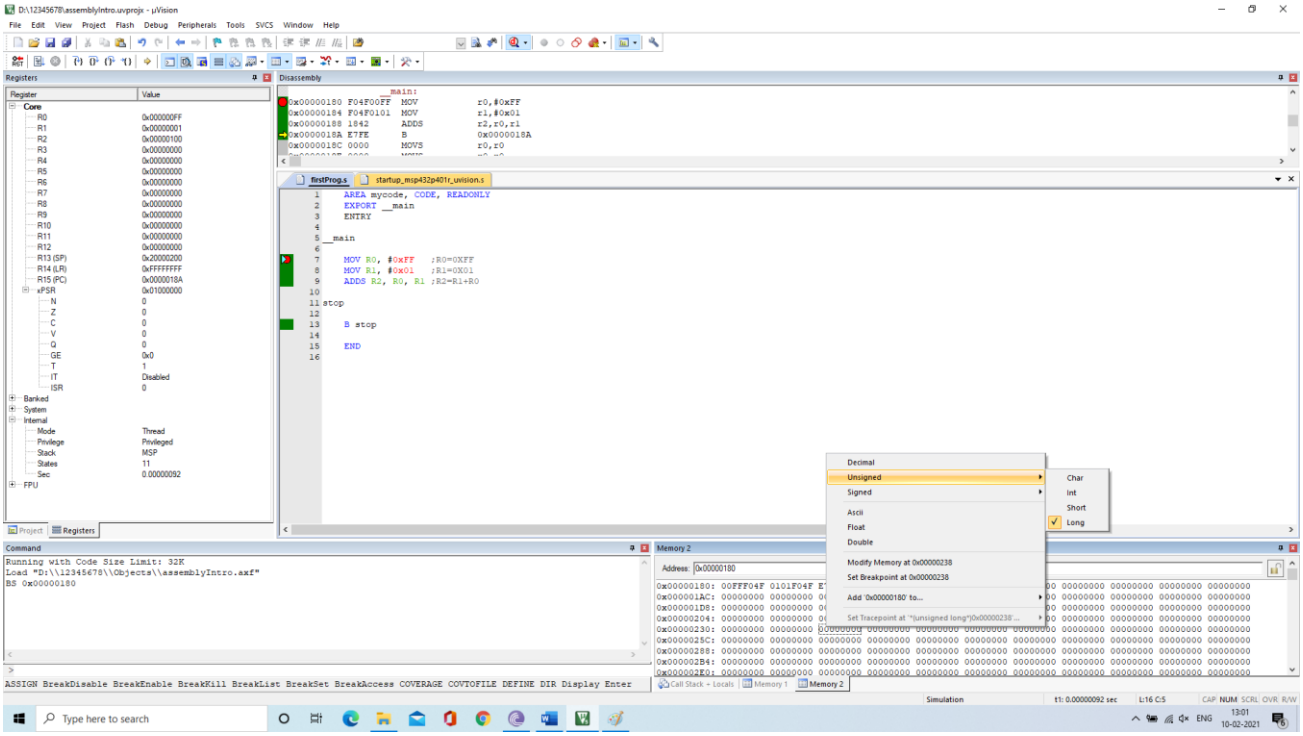
Output (Register contents or memory contents)

Inference (Your observations on output)

To view memory:



Click on View→Memory Windows (Memory 2)



Note: Check for errors; Press start debugging only if zero errors. Otherwise note the errors and line numbers, correct the errors and build again.

You can directly load the data to memory after debug.

Stop debug and make changes and rebuild in case if you change anything in the program.

Example 1: To Familiarize with data movement, arithmetic instructions in Keil µVision with MSP432P401R

```
        AREA mycode, CODE, READONLY
        EXPORT __main
        ENTRY

__main

        MOV R0, #0xFF    ;R0=0XFF
        MOV R1, #0x01    ;R1=0X01
        ADDS R2, R0, R1 ;R2=R1+R0

stop

        B stop

        END
```

Source: Please utilize the following links for using Keil and ARM (MSP432P401R).

1. "Getting Started with MDK Create Applications with µVision® for ARM® Cortex® -M Microcontrollers". https://armkeil.blob.core.windows.net/product/mdk5-getting-started.pdf
2. "µVision User's Guide". http://www.keil.com/support/man/docs/uv4/
3. "Arm® Keil® MDK Version 5 for SimpleLink™ MSP432™ Microcontrollers", http://www.ti.com/lit/ug/slau590j/slau590j.pdf
4. "MSP432P401R Simple Link™ Microcontroller Launch Pad™ Development Kit(MSP-EXP432P401R)", http://www.ti.com/lit/ug/slau597f/slau597f.pdf
5. "Texas Instruments MSP432: Cortex™-M4 Tutorial Using the MSP432P401R Launch Pad Board and ARM Keil MDK 5 Toolkit". http://www.keil.com/appnotes/files/apnt_276_v3_8.pdf
6. "General Purpose Input Output – MSP432", http://www.ti.com/lit/ml/swrp156/swrp156.pdf
7. "Teaching Materials TI University Program", https://university.ti.com/en/faculty
8. "ARM University Program Teaching Material", https://university.ti.com/en/faculty

Example 2: Write an assembly language program to perform addition between 25 and 35 and store the output in R3 register of MSP432P401R. Show the output in Debugging window. Syntax: ADD Operand 1 + Operand 2

Solution:

```
AREA main, CODE, READONLY
EXPORT __main        ; make __main visible to linker
ENTRY

__main
MOV R1, #0x25
MOV R2, #0x34
ADD R3, R2, R1


ALIGN
AREA allocations, DATA, READ WRITE
END
```

Example 3: Write an assembly language program to perform subtraction between 25 and 35, and store the output in R3 register of MSP432P401R. Show the output in Debugging window. Syntax: SUB Operand 1 - Operand 2

Solution:

```
AREA main, CODE, READONLY
EXPORT __main          ; make __main visible to linker

ENTRY
__main
 MOV  R1, #0x40
 MOV R2, #0x20
 SUB R3, R1, R2

 END
```

EXPERIMENT NO.2: INTRODUCTION TO INSTRUCTION SETS IN ARM CORTEX M4

AIM: To familiarize Instruction set in ARM Cortex M4.

Data Movement Instructions:

- MOV{s}{cond} Rd, Operand2
- MOV{cond} Rd, #imm16
- MVN{s}{cond} Rd, Operand2
- MOVW{cond} Rd, #imm16
- MOVT{cond} Rd, #imm16

Example 1: Write an Assembly language code to perform data movement in ARM Cortex M4F core.

Solution:

```
        AREA main, CODE, READONLY
        EXPORT __main       ; make     main visible to linker
        ENTRY
__main
        MOV   R1, #0x40
        MOV   R2, #0x20
        MOVT R3, #0xFFFF
        MOVW R4, #0xFFFF
        MVN R5, #0x0FF
Here    B       Here

        END
```

Shift and Rotate Instructions:

- LSR{s}{cond}{.size} Rd, Rm, Rs or #n
- ASR{s}{cond}{.size} Rd, Rm, Rs or #n
- LSL {s}{cond}{.size} Rd, Rm, Rs or #n
- ROR {s}{cond}{.size} Rd, Rm, Rs or #n
- RRX{s}{cond} Rd, Rm

Example 2: Write an Assembly language code to perform shift and rotate in ARM Cortex

M4F core.

Solution:

```
    AREA main, CODE, READONLY
    EXPORT __main      ; make    main visible to linker
    ENTRY
__main
    MOV   R1, #0x4
    MOV   R2, #0x5A5A
    LSR R2, R1
    ASR R2, R1
    LSL R2, R1
    ROR R2, R1
    RRX R2, R1
Here  B       Here

    END
```

Logical Instruction:

- AND{s}{cond}{.size} Rd, Rn, Operand2
- BIC{s}{cond}{.size} Rd, Rn, Operand2
- EOR{s}{cond}{.size} Rd, Rn, Operand2
- ORN{s}{cond}{.size} Rd, Rn, Operand2
- ORR{s}{cond}{.size} Rd, Rn, Operand2

Example 3: Write an Assembly language code to perform Logical operations in ARM Cortex M4F core and mention the flag updates.

Solution:

```
    AREA main, CODE, READONLY
    EXPORT __main      ; make    main visible to linker
    ENTRY
 __main
    MOV   R1, #0x01
    MOV   R2, #0x01
    ANDS R1, R2
    BICS R1, R2
    EORS R1, R2
    ORNS R1, R2
    ORRS R1, R2
 Here B       Here
```

```
    END
```

Arithmetic Instruction:

- ADD{s}{cond}{.size} Rd, Rn, Operand2
- ADC{s}{cond}{.size} Rd, Rn, Operand2
- SUB{s}{cond}{.size} Rd, Rn, Operand2
- SBC{s}{cond}{.size} Rd, Rn, Operand2
- RSB{s}{cond}{.size} Rd, Rn, Operand2
- ADDW{cond} Rd, Rn, #imm12
- SUBW{cond} Rd, Rn, #imm12

Example 4: Write an Assembly language code to perform Arithmetic operations in ARM Cortex M4F core and mention the flag updates.

Solution:

```
    AREA main, CODE, READONLY
    EXPORT __main       ; make     main visible to linker
    ENTRY
__main
    MOV   R1, #0x01
    MOV   R2, #0x01
    ADDS R1, R2
    ADCS R1, R2
    SUBS R1, R2
    SBCS R1, R2
    RSBS R1, R2
Here B        Here

    END
```

Multiplication & Divide Instructions:

- MUL{S}<cond> <Rd>, <Rn>, <Rm>
- MLA{s}<cond> <Rd>, <Rn>, <Rm>, <Ra>
- MLS <cond> <Rd>, <Rn>, <Rm>, <Ra>
- UMULL{cond} RdLow, RdHigh, Rn, Rm
- UMLAL{cond} RdLow, RdHigh, Rn, Rm
- SMULL {cond} RdLow, RdHigh, Rn, Rm
- SMLAL{cond} RdLow, RdHigh, Rn, Rm
- SDIV {cond} Rd, Rn, Rm

- UDIV{cond} Rd, Rn, Rm

Example 5: Write an Assembly language code to perform Multiplication and Division in ARM Cortex M4F.

Solution:

```
    AREA main, CODE, READONLY
    EXPORT __main       ; make     main visible to linker
    ENTRY
__main
    MOV   R1, #0x0FF
    MOV   R2, #0x0FF
    MUL R3, R1, R2
    MLA R4, R3, R1, R2
    MLS R5, R3, R1, R2
    UMULL R1, R2, R3, R4
    UMLAL R1, R2, R3, R4
    SMULL R1, R2, R3, R4
    SMLAL R1, R2, R3, R4
    SDIV R3, R1, R2
    UDIV R1, R2, R3
Here B        Here
    END
```

Bitfield Instruction:

- REV{cond} Rd, Rn
- REV16{cond} Rd, Rn
- REVSH{cond} Rd, Rn
- RBIT {cond} Rd, Rn
- SBFX{cond} Rd, Rn, #lsb, #width
- UBFX{cond} Rd, Rn, #lsb, #width
- SXTB{cond} Rd, Rm, ROR#imm
- SXTH{cond} Rd, Rm, ROR#imm
- UXTB{cond} Rd, Rm, ROR#imm
- UXTH{cond} Rd, Rm, ROR#imm
- CLZ{cond} Rd, Rm

Example 6: Write an assembly language program to perform bit field operations.

Solution:

```
        AREA main, CODE, READONLY
        EXPORT __main
        ENTRY
__main
        LDR R0, =0XFAF3AF0A
        REV    R1, R0
        REV16 R1, R0
        REVSH R1, R0
        RBIT R1,R0
        BFC R0, #4, #8
        BFI R1, R0, #4, #12
        SBFX R1,R0, #6, #10
        UBFX R1, R0, #6, #10
        SXTB R1, R0
        UXTB R1, R0
        SXTH R1, R0
        UXTH R1, R0
        SXTB R1, R0, ROR #8
        CLZ R2, R1
Here B Here
        END
```

Memory Access Instruction:

- LDR{type}{cond} Rt, Rn,{#offset}
- STR{type}{cond} Rt, Rn,{#offset}
- PUSH{cond} reglist
- POP{cond} reglist **movement**

Example 7: Write an assembly language program to perform data in accessed memory locations.

Solution:

```
        AREA main, CODE, READONLY
        EXPORT __main
        ENTRY
__main
        LDR    R1, =0x20000000      ;R1=0x20000000
        LDR    R2, =0x64156415
        STR    R2, [R1]             ;Store R2 to location 0x20000000
        ADD    R1, R1, #1           ;R1 = R1 + 1 = 0x20000001
        STR    R2, [R1]             ;Store R2 to location 0x20000001
        ADD    R1, R1, #1           ;R1 = R1 + 1 = 0x20000002
        STR    R2, [R1]             ;Store R2 to location 0x20000002
```

```
ADD   R1, R1, #1          ;R1 = R1 + 1 = 0x20000003
STR   R2, [R1]            ;Store R2 to location 0x20000003

END
```

Branch Instruction:

- B <Cond> <LABEL>
- BL <Cond> <LABEL>
- BX <Cond> <Rm>
- BLX <Cond> <Rm>
- CBZ <Rn>, <LABEL>
- CBNZ <Rn>, <LABEL>
- TBB <Cond> [<Rn>, <Rm>]
- TBH <Cond> [<Rn>, <Rm>, LSL #1]

Branch Condition

<Cond>	MEANING	FLAGS
EQ	EQUAL	Z=1
NE	NOT EQUAL	Z=0
CS, HS	CARRY SET, UNSIGNED HIGHER OR SAME	C=1
CC, LO	CARRY CLEAR, UNSIGNED LOWER	C=0
MI	MINUS, NEAGATIVE	N=1
PL	PLUS, POSITIVE OR ZERO	N=0
VS	OVERFLOW	V=1
VC	NO OVERFLOW	V=0
HI	UNSIGNED HIGHER	C=1 AND Z=0
LS	UNSIGNED LOWER OR SAME	C=0 OR Z=1
GE	SIGNED GREATER OR EQUAL	N=V
LT	SIGNED LESS	N NOT EQUAL TO V
GT	SIGNED GREATER	Z=0 AND N=V
LE	SIGNED LESS OR EQUAL	Z=1 OR N NOT EQUAL TO V
AL, <OMIT>	ALWAYS	ANY

Example 8: Write an assembly language to perform down counting operation continuously from 10hex to zero.

Solution:

```
AREA SATURATION, CODE, READONLY
```

```
        EXPORT __main
        ENTRY
__main
        LDR R1, =0x0000010
Here    SUBS R1, R1, #1;
        BNE Here

        END
```

Exercise:

1. Write an assembly language code for the following expression in MSP432 launchpad.
    a. $(X^2+Y^3) + Z^2$
    b. $X^2+Y^2 + 2XYZ$
    c. $Y = MX + C^2$

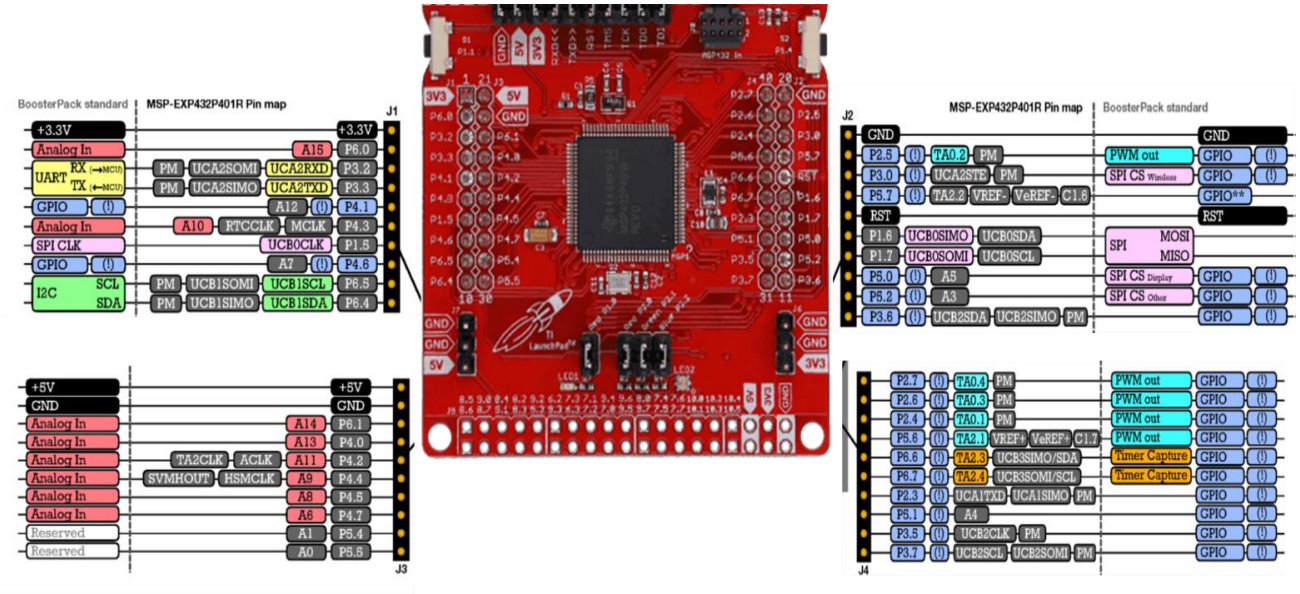
    Condition: X, Y, Z are the last three digits of your registration number. Assume missing variables as a variable.

2. Write an assembly language code to separate odd and even numbers between 0-100. Store Odd numbers from 0x20000000 memory location and Even number in 0x2000F000.
3. Write an assembly language code to store the 10 prime numbers into a memory location (0x20000000) onwards, starting from 1.

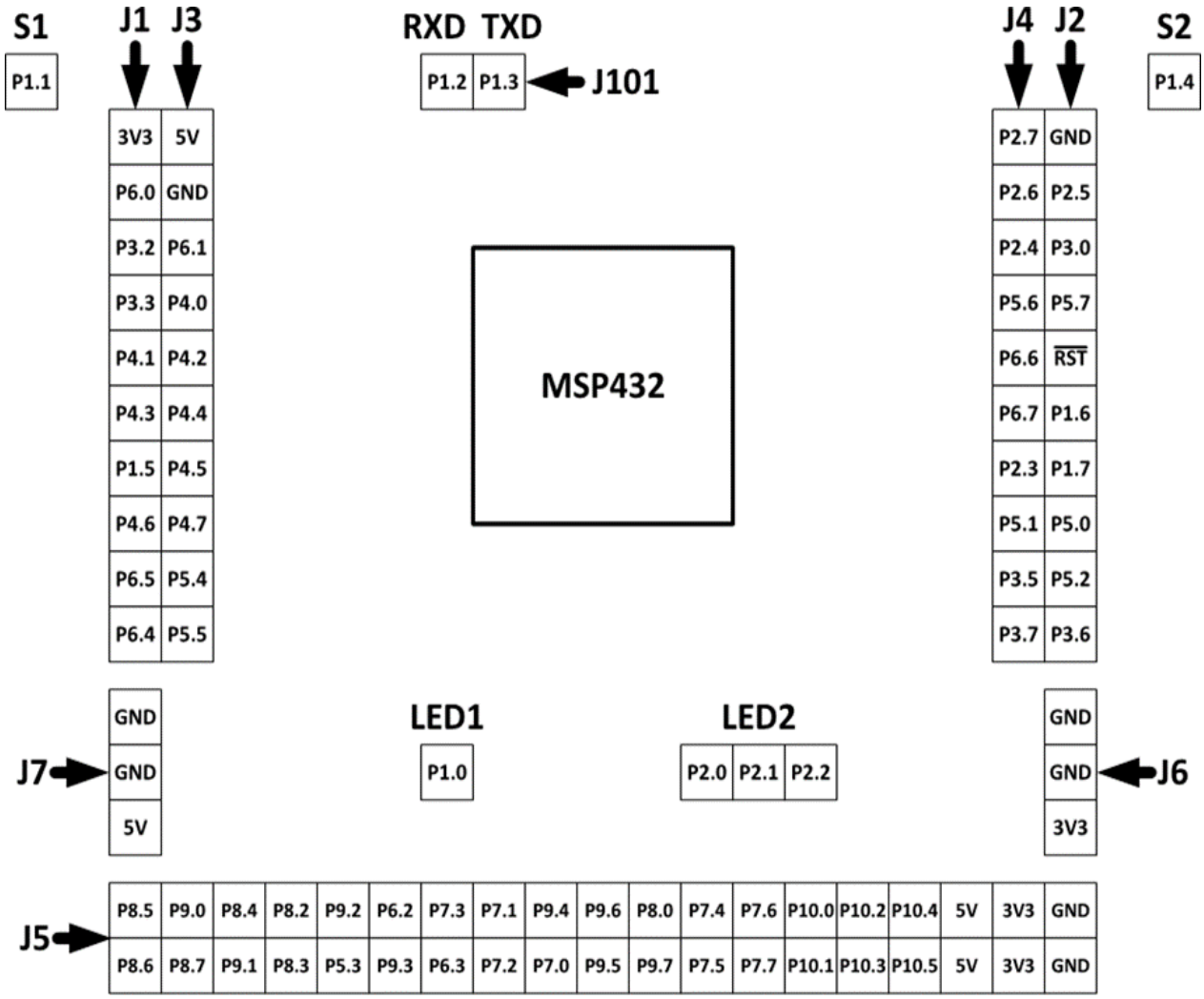
EXPERIMENT NO. 3: INTRODUCTION TO EMBEDDED C AND DEVELOPING ALGORITHMS FOR GPIO & NVIC

AIM: To familiarize the Embedded C Programming and develop applications for GPIO and NVIC

General Purpose Input and Output: GPIO

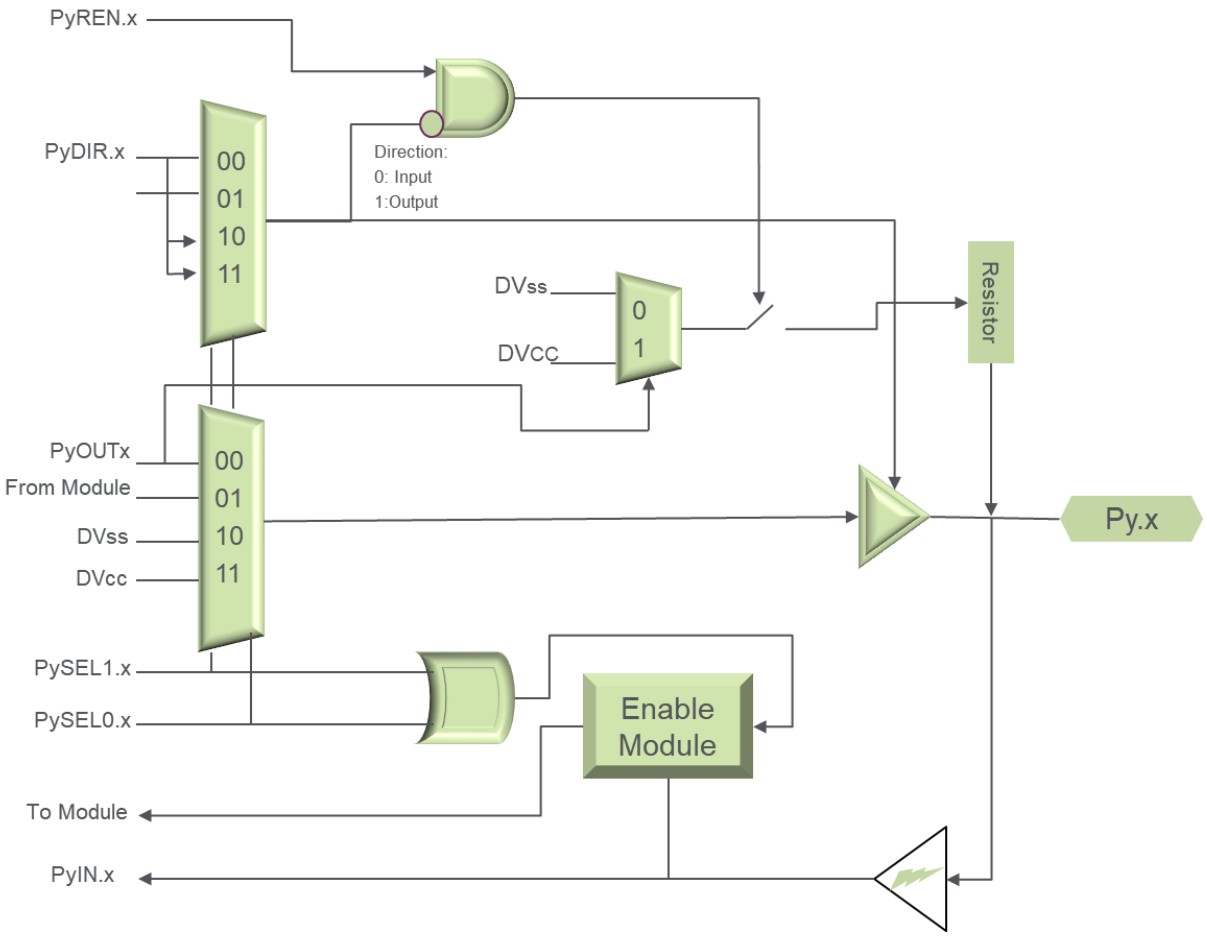


MSP432P401R Launchpad Pin Configuration

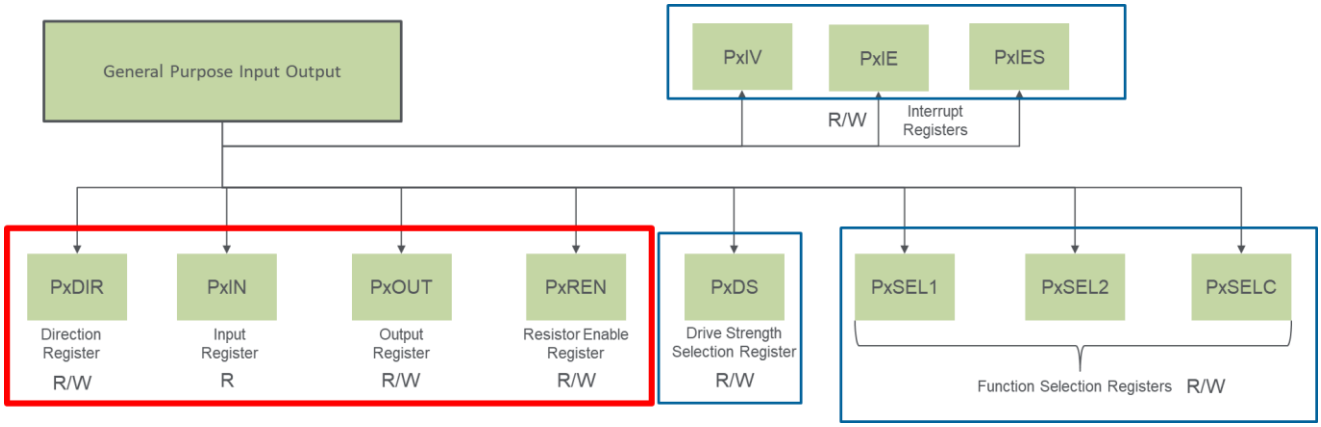


Port P1	P1.5–P1.7
Port P2	P2.3–P2.7
Port P3	P3.0, P3.2–P3.3, P3.5–P3.7
Port P4	P4.0–P4.7
Port P5	P5.0–P5.7
Port P6	P6.0–P6.7
Port P7	P7.0–P7.7
Port P8	P8.0, P8.2–P8.7
Port P9	P9.0–P9.7
Port P10	P10.0–P10.5

Each PIN Circuit Diagram:

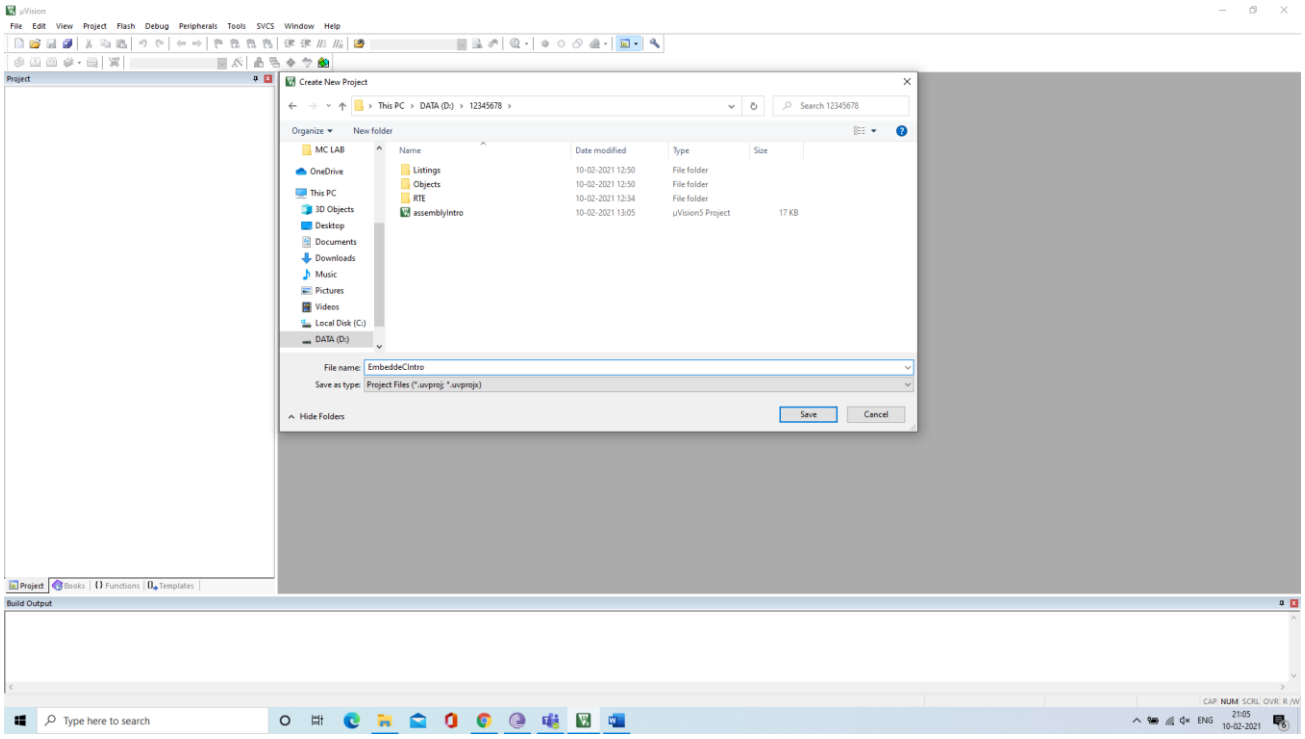


Registers of GPIO:

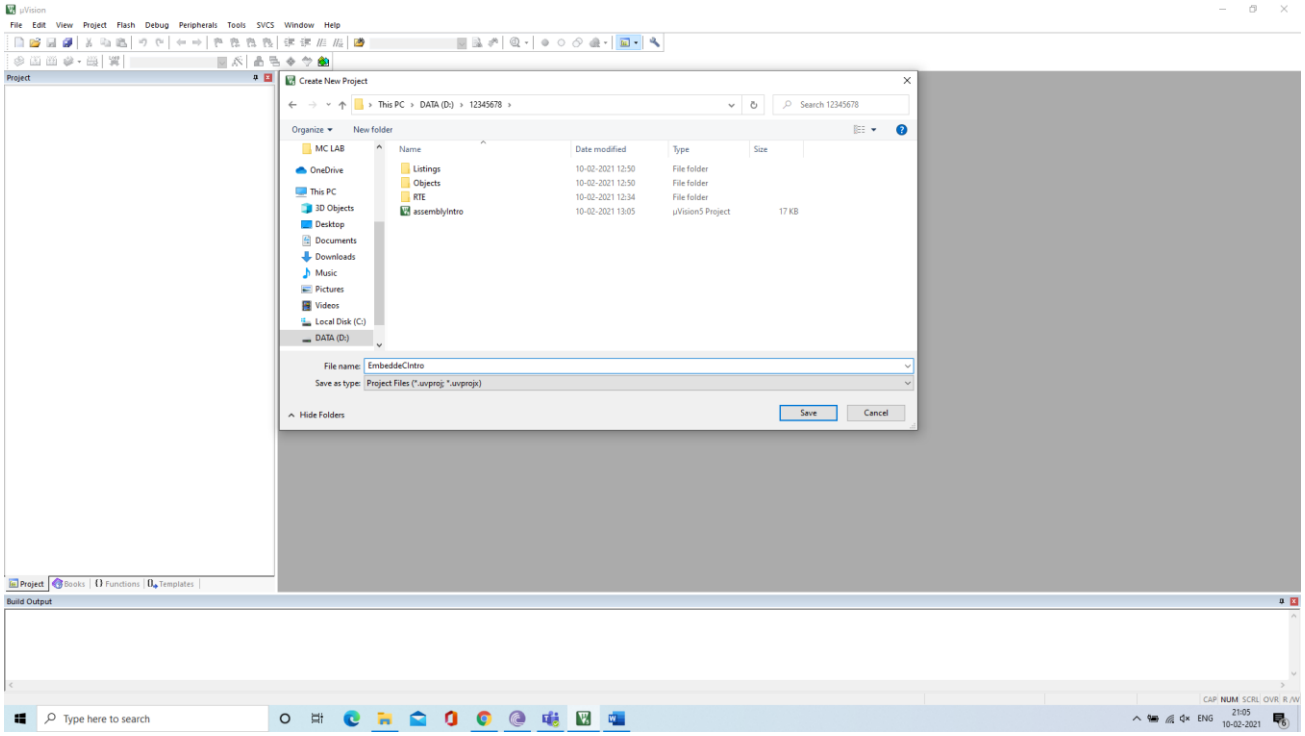


Procedure for connecting MSP432P401R using Keil Software.

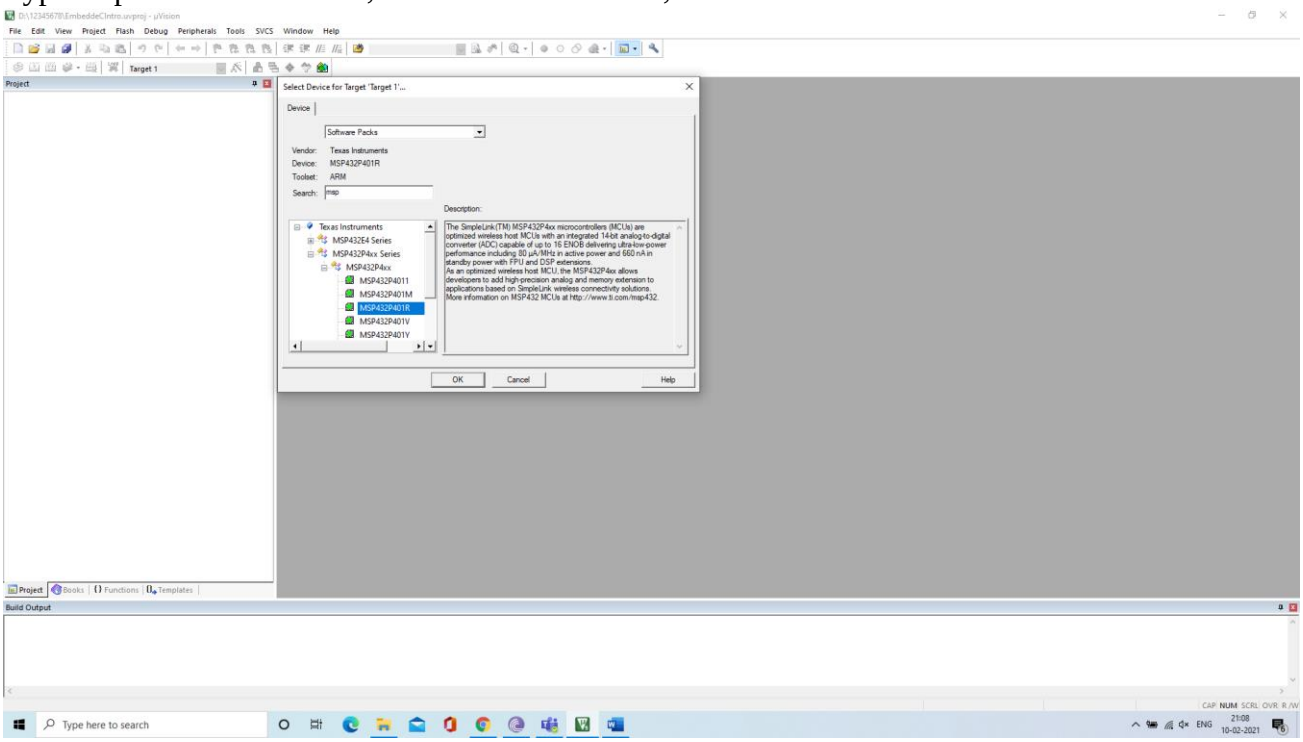
1. Project→Create New Project



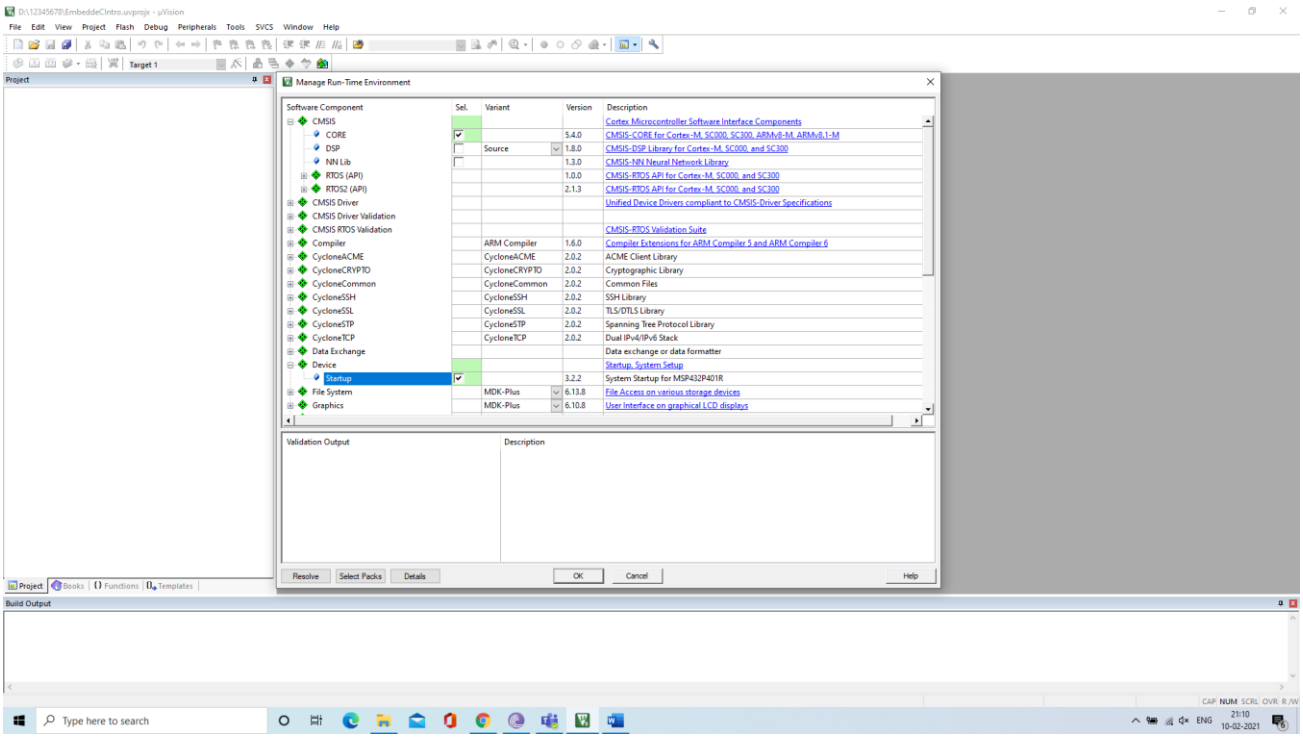
2. Enter the file name as EmbeddedCIntro (anything you wish), Click Save.



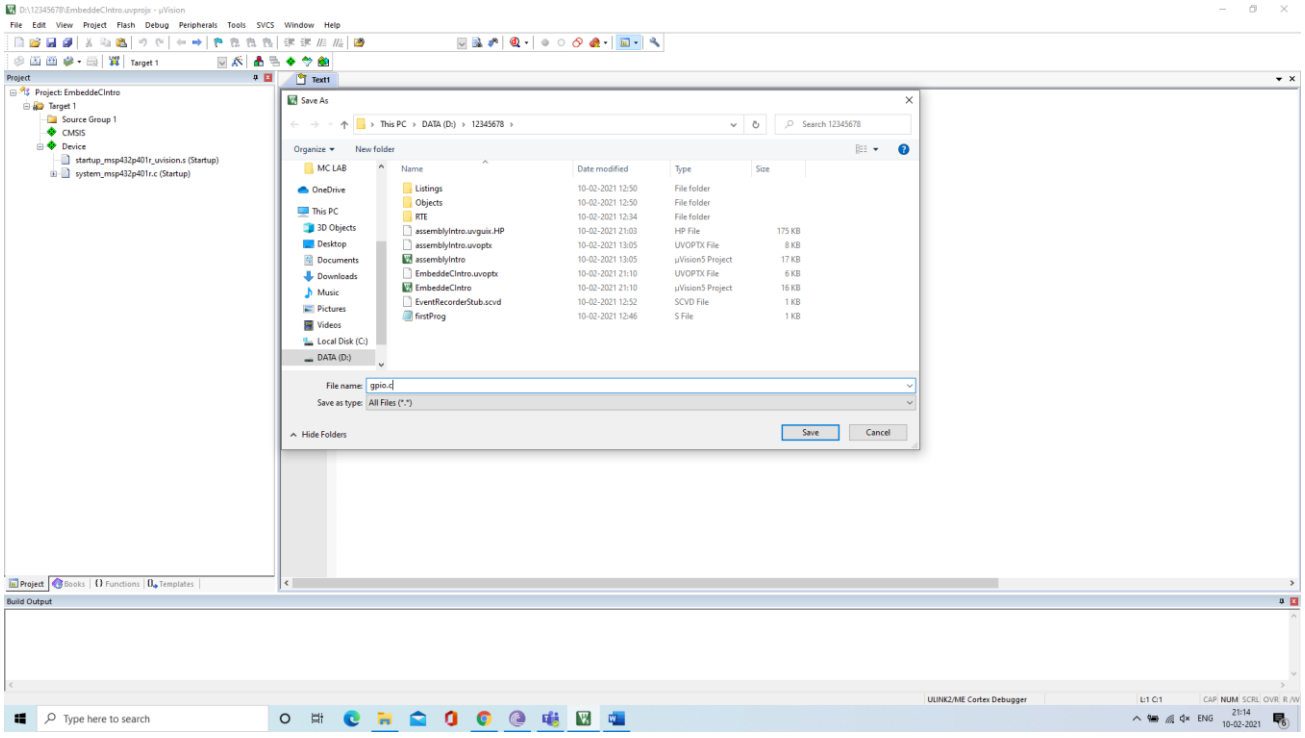
3. Type msp in the search field, select MSP432P401R, click OK.



4. Check the right side box of CORE in CMSIS, and Startup in Device, Click OK.



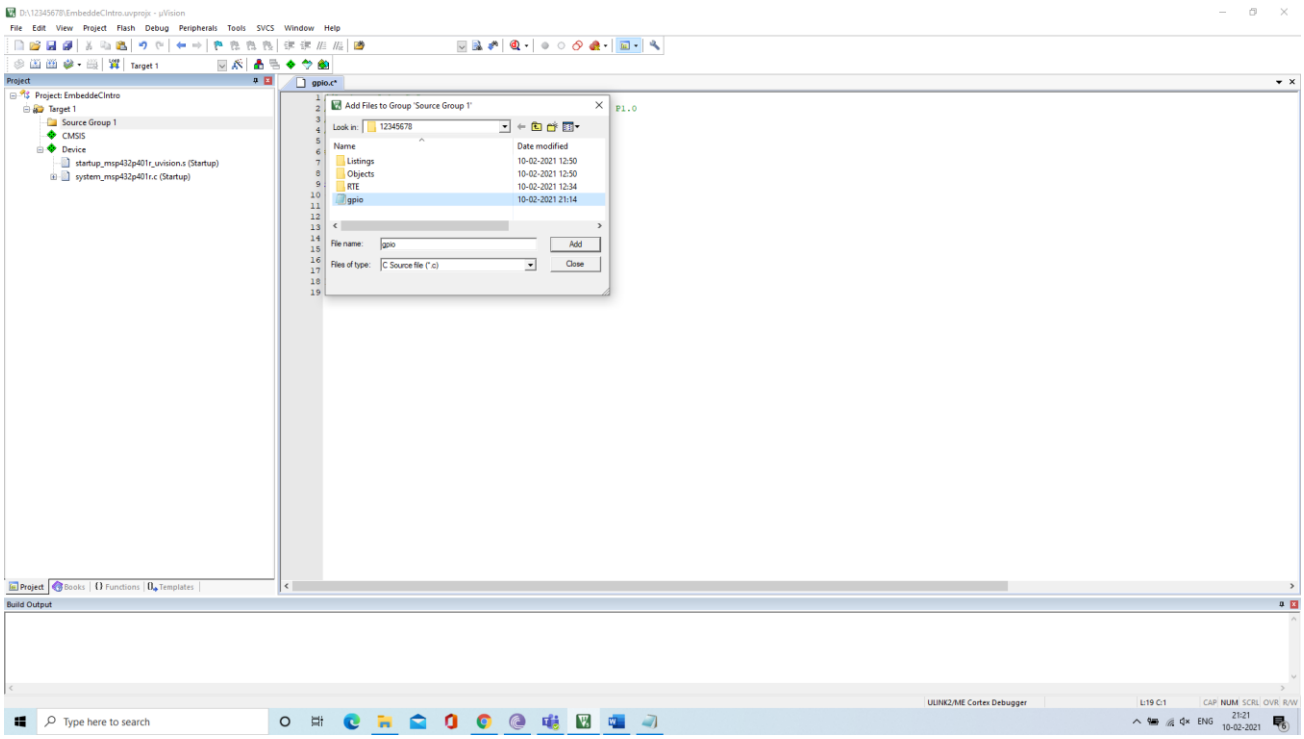
5. File→ New
6. Save the file as filename.c (Example gpio.c), Click Save.



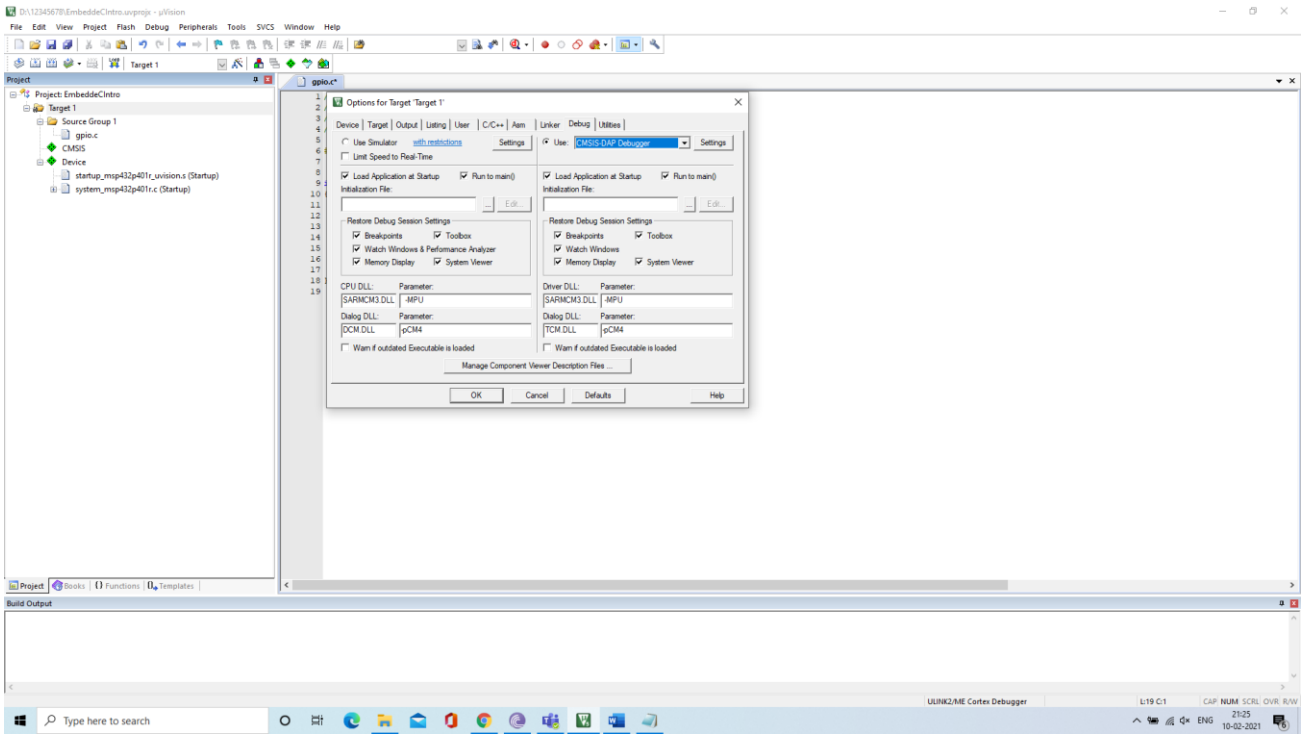
7. Type the program.

```
//Author: Asha C S
//Embedded C code to glow LED connected at pin number P1.0
//Date: 10-02-2021
```

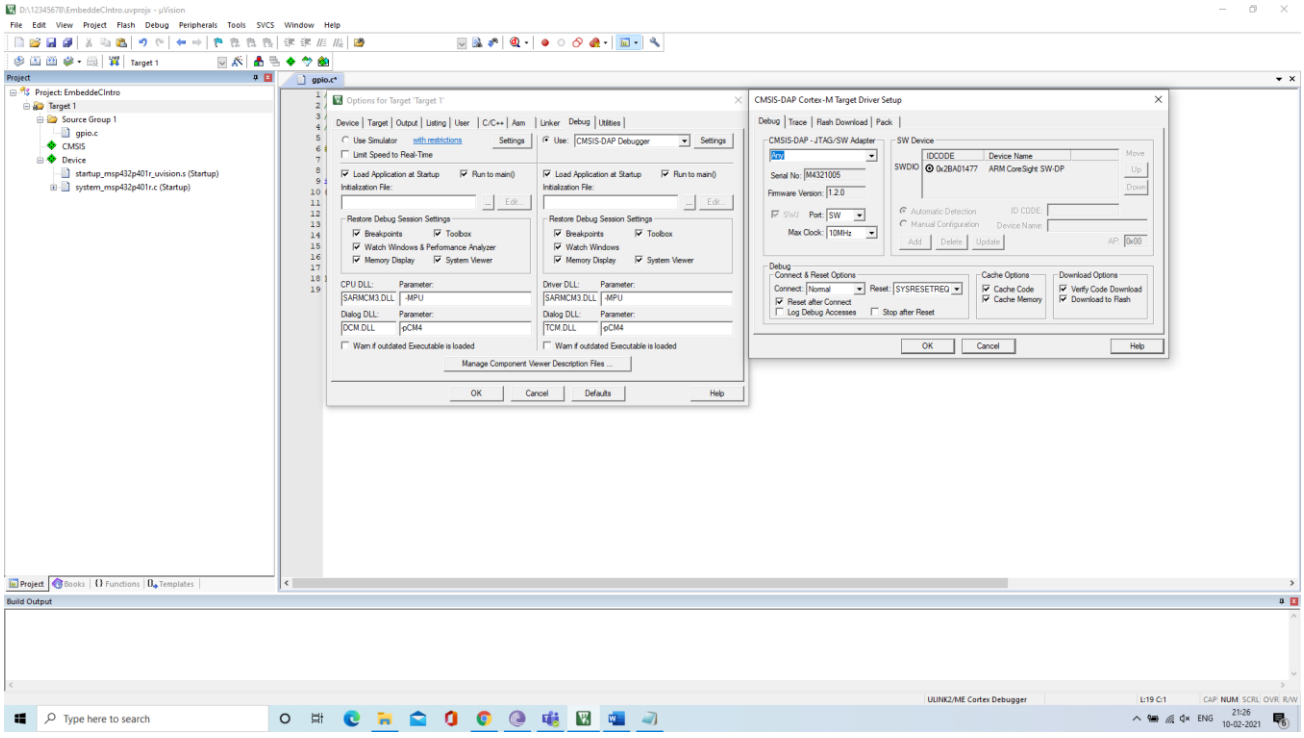
```
//Kit used: MSP432P401R

#include "msp.h"


int main(void)
{

    //stop watch dog timer
    WDT_A->CTL=WDT_A_CTL_PW|WDT_A_CTL_HOLD;
    // Make P1.0 as output pin
    P1->DIR=0x01;
    //Glow P1.0
    P1->OUT=0x01;
}
```

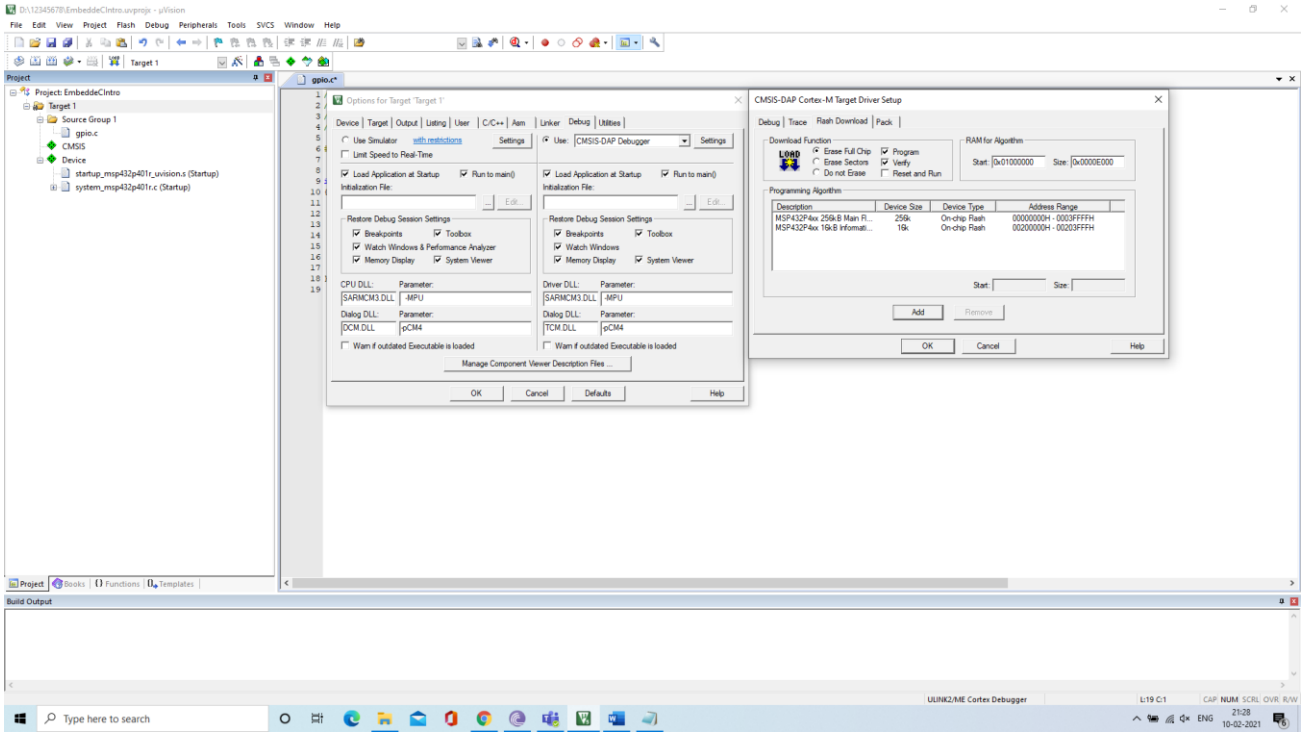


8. Right click on Source Group 1→ Add existing file→Click gpio.c file→Add→Close
9. Right click on Target 1→ options for target

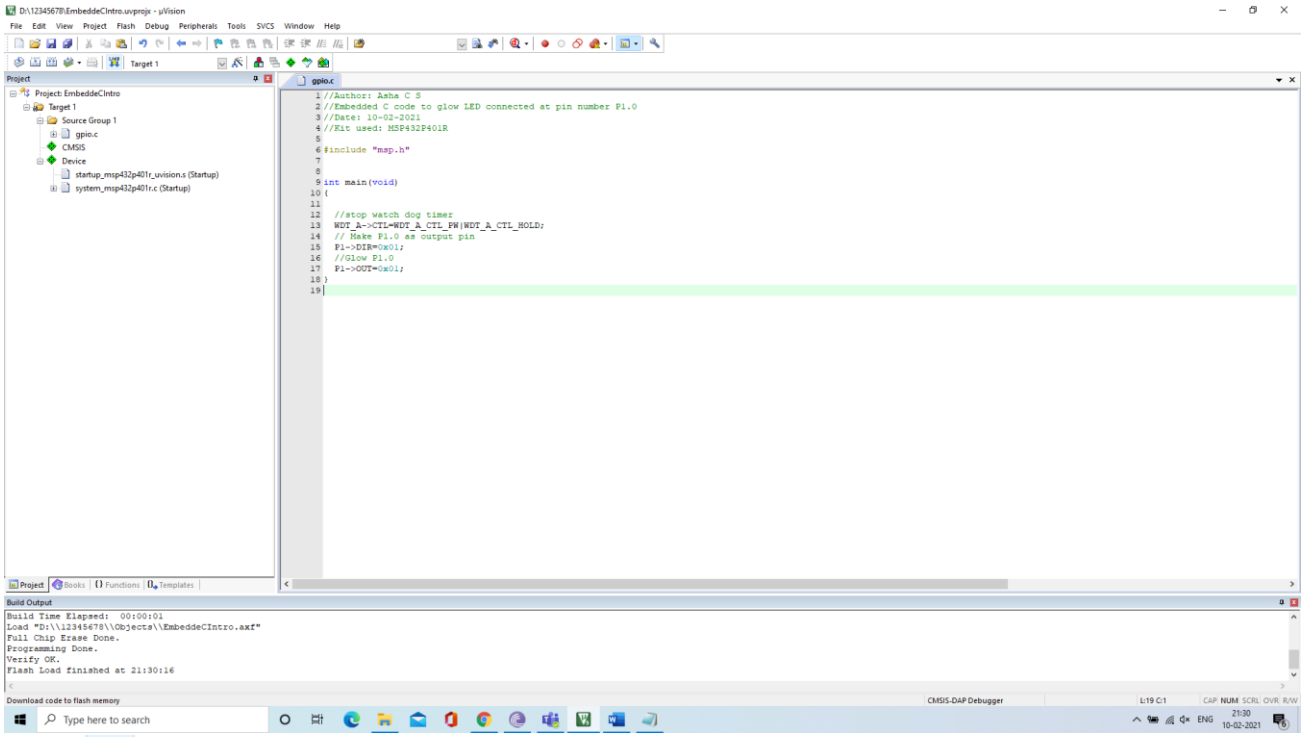


10. Debug→Click Use→CMSIS DAP debugger
11. Click settings→Check verify code download and download to flash



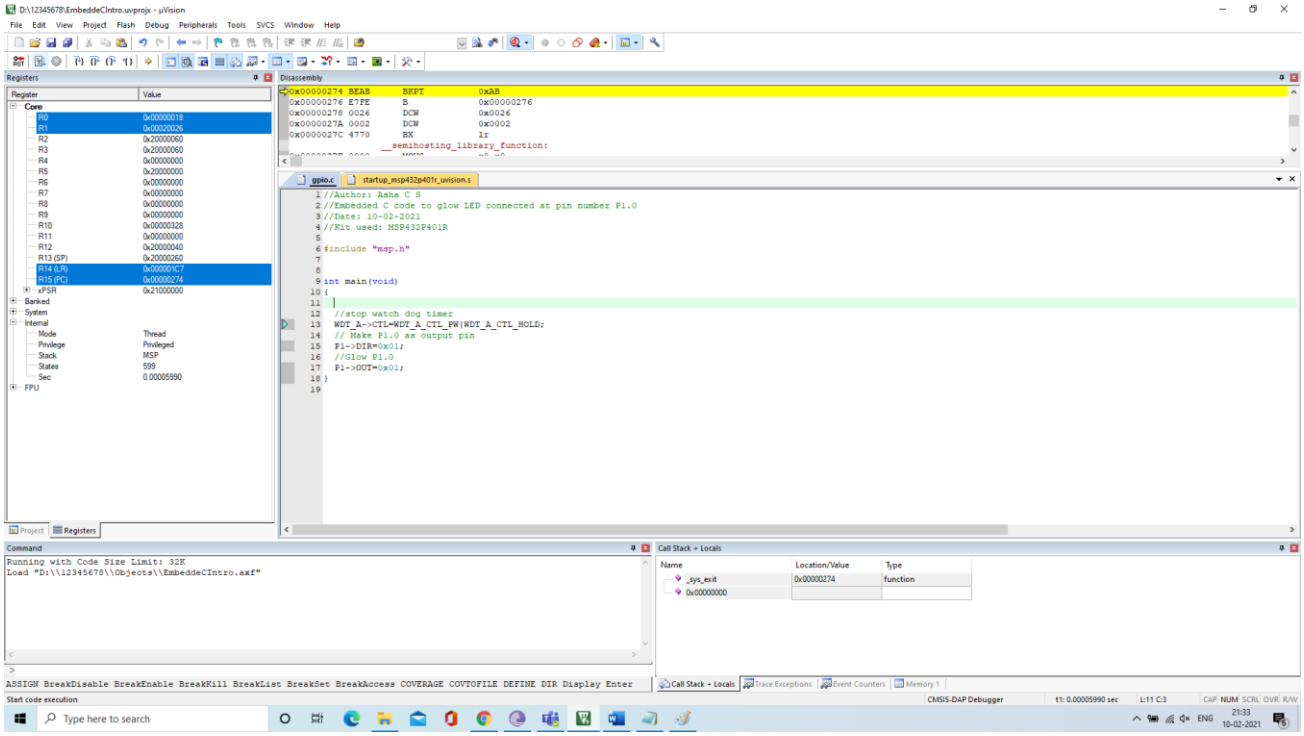


12. Erase Full chip in Flash download tab, click OK.
13. Click  build button, check for zero errors.



14. Click  download button. Press Reset button of microcontroller.

    Or Click debug  and  run button.
15. Observe LED.



Example 1: Write an embedded C programming language to toggle an RED LED in MSP432P401R.

Solution:

```
#include "msp.h"

int main(void)
{
        volatile uint32_t i;
        // Stop watchdog timer

        WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;

        // The following code toggles P1.0 port
         P1->DIR |= BIT0;                    // Configure P1.0 as output

        while(1)
        {
                P1->OUT ^= BIT0;             // Toggle P1.0
                for(i=10000; i>0; i--);      // Delay
```

```
        }
}
```

Example 2: Write an Embedded C code to perform LED blinking after pressing switch P1.1
Solution:

```
#include "msp.h"

int main(void)
 {
   WDT_A->CTL = WDT_A_CTL_PW |WDT_A_CTL_HOLD;

        /*** Write pin configuration here***/

                while (1)
                 {
                /*** Write your Logic Here***/
                 }
}
```

Exercise:

1. **Write an Embedded C code to perform LED Blink operation on RGB led P2.0, P2.1, P2.2 one after another for five times in a loop.**
2. **Write an Embedded C code to blink the LED to only if P1.1 and P1.4 switches are pressed.**
3. **Write an Embedded C code to blink the LED to only if P1.1 or P1.4 switches are pressed.**

Nested Vector Interrupt controller:

Example 3: Write an Embedded C code to turn on LED P1.0 when P1.1 switch is pressed. Write an ISR to perform the operation when the P1.1 is pressed.

Solution:

```
#include "msp.h"
#define LED1 BIT0
#define S1 BIT1
#define DELAY 500
int i;

int main (void)
        {
                WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;
```

```
            /*** Write pin configuration here***/

            NVIC->ISER [1] = 0x00000008;
            __enable_irq();
            while (1);
    }

void PORT1_IRQHandler (void)
    {
            /*** Write your Logic Here***/
    }
```

Exercise:

1. **Write an Embedded C Code to perform display letters 0 to 9 using seven segments display with the delay. Refer https://www.electronics-tutorials.ws/blog/7-segment-display-tutorial.html for theory.**

EXPERIMENT NO. 4: Developing Algorithms for Timers and Counters (SysTick and Timer 32).

AIM: To utilize the delay functions using hardware and software-based delays.

Example 1: Write an Embedded C programming language to generate a delay of one second using software delay (for loop condition.)

Solution:

```
#include "msp.h"
int main(void)
{
    volatile uint32_t i;

    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD; // Stop watch dog timer
    P2->DIR |= 0x07;
    while(1)
```

```
        {
            P2->OUT ^= 0xff;
            for (i=1000; i>0; i--);
        }
}
```

Example 2: Write an Embedded C programming language to generate a delay of one second using SysTick timer.

Solution:

```
#include "msp.h"
int main(void)
{
        WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;
        P1->DIR |= BIT0;
        P1->OUT &= ~BIT0;

   /***Initialize SysTick Here***/

        __enable_irq();
        while (1);

}

void SysTick_Handler(void)
    {
        P1->OUT ^= BIT0;
    }
```

Example 3: Write an Embedded C programming language to generate a delay of one second using Timer 32 interrupt.

Solution:

```
#include "msp.h"
#include <stdint.h>
int main(void)
    {
        volatile uint32_t i;
        WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;
        P1->DIR |= BIT0;
        P1->OUT &= ~BIT0;
```

```
    /*** Initialize Timer setup here***/

        while (1);
}

void T32_INT1_IRQHandler(void)
{
    /***timer clear/set here***/
}
```

Exercise:

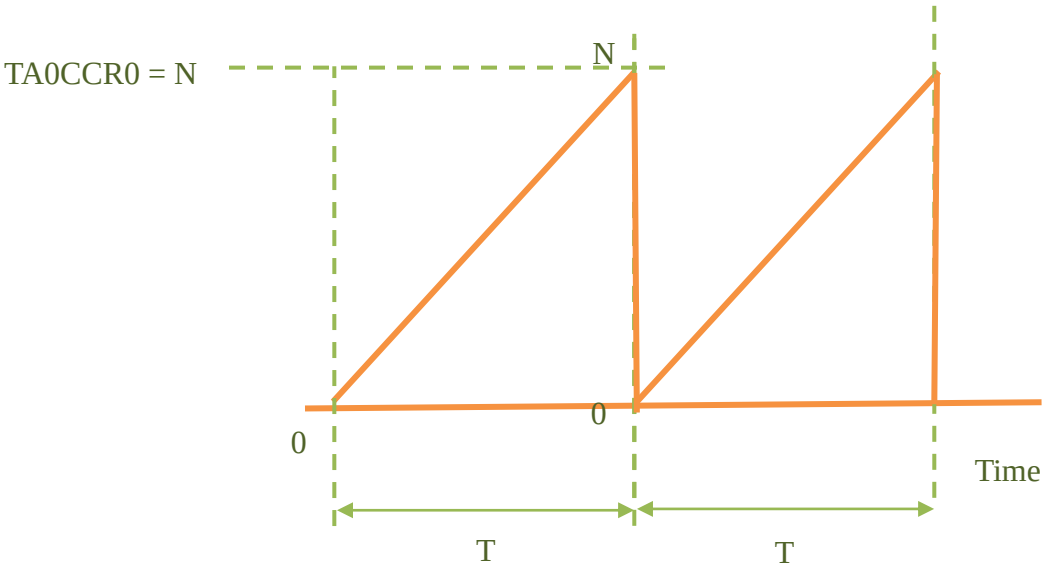
1. **Write an Embedded C programming language for generating a delay of 4 seconds using default frequency of Timer 32 and toggle RED LED connected to P1.0.**

2. **Write an Embedded C code to rotate DC motor in clockwise and anti-clockwise a delay of 1 minute using Timer 32 interrupts.**

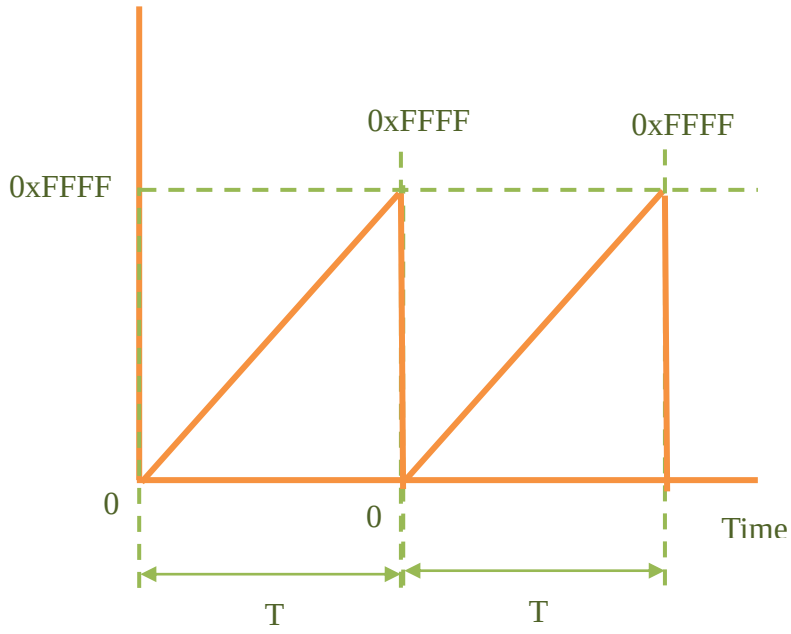
3. Write an Embedded C code to rotate DC motor in clockwise and anti-clockwise while pressing P1.1 & P1.4 respectively with a delay of for 1 minute using Timer 32 interrupts.

EXPERIMENT NO. 5: USE OF TIMER A AND GENERATION OF PWM SIGNALS USING TIMER A

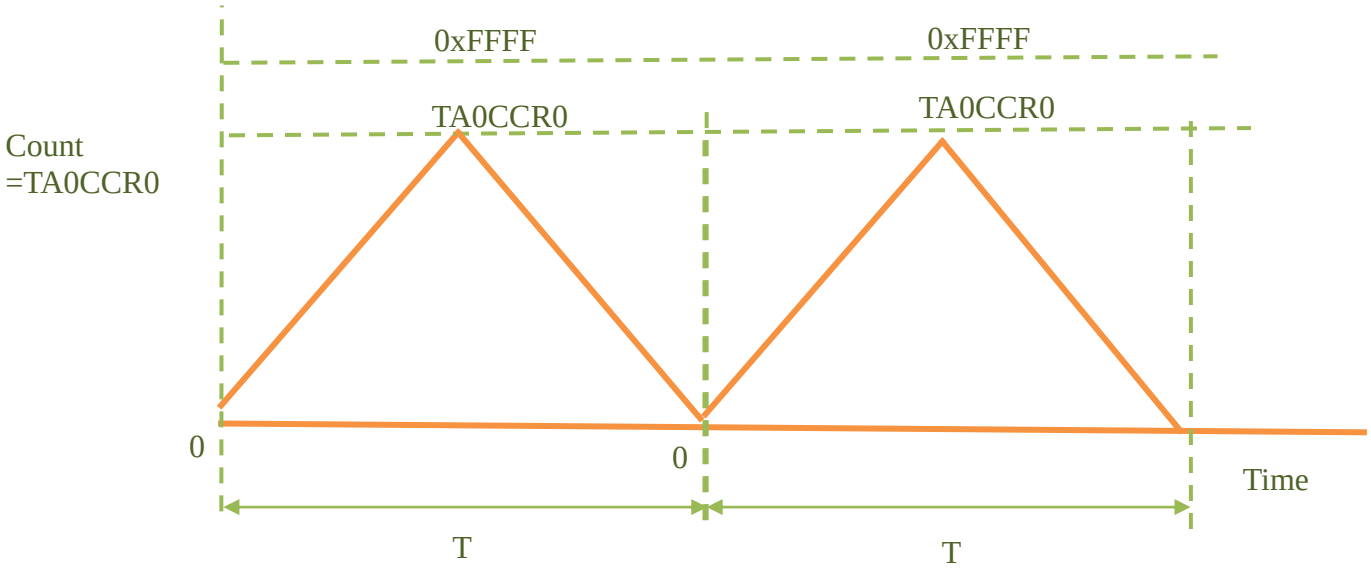
AIM: To Familiarize the concept of TIMER A for delay generation, and Pulse Width Modulation.



$$T = \frac{(TA0CCR0+1)}{Freq}$$



$$T = \frac{(0xFFFF+1)}{Freq}$$



$$T = \frac{(2TA0CCR0)}{Freq}$$

Steps	Details
1.	Set the port pins output (such as LED, speakers, display devices) by setting Px->DIR regist pins
2.	Select the clock (TIMER_A_CTL_TASSEL_x), select the mode (TIMER_A_CTL_MC_3) enable timer interrupt (TIMER_A_CTL_IE), clear (TIMER_A_CTL_CLR)
3.	Load the count value as TIMER_A0->CCR[0] = 50000;
4.	Enable TIMER_A in NVIC using NVIC->ISER[0]=1<<TA0_0_IRQn & 31

	TIMER_A0->CCTL[0] &= ~TIMER_A_CCTLN_CCIFG; TIMER_A0->CCTL[0] = TIMER_A_CCTLN_CCIE;
***	Enable TIMER_A in NVIC using NVIC->ISER[0]=1<<TA0_N_IRQn & 31
5.	Enable interrupt in PRIMASK register using _enable_irq();
	Stay in infinite loop (while(1);) or sleep mode (__sleep();)
6.	Write ISR in void TA0_0_IRQHandler(void){} (CCIFG flag)
***	Write ISR in void TA0_N_IRQHandler(void){} (TAIFG flag)
7.	Clear the flag using TIMER_A0->CCTL[0] &= ~ TIMER_A_CCTLN_CCIFG
***	Clear the flag using TIMER_A0->CTL &= ~ TIMER_A_CTL_IFG
8.	Toggle the LED as P2->OUT ^= BIT0

Example 1: Write an Embedded C program to generate delay using continuous mode of TIMER A.

Solution:

```
#include "msp.h"
int main(void)
{
        WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;
        /*** Initalize timer A in continuous mode here***/
        NVIC->ISER[0] = 1 << ((TA0_N_IRQn) & 31);
        __enable_irq();
        while (1);
```

```
}

void TA0_N_IRQHandler(void)
        {
        /***set/reset the timer***/
        }
```

Example 2: Write an Embedded C program to generate delay using UP mode of TIMER A overflow.

Solution:

```
#include "msp.h"
int main(void)
{
        WDT_A→CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;
        NVIC→ISER[0] = 1 << ((TA0_N_IRQn) & 31);

        /***Initialize the timer here under UP mode***/

        __enable_irq();
        while(1);
}
void TA0_N_IRQHandler(void)
        {
        /***Initialize the Set/Reset logic here***/
        }
```

Exercise:

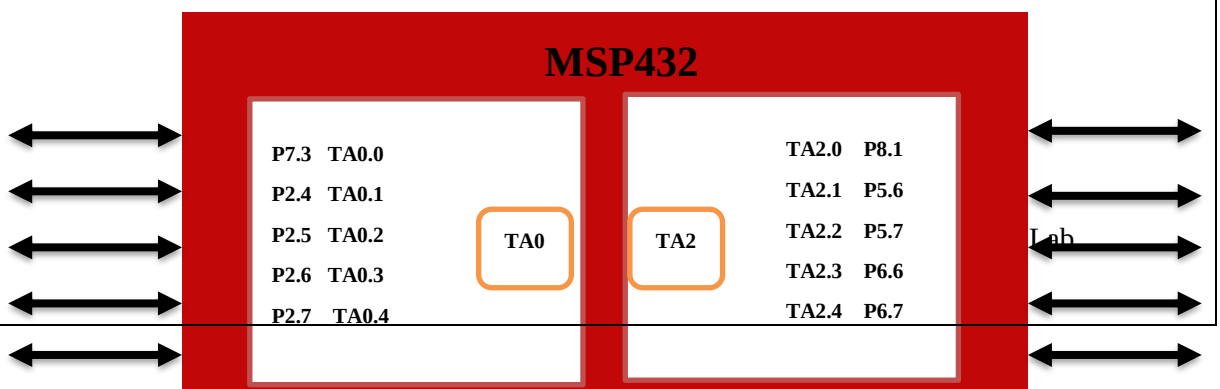
Write an embedded C code to generate the time delay of 1 second using Timer A0 overflow to blink LED connected to P2.0, Timer A1 overflow to blink P2.1, Timer A3 overflow to blink P2.2.

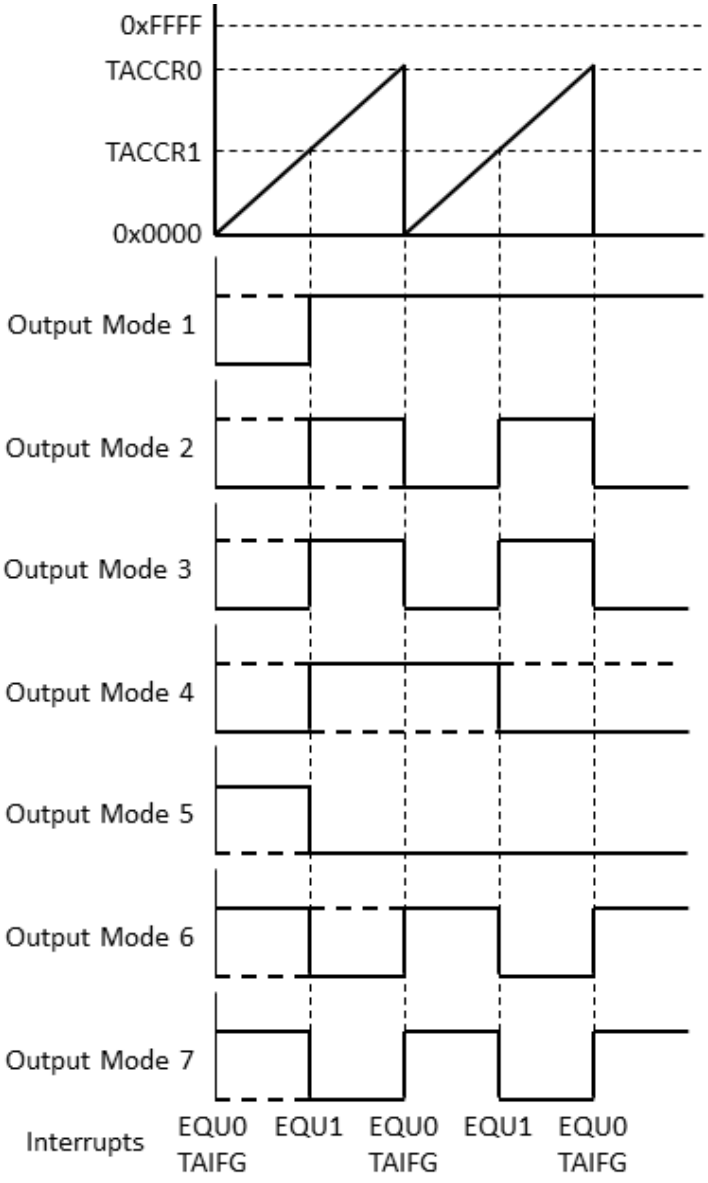
Example 3: Write an Embedded C program to generate delay using capture compare overflow flag. Find the difference between delay generated using TAIFG and CCIFG flag using UP/CONTINUOUS/ UP-DOWN mode.

Solution:

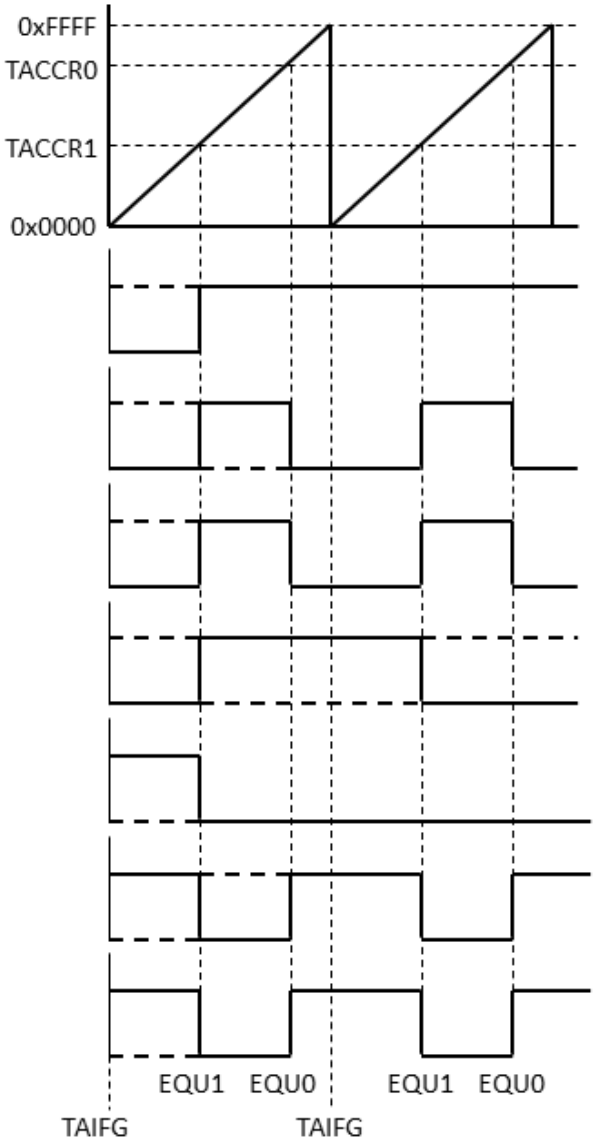
```
#include "msp.h"
int main(void)
{
        WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;
        P1->DIR |= BIT0;
        P1->OUT &= ~BIT0;
        P2->DIR |= BIT4;
        P2->SEL0 |= BIT4;

        /***Initialize the timer here***/

        __enable_irq();

        NVIC->ISER[0] |= 1 << ((TA0_N_IRQn) & 31);
        NVIC->ISER[0] |= 1 << ((TA0_0_IRQn) & 31);
        while(1);
}

void TA0_0_IRQHandler(void)
{
        /***Set/Reset and Load the value here*** /
}

void TA0_N_IRQHandler(void)
{
        /***write the logic here***/
}
```

\

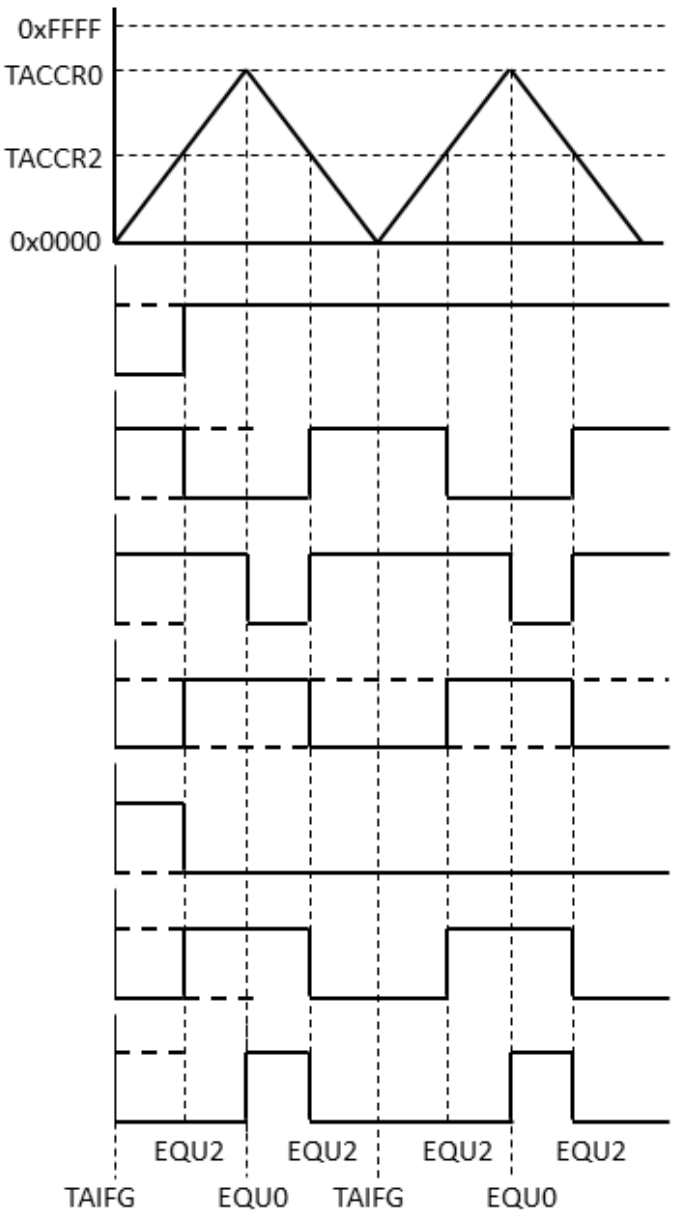




Up mode



Continuous mode



Up/down mode

Steps	Details
1.	Set the port pins output (such as LED, speakers, display devices) by setting Px->DIR register pins
2.	Modifications required in the connection. Remove jumper of P2.0 and connect the PWM out pin P2.4 to lower pin of P2.0
3.	Change the clock frequency of ACLK or SMCLK to desired frequency using clock select register settings. Example shows the SMCLK clock selected to be DCO at 12 MHz.

4.	Select the duty cycle and set the appropriate count values in CCR0 and CCRx registers. The value depends on the output mode and timer mode.
5.	Select the output mode in the TIMER_A0_CCTL[x] register.
6.	Select the clock using TASSEL, mode using MC in the Timer A control register.
7.	Stay in an infinite loop using while(1);
	For up/out7 mode duty cycle $D = \frac{TACCR_x}{TACCR0 + 1}$ frequency of PWM $\frac{f_{CLK}}{0xFFFF+1}$
	For continuous/out7 mode duty cycle $D = \frac{TACCRx+(0xFFFF-TACCR0)}{0xFFFF+1}$ frequency of PWM $\frac{f_{CLK}}{0xFFFF+1}$
	For up down/out7 mode duty cycle $\frac{TACCR0-TACCRx}{2TACCR0}$, frequency of PWM $\frac{f_{CLK}}{2TACCR0}$

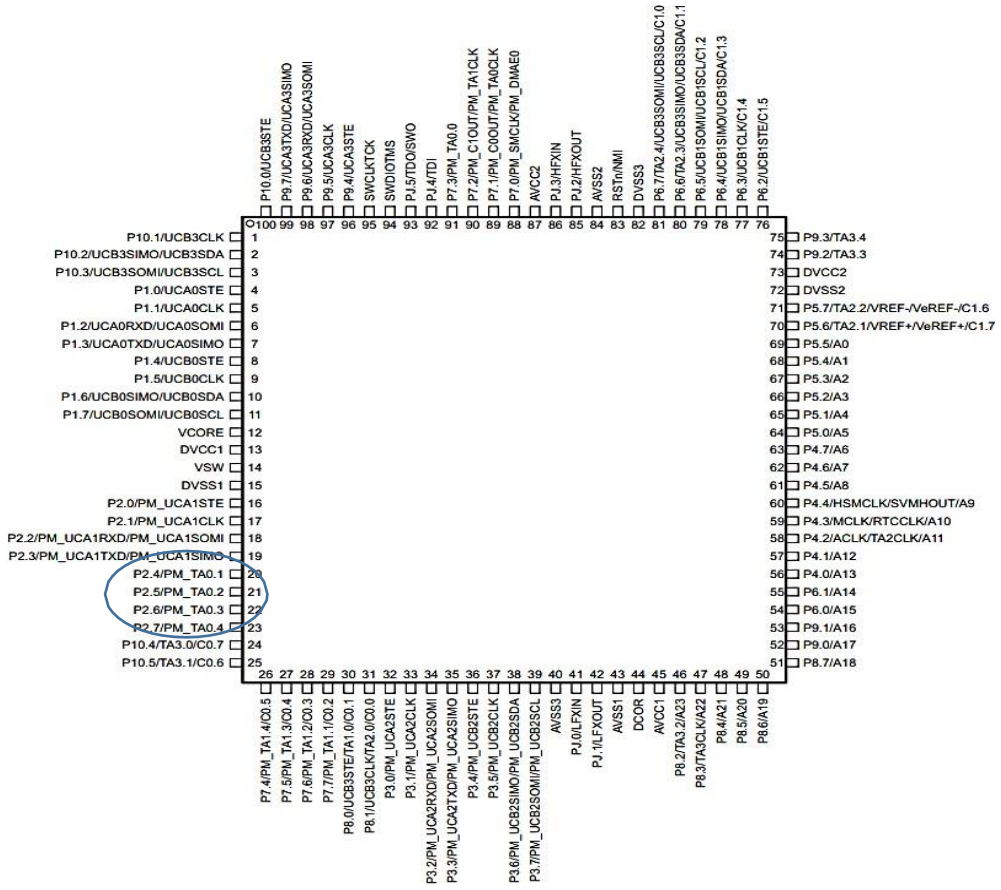
Example 3: Write an Embedded C code to generate the PWM signal with various modes (UP, Continuous, Up/Down).

```
#include "msp.h"
int main(void)
{
        WDT_A->CTL = WDT_A_CTL_PW |WDT_A_CTL_HOLD;
        P2->DIR |= BIT4;
        P2->SEL0 |= BIT4;
        P2->SEL1 &= ~(BIT4);

        /***Initialize the Timer here***/

        while(1);
```

}

Example 4: Write an Embedded C code to generate the PWM signal using TIMER A0 Module on MSP432P401R to vary the intensity of the LED when the switch P1.1 is pressed (Use interrupt for PORT1).

For more details:



Solution:

```
#include "msp.h"
int main(void)
{
        WDT_A->CTL = WDT_A_CTL_PW |WDT_A_CTL_HOLD;
        P2->DIR |= BIT4;
        P2->SEL0 |= BIT4;
        P2->SEL1 &= ~(BIT4);

        /***Initalize the PWM here***/
```

```
        while(1);

        void PORT1_IRQHandler(void)
                {
                /**Make the changes to vary the intensity of LED**/
                }
}
```

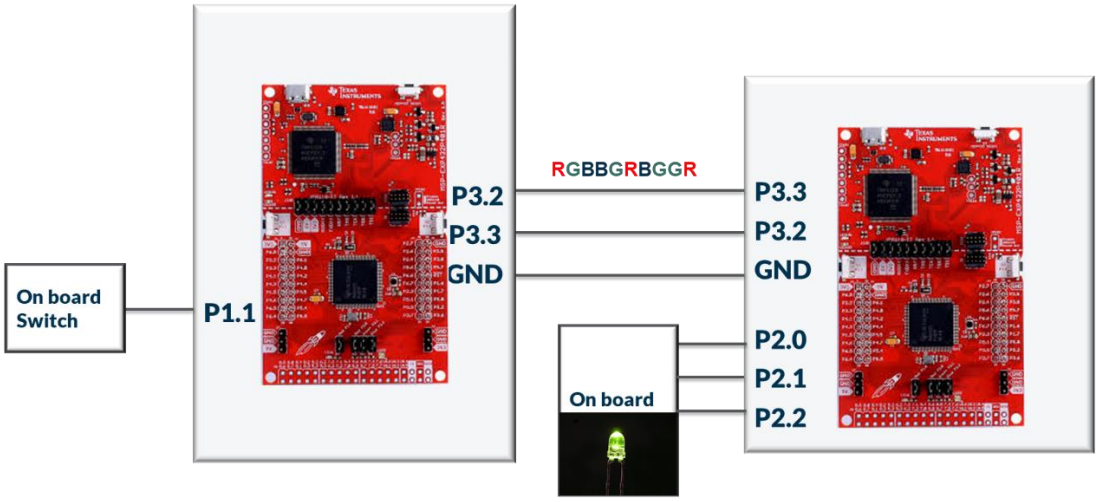
Exercise 2: Write an embedded C code to rotate servo motor in clockwise, anti-clockwise direction and to stop. Refer **https://circuitdigest.com/article/servo-motor-working-and-basics**

EXPERIMENT NO.6: PERFORMING SERIAL COMMUNICATION USING UART.

AIM: To perform serial communication using Universal Asynchronous Receiver and Transmitter.

2. The UART initialization and configuration
3. Initialize and configure GPIO related ports and pins used for the UART mode
4. Disable the EUSCI_A module by setting the **UCSWRST bit in the UCAxCTLW0 Register**
5. Set up UART protocol, for length of data byte, parity bit, stop bits, MSB/LSB first, UART mode etc
6. Compute the value to be loaded in to UCAxBRW register based on clock source and baud rate
7. Compute the baud rate and calculate the value to be loaded into UCBRFx and UCBRSx, OS16 in the UCAxMCTLW register.
8. The UART transmit enable and data transmit operation
9. The UART receive enable and data receive operation
10. Enable EUSCI_A by clearing UCSWRST in the UCAxCTLW0 register
11. Enable UART related interrupts in UCAxIE register.

Example 1: Write an Embedded C Code to generate serial communication between 2 microcontrollers using UART and blink LED on receiving a character like ‘R’ ‘G’ ‘B’.



Receiver Code:

```
#include "msp.h"
int main(void)
```

```
{
    WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD; // Stop watchdog timer
    // Unlock CS module for register access
    // Reset tuning parameters
    // Set DCO to 12MHz
    // SMCLK = DCO
   // Lock CS module from unintended accesses

/*******clock settings*******/

   P2->DIR |= 0x07;
   // Configure UART pins
   P1->SEL0 |= BIT2 | BIT3;              // set 2-UART pin as secondary function


            // Configure UART
          /*******UART configuration********/
          // Configure eUSCI clock source for SMCLK
            // Baud Rate calculation
            // 12000000/(16*9600) = 78.125
            // Fractional portion = 0.125
            // User's Guide Table 21-4: UCBRSx = 0x10
            // UCBRFx = int ( (78.125-78)*16) = 2

          /*******Baud rate Settings*********/

          EUSCI_A0->CTLW0 &= _____ // Initialize eUSCI
          EUSCI_A0->IFG &= _____       // Clear eUSCI RX interrupt flag
          EUSCI_A0->IE |= _____        // Enable USCI_A0 RX interrupt
                                       // Enable global interrupt
            __enable_irq();

            // Enable eUSCIA0 interrupt in NVIC module
            NVIC->ISER[0] = 1 << ((EUSCIA0_IRQn) & 31);
```

```
        while(1);
        }

// UART interrupt service routine
void EUSCIA0_IRQHandler(void)
{
   if (EUSCI_A0->IFG & EUSCI_A_IFG_RXIFG)
   {
      /***Write logic here***/
  }
}
```

Transmitter Code:

```
#include "msp.h"
#define S1 BIT1
char TXData[10] = "RGBBGRBGGR";
int i = 0;

int main(void)
{
        WDT_A->CTL = WDT_A_CTL_PW | WDT_A_CTL_HOLD;

        /***Enable P1.1 as an interrupt***/

        P3->SEL0 |= BIT2 | BIT3;
        // Set P3.2 and P3.3 as UCA2RXD and UCA2TXD
        // Configure SMCLK as 12 MHz from DCO

        CS->KEY = CS_KEY_VAL;
        CS->CTL0 &= _____;
        CS->CTL0 |= _____ ;
        CS->CTL1 |= _____;
        CS->KEY = 0;
```

```
        EUSCI_A2->CTLW0 |= _____  ;// Hold eUSCI module in reset state
        EUSCI_A2->CTLW0 |= _____  ;// Select SMCLK as EUSCI_A2 clock
        // Set baud-rate to 57600 from Table
        EUSCI_A2->BRW = _____;
        EUSCI_A2->MCTLW = _____;
        EUSCI_A2->CTLW0 &= _____; // Clear SWRST to resume operation

        NVIC->ISER[1] = 0x00000008; // Port P1 interrupt is enabled in NVIC
        NVIC->ISER[0] = 0x00040000; // EUSCI_A2 interrupt is enabled in NVIC

        __enable_irq(); // All interrupts are enabled
        __sleep(); // enter LPM0
}

void EUSCIA2_IRQHandler(void)
{
        /*** Write your logic here***/
}

void PORT1_IRQHandler(void)
{
        /*** Write your logic here***/
}
```

Exercise 1: Write an Embedded C code to rotate DC motor clockwise attached to MSP1 board when the signal from IR Sensor is sensed from MSP2 board.

EXPERIMENT NO.7: DEVELOPING ALGORITHMS FOR AUTONOMOUS TRANSPORTATION SYSTEM AND ADDRESS THE SOCIETAL FUNCTIONAL SAFETY.

AIM: To design the case study file for autonomous transportation system and develop the algorithm for desired societal transportation needs.

Directed Study:

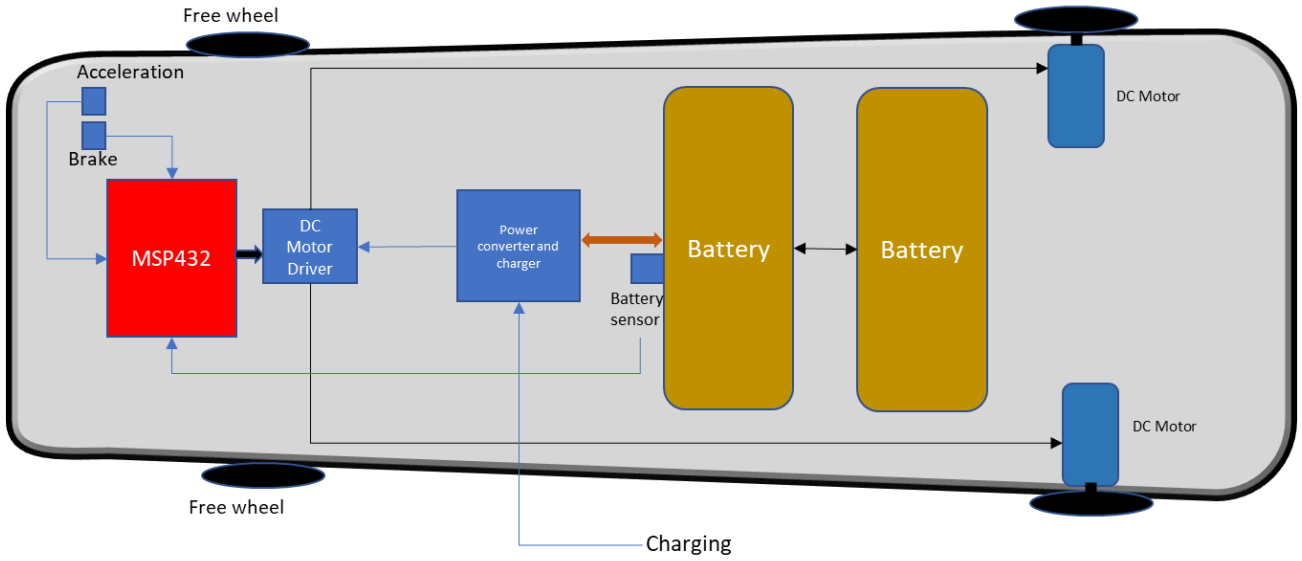
1. "Application of Functional Safety in Autonomous Vehicles Using ISO 26262 Standard: A Survey", Mukul Anil Gosavi, Benjamin B. Rhoades and James M. Conrad. Southeast Con 2018. IEEE Publisher.

2. "Autonomous Vehicle Safety: An Interdisciplinary Challenge", Philip Koopman & Michael Wagner, IEEE Intelligent transportation systems magazine, 2017.

3. "Recent Advances and Future Trends for Automotive Functional Safety Design Methodologies", Guoqi Xie, Yanwen Li , Yunbo Han, Yong Xie, Gang Zeng and Renfa Li, IEEE Transactions on Industrial Informatics, 2020.

Part 1: Electric vehicle:

Exercise 1: Design and develop algorithms for electric vehicle shown in figure:

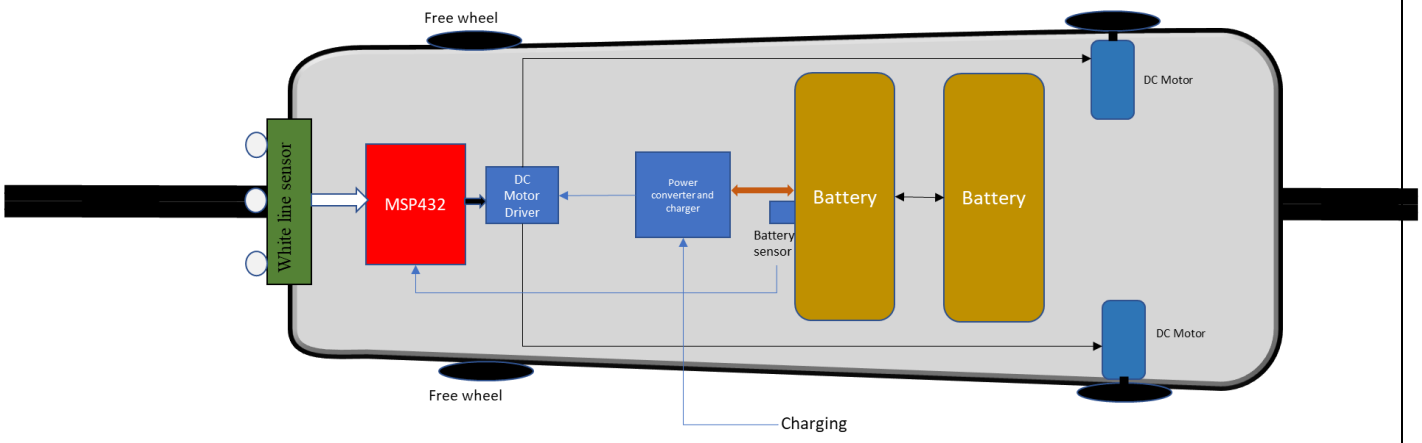


Conditions:

1. Brake and acceleration are connected as switch buttons, upon pressing the action must be taken.
2. Battery sensor monitors battery module. If LOW output is sensed, then the battery module requires charging else the sensor is high.
3. DC motor driver is connected to 2 DC motors for traction. Front the vehicle is attached to free wheels.
4. Assume the GPIO pins.
5. Calculate the machine cycles used and analyze the latency challenges.

Part 2: Autonomous Vehicles:

Exercise 2: Develop algorithms for the autonomous vehicle shown in the figure and develop case study file for the functional safety standards for each limitation in the system shown.

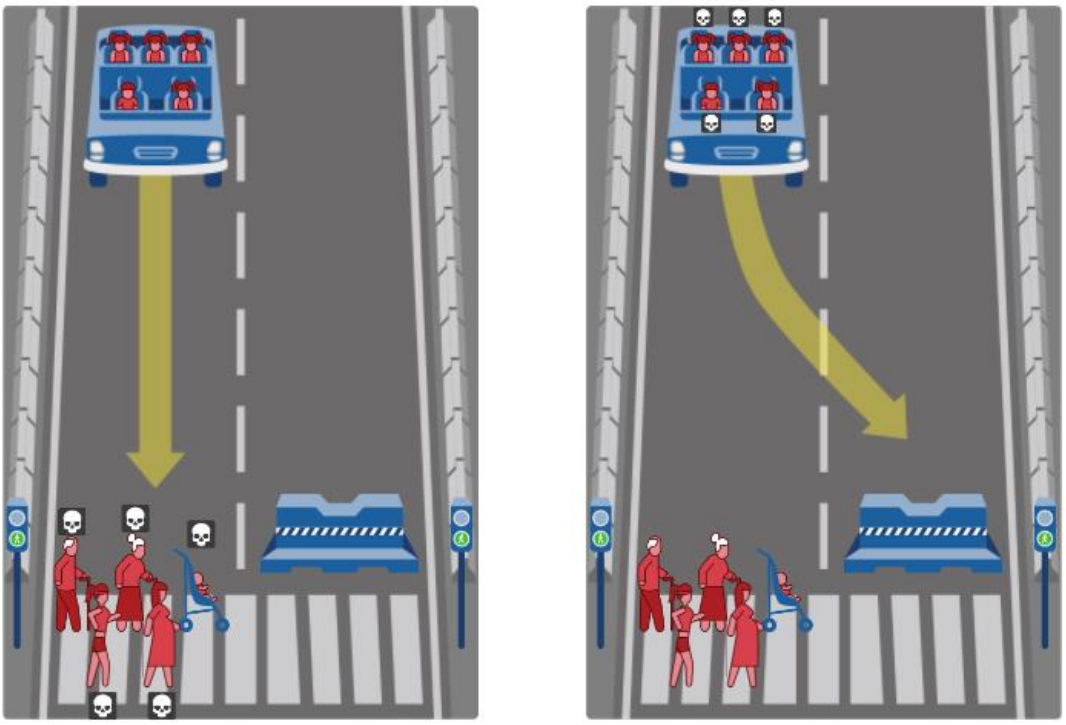


Condition:

1. White line sensor is connected to P2.0, P2.1, P2.2 and while sensing white color the sensor output is high.
2. Calculate the machine cycles used and analyze the latency challenges.

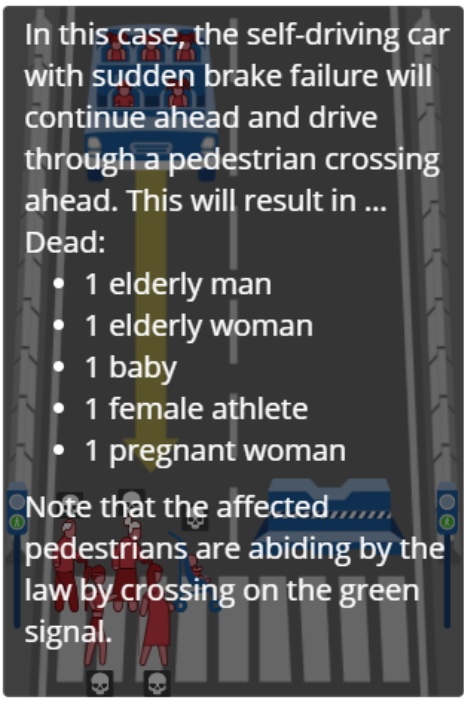
Part 3: Ethical dilemma:

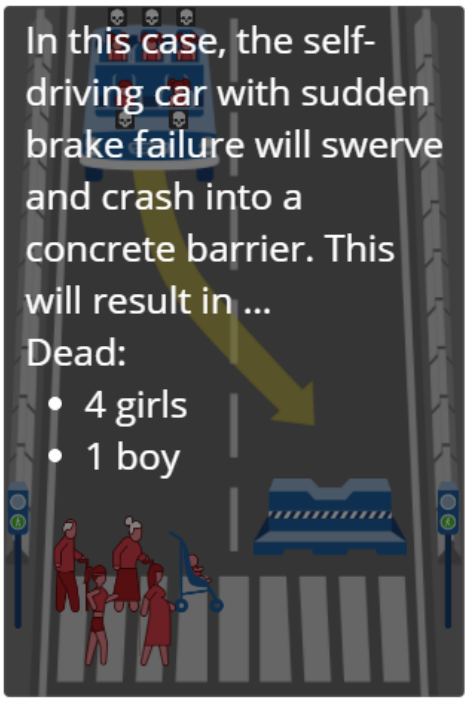
Exercise 3: Analyze and design the case study file for the ethical outcome shown in figure below, also suggest the functional safety required for such a system in terms of digital sensors.



An autonomous vehicle as a school bus

Condition:





Source: Designed using moralmachine.net.

EXPERIMENT NO.8: CASE STUDY 2: DEVELOPING ALGORITHMS FOR INDUSTRIAL AUTOMATION AND ADDRESS THE INDUSTRIAL FUNCTIONAL SAFETY.

Aim: To develop algorithms and design FSM for industrial automation.

Directed Study:

1. "Functional safety concept for hazardous systems and new challenges", Kazimierz T. Kosmowski, Journal of Loss Prevention in the Process Industries, 2006.

2. "Unified Functional Safety Assessment of Industrial Automation Systems", Zeeshan E Bhatti, Partha S Roo and Roopak Sinha. IEEE Transactions on Industrial Informatics, 2017.

3. "Functional Safety IEC 61508 / IEC 61511: The Impact to Certification and the User", Heinz Gall, IEEE/ACS International Conference on Computer Systems and Application, 2008.

Exercise 1: Develop embedded c algorithms for the following automation operation. Assume the GPIO and use NVIC & Timers wherever necessary.

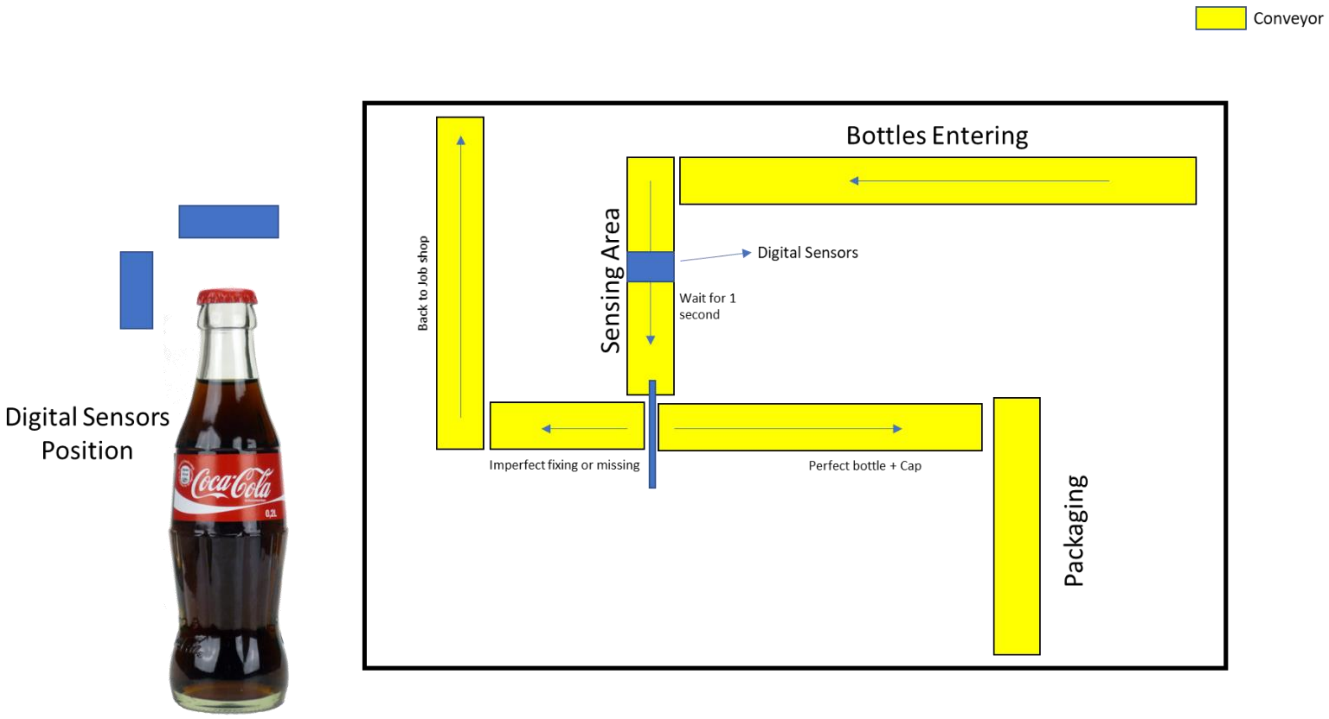
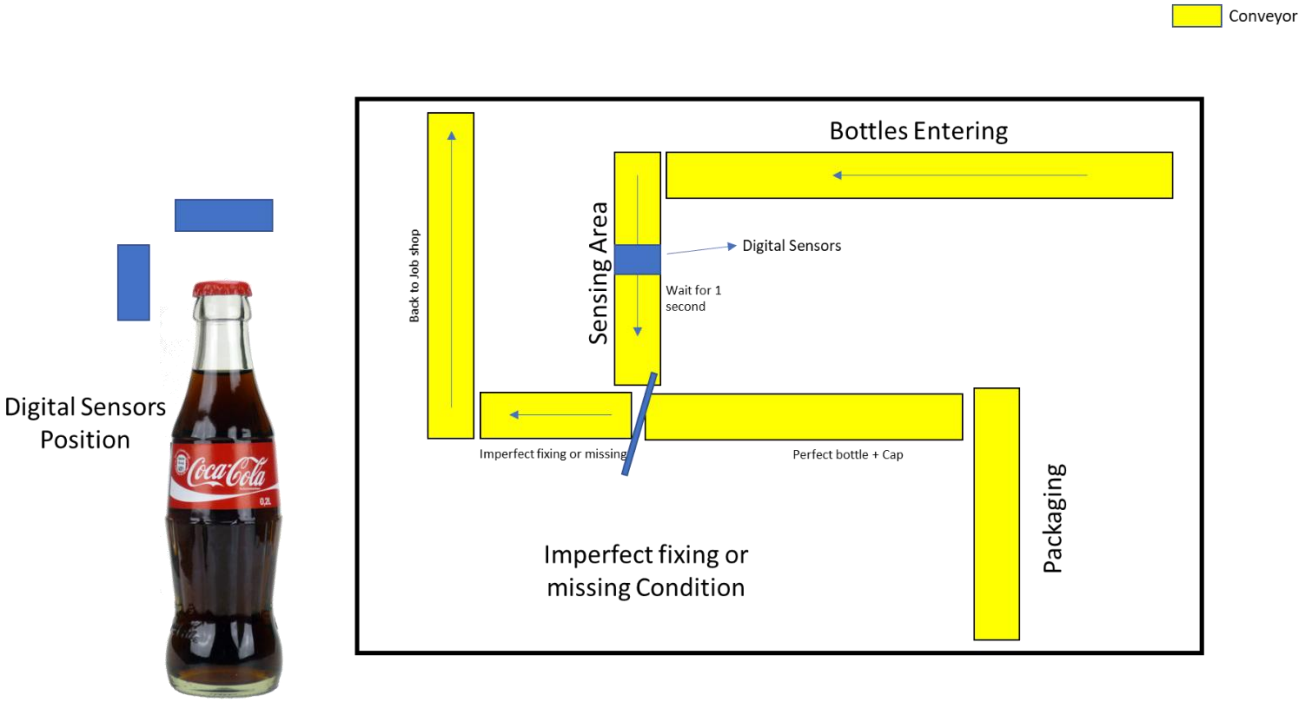


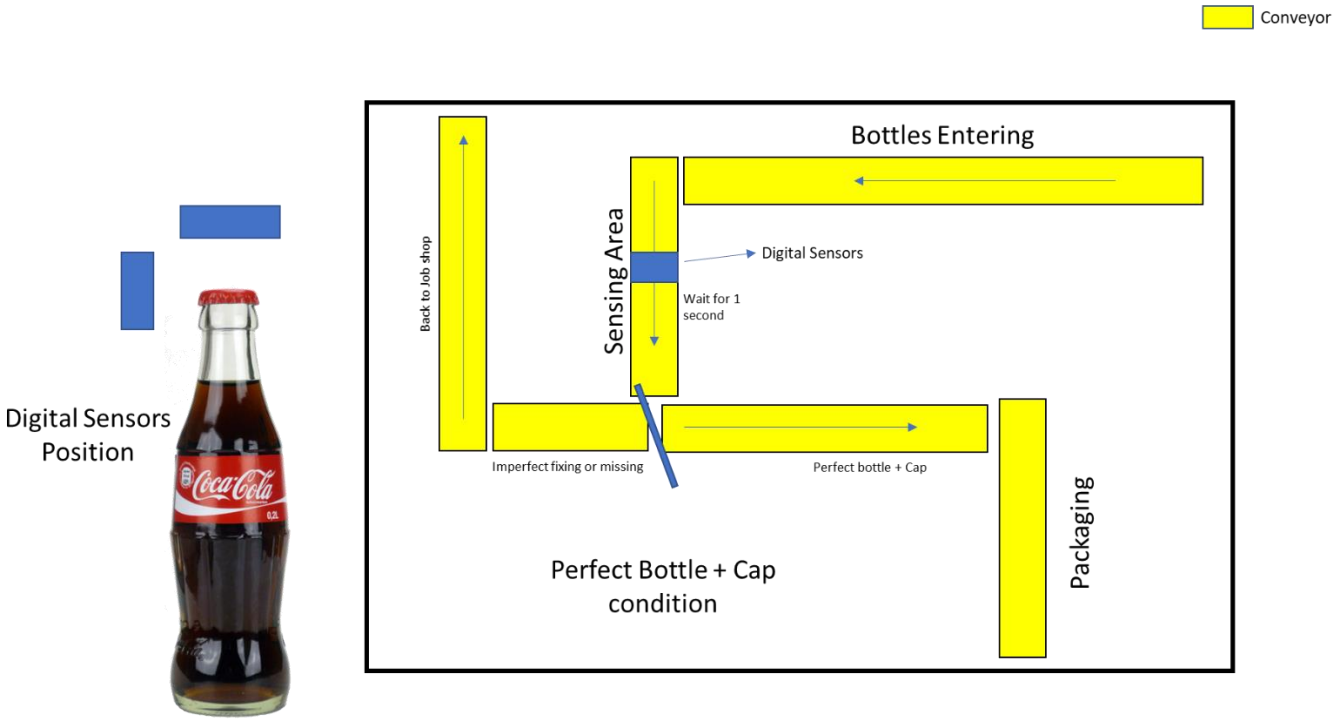
Figure: The bottle inspection line

Every conveyor is connected to a DC motor at one end and other end its free wheel. Separator is connected to a servo motor (Use PWM concept). The sensors used are digital sensors.

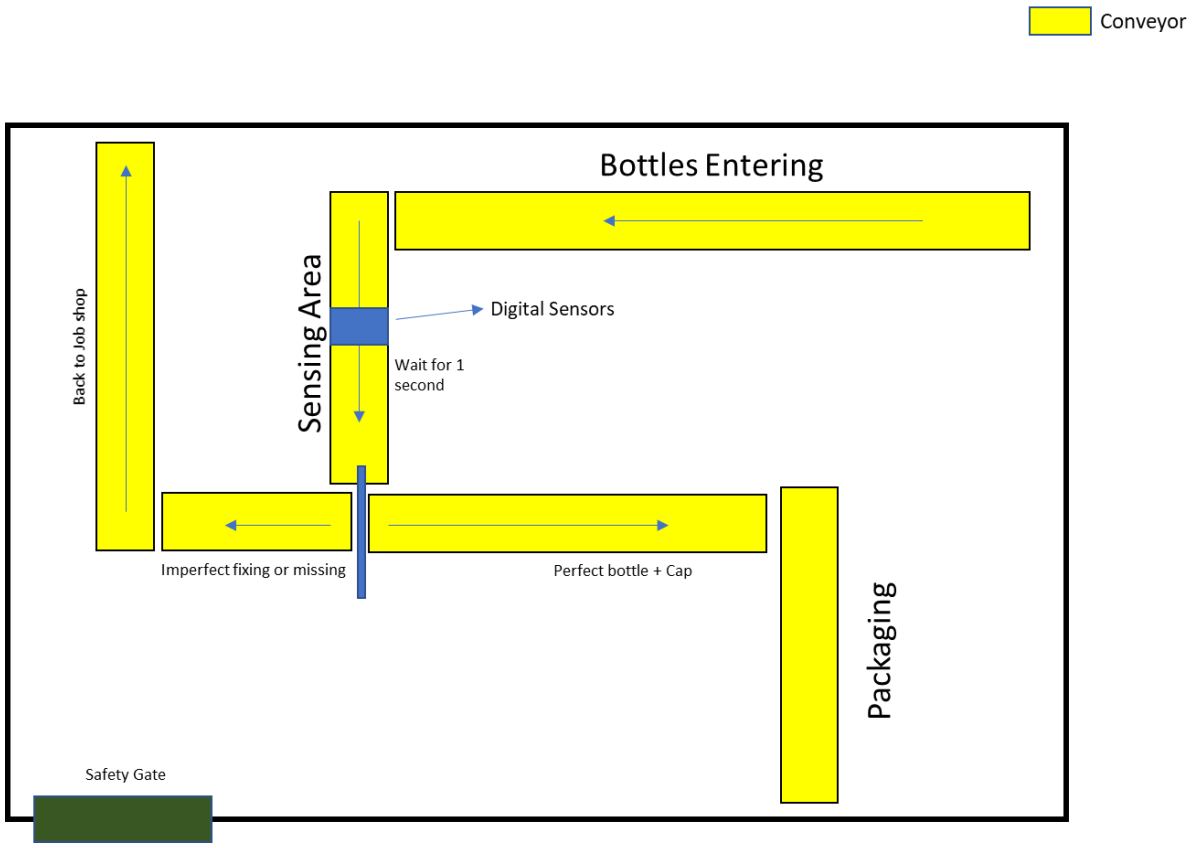
Condition 1:



Condition 2:



Exercise 2: Write an Embedded C code to perform serial communication-based safety system for the figure: The bottle inspection line.



Condition:

1. The safety gate is connected to MSP432(a) and gives a serial communication signal to MSP432(b).
2. The closing of the safety gate sends a signal across MSP432(b) to start the operation.
3. The opening of the safety gate should stop the operation/ halt the operation in MSp432(b). i.e., as shown in figure: The bottle inspection line.
4. Address the functional safety standards required.

EXPERIMENT NO.9: CASE STUDY 3: DEVELOPING ALGORITHMS FOR COLLECTING DATA FOR MEDICAL APPLICATIONS AND ADDRESS THE MEDICAL FUNCTIONAL SAFETY.

AIM: To develop algorithms for collecting data from medical applications and address medical FSM.

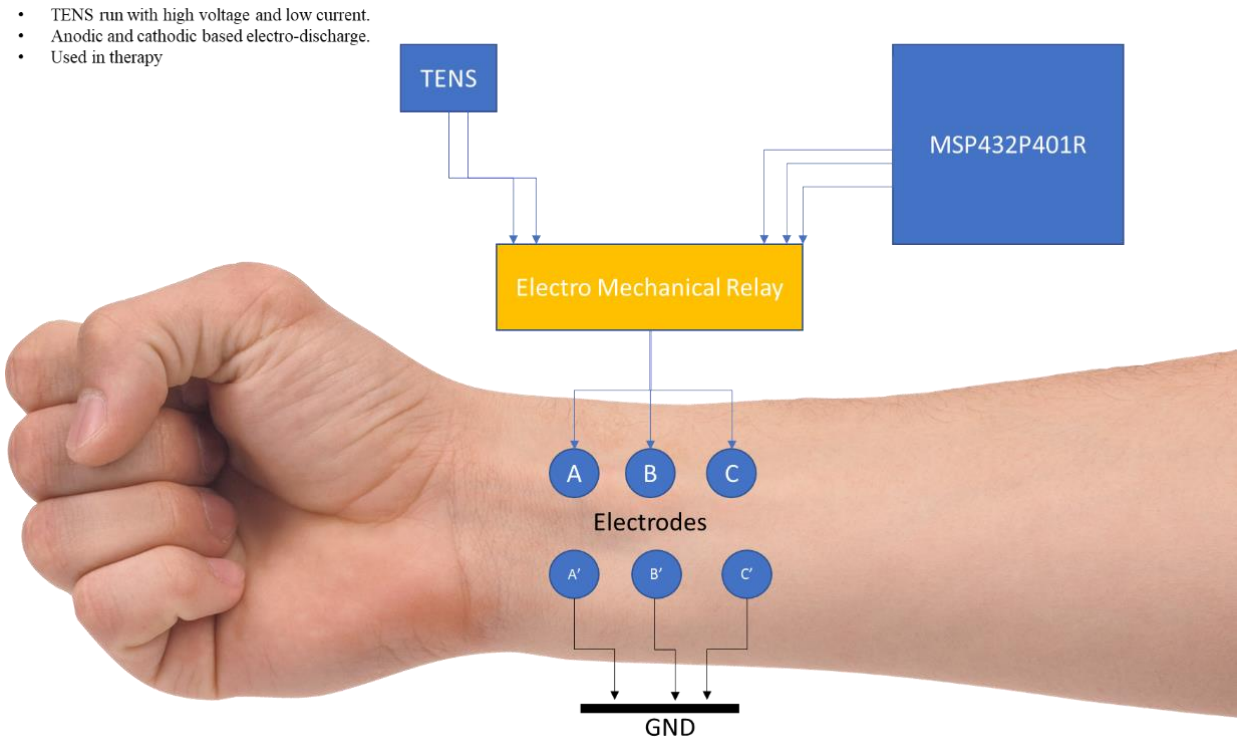
Directed Study:

1. "Implementing Functional Safety", Christof Ebert, IEEE Software, 2015.

2. "International IEC standard 62304: Medical device software – Software life cycle processes". https://webstore.iec.ch/preview/info_iec62304%7Bed1.0%7Den_d.pdf . Accessed on 19/01/2021.

3. "Regulatory Strategies for the Third Edition of IEC 60601-1", Leo Eisner, Garry Lee, Mark Leimbeck, https://www.mddionline.com/news/regulatory-strategies-third-edition-iec-60601-1, Accessed on 19/01/2021.

Exercise 1: Write an Embedded C code for performing a medical therapy using electro-nerve stimulus.



Condition:

1. Assume the GPIO connected.
2. Transcutaneous Electrical Nerve Stimulus device can connect at a time to only two electrodes.

3. Perform a random therapy by programming the MSP432PXXXR (for example: AC-BC-CB-AB-CA……,)
4. A', B', C' are grounds through which anodic discharge happen.
5. Electromechanical relay is a switching operation to change or divert the voltage generated by TENS device.

Uses: TENS is used in regenerating the neuron impulse sensation in Axon if myelin sheath stops insulating or is degenerated. Patients like ALS, Parkinson, etc., with medication also use electro therapy for regeneration.

Exercise 2: Study the TENS device from Internet sources and address the functional safety standards.

EXPERIMENT NO.10: TASK SPECIFIC/APPLICATION BASED PROGRAMMING

AIM: To design a logic for application-based programming on MSP432P401R.

Note: Use only one application (Autonomous Transportation, Industrial Automation or Medical Devices)

To be addressed in designing case study file: -

1. AIM
2. Literature – journals and articles.
3. Procedure or working principle (with block diagram for working module).
4. Code (Embedded C or Assembly with comments)
5. Ethical issues.
6. Functional Safety Standards or Regulatory.
7. Discussion and Future Scope.

Mini Project Guidelines

Outcome:

- The goal is to provide deeper understanding of the subject and its application in solving industry-based problems. In addition, it also provides hands-on experience with various tools such as Sensors, Actuators, MSP432 Launchpad.

- It also helps to improve the project skills and makes students ready for taking up final year major projects. The work includes the project proposal, independent learning, project execution, and presentation.

- Individual or a team of students will choose a mini project from real time problem. Complexity of the work depends on the number of team members.

FORMAT OF MINI PROJECT REPORT

1. **TITLE OF MINI PROJECT** (Font Size 16, Bold, Uppercase, center aligned)
2. **Margins:** Left - 1.25", Right - 1", Top & Bottom - 1".
3. **Line Spacing:** 1.5
4. Title of the report
    - Font: Times New Roman (Bold)
    - Size: 16
    - Alignment: center
5. Section Headings: First level
    - Font: Times New Roman (Bold)
    - Size: 14
    - Alignment: Left
6. Section Headings: Second level
    - Font: Times New Roman (Bold)
    - Size: 12
    - Alignment: Left
7. Text of paragraph:
    - Font: Times New Roman (Normal)
    - Size: 12
    - Alignment: Left and Right justified.
8. Figures and Tables **:**
    - Caption (placed below the figure and above the table)
    - Font : Times New Roman (Bold)
    - Size : 10 point
    - Alignment : Centered
10. Page Numbering (Right)
    - Roman numbered from certificate page to list of figures page
    - Arabic 1, 2 from first chapter (introduction) to end
11. References

- Spacing : 1.5
- Font : Times New Roman (normal)
- Size : 12 point
- [Citation number] Author's Name, "Article Title", Journal, Publisher,
- [Citation number] Author's Name, "Title of the Book", Publication, Edition, Year of Printing.

TITLE (16, BOLD)
Provide proper title that suits to your problem.

MAIN HEADING (14, Bold)

Explain the topic in own words, with relevant figures, tables, graphs, etc. (12, Normal). Figures, graphs, tables, etc. should be center aligned.

SUB-HEADING (12, BOLD)

You can use sub-headings such as Experimental Setup, Results & Discussions, Advantages & Discussions, Applications; etc.
Figures must be drawn using draw.io online editor/ paint or anything which looks professional.

CONCLUSION (14, Bold)

Summarize the topic and write concluding remarks in your own words. (12, Normal)

REFERENCES (14, Bold)

[1] List the reference papers in the order in which they are referred, including the main reference paper. (12, Normal)
[2] Author1, Author2, and others, "Title of the paper", Journal/Conference name, Volume No., Serial No., Month, Year.

- Report should be typed and printed back-to-back of A4-size papers.
- Monochrome printing is suggested.
- Minimum No. of pages: 15, Maximum No. of pages: 40.
- Total No. of copies to be made: 1.
- Before printing, upload the soft copy to me via MS team's assignment section
- (naming report to be strictly followed as Regno_Name)

The report shall be arranged under the following headings:

Scoring rubrics include the quantity, quality, complexity of the work, results, experimental analysis, clarity of presentation, and depth of understanding, report writing. Reports should contain the following material, organized logically into sections:

Title page

Abstract

Each problem should have a description of your method, solution, and the results of the experiments.

Introduction

Introduction explaining the topic, and motivation. Specific aims of your project use tables or figures to explain them. Mention why you have chosen this problem, what is the advantage of your work to society/industry. Mention the contribution for the work.

Literature

Discuss the previous works done in the area related to your work. Cite some resources that fits to your work. Describe the work in detail and explain how it works on your data. Mention the advantages and drawbacks in your words in one paragraph. Tabular format is more appreciated.

- Write the main contribution to the problem statement in your words.
- How it connects to your work.
- What is the strength you observed to choose as base work for your project?
- Mention the advantages and limitation of the paper according to authors.

Problem statement & Objectives

State the problem, its need in industry/society. List one or 2 objectives that you are going to achieve.

Methodology

Describe the work you used to solve the problem or to perform the experiments. Explain the methods/algorithms you used for experimentation. Explain the experimental setup, detailed dataset, programing plat form, algorithm. Provide the code used for experimentation with test results.

Result Analysis and Discussion

Present the data used for experimentation, make the list of data in a table. Provide images related to your work. Describe results on images and/or on your test data sets. Put quantitative results in tables or graphs. Use quantitative metrics such as accuracy, mean, standard deviation etc based on your application. If the word is based on industry/health application, then discuss health, safety, risk analysis, risk assessment in the report.

Output/Inference

Provide the output of your experiment and discuss how well it works. In addition, explain where the method fails, how it could be improved.

Conclusion ad Future work

Conclude your work by mentioning the methods you used for the work, result analysis, advantages and limitations, and possible future work.

References.

List of papers cited in the text. Use a consistent citation style - see examples of styles in published journal or conference papers.

List of Equipment's:

1. ARM Cortex M4 – MSP432P401R Launch Pad (Texas Instruments 32 Bit ARMMicrocontroller).
2. Sensors, DC motors and Servo Motors.
3. Keil Microvision 5.
4. Code Composer Studio v9.

Evaluation Guidelines:

In-Semester:

- 9 Lab experiments - Each lab 10 Marks – (5 Marks Conduction, 3 Marks Write-up, 2 Marks Viva examination). 9labs X 10 Marks = 90 Marks.
- 10^{th} Lab Experiment – 6 Marks report + 4 Marks presentation = 10 Marks
- 100 Marks Converted to 60 Marks.

End Semester:

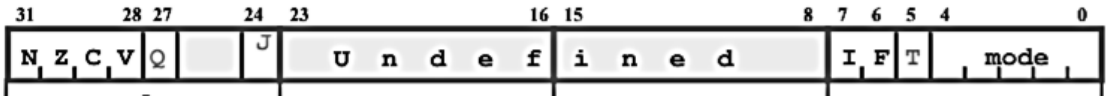
- Total 40 Marks.
- 10 Marks Viva + 18 Marks Conduction of Experiment + 12 Marks Writeup.

Annexure
Data Sheet for MSP432P401R:

Assembly Language:

Opcode	Operand

1. ARM Microcontroller – 32 Bit Length distributed over Opcode and Operand based on the distribution assembly language has 2 taxonomies, variable length and fixed length.
2. Flag to check the status of assembly language execution.



N- Negative
Z- Zero
C- Carry
V- Overflow

S. No.	Instruction	Syntax(s)
1	Data Movement Instructions	MOV{s}{cond} Rd, Operand2 MOV{cond} Rd, #imm16 MVN{s}{cond} Rd, Operand2 MOVW{cond} Rd, #imm16 MOVT{cond} Rd, #imm16
2	Shift and Rotate Instructions	LSR{s}{cond}{.size} Rd, Rm, Rs or #n ASR{s}{cond}{.size} Rd, Rm, Rs or #n LSL {s}{cond}{.size} Rd, Rm, Rs or #n ROR {s}{cond}{.size} Rd, Rm, Rs or #n RRX{s}{cond} Rd, Rm
3	Logical Instruction	AND{s}{cond}{.size} Rd, Rn, Operand2 BIC{s}{cond}{.size} Rd, Rn, Operand2 EOR{s}{cond}{.size} Rd, Rn, Operand2 ORN{s}{cond}{.size} Rd, Rn, Operand2 ORR{s}{cond}{.size} Rd, Rn, Operand2

4	Arithmetic Instruction	ADD{s}{cond}{.size} Rd, Rn, Operand2 ADC{s}{cond}{.size} Rd, Rn, Operand2 SUB{s}{cond}{.size} Rd, Rn, Operand2 SBC{s}{cond}{.size} Rd, Rn, Operand2 RSB{s}{cond}{.size} Rd, Rn, Operand2 ADDW{cond} Rd, Rn, #imm12 SUBW{cond} Rd, Rn, #imm12
5	Multiplication & Divide Instructions	UMULL{cond} RdLow, RdHigh, Rn, Rm UMLAL{cond} RdLow, RdHigh, Rn, Rm SMULL {cond} RdLow, RdHigh, Rn, Rm SMLAL{cond} RdLow, RdHigh, Rn, Rm SDIV {cond} Rd, Rn, Rm UDIV{cond} Rd, Rn, Rm
6	Compare Instructions	CMN {cond} Rn, Operand2 CMP {cond} Rn, Operand2 TEQ {cond} Rn, Operand2 TST {cond} Rn, Operand2
7	Bit Field Instructions	REV{cond} Rd, Rn REV16{cond} Rd, Rn REVSH{cond} Rd, Rn RBIT {cond} Rd, Rn SBFX{cond} Rd, Rn, #lsb, #width UBFX{cond} Rd, Rn, #lsb, #width SXTB{cond} Rd, Rm, ROR#imm SXTH{cond} Rd, Rm, ROR#imm UXTB{cond} Rd, Rm, ROR#imm UXTH{cond} Rd, Rm, ROR#imm CLZ{cond} Rd, Rm

<table>
<tr><td>8</td><td>Memory Access Instructions</td><td>LDR{type}{cond} Rt, Rn,{#offset}
STR{type}{cond} Rt, Rn,{#offset}
PUSH{cond} reglist
POP{cond} reglist</td></tr>
<tr><td>9</td><td>Branch Instruction</td><td>B <Cond> <LABEL>
BL <Cond> <LABEL>
BX <Cond> <Rm>
BLX <Cond> <Rm>
CBZ <Rn>, <LABEL>
CBNZ <Rn>, <LABEL>
TBB <Cond> [<Rn>, <Rm>]
TBH <Cond> [<Rn>, <Rm>, LSL #1]</td></tr>
<tr><td>10</td><td>Branch Conditions</td><td>
<table>
<tr><th><Cond></th><th>MEANING</th><th>FLAGS</th></tr>
<tr><td>EQ</td><td>equal</td><td>Z=1</td></tr>
<tr><td>NE</td><td>not equal</td><td>Z=0</td></tr>
<tr><td>CS, HS</td><td>carry set, unsigned higher or same</td><td>C=1</td></tr>
<tr><td>CC, LO</td><td>carry clear, unsigned lower</td><td>C=0</td></tr>
<tr><td>MI</td><td>minus, neagative</td><td>N=1</td></tr>
<tr><td>PL</td><td>plus, positive or zero</td><td>N=0</td></tr>
<tr><td>VS</td><td>overflow</td><td>V=1</td></tr>
<tr><td>VC</td><td>no overflow</td><td>V=0</td></tr>
<tr><td>HI</td><td>unsigned higher</td><td>C=1 AND Z=0</td></tr>
<tr><td>LS</td><td>unsigned lower or same</td><td>C=0 OR Z=1</td></tr>
<tr><td>GE</td><td>signed greater or equal</td><td>N=V</td></tr>
<tr><td>LT</td><td>signed less</td><td>N NOT EQUAL TO V</td></tr>
<tr><td>GT</td><td>signed greater</td><td>Z=0 AND N=V</td></tr>
<tr><td>LE</td><td>signed less or equal</td><td>Z=1 OR N NOT EQUAL TO V</td></tr>
<tr><td>AL, <OMIT></td><td>always</td><td>ANY</td></tr>
</table>
</td></tr>
</table>

Embedded C - Register Directive Specific programming for ARM Cortex M4F

TIMER 32 REGISTERS

Bits	**7**	**6**	**5**	**3-2**	**1**	**0**
TIMER32_y->CONTROL	**ENABLE**	**MODE**	**IE**	**PRESCALE**	**SIZE**	**ONESHOT**
TIMER32_CONTROL_**XX**_x	1-enable, 0-disable	0- free running mode, 1- periodic	1- enable interrupt 0- disable	00=/1, 01=/16, 10=/256	1- 32 bit, 0- 16 bit	1-oneshot, 0-wrap

25	Timer32_INT1	T32_INT1_IRQn	T32_INT1_IRQHandler
26	Timer32_INT2	T32_INT2_IRQn	T32_INT2_IRQHandler

TIMER32_y->LOAD	Load register
TIMER32_y->INTCLR	Clear flag
TIMER32_y->VALUE	Check the counter value

TIMER A REGISTERS

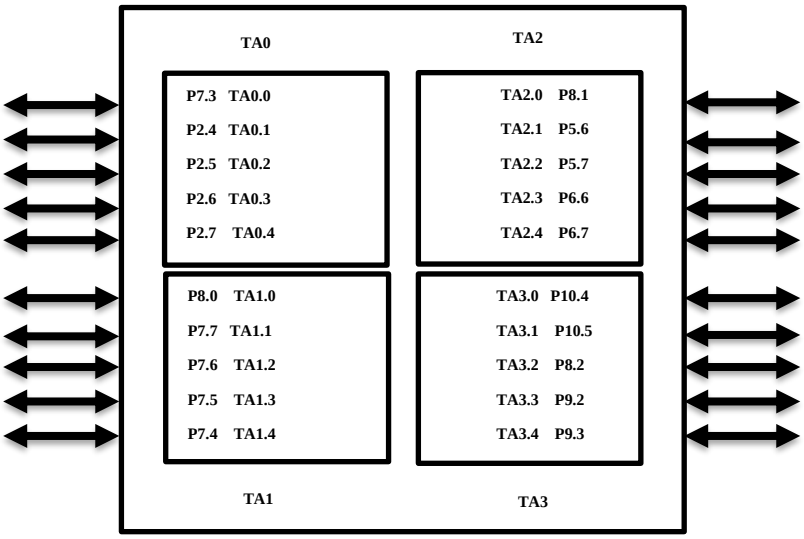
Bits	**9-8**	**7-6**	**5-4**	**2**	**1**	**0**
TIMER_A**y**->CTL	**TASSEL**	**ID**	**MC**	TA**CLR**	TA**IE**	TA**IFG**
TIMER_A_CTL_**XX**_x	00-TACLK, 01-ACLK 10-SMCLK, 11-INCLK	00-/1, 01-/2, 10-/4, 11-/8	00- stop, 01- up 10- continuous, 11- up/down	1- clear counter	1-enable timer interrupt	1-timer overflow flag set

Bits	15-14	13-12	11	10	8	7-5	4	3	2	1	0
TIMER_Ay->CCTL	**CM**	**CCIS**	**SCS**	**SCCI**	**CAP**	**OUTMOD**	**CCIE**	**CCI**	**OUT**	**COV**	**CCIFG**
TIMER_A_CCTLN_**XX**_x	00-no edge, 01-rising, 10-falling, 11- both	00-CCInA, 01-CCInB, 10-GND, 11-VCC	Synchronize timer clock	Observe synchronized input	1-capture 0-compare	000-output, 001-set, 010-toggle/reset, 011- set/reset, 100-toggle, 101-reset, 110- toggle/set, 111- reset/set	1-enable interrupt	Capture input value	Bit value		Set for capture in capture mode Set if compare is true in compare mode

TAIV Content	**Interrupt Source**	**Interrupt Flag**	**Interrupt Priority**
0x00	No interrupt pending	—	—
0x02	Capture/Compare 1	TAxCCR1 CCIFG	Highest
0x04	Capture/Compare 2	TAxCCR2 CCIFG	
0x06	Capture/Compare 3	TAxCCR3 CCIFG	
0x08	Capture/Compare 4	TAxCCR4 CCIFG	
0x0A	Capture/Compare 5	TAxCCR5 CCIFG	
0x0C	Capture/Compare 6	TAxCCR6 CCIFG	
0x0E	Timer overflow	TAxCTL TAIFG	Lowest

Bits	**3-0**
TIMER_Ay->EX0	**TAIDEX**
TIMER_A_EX0_**XX**_x	**0-/1, 1=/2, 2=/3, 3=/4, 4=/5, 5=/6, 6=/7, 7=/8.**

8	Timer_A0	TA0_0_IRQn	TA0_0_IRQHandler	TA0CCR0-CCIFG
9	Timer_A0	TA0_N_IRQn	TA0_N_IRQHandler	TA0CCR1-6, TAIFG
10	Timer_A1	TA1_0_IRQn	TA1_0_IRQHandler	TA1CCR0-CCIFG
11	Timer_A1	TA1_N_IRQn	TA1_N_IRQHandler	TA1CCR1-6, TAIFG
12	Timer_A2	TA2_0_IRQn	TA2_0_IRQHandler	TA2CCR0-CCIFG
13	Timer_A2	TA2_N_IRQn	TA2_N_IRQHandler	TA2CCR1-6, TAIFG
14	Timer_A3	TA3_0_IRQn	TA3_0_IRQHandler	TA3CCR0-CCIFG
15	Timer_A3	TA3_N_IRQn	TA3_N_IRQHandler	TA3CCR1-6, TAIFG



SYSTEM TIMER

Bits	16	2	1	0
SysTick->CTRL	**COUNTFLAG**	**CLKSOURCE**	**TICKINT**	**ENABLE**
SysTick_CTRL_**XX**_Msk	Set when counter reaches 0	1-Select clock from MCLK	1-enable interrupt	1-enable SysTick timer

WATCH DOG REGISTERS

Bits	15-8	7	6-5	4	3	2-0
WDT_A->CTL	WDT**PW**	WDT**HOLD**	WDT**SSEL**	WDT**TMSEL**	WDT**CNTCL**	WDT**IS**
WDT_A_CTL_**XX**_x	**Set password**	**1-stop WDT**	**00-SMCLK, 01- ACLK, 10- VLOCLK, 11- BCLK**	**1-interval, 0-watchdog**	**Clear counter to 0**	**000=/** 2^{31} **001-/** 2^{27} **……** **111-** 2^{6} **/** 2^{x} **(x=31,27,23,19,15,13,9,6)**

3	WDT_A	WDT_A_IRQn	WDT_A_IRQHandler

CLOCK SYSTEM

CLOCK	USE	CLOCK SOURCE
MCLK	CPU, DMA	LFXT, VLO, REFO, DCO, MODSC, HFXT
SMCLK	Peripherals	LFXT, VLO, REFO, DCO, MODSC, HFXT
HSMCLK	Peripherals	LFXT, VLO, REFO, DCO, MODSC, HFXT
ACLK	Peripherals	LFXTCLK, VLOCLK, or REFOCLK
BCLK	Peripherals	LFXT or REFO

CLOCK	FREQUENCY
DCO	1–48 MHz
HFXT	1–48 MHz
LFXT	32 kHz
MODOSC	24 MHz
SYSOSC	5 MHz
REFO	32–128 kHz
VLO	10 kHz

Bits	23	22	18-16	9-0
CS->CTL0	**DCOEN**	**DCORES**	**DCORSEL**	**DCOTUNE**
CS_CTL0_**XX**_x	1-enable DCO clock source		0-1.5MHz,1-3MHz, 2-6MHz, 3-12MHz, 4-24MHz, 5- 48MHz	

Bits	**30-28**	**26-24**	**22-20**	**18-16**	**12**	**10-8**	**6-4**	**2-0**
CS->CTL1	**DIVS**	**DIVA**	**DIVHS**	**DIVM**	**SELB**	**SELA**	**SELS**	**SELM**

CS_CTL1_**XX**_x	x=/ 2^x for SMCLK	x=/ 2^x for ACLK	x=/ 2^x for HSMCLK	x=/ 2^x for MCLK	BCLK 1- REFO 0- LFXT	ACLK 000- LFXT/REFO 001- VLO 010- REFO	SMCLK, HSMCLK 000- LFXT/REFO, 001- VLO, 010- REFO, 011- DCO, 100- MODOSC, 101- HFXT/REFO	MCLK 000- LFXT/REFO 001- VLO, 010- REFO 011- DCO, 100- MODOSC 101- HFXT/DCO

UART REGISTERS

Bits	15-8	7-4	0
EUSCI_Ax->MCTLW	UC**BRS**x	UC**BRF**x	UC**OS16**
EUSCI_A_MCTLW_**XX**	Modulation property for BITCLK XX-BRS_OFS	Modulation property for BITCLK16 XX-BRF_OFS	1- oversampling, 0- disable over sampling

Bits	**15**	**14**	**13**	**12**	**11**
EUSCI_Ax->CTLW0	UC**PEN**	UC**PAR**	UC**MSB**	UC7BIT (UC**SEVENBIT**)	UC**SPB**
EUSCI_A_CTLW0_**XX**_x	1- enable parity	1-even parity, 0-odd parity	1- MSB first, 0- LSB first	1- 7 bits, 0- 8 bits	1- 2 stop bits, 0- 1 stop bit
Bits	**10-9**	**8**	**7-6**	**0**	
Bits	UC**MODE**x	UC**SYNC**	UC**SSEL**x	UC**SWRST**	
	00-UART, 01-idle, 10-address, 11- automatic baud rate	1- asynchronous 0- synchronous	0-UCLK, 1-ACLK, 2-SMCLK	1- reset the module	

Start Bit	D0…D6	D7	Address Bit	Parity Bit	Stop Bit	Second Stop Bit

16	EUSCI_A0	EUSCIA0_IRQn	EUSCIA0_IRQHandler
17	EUSCI_A1	EUSCIA1_IRQn	EUSCIA1_IRQHandler
18	EUSCI_A2	EUSCIA2_IRQn	EUSCIA2_IRQHandler
19	EUSCI_A3	EUSCIA3_IRQn	EUSCIA3_IRQHandler

PIN	PxSEL1=0, PxSEL0=1
P1.2	UCA0RXD
P1.3	UCA0TXD
P2.2	UCA1RXD
P2.3	UCA1TXD
P3.2	UCA2RXD
P3.3	UCA2TXD
P9.6	UCA3RXD
P9.7	UCA3TXD

UCIVx	Interrupt Source	Interrupt Flag	Interrupt Priority
0x00	No interrupt pending		
0x02	Receive buffer full	UCRXIFG	Highest
0x04	Transmit buffer empty	UCTXIFG	
0x06	Start bit received	UCSTTIFG	
0x08	Transmit complete	UCTXCPTIFG	Lowest

Fractional Portion of N	UCBRSx	Fractional Portion of N	UCBRSx
0.0000	0x00	0.5002	0xAA
0.0529	0x01	0.5715	0x6B

0.0715	0x02	0.6003	0xAD
0.0835	0x04	0.6254	0xB5
0.1001	0x08	0.6432	0xB6
0.1252	0x10	0.6667	0xD6
0.1430	0x20	0.7001	0xB7
0.1670	0x11	0.7147	0xBB
0.2147	0x21	0.7503	0xDD
0.2224	0x22	0.7861	0xED
0.2503	0x44	0.8004	0xEE
0.3000	0x25	0.8333	0xBF
0.3335	0x49	0.8464	0xDF
0.3575	0x4A	0.8572	0xEF
0.3753	0x52	0.8751	0xF7
0.4003	0x92	0.9004	0xFB
0.4286	0x53	0.9170	0xFD
0.4378	0x55	0.9288	0xFE

EUSCI_Ax->BRW	**Baudrate control word**
EUSCI_Ax->RXBUF (7:0)	Receiver buffer
EUSCI_Ax->TXBUF (7:0)	Transmit buffer

Bits	1	0
EUSCI_Ax->IE	UC**TXIE**	UC**RXIE**
EUSCI_A_IE_**XX**	1- enable transmit interrupt 0- disable transmit interrupt	1- enable receive interrupt 0- disable receive interrupt

Bits	1	0
EUSCI_Ax->IFG	UC**TXIFG**	UC**RXIFG**
EUSCI_A_IFG_**XX**	Set when transmit buffer is empty. Cleared in the ISR.	Set when receive buffer is full. Cleared in the ISR.