

**MTE 2161: MICROCONTROLLER LAB**

**THIRD SEMESTER**

**NAME: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**REG NO: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**ROLL NO: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**DEPARTMENT OF**

**MECHATRONICS ENGINNEERING**

**LIST OF EXPERIMENTS:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Sr No.** | **TITLE OF EXPERIMENT** | **DATE** | **PAGE NO.** | **FACULTY**  **SIGN** | **MARKS** |
| 1. | INTRODUCTION TO MSP432P401R SIMULATION SOFTWARE |  |  |  |  |
| 2. | INTRODUCTION TO ARITHMETIC OPERATIONS USING MSP432P401R |  |  |  |  |
| 3. | INTRODUCTION TO COMPARISON AND LOGICAL OPERATIONS USING MSP432P401R |  |  |  |  |
| 4. | INTRODUCTION TO DATA MOVEMENT |  |  |  |  |
| 5. | ADDRESSING MODES OF MSP432P401R |  |  |  |  |
| 6. | INTRODUCTION OF MSP432P401R WITH EMBEDDED C PROGRAMMING USING CODE COMPOSER STUDIO |  |  |  |  |
| 7. | USE OF DELAY IN MSP432P401R |  |  |  |  |
| 8. | READING I/O PORTS IN MSP432P401R |  |  |  |  |
| 9. | PERFORMING TIMERS AND COUNTER OPERATION FOR MSP432P401R |  |  |  |  |
| 10. | TASK SPECIFIC/APPLICATION BASED PROGRAMMING |  |  |  |  |

**EXPERIMENT NO.1: INTRODUCTION TO MSP432P401R SIMULATION SOFTWARE**

**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:

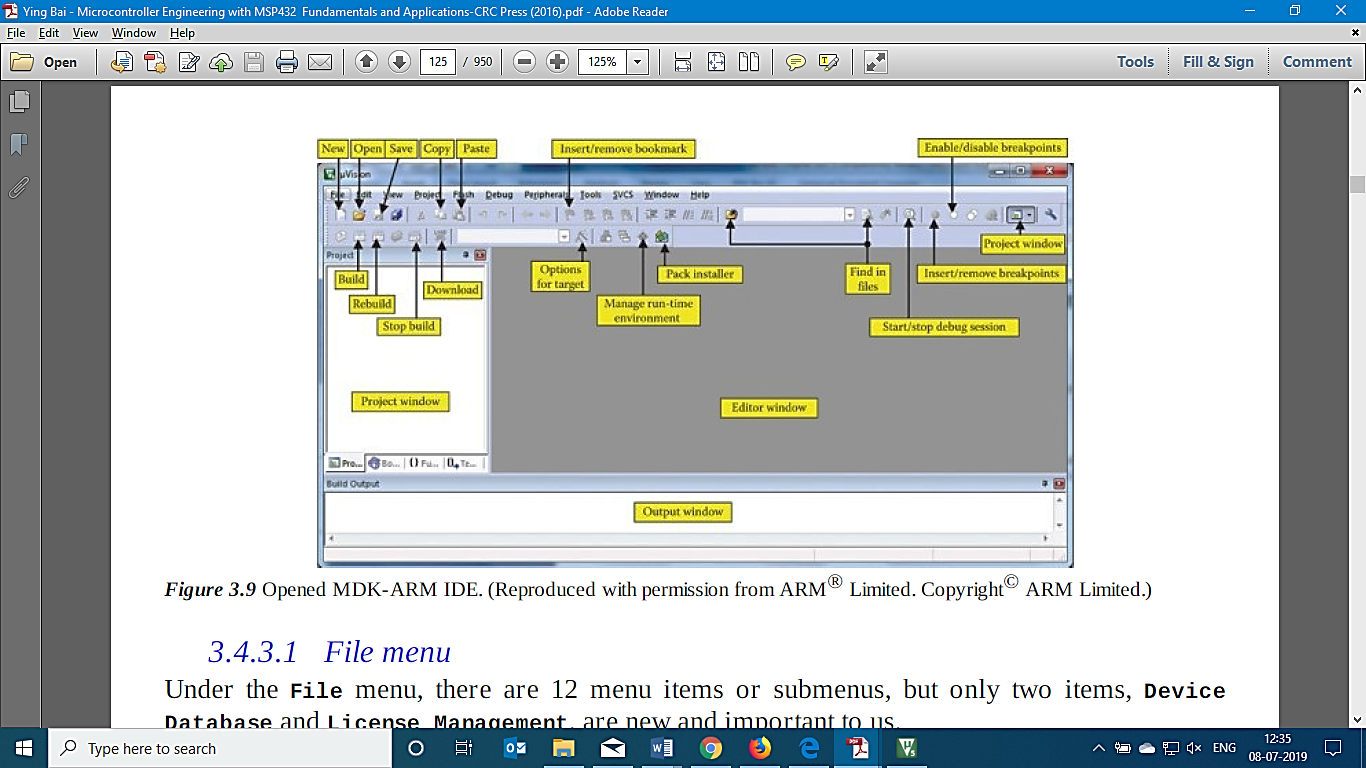
1. Keil MDK-ARM Microcontroller Development Kit (MDK) IDE.
2. Texas Instruments’ Code Composer Studio™ (CCS) IDE.
3. IAR Embedded Workbench for ARM.
4. Mentor Graphics Sourcery Code Bench.
5. GNU Compiler Collection (GCC).

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 must and should requirement for programming the internal/external modes.

**STEPS OF INSTRUCTIONS:**

1. Open Keil MDK ARM µVision 5.

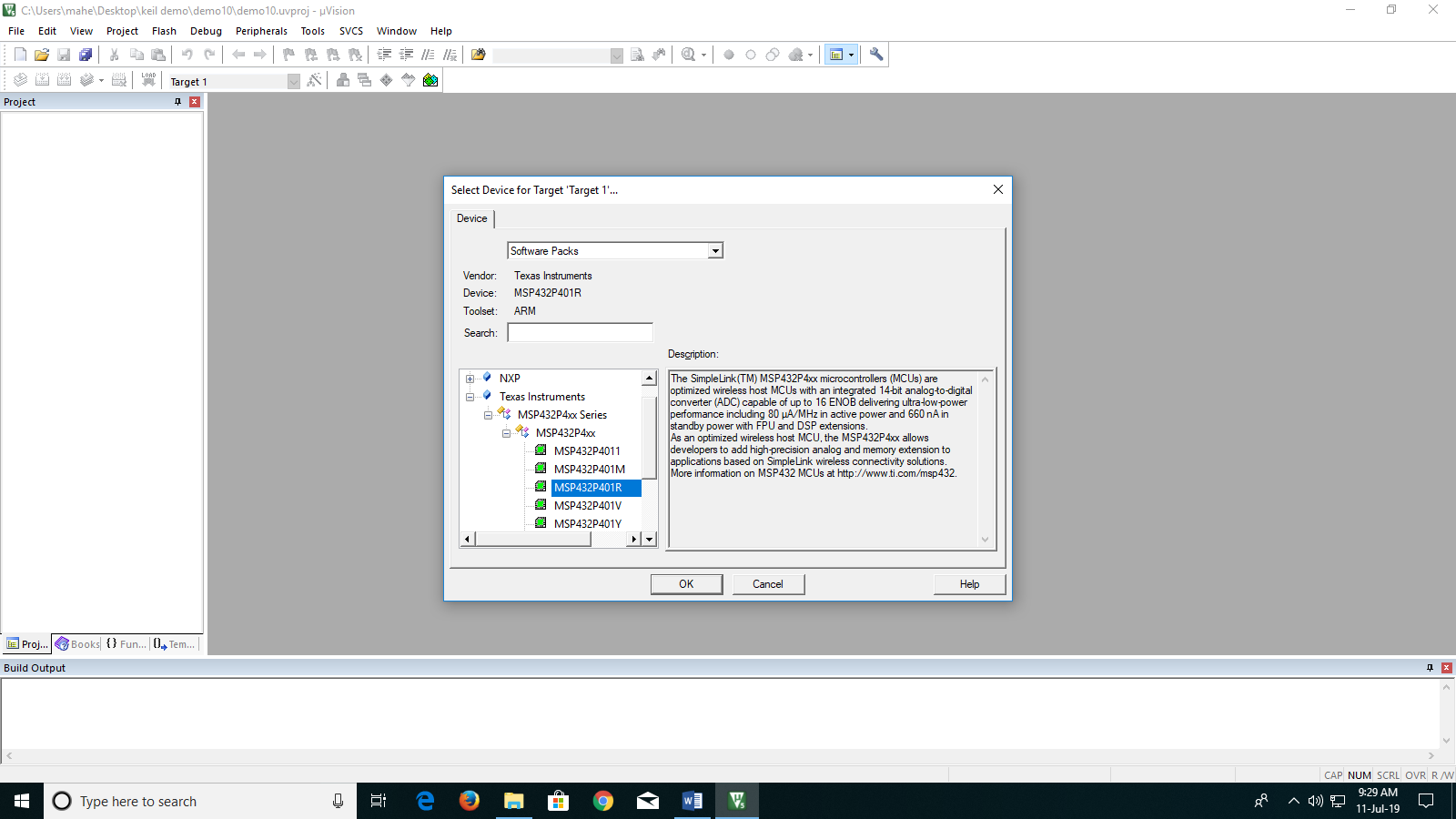


2. Click on the new μVersion Project item.

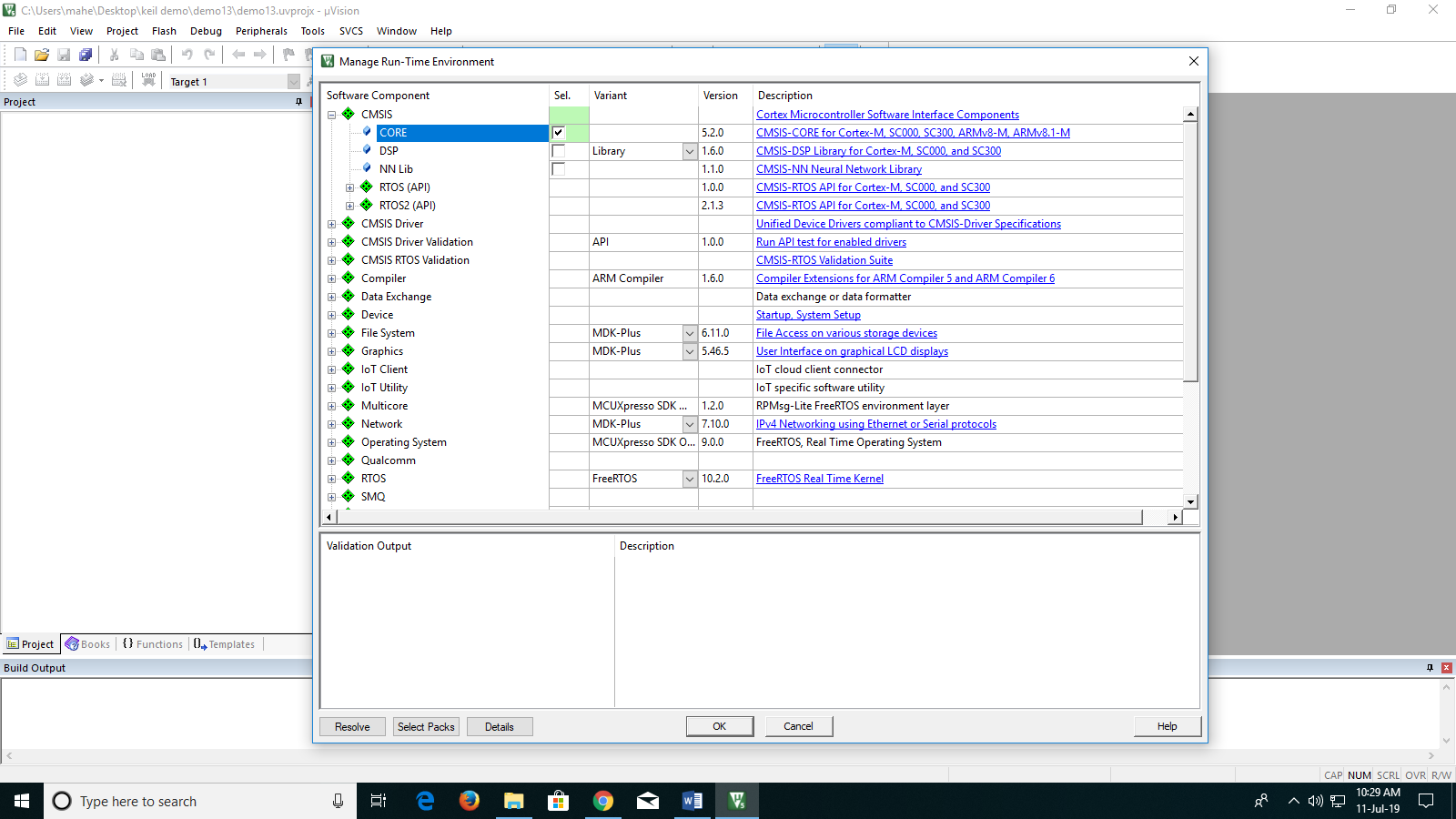
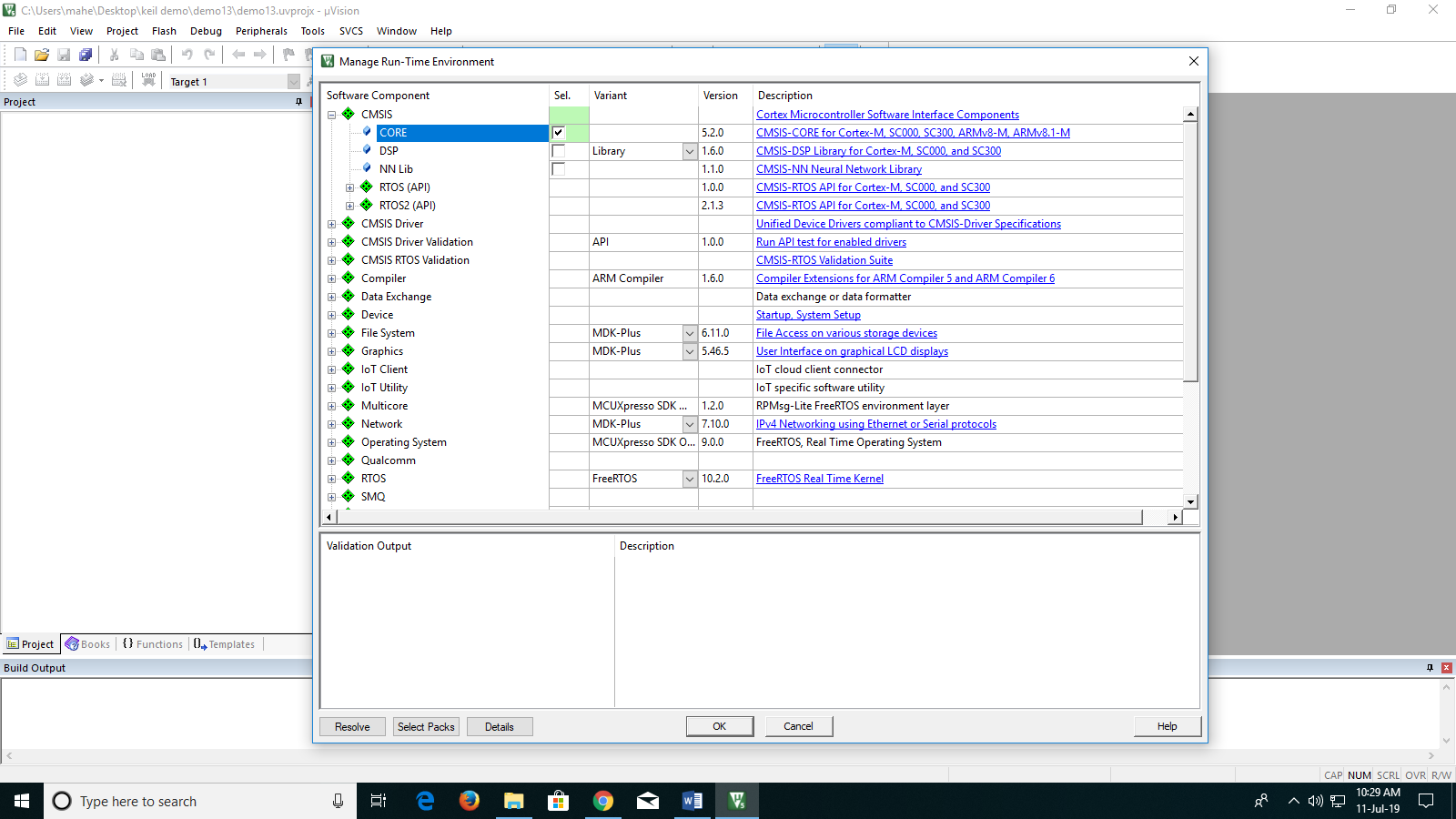
3. Create a folder in drive with registration number.

4. Create another folder as “Experiment 1” and save file name as introduction (format type **.uvproj**).

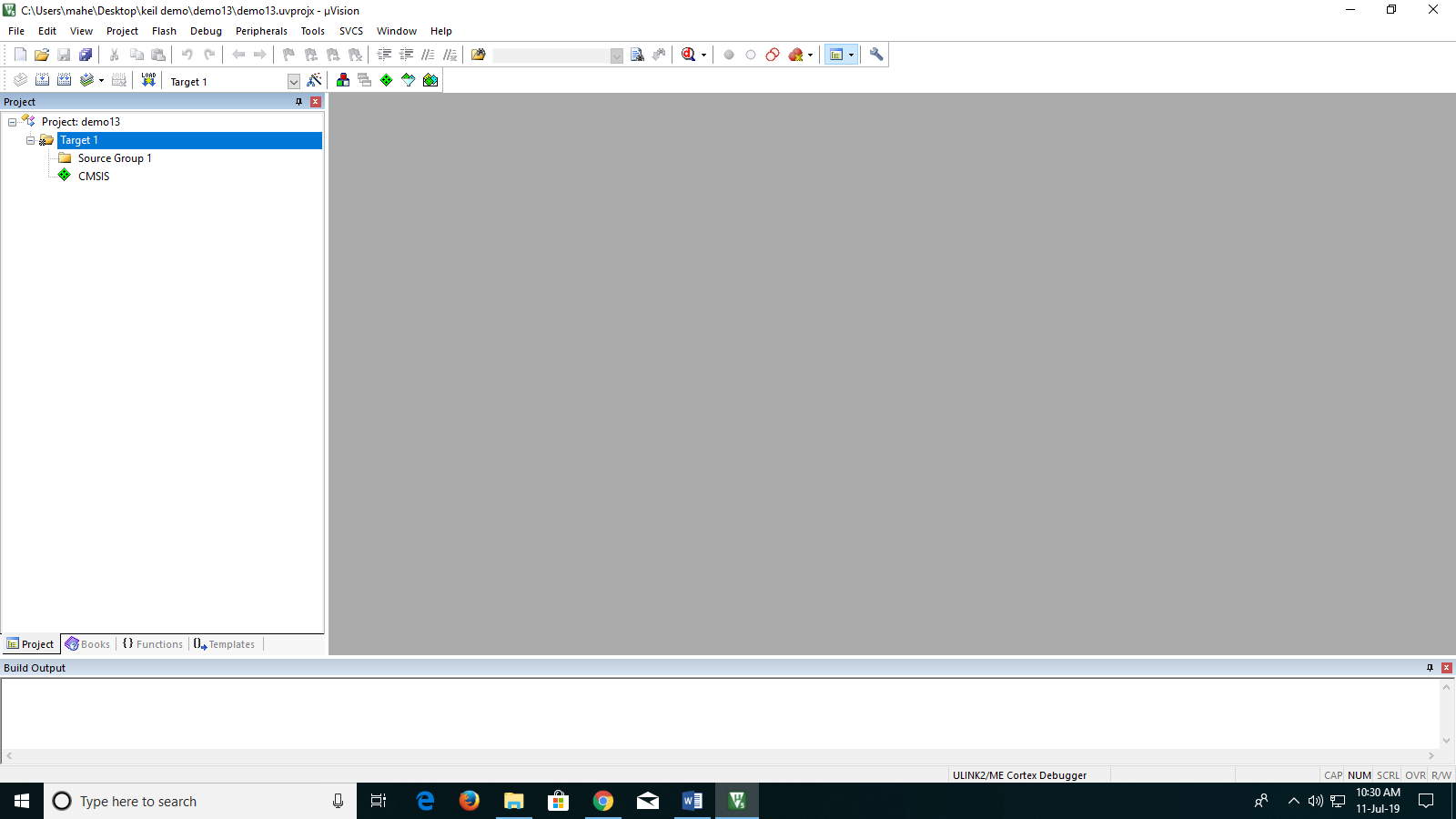
5. The following save option will open up a device selection window. Select MSP432P401R under vendor option.



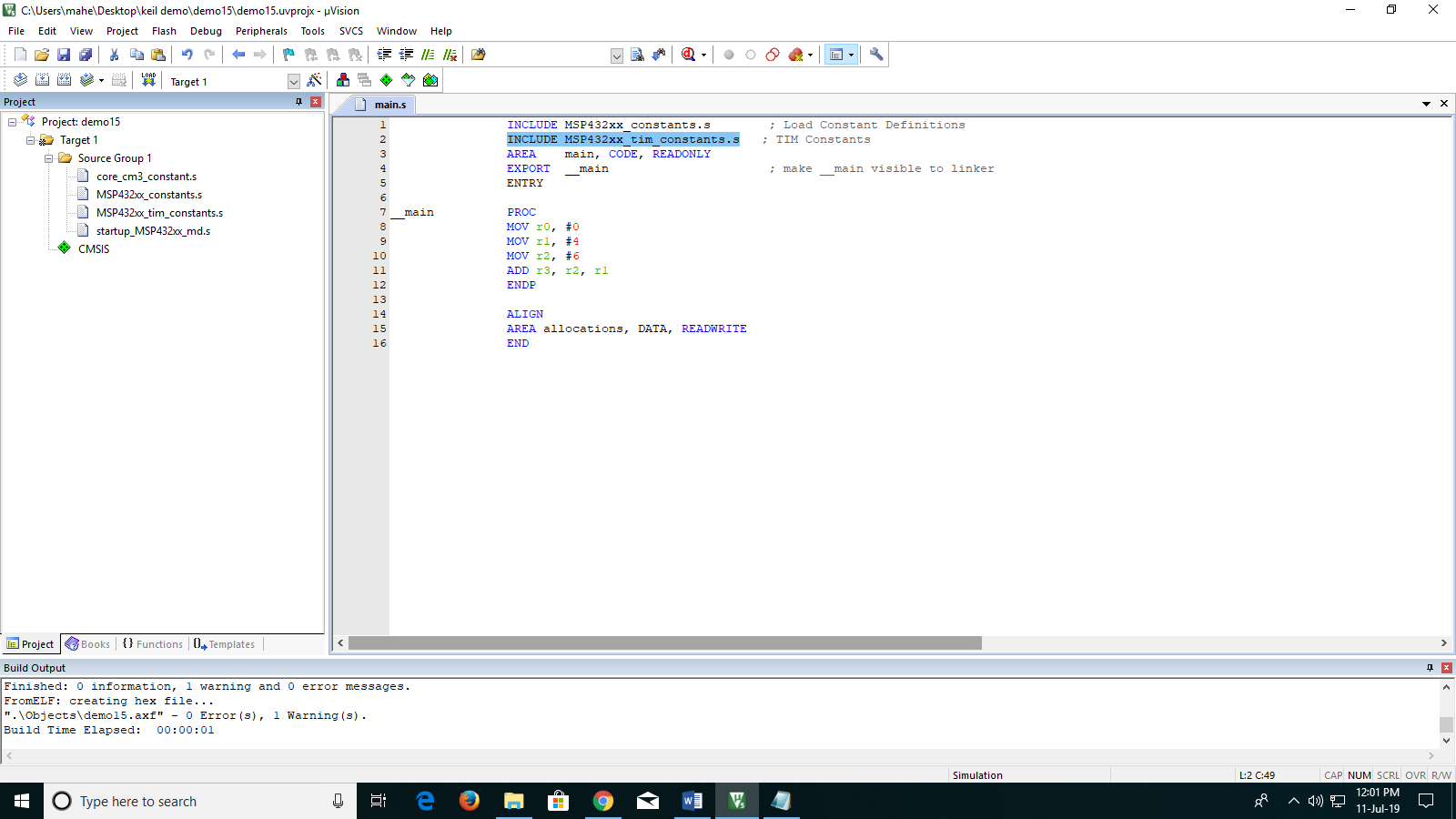
1. Select the device and specific target, also check the specifications as per data sheet provided over right side in description box.
2. A run time manager will open after selecting the micro controller.



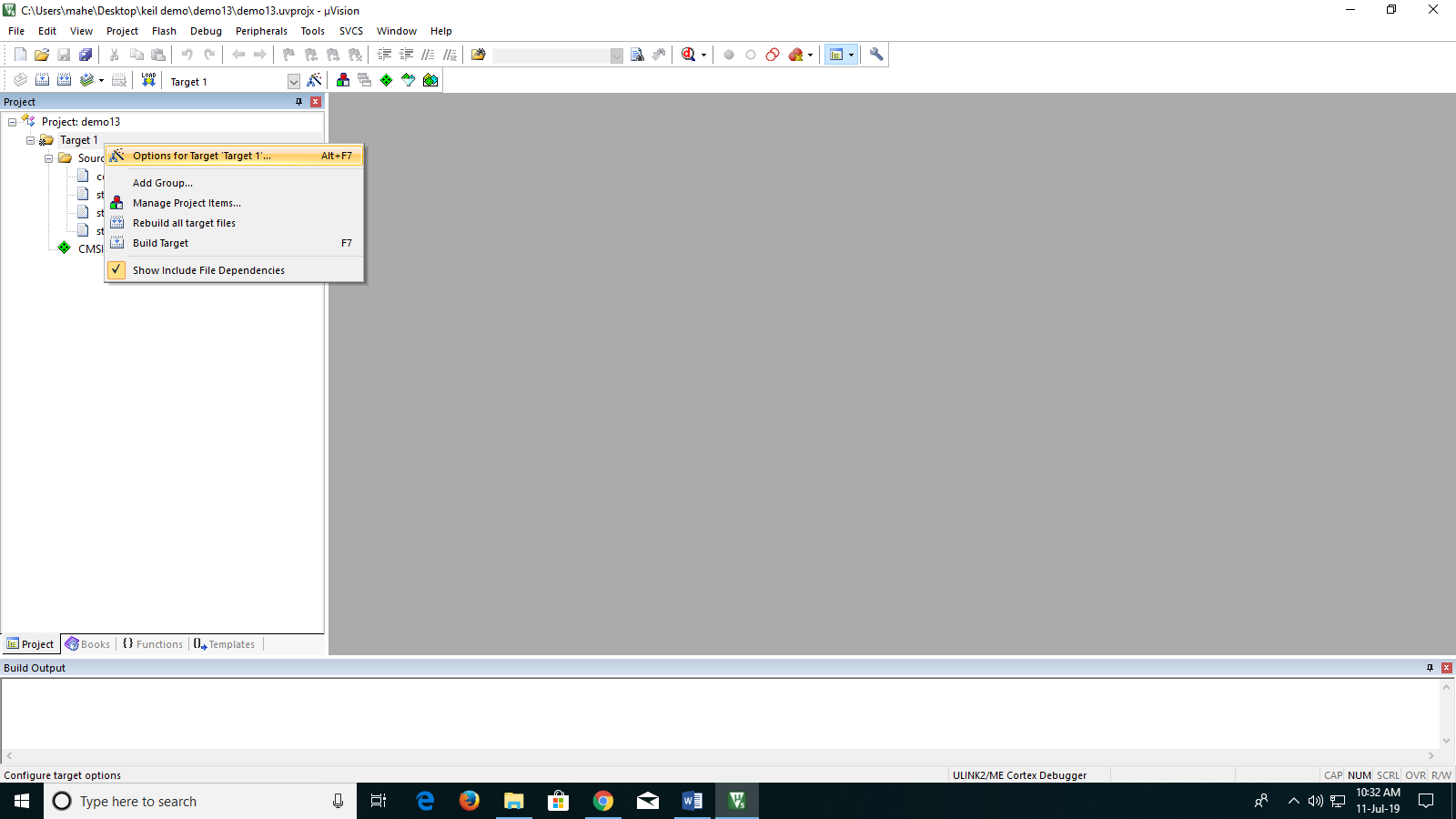
1. Under CMSIS (Cortex Microcontroller Software Interface Standard) select core.
2. These files are required for simulation software to understand the algorithm based on core compiling and through Assembly or C programming instruction sets.



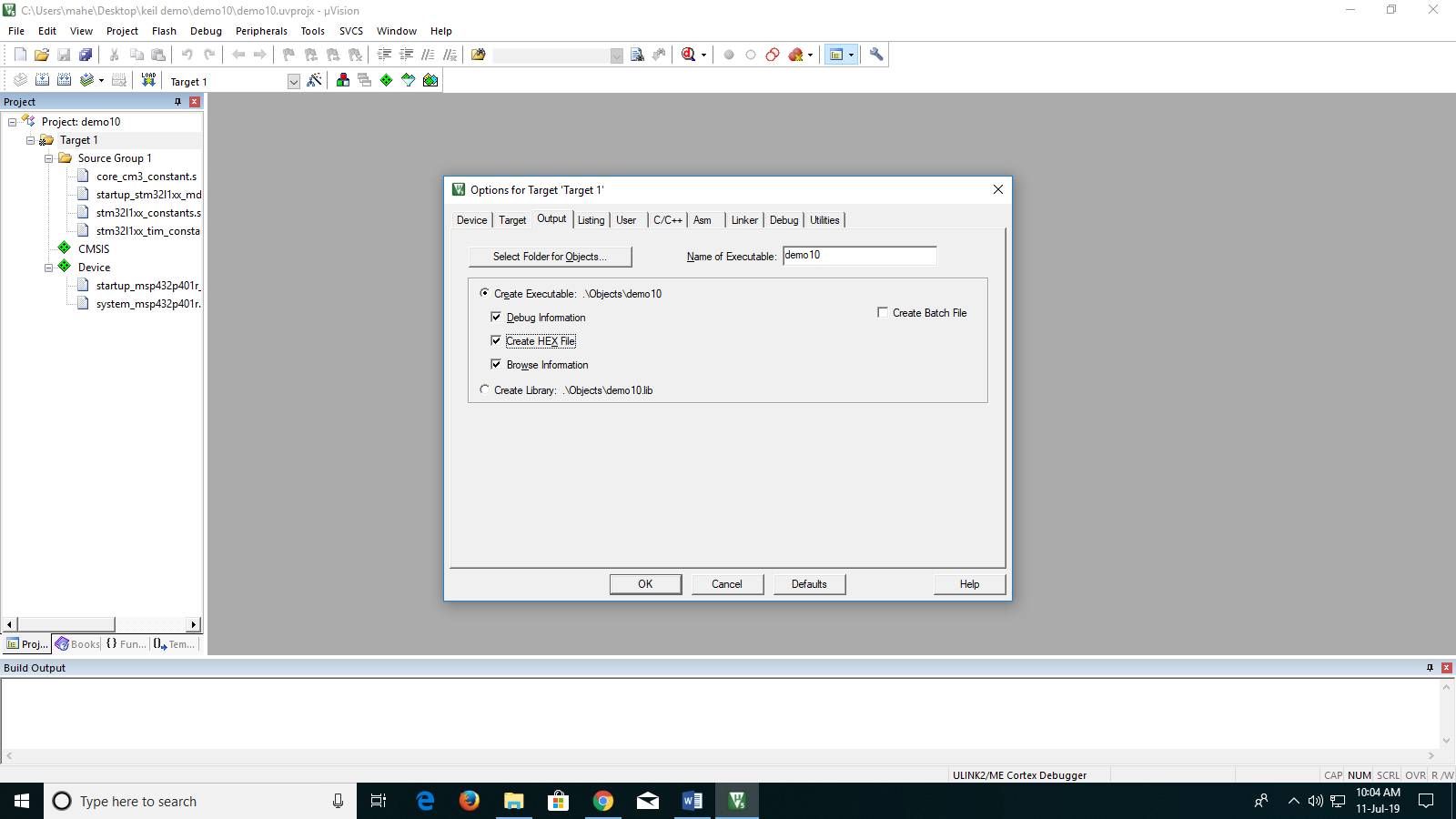
1. The files are added under project created as .C and .S extension.
2. Right click on source group and click on add existing files.



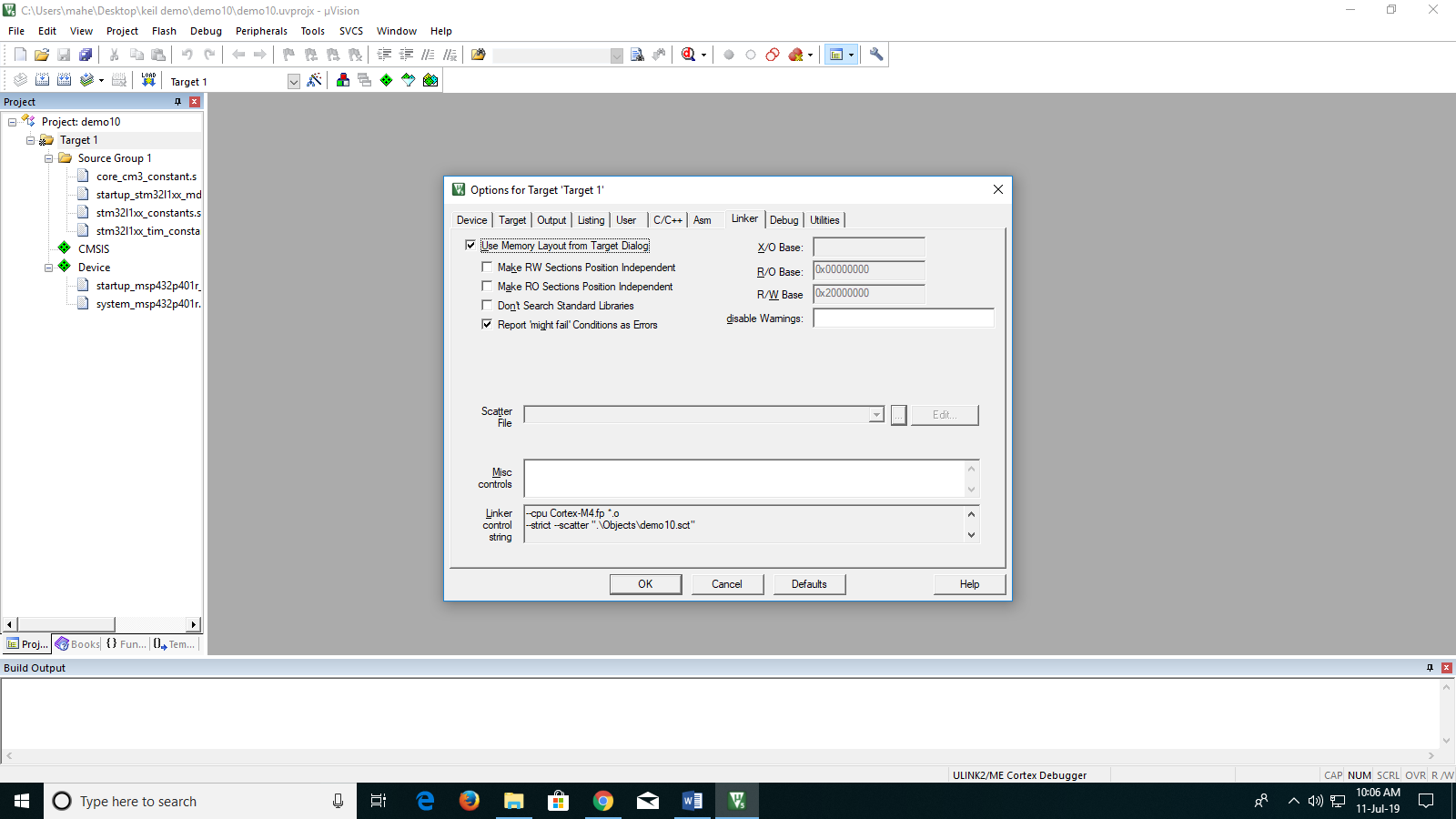
1. Go to file name Keil Startup files and add all the files with .S extension. These are used to compile as a reference document for Assembly programming only.
2. After Add, the files will reflect under source group1.
3. Now go to Target 1 and press right click on mouse, and go to “options for target”.



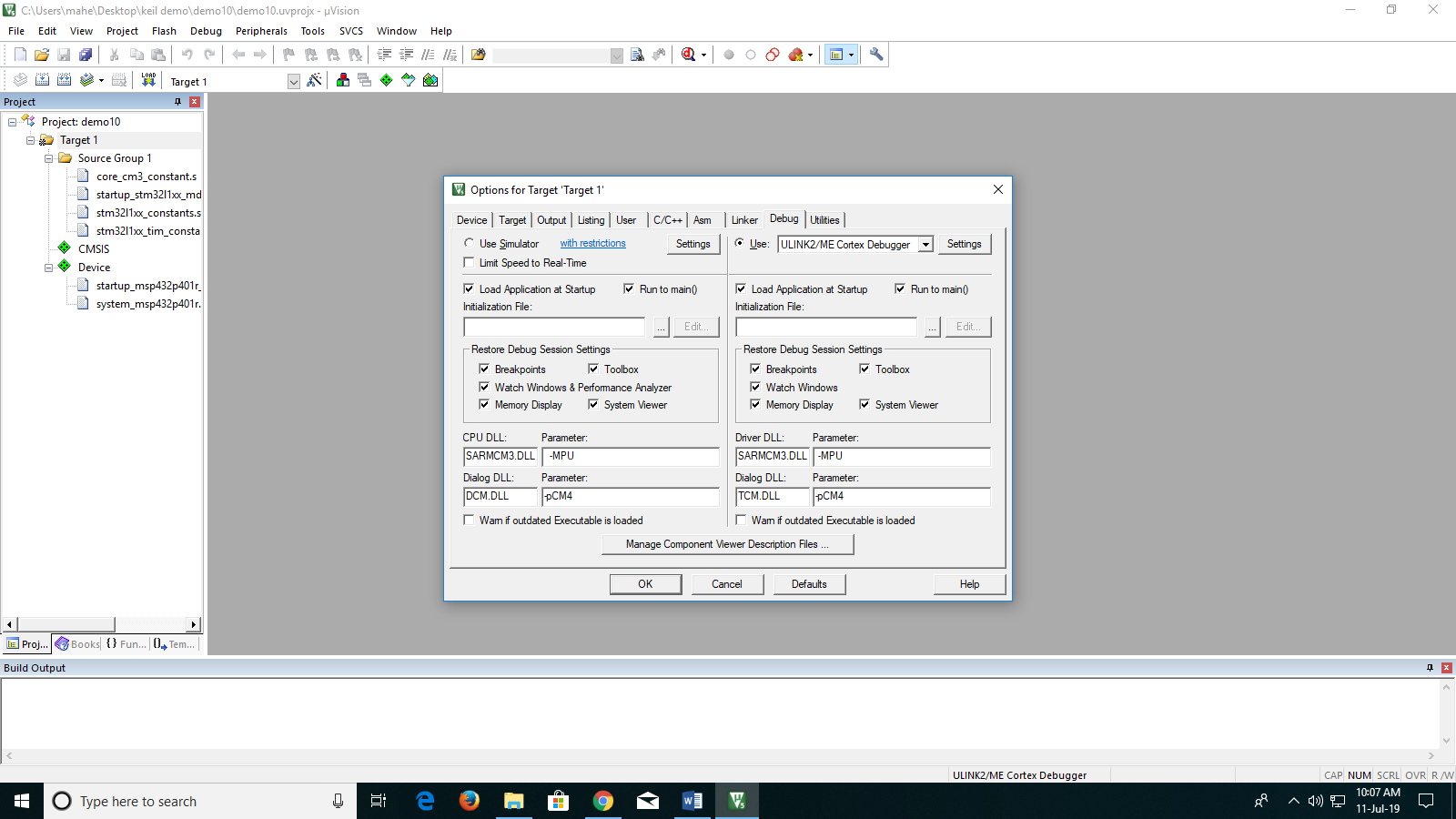
1. In options for target the specifications of microcontroller is given or can be customized based on vendor updates.
2. Go to output option and click on Create HEX file.



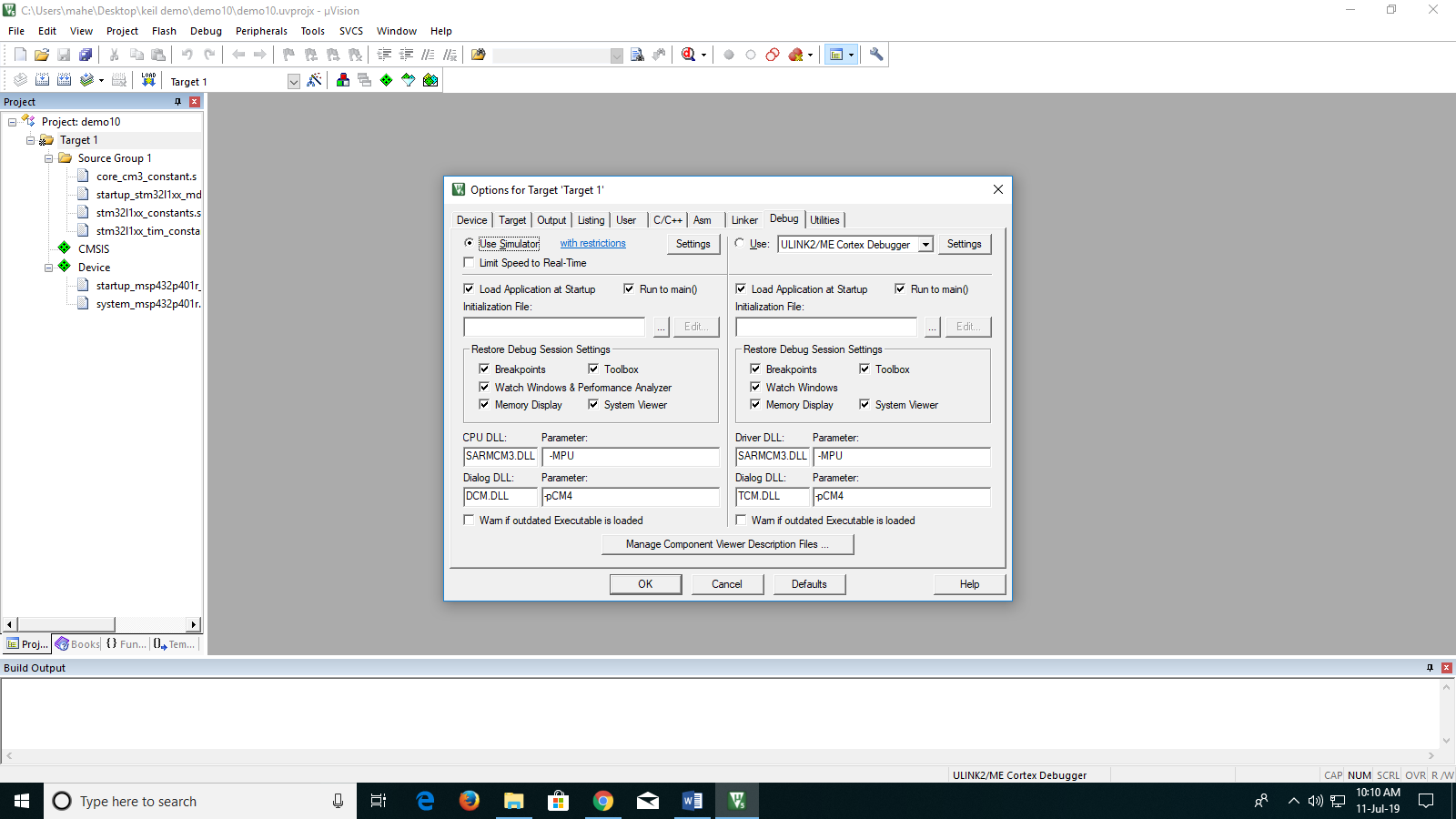
1. Go to Linker and click the option “Use Memory layout from Target Dialog”.



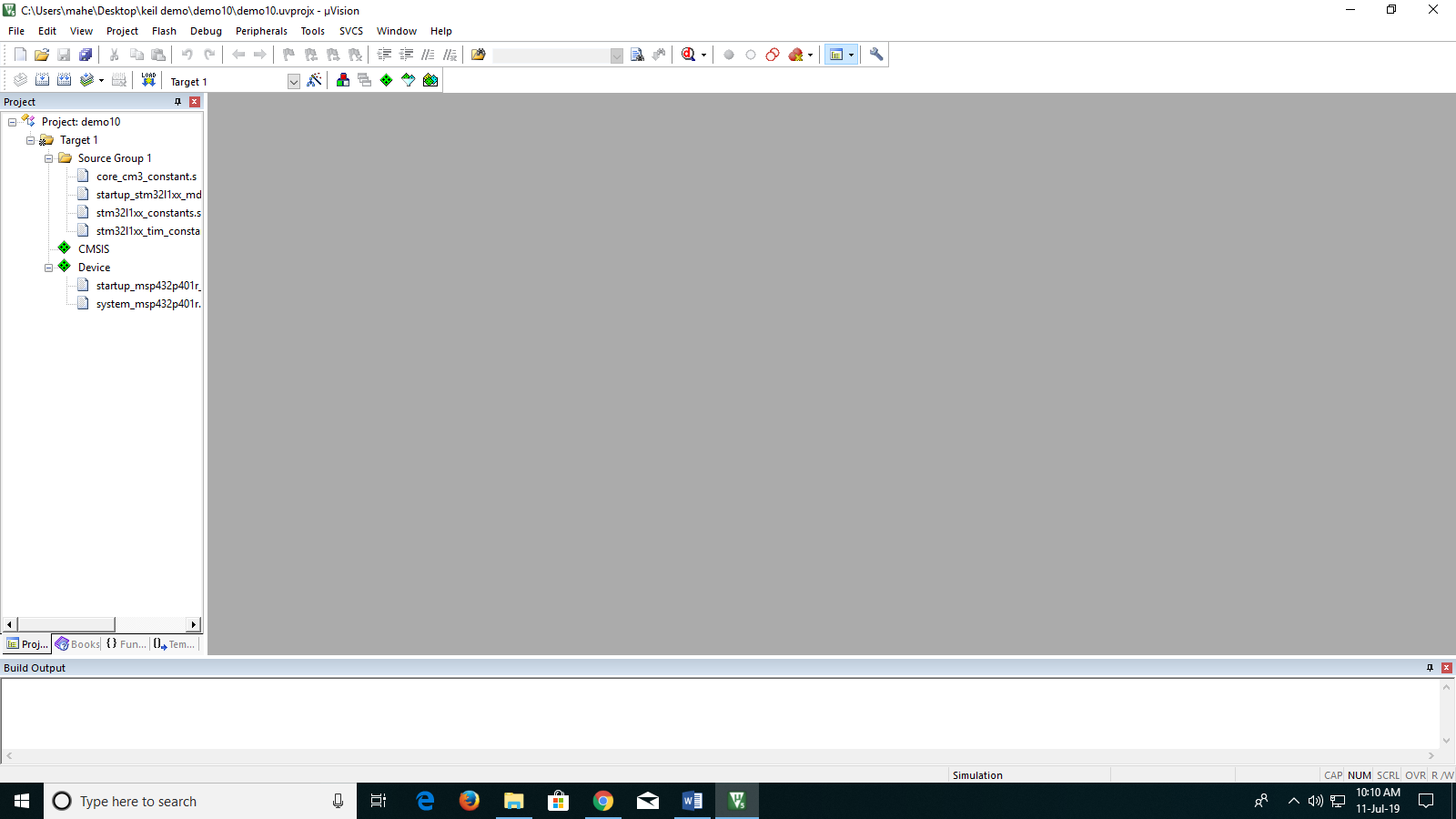
1. At the bottom part of Keil one can see the debugging mode as ULINK2/ME Cortex Debugger.



1. To Change that option and perform online simulation go to debug option and click on Simulation.



1. After that press OK, then the option changes from ULINK to Simulation.



1. Now click on New Empty Document on left top corner or just press Ctrl+N as a shortcut to have an empty document for coding.

**Example 1: To Familiarize with Keil and MSP432P401R**

INCLUDE MSP432xx\_constants.s ; Load Constant Definitions

INCLUDE MSP432xx\_tim\_constants.s ; TIM Constants

AREA main, CODE, READONLY

EXPORT \_\_main ; make \_\_main visible to linker

ENTRY

The helper file to be used for reading the program.

\_\_main PROC

MOV r0, #0

MOV r1, #4

Main Program

MOV r2, #6

ADD r3, r2, r1

ENDP

ALIGN

AREA allocations, DATA, READWRITE

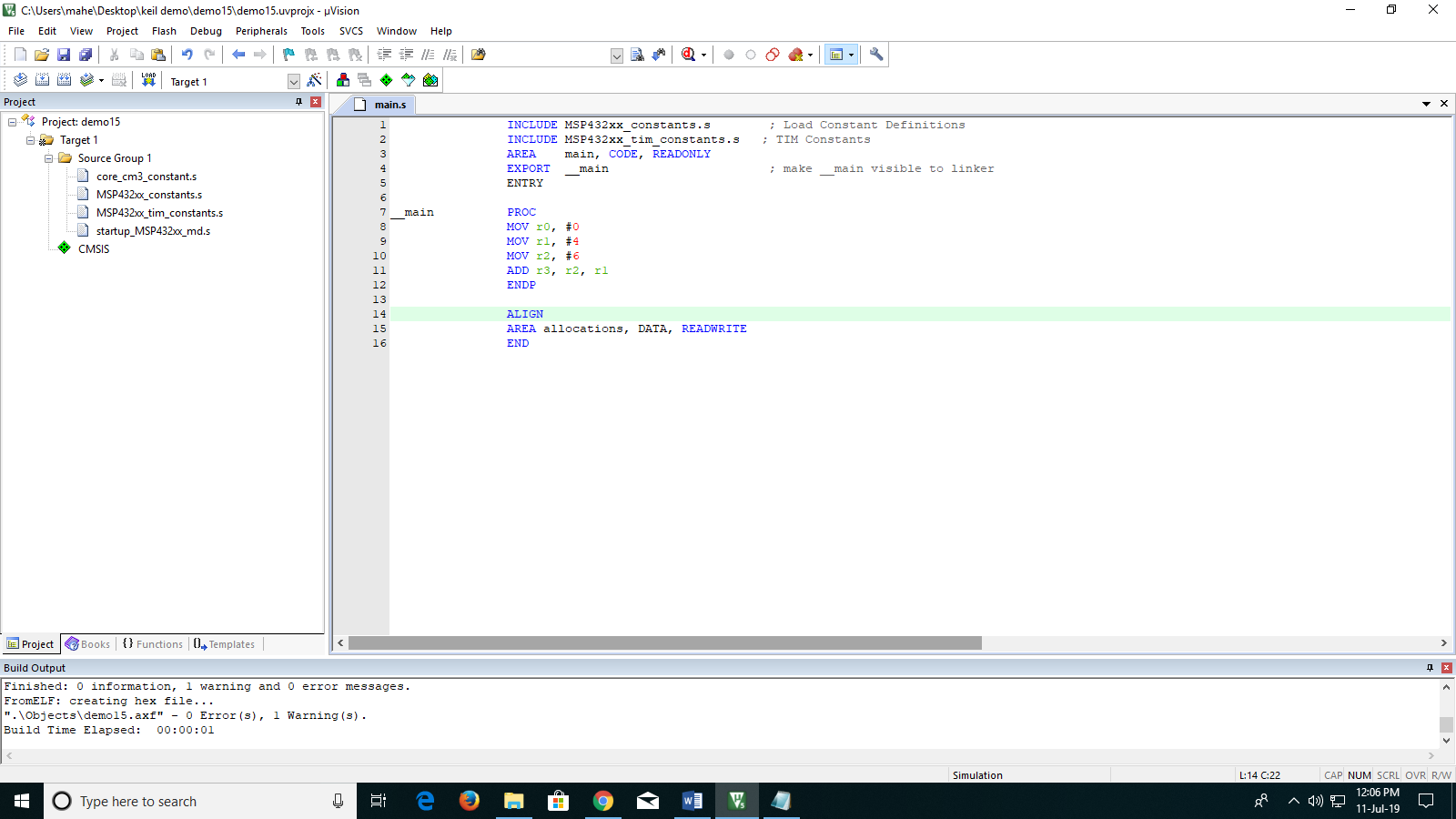
END

Program to write on core or DSP setup based on memory management.

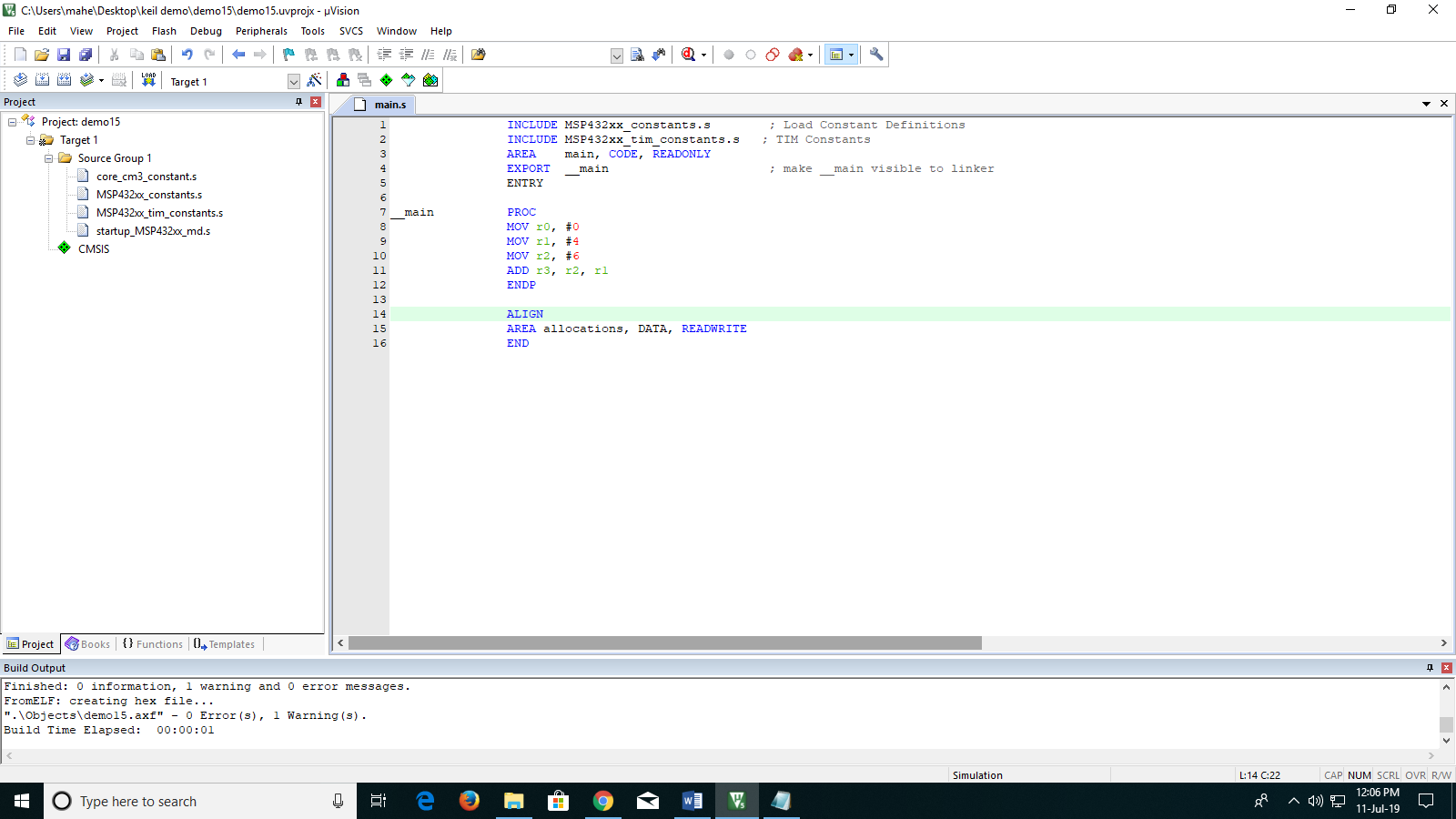
1. Write the sample program (Example 1) on new empty document.

Note: The Memory area in ARM Cortex M4 architecture has distinguished with the header to specify the allocation required.

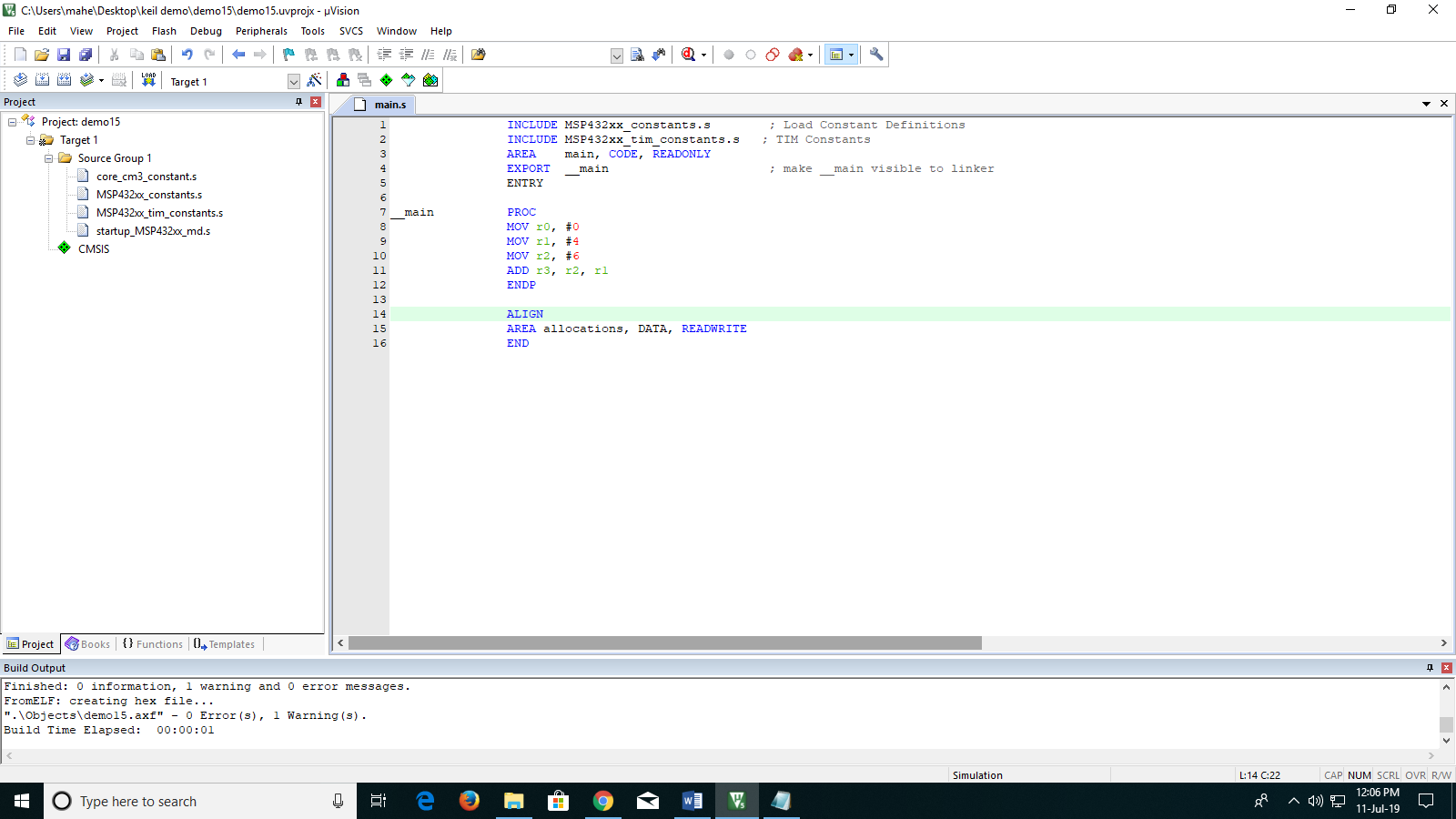
1. After writing the code save it as MAIN.S (.S/.ASM is extension for assembly code). And also save it in the same folder where keil startup files are saved. Always it’s better to save each Experiment with all the files individually with MAIN and Keil startup files.
2. Now right click on source group and click “Add Existing file” and redirect if required to MAIN.S or MAIN.asm and add it to source group1.
3. After adding save all the files.



1. Then translate the files by pressing Ctrl +F7 or by pressing the following tab.



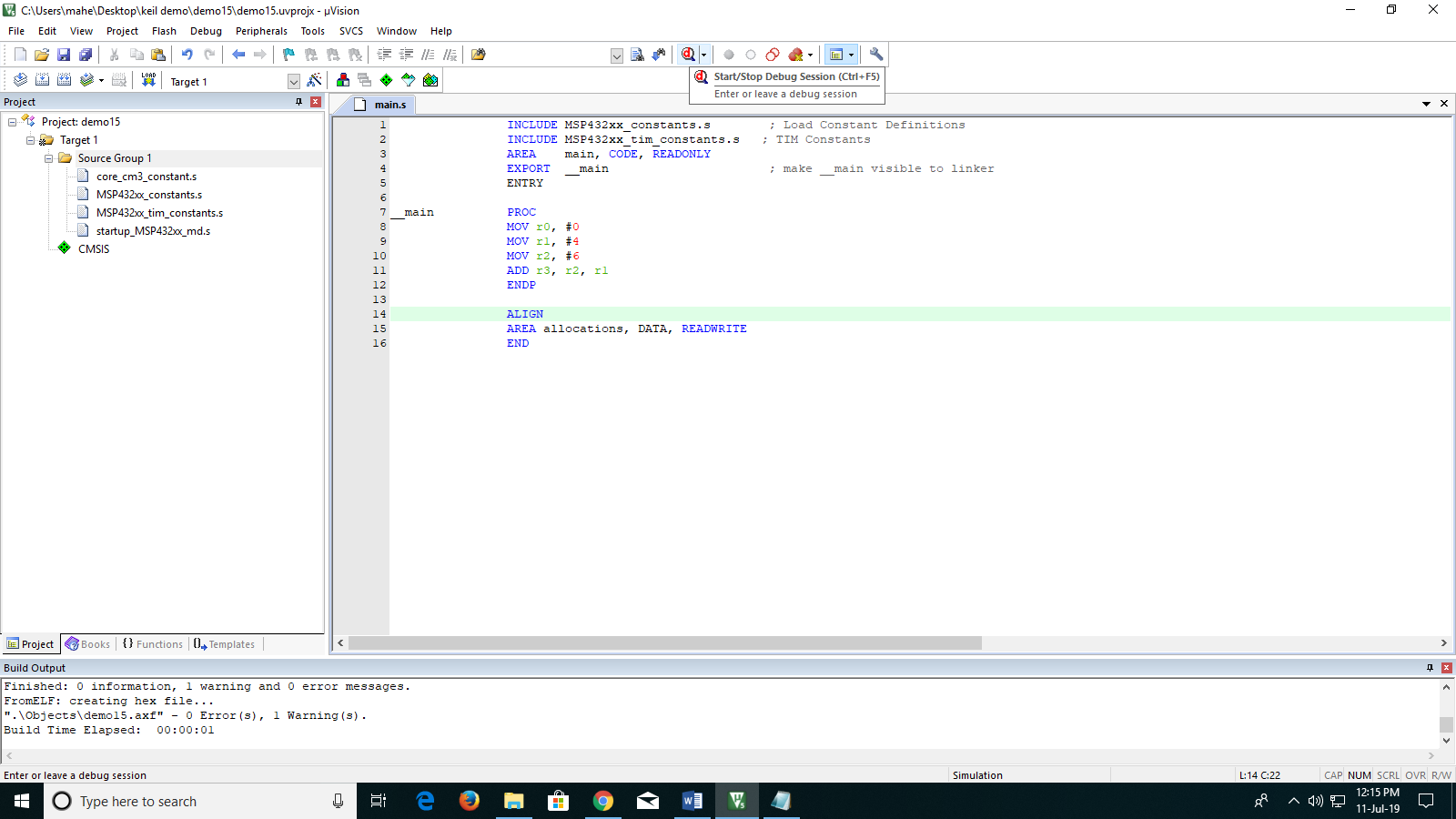
1. Then build the file by pressing F7 or by the following tab.



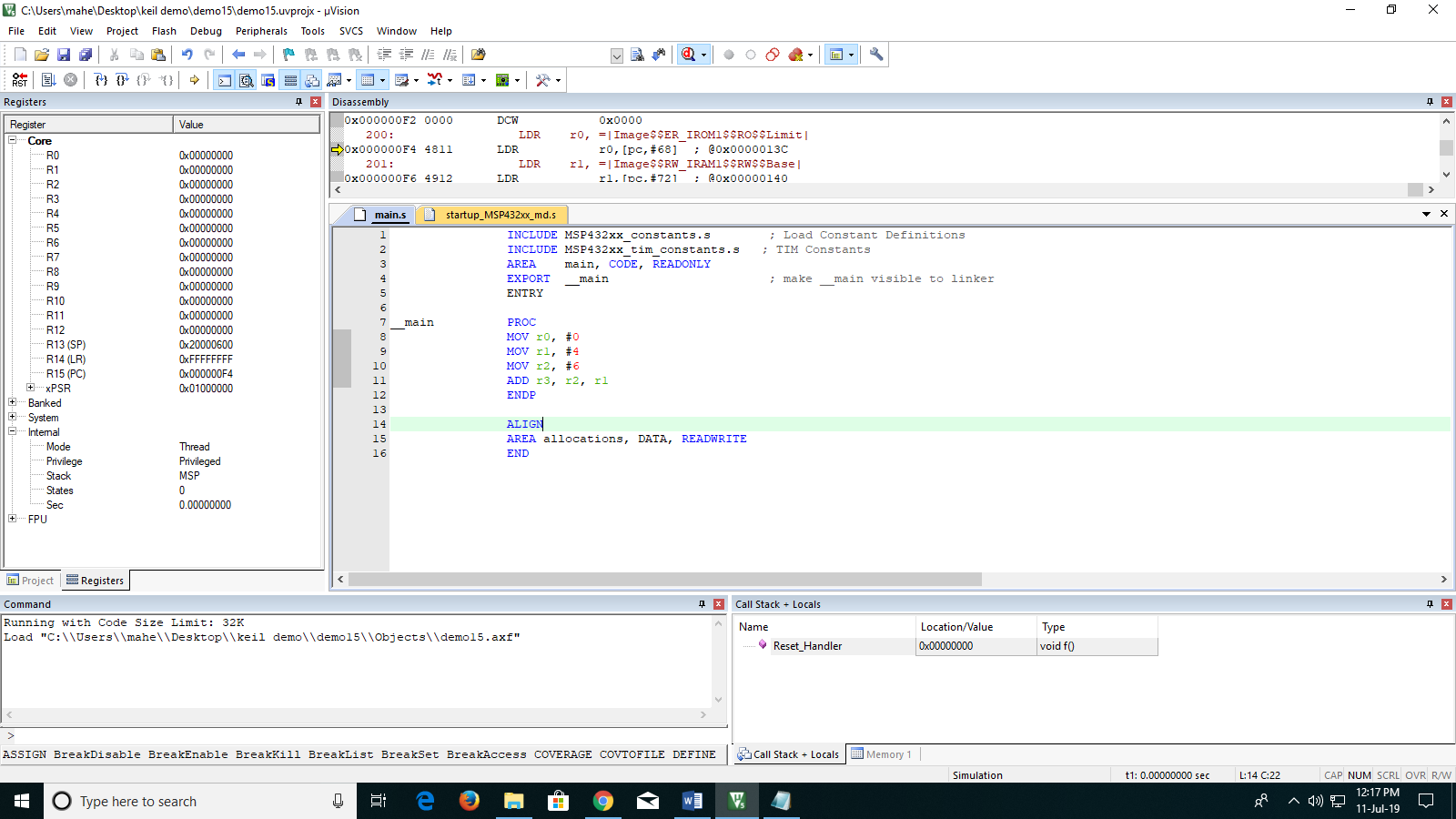
1. Check for the errors or warnings at output build window below the project window.



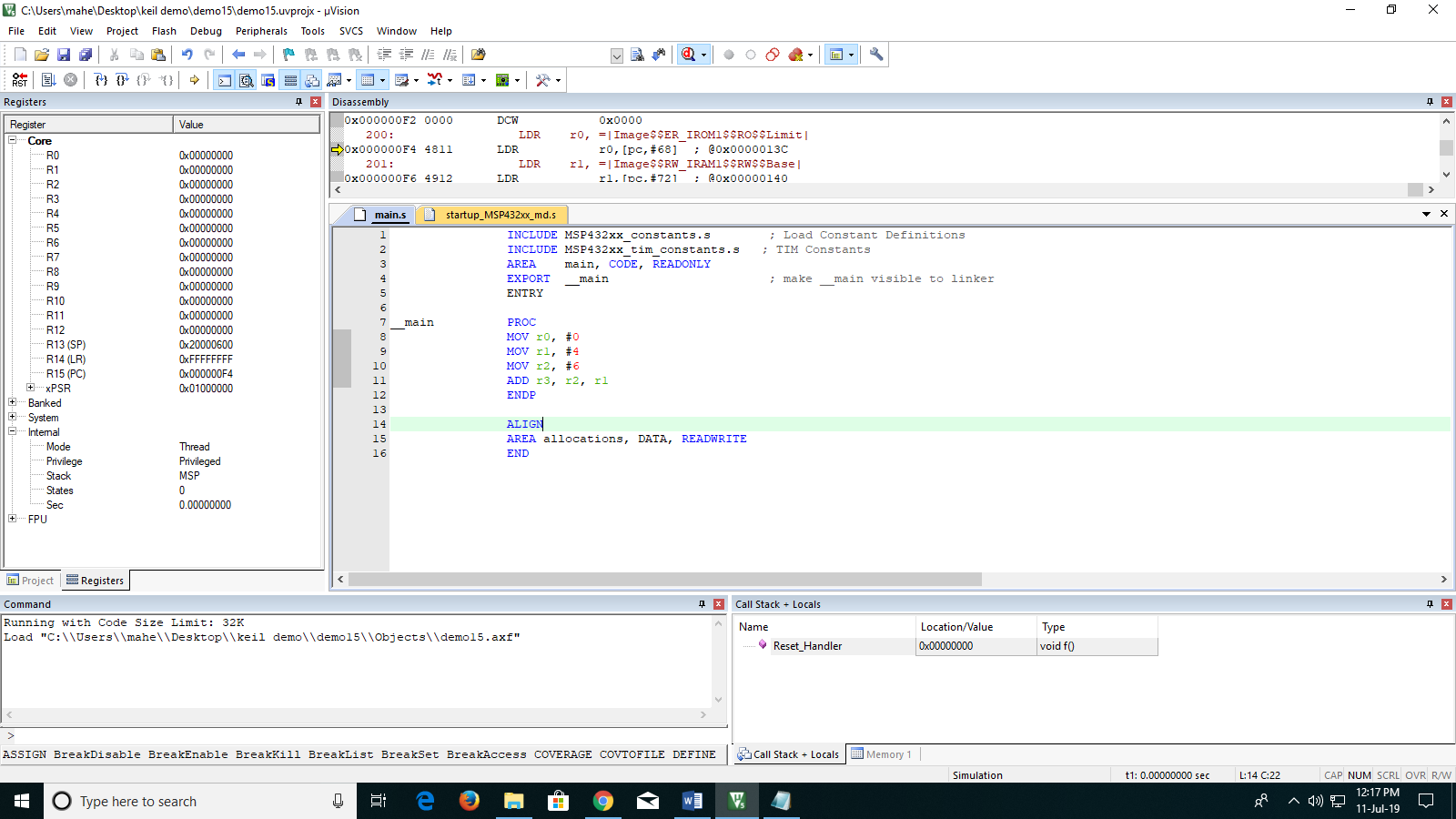
1. If zero errors and software warning 1 then press debug option or press Ctrl+F5.

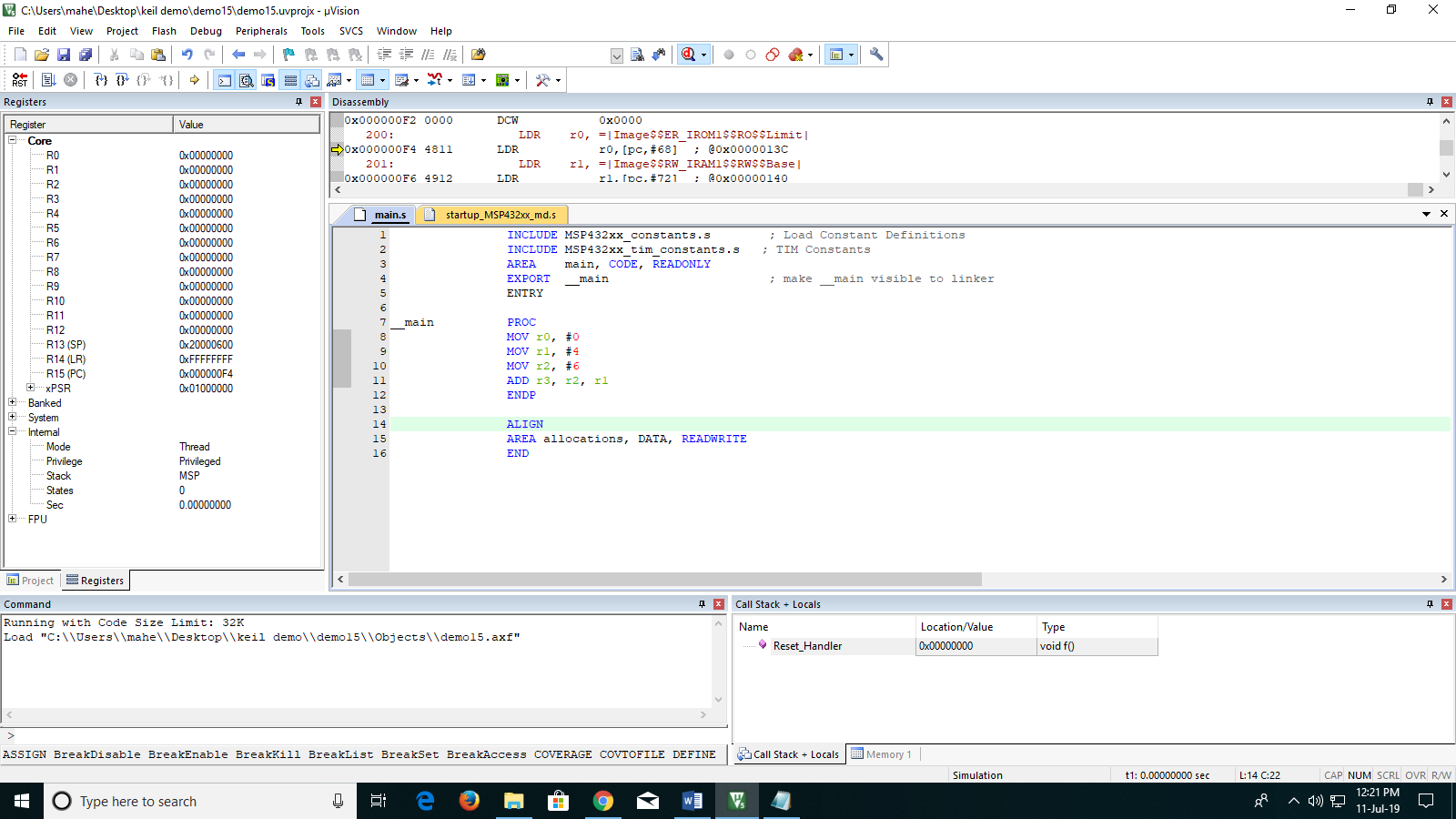


1. As its simulation the memory will be limited to 32K. Press ok and enter into debugging mode.
2. In debugging mode one can see the movement of data into register or CPSR or interrupts/timers performance. Also can check the pulsewidth modulation in analyzer.

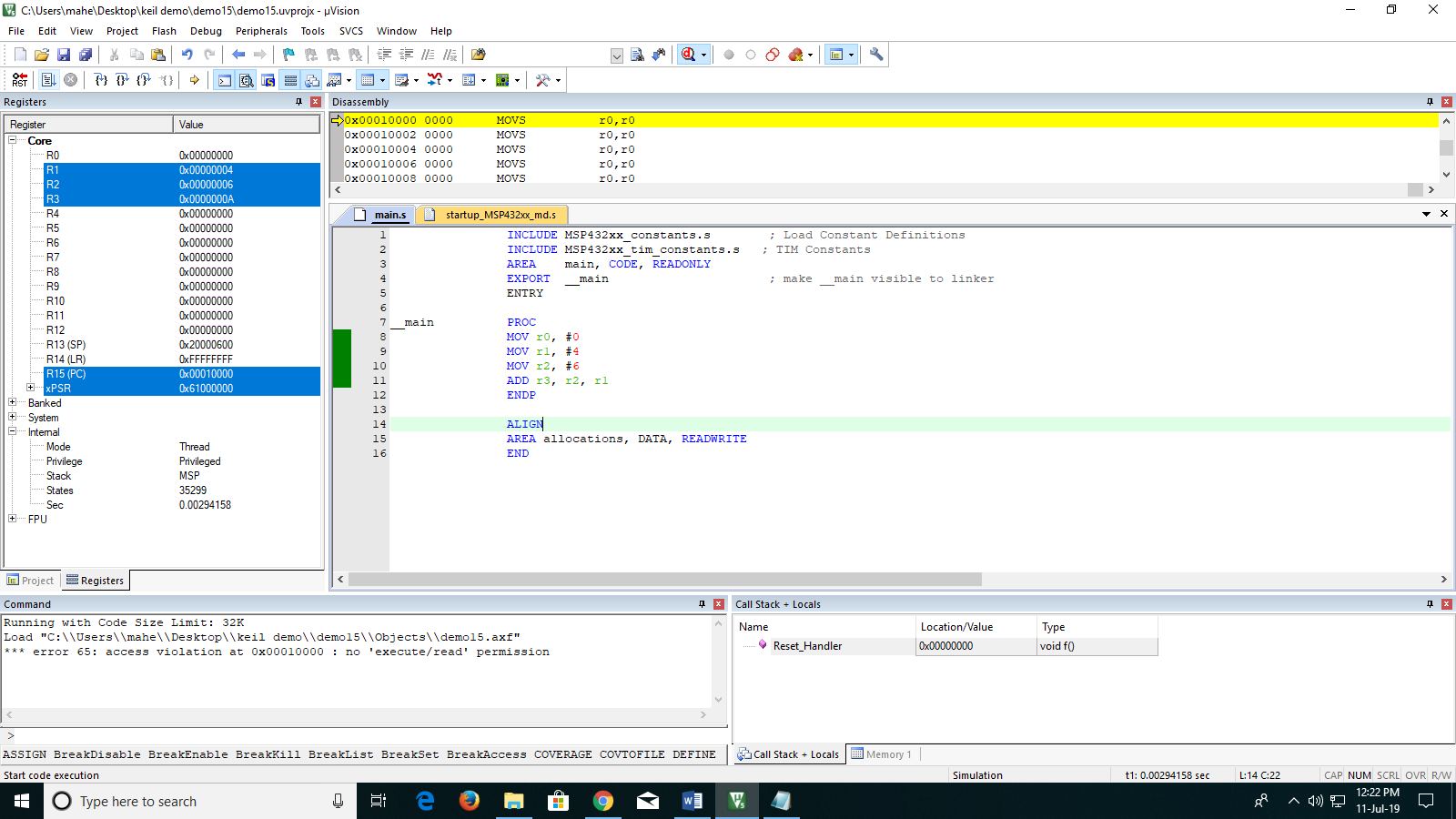
(RUN and Reset options).

Note: The run and reset options will set and clear the data in registers. If need to see step by step then one can press step option.

(Step and Step over)



**Before RUN**



**After RUN**

**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>

1. “µVision User's Guide”.

<http://www.keil.com/support/man/docs/uv4/>

1. “Arm® Keil® MDK Version 5 for SimpleLink™ MSP432™ Microcontrollers”

<http://www.ti.com/lit/ug/slau590j/slau590j.pdf>

1. “MSP432P401R SimpleLink™ Microcontroller LaunchPad™ Development Kit (MSP‑EXP432P401R)”.

<http://www.ti.com/lit/ug/slau597f/slau597f.pdf>

1. “Texas Instruments MSP432: Cortex™-M4 Tutorial Using the MSP432P401R LaunchPad Board and ARM Keil MDK 5 Toolkit”.

<http://www.keil.com/appnotes/files/apnt_276_v3_8.pdf>

1. “General Purpose Input Output – MSP432”

<http://www.ti.com/lit/ml/swrp156/swrp156.pdf>

1. “Teaching Materials TI University Program”.

<https://university.ti.com/en/faculty>

1. “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:**

INCLUDE MSP432xx\_constants.s ; Load Constant Definitions

INCLUDE MSP432xx\_tim\_constants.s ; TIM Constants

AREA main, CODE, READONLY

EXPORT \_\_main ; make \_\_main visible to linker

ENTRY

\_\_main PROC

MOV R1, #0x25

MOV R2, #0x34

ADD R3, R2,R1

ENDP

ALIGN

AREA allocations, DATA, READWRITE

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:**

INCLUDE MSP432xx\_constants.s ; Load Constant Definitions

INCLUDE MSP432xx\_tim\_constants.s ; TIM Constants

AREA main, CODE, READONLY

EXPORT \_\_main ; make \_\_main visible to linker

ENTRY

\_\_main PROC

MOV R1, #0x40

MOV R2, #0x20

SUB R3, R1, R2

ENDP

ALIGN

AREA allocations, DATA, READWRITE

END

Exercise: Note: All the values are given in Decimal value convert them to hexadecimal while performing the experiment.

1. Write an assembly language to ADD 16, 19, 22, 47 and store the value in different register.

2. Write an assembly language to perform following operation (10-9-3-1) and store the output value in different register.

3. Write an assembly language to perform following operation R0 = 9 + (34 – 4 – 2).

**EXPERIMENT NO.2: INTRODUCTION TO ARITHMETIC OPERATIONS USING MSP432P401R**

**AIM**: To be familiarized Instruction set using ARITHEMETIC Operations.

Syntax:

<Operation>{<Cond>}{S} Rd, Rn, Operand2.

1. ADD operand1 + operand2
2. ADC operand1 + operand2 + carry
3. SUB operand1 - operand2
4. SBC operand1 - operand2 + carry -1
5. RSB operand2 - operand1

**Example 1: WAP to analyze the arithmetic operation using ADD and ADC**

Solution:

INCLUDE MSP432xx\_constants.s ; Load Constant Definitions

INCLUDE MSP432xx\_tim\_constants.s ; TIM Constants

AREA main, CODE, READONLY

EXPORT \_\_main ; make \_\_main visible to linker

ENTRY

\_\_main PROC

MOV R1, #0x40

MOV R2, #0x20

MOV R3, #0x40

MOV R4, #0x20

ADD R5, R1, R2

ADC R6, R3, R4

ENDP

ALIGN

AREA allocations, DATA, READWRITE

END

Output:

**Example 2: WAP to analyze the arithmetic operation using SUB and SBC**

Solution:

INCLUDE MSP432xx\_constants.s ; Load Constant Definitions

INCLUDE MSP432xx\_tim\_constants.s ; TIM Constants

AREA main, CODE, READONLY

EXPORT \_\_main ; make \_\_main visible to linker

ENTRY

\_\_main PROC

MOV R1, #0x40

MOV R2, #0x20

MOV R3, #0x40

MOV R4, #0x20

SUB R5, R1, R2

SBC R3, R4, #0x01

ENDP

ALIGN

AREA allocations, DATA, READWRITE

END

**Example 3: WAP to analyze the arithmetic operation using RSB.**

Solution:

INCLUDE MSP432xx\_constants.s ; Load Constant Definitions

INCLUDE MSP432xx\_tim\_constants.s ; TIM Constants

AREA main, CODE, READONLY

EXPORT \_\_main ; make \_\_main visible to linker

ENTRY

\_\_main PROC

MOV R1, #0x40

MOV R2, #0x20

MOV R1, #0x40

MOV R2, #0x20

RSB R5, R1, R2

ENDP

ALIGN

AREA allocations, DATA, READWRITE

END

**Exercise**:

*Note: All the values are given in decimal value convert them to hexadecimal while performing the experiment.*

1. Write an assembly language to ADD 16 & 19 and the output to perform ADC with 6. Analyze the register behavior.

2. Write an assembly language to perform following operation R0= 24-16 with a carry 3.

3. Write an assembly language to perform the following example:

Ramesh has a habit of reverse subtraction, one day he went to retailer store to buy a packet of milk for rupees 18/- and chocolate for rupees 20/-. He gave rupees 100/- to the shopkeeper and asked for rupees 3536/-. Check in debugging whether Ramesh calculations were correct or wrong.

**EXPERIMENT NO.3: INTRODUCTION TO COMPARISON AND LOGICAL OPERATIONS USING MSP432P401R**

**AIM**: To be familiarized Instruction set using ARITHEMETIC Operations.

Syntax:

1. CMP operand1 - operand2, but result not written
2. CMN operand1 + operand2, but result not written
3. TST operand1 AND operand2, but result not written
4. TEQ operand1 EOR operand2, but result not written
5. AND operand1 AND operand2
6. EOR operand1 EOR operand2
7. ORR operand1 OR operand2
8. BIC operand1 AND NOT operand2 [ie bit clear]

**Example 1: To familiarize the instruction sets used for comparison operator CMP, CMN, TST, TEQ with variations in CPSR.**

Solution:

INCLUDE MSP432xx\_constants.s; Load Constant Definitions

INCLUDE MSP432xx\_tim\_constants.s ; TIM Constants

AREA main, CODE, READONLY

EXPORT \_\_main ; make \_\_main visible to linker

ENTRY

\_\_main PROC

MOV R1, #0x10

MOV R2, #0x09

**CMP R1, R2**

**;CMN R1, R2**

**;TST R1, R2**

**;TEQ R1, R2**

ENDP

ALIGN

AREA allocations, DATA, READWRITE

END

**Example 2: To familiarize the instruction sets used for logical operation AND, EOR, ORR, BIC**

Solution: **a. AND Gate**

INCLUDE MSP432xx\_constants.s ; Load Constant Definitions

INCLUDE MSP432xx\_tim\_constants.s ; TIM Constants

AREA main, CODE, READONLY

EXPORT \_\_main ; make \_\_main visible to linker

ENTRY

\_\_main PROC

MOV R1, X ;(The value is based on logic table)

MOV R2, Y ;(The value is based on logic table)

AND R3, R2, R1

ENDP

ALIGN

AREA allocations, DATA, READWRITE

END

**b. ORR Gate**

INCLUDE MSP432xx\_constants.s ; Load Constant Definitions

INCLUDE MSP432xx\_tim\_constants.s ; TIM Constants

AREA main, CODE, READONLY

EXPORT \_\_main ; make \_\_main visible to linker

ENTRY

\_\_main PROC

MOV R1, X ;(The value is based on logic table)

MOV R2, Y ;(The value is based on logic table)

ORR R3, R2, R1

ENDP

ALIGN

AREA allocations, DATA, READWRITE

END

**c. EOR Gate**

INCLUDE MSP432xx\_constants.s ; Load Constant Definitions

INCLUDE MSP432xx\_tim\_constants.s ; TIM Constants

AREA main, CODE, READONLY

EXPORT \_\_main ; make \_\_main visible to linker

ENTRY

\_\_main PROC

MOV R1, X(The value is based on logic table)

MOV R2, Y(The value is based on logic table)

EOR R3, R2, R1

ENDP

ALIGN

AREA allocations, DATA, READWRITE

END

**d. BIC (Bit Clear – AND NOT)**

INCLUDE MSP432xx\_constants.s ; Load Constant Definitions

INCLUDE MSP432xx\_tim\_constants.s ; TIM Constants

AREA main, CODE, READONLY

EXPORT \_\_main ; make \_\_main visible to linker

ENTRY

\_\_main PROC

MOV R1, X ;(The value is based on logic table)

MOV R2, Y ;(The value is based on logic table)

BIC R3, R2, R1

ENDP

ALIGN

AREA allocations, DATA, READWRITE

END

**Exercise**:

*Note: All the values are given in decimal value convert them to hexadecimal while performing the experiment.*

1. Write an assembly language to design a system based on three sensors decision, i.e. if all are true only then the system should glow light. Assume that light is interfaced with R7 register.

2. Write an assembly language to connect four sensors in two sub systems. Each system has 2 sensors divided, so create a logic for the system with given condition.

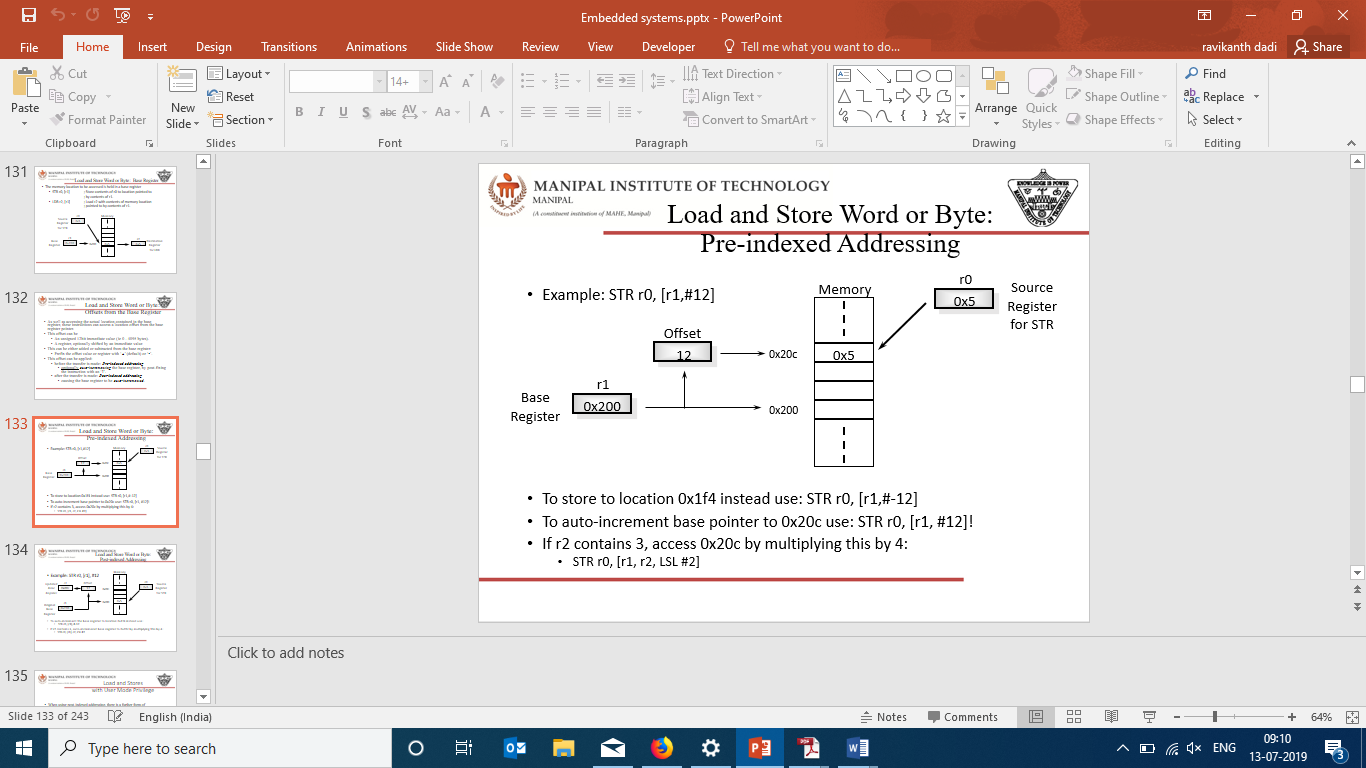
* The First system holds 2 sensors interfaced with R0 & R1 registers.
* Second System holds 2 sensors interfaced with R3 & R4 registers.
* If all the sensors are high in input then make R7 Register high which is interfaced with a motor of a conveyor belt for a manufacturing application in an industry.

**EXPERIMENT NO.4: INTRODUCTION TO DATA MOVEMENT**

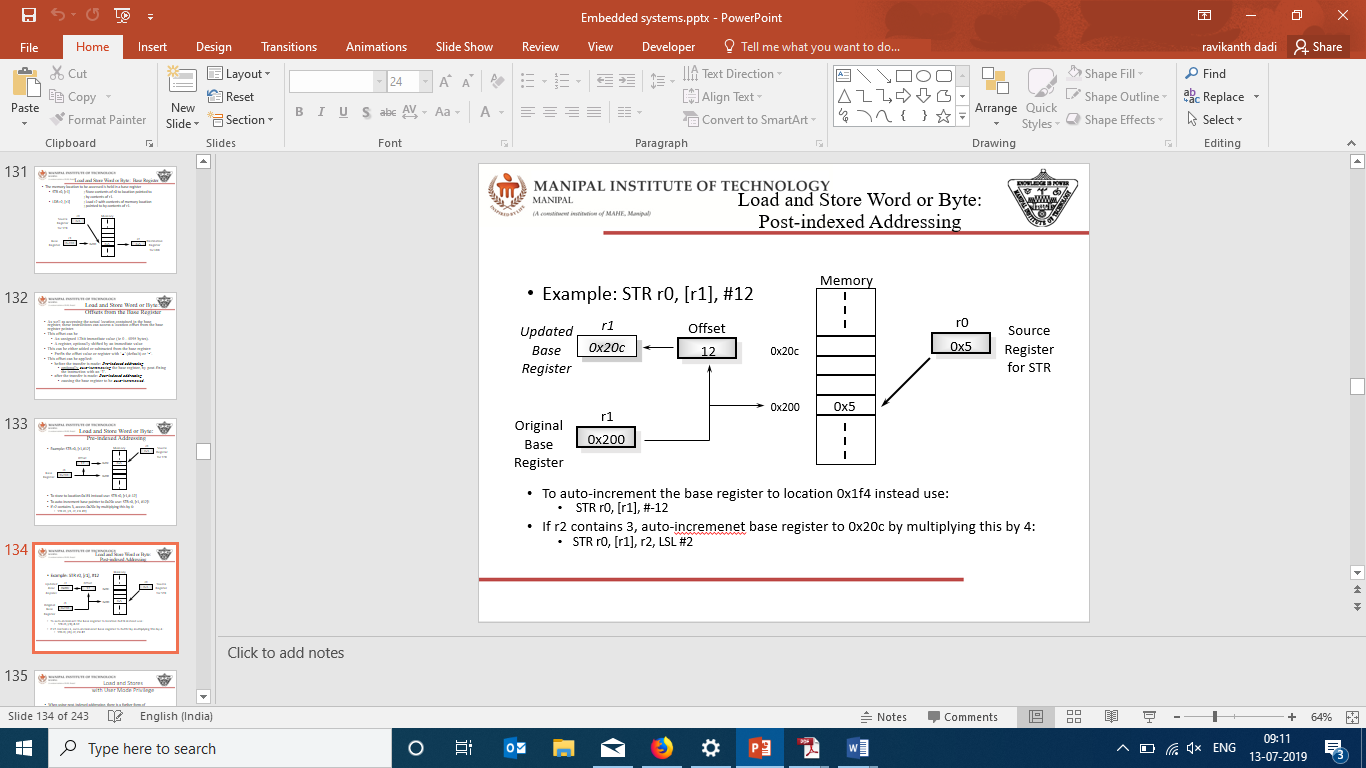
**AIM**: To be familiarized the basics of data movement.

**Syntax:**

1. **LSL #X = multiply by 2X**
2. **LSR #X = divide by 2X**
3. **ASR #X = divide by 2X**
4. **ROR #5**
5. **STR r0, [r1] ; Store contents of r0 to location pointed to  
    ; by contents of r1.**
6. **LDR r2, [r1] ; Load r2 with contents of memory location  
    ; pointed to by contents of r1.**
7. **Pre Indexing and Post Indexing of STR.** 
   1. **STR r0, [r1,#12]**



b. Example: STR r0, [r1], #12



Example 1: To analyze the barrel shifter and movement of data within the architecture.

Solution:

INCLUDE MSP432xx\_constants.s ; Load Constant Definitions

INCLUDE MSP432xx\_tim\_constants.s ; TIM Constants

AREA main, CODE, READONLY

EXPORT \_\_main ; make \_\_main visible to linker

ENTRY

\_\_main PROC

MOV R1, 0x01(The value is based on logic table)

MOV R1, R1, LSL #1

MOV R1, R1, LSR #2

MOV R1, R1, ASR #2

MOV R1, R1, ROR #16

ADD R1, R1, R1, LSL #4

RSB R1, R1, R1, LSL #5

ENDP

ALIGN

AREA allocations, DATA, READWRITE

END

Example 2: To analyze the data movement for pre indexing and post indexing of data.

Solution:

INCLUDE MSP432xx\_constants.s ; Load Constant Definitions

INCLUDE MSP432xx\_tim\_constants.s ; TIM Constants

AREA main, CODE, READONLY

EXPORT \_\_main ; make \_\_main visible to linker

ENTRY

\_\_main PROC

MOV R1, 0x01 ;(The value is based on logic table)

STR R0, [R1,#4]

STR R0, [R1], #4

LDR R2, [R0]

ENDP

ALIGN

AREA allocations, DATA, READWRITE

END

Exercise:

*Note: All the values are given in decimal value convert them to hexadecimal while performing the experiment.*

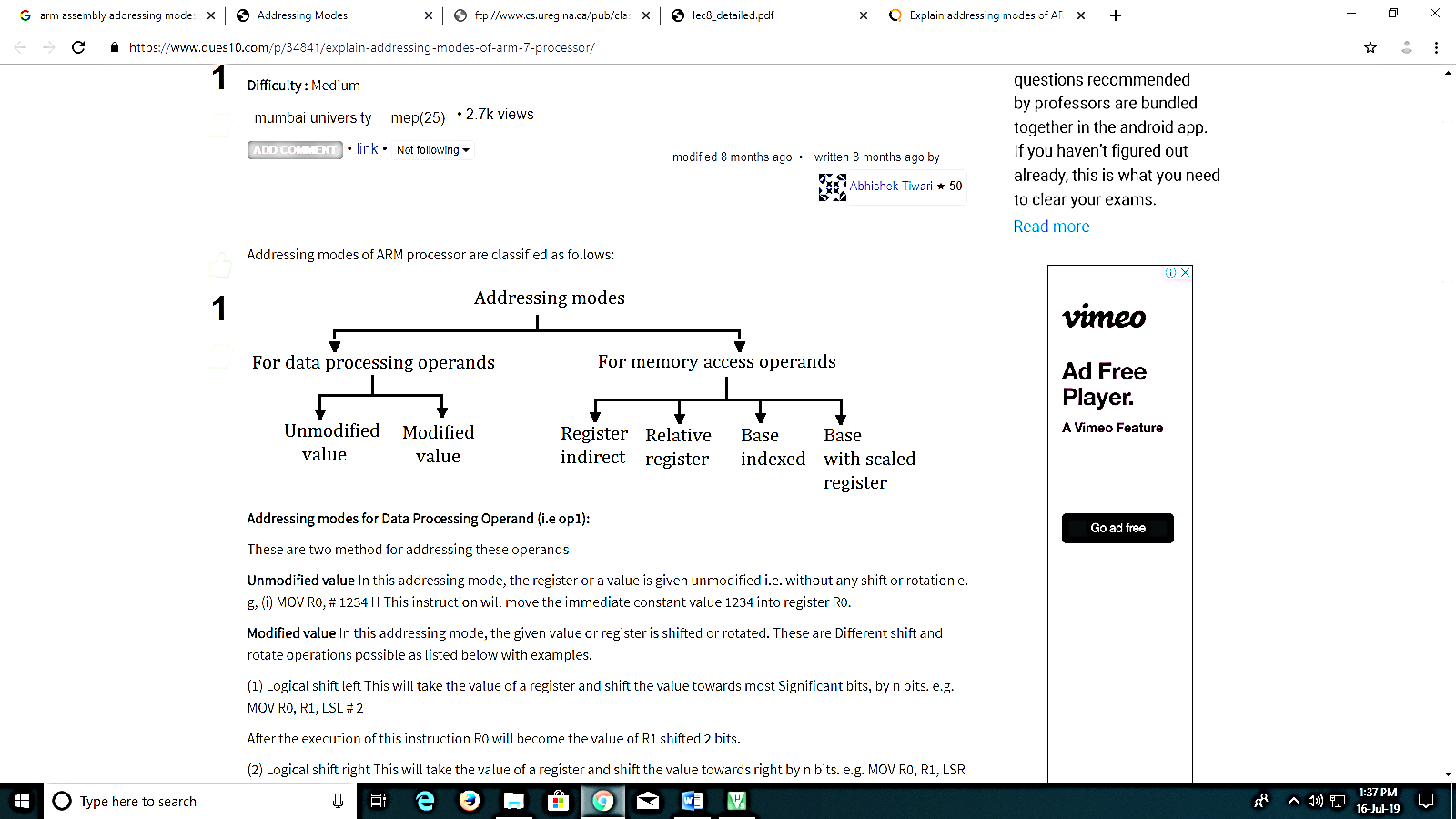
1. Write an assembly language programming for moving the sensor value 20 to register bank with a pre indexing value 12 rotated left by 2 and load into an another register which is interfaced with a DC motor for rotation.

2. Write an assembly language to store a valve 11 in any assumed register to be pre-indexed by #2 and right rotated by the multiplicand of 2.

**EXPERIMENT NO.5: ADDRESSING MODES OF MSP432P401R**

**AIM**: To be familiarized the basics of addressing modes of ARM in assembly language to understand the data movement in specific with registers.

**“Opcode Destination, Operand\_1, Operand\_2”**



**Syntax:**

**A. Data Processing Operands:**

**1. Unmodified:** MOV R0, # 1234 H

**2. Modified:** MOV R0, R1, LSL # 2 or ROR #2 or ASR #2.

**B. Memory Access Operands:**

**1. Register indirect addressing mode:** LDR R0, [R1]

This instruction will load the register R0 with the 32-bit word at the memory address held in the register R1.

2. **Relative register indirect addressing mode:** LDR R0, [R1, #4]

In this addressing mode the memory address is generated by an immediate value added to a register. Pre index and post index are supported in this addressing mode.

3. **Base indexed indirect addressing mode:** LDR R0, [R1, R2]

In this addressing mode the memory address is generated by adding the values of two registers. Pre-index and post-index are supported also in this addressing mode.

4. **Base with scaled register addressing mode:** LDR R0, [R1, R2, LSL #2]

 In this addressing mode the memory address is generated by a register value added to another register shifted left. Pre-index and post-index are supported in this addressing mode.

Example 1: Write an assembly language programming to send the data value from register to memory location and load it back from different address of any specified memory.

Solution:

INCLUDE MSP432xx\_constants.s ; Load Constant Definitions

INCLUDE MSP432xx\_tim\_constants.s ; TIM Constants

AREA main, CODE, READONLY

EXPORT \_\_main ; make \_\_main visible to linker

ENTRY

\_\_main PROC

LDR R1, =1000000000

LDR R2, =2000000000

LDR R3, =55

LDR R3, [R1] ;get from RAM1

STR R3, [R2] ;send it to RAM2

ENDP

ALIGN

AREA allocations, DATA, READWRITE

END

Example 2: Write an assembly language programming to send the data value from register to memory location and load it back from different address of any specified memory after performing athematic operation in a loop.

Solution:

INCLUDE MSP432xx\_constants.s ; Load Constant Definitions

INCLUDE MSP432xx\_tim\_constants.s ; TIM Constants

AREA main, CODE, READONLY

EXPORT \_\_main ; make \_\_main visible to linker

ENTRY

\_\_main PROC

MOV R0, #10 ;counter

\_\_loop LDR R2,=0x55555555

STR R2, [R1] ;send it to RAM

ADD R1, R1, #4 ;R1 = R1 + 4 to increment pointer

SUBS R0, R0, #1 ;R0 = R0 - 1 for dec counter

B \_\_loop

ENDP

ENDP

ALIGN

AREA allocations, DATA, READWRITE

END

Exercises:

1. Write an assembly program to store data of a memory location 255 to 260 in register R0 onwards.

2. Write an assembly program to store all the even numbers in register R6 and all the odd numbers in register R7. The process should be executed in a loop.

3. Write an assembly program to Load R0 to R9 from memory pointed to by R11 and store R0 to R9 to memory pointed by R12.

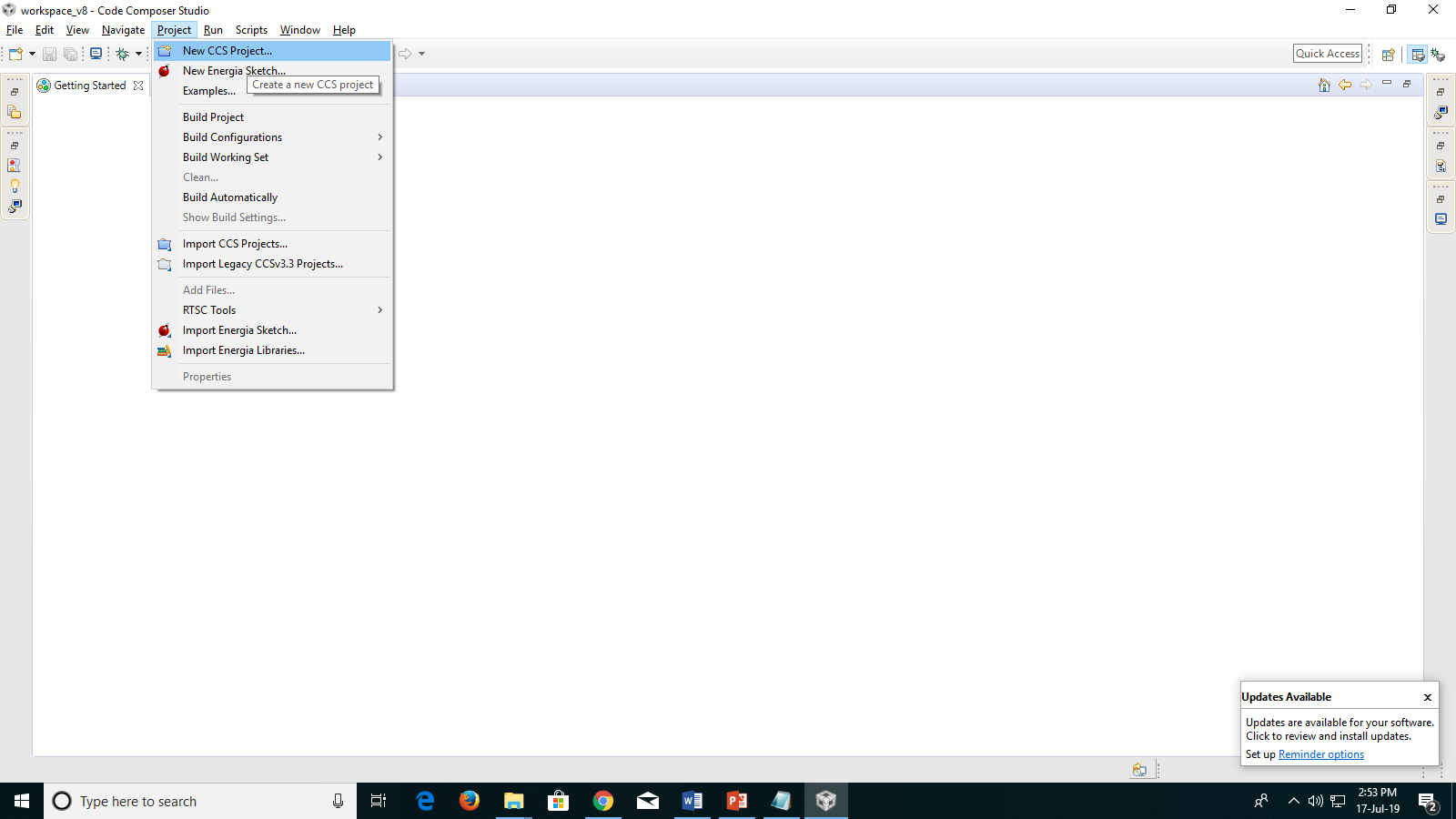
**EXPERIMENT NO.6: INTRODUCTION OF MSP432P401R WITH EMBEDDED C PROGRAMMING USING CODE COMPOSER STUDIO**

**AIM:** To familiarize the Embedded C Programming with online programming by connecting MSP432P401R using Code Composer.

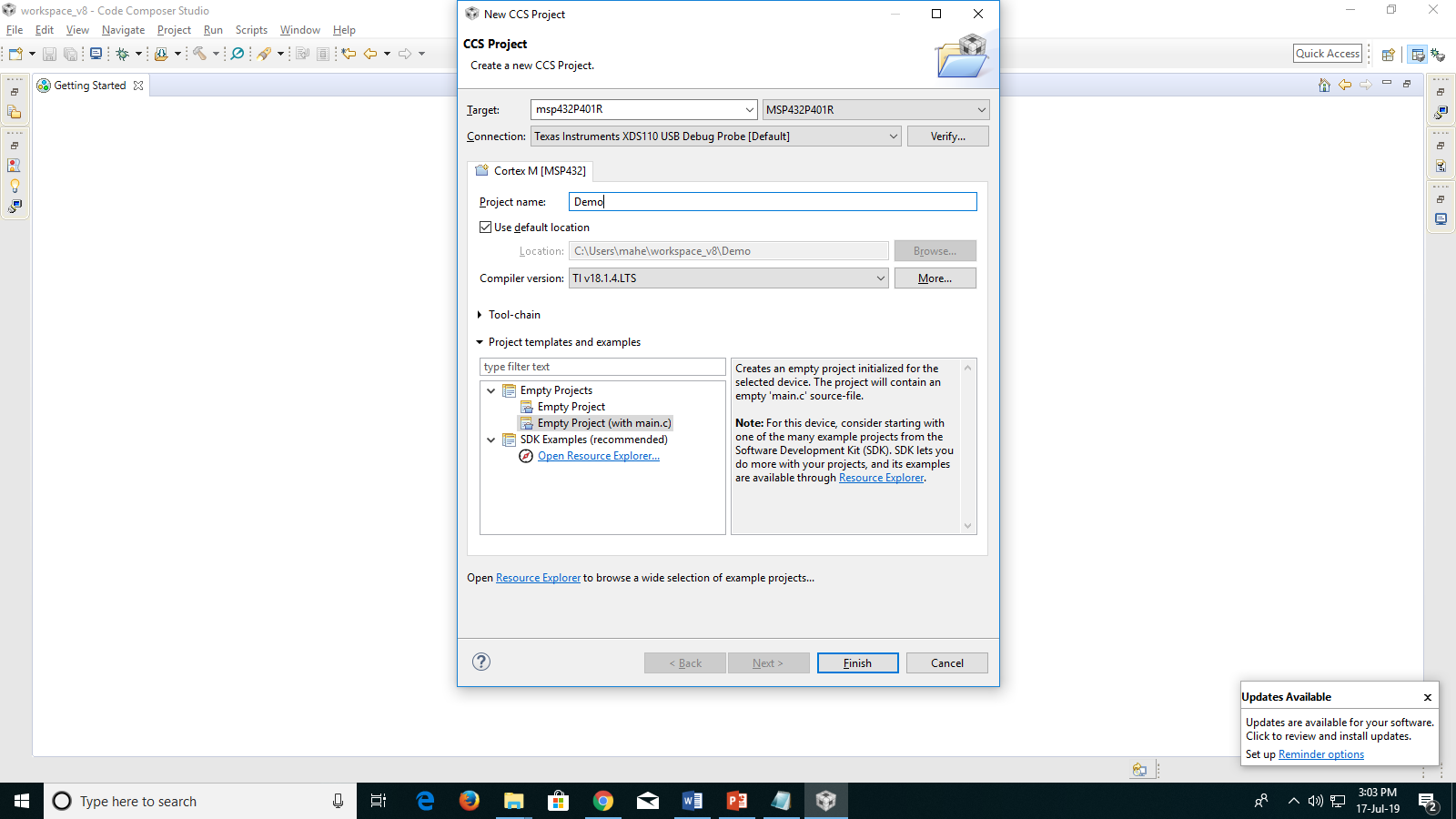
**STEPS OF INSTRUCTIONS:**

1. Open code composer and connect MSP432P401 using USB cable.

2. Click on new CCS project.



3. A project window will open. Under **target** type MSP432P401R. Name the Project and save it with just ALPHABETS.



4. Select Empty project with main.c and press finish or can create an empty file and follow below process.

5. Left top under project explorer click on expand symbol and press right button on mouse, and create a new C source file.

6. Create a file name and press ok.

7. The file will be present under project explorer, double click it and the editor will open.

8. All the programs are written under editor.

**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;

WDT\_A->CTL = WDT\_A\_CTL\_PW | WDT\_A\_CTL\_HOLD; // Stop watchdog timer

// 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 a C programming language to toggle RGB LED’s at pin P2.0, P2.1, P2.2.

Solution:

#include "msp.h"

int main(void)

{

volatile uint32\_t i;

WDT\_A->CTL = WDT\_A\_CTL\_PW | WDT\_A\_CTL\_HOLD; // Stop watchdog timer

// The following code toggles P2 port

P2->DIR |= 0xff; // Configure P2 as output

while(1)

{

P2->OUT ^= 0xff; // Toggle P2

for(i=10000; i>0; i--); // Delay

}

}

Exercise:

1. Write an assembly language program to toggle RGB LED’s individually. Set a visible frequency.

2. Write an assembly language program to switch RGB LED individually for one cycle and after RGB switch RED LED at P1.0. Repeat this cycle for 5 times and stop.

**EXPERIMENT NO.7: USE OF DELAY IN MSP432P401R**

AIM: To familiarize the Embedded C Programming with the use of delay.

Example 1: Write an Embedded C programming language to generate a delay of one second using 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 watchdog timer

P2->DIR |= 0xff;

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 XTML as 48MHz (crystal frequency).

Solution:

#include "msp.h"

uint32\_t ClockFrequency = 48000000;

int x, y, z;

void delay(unsigned long ulCount)

{

\_\_asm ( "pdloop: subs r0, #1\n" " bne pdloop\n");

}

void Clock\_Delay1ms(uint32\_t n)

{

while(n)

{

delay(ClockFrequency/9162); // 1 msec, tuned at 48 MHz

n--;

}

}

int main(void)

{

volatile uint32\_t i;

WDT\_A->CTL = WDT\_A\_CTL\_PW | WDT\_A\_CTL\_HOLD; // Stop watchdog timer

P2->DIR |= 0xff;

while(1)

{

P2->OUT ^= 0xff;

Clock\_Delay1ms(100);

}

}

Exercise:

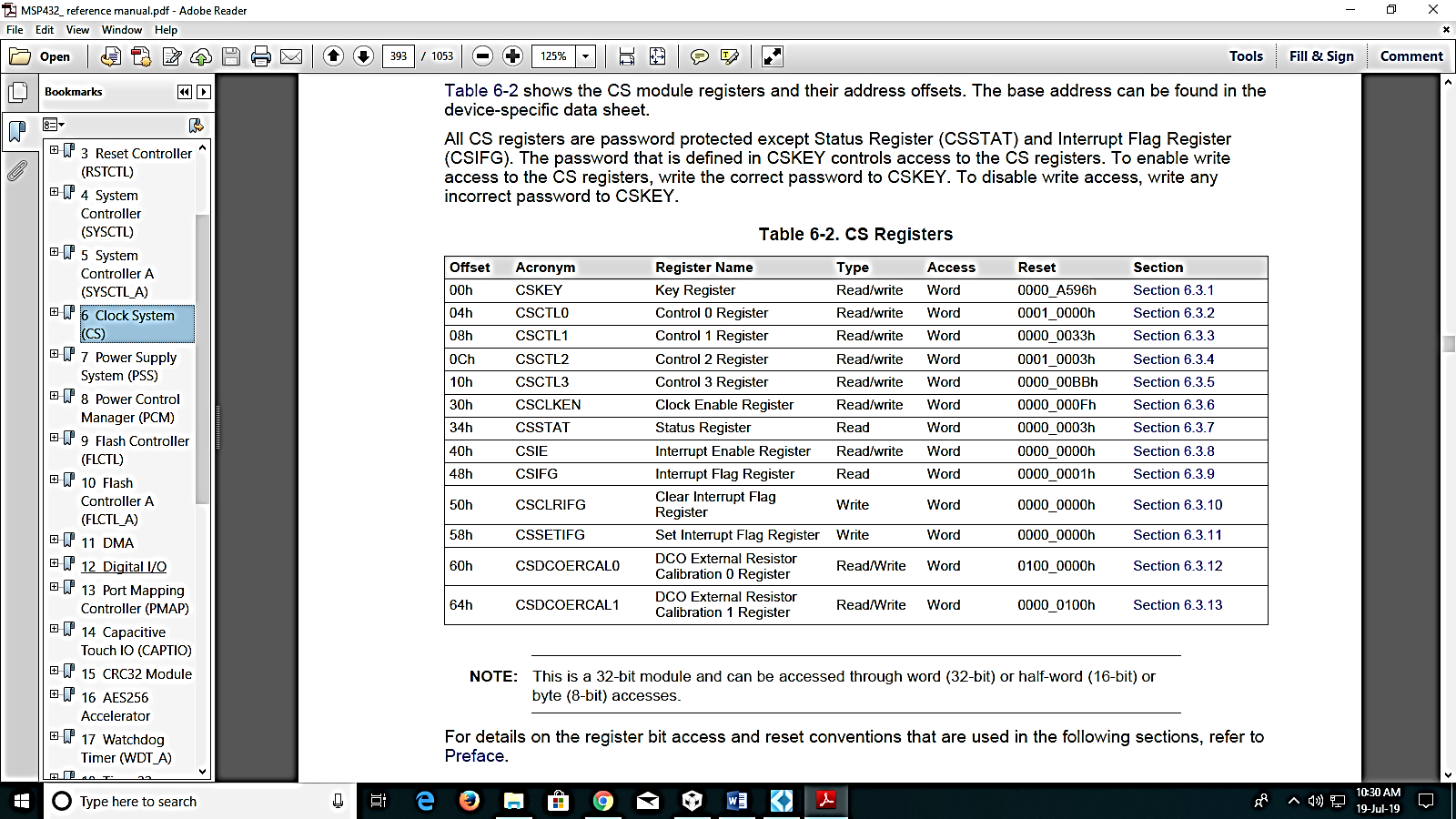
1. Write an Embedded C programming language for generating a delay of 4 seconds using a 3MHz crystal frequency and toggle RED LED at P1.0.

2. Write an Embedded C programming language for switching RED LED, GREEN LED, BLUE LED each individually ON after 2 seconds of delay simultaneously use 12MHz clock for scaling.

3. Write an Embedded C programming language to 2 LED’s with different clocks 12MHz and 48MHz simultaneously. Condition: Both the LED’s should be scaled to 1sec delay of toggle.

**EXPERIMENT NO.8: READING I/O PORTS IN MSP432P401R**

AIM: To familiarize the Embedded C Programming with the use of INPUT AND OUTPUT ports/pins for interfacing.



*NOTE: Data sheet of MSP432*

Example 1: Write an Embedded C programming language to read data from P1.1 and write a toggling value to RGB green LED.

Solution:

#include "msp.h"

void main(void)

{

WDT\_A->CTL = WDT\_A\_CTL\_PW | WDT\_A\_CTL\_HOLD; // stop watchdog timer

//CS->KEY = CS\_KEY\_VAL;

//CS->CTL0 = CS\_CTL0\_DCOEN | CS\_CTL0\_DCORSEL\_1;

//CS->CTL1 = CS\_CTL1\_SELM\_3;

//CS->KEY = 0;

P2->DIR |= BIT1;

P1->DIR &= ~BIT1;

P1->REN = BIT1;

P1->DIR &= ~BIT1;

P1->IES = BIT1;

P1->IFG = 0;

P1->IE = BIT1;

NVIC-> ISER[1] = 1<< (PORT1\_IRQn & 31);

\_\_enable\_irq();

}

void PORT1\_IRQHandler(void)

{

if (P1->IFG & BIT1)

{

P2->OUT^= BIT1;

}

P1->IFG&= ~BIT1;

}

Example 2: Write an Embedded C programming language to read data from P1.1 and write a toggling value to RGB green LED with a delay of 1 second using a clock frequency of 48MHz.

Solution:

#include "msp.h"

uint32\_t ClockFrequency = 48000000;

int x, y, z;

void delay(unsigned long ulCount)

{

\_\_asm ( "pdloop: subs r0, #1\n" " bne pdloop\n");

}

void Clock\_Delay1ms(uint32\_t n)

{

while(n)

{

delay(ClockFrequency/9162); // 1 msec, tuned at 48 MHz

n--;

}

}

void main(void)

{

WDT\_A->CTL = WDT\_A\_CTL\_PW | WDT\_A\_CTL\_HOLD; // stop watchdog timer

CS->KEY = CS\_KEY\_VAL;

CS->CTL0 = CS\_CTL0\_DCOEN | CS\_CTL0\_DCORSEL\_1;

CS->CTL1 = CS\_CTL1\_SELM\_3;

CS->KEY = 0;

P2->DIR |= BIT1;

P1->DIR &= ~BIT1;

P1->REN = BIT1;

P1->DIR &= ~BIT1;

P1->IES = BIT1;

P1->IFG = 0;

P1->IE = BIT1;

NVIC-> ISER[1] = 1<< (PORT1\_IRQn & 31);

\_\_enable\_irq();

}

void PORT1\_IRQHandler(void)

{

if (P1->IFG & BIT1)

{

Clock\_Delay1ms(200);

P2->OUT^= BIT1;

}

P1->IFG&= ~BIT1;

}

Exercise:

1. Write an Embedded C programming for toggling RGB LED simultaneously after pressing P1.4 with a delay of 1 second using predefined clock 48MHz.

2. Write an Embedded C programming for interfacing P1.1 and P1.4 switches, with a condition of AND gate interfaced with RED status LED at P1.0.

3. Write an Embedded C programming for interfacing P1.1 and P1.4 switches, with a condition of OR gate interfaced with RED status LED at P1.0.

**EXPERIMENT NO.9: PERFORMING TIMERS AND COUNTER OPERATION FOR MSP432P401R**

AIM: To familiarize the Embedded C Programming with the use of Timers.

Example 1: Write an Embedded C programming language to utilize timers as an interrupt to blink a RED LED at P1.0.

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;

// Timer32 set up in one-shot, free run, 32-bit, no pre-scale

TIMER32\_1->CONTROL = TIMER32\_CONTROL\_SIZE | TIMER32\_CONTROL\_ONESHOT;//0x03;

// Load Timer32 Counter with initial value

TIMER32\_1->LOAD= 0xFFFFFF;

// Enalbe the Timer32 interrupt in NVIC

\_\_enable\_irq();

NVIC->ISER[0] = 1 << ((T32\_INT1\_IRQn) & 31);

// Start Timer32 and enable interrupt

TIMER32\_1->CONTROL |= TIMER32\_CONTROL\_ENABLE |

TIMER32\_CONTROL\_IE;//0xA0;

// Disable sleep on exit from ISR

SCB->SCR &= ~SCB\_SCR\_SLEEPONEXIT\_Msk;

// Enter LPM0 and wait for the timer interrupt to wake-up

\_\_sleep();

\_\_no\_operation();

// Timer interrupt wakes-up

while (1)

{

P1->OUT ^= BIT0; // Toggle P1.0 LED

for (i = 100000; i > 0; i--); // Delay

}

}

void T32\_INT1\_IRQHandler(void)

{

TIMER32\_1->INTCLR |= BIT0; // Clear Timer32 interrupt flag

// Disable the timer and interrupt

TIMER32\_1->CONTROL &= ~(TIMER32\_CONTROL\_ENABLE | TIMER32\_CONTROL\_IE);

}

Example 2: Write an Embedded C programming language to generate accurate hardware delay for blinking LED by initializing the PORT inside an Interrupt handler.

Solution:

#include "msp.h"

int main(void)

{

WDT\_A->CTL = WDT\_A\_CTL\_PW |WDT\_A\_CTL\_HOLD;

// Configure GPIO

P1->DIR |= BIT0;

P1->OUT &= ~BIT0;

// Timer32 set up in periodic mode, 32-bit, no pre-scale

TIMER32\_1->CONTROL = TIMER32\_CONTROL\_SIZE |TIMER32\_CONTROL\_MODE;

// Load Timer32 counter with period = 0x20000

TIMER32\_1->LOAD= 0x2000000;

// Enalbe the Timer32 interrupt in NVIC

\_\_enable\_irq();

NVIC->ISER[0] = 1 << ((T32\_INT1\_IRQn) & 31);

// Enable sleep on exit from ISR

SCB->SCR |= SCB\_SCR\_SLEEPONEXIT\_Msk;

// Start Timer32 with interrupt enabled

TIMER32\_1->CONTROL |= TIMER32\_CONTROL\_ENABLE |TIMER32\_CONTROL\_IE;

// Enter LPM0;

\_\_sleep();

// For the debugger when halt is asserted. This is only needed for debug mode

\_\_no\_operation();

}

void T32\_INT1\_IRQHandler(void)

{

TIMER32\_1->INTCLR |= BIT0; // Clear Timer32 interrupt flag

P1->OUT ^= BIT0; // Toggle P1.0 LED

}

Exercise:

1. Write an Embedded C programming language to toggle RED LED for every 1 second using timers’ concept. (Set the clock from Pre scalar, refer system\_MSP432P401R.c source file with data sheet).

2. Write an Embedded C Programming language to toggle RED LED for 1 second and RGB LED each 1second simultaneously. Strictly use the timer32 and IQR Handlers for invoking delay.

**EXPERIMENT NO.10: TASK SPECIFIC/APPLICATION BASED PROGRAMMING**

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

*Note: One can use all the peripheral I/O’s available in MSP432P401R to execute the output, like the switches and LED’s. Use of Hardware delay would be appreciated.*

1. Design a logic and perform the logic through embedded C programming.

a. Inspection system in a cola factory to detect the defective bottle.

b. Safety system in vehicle.

c. Pick and Place robot.

d. Object detection by a mobile robot.

Write the logic by picking any two topics and create an application that is specific with mechatronics. Design a logical code that is best fit for the application selected and also make a write-up justifying the application. Students can pick Keil or Code composer (any).