Faculty of Engineering
Computer and Systems Engineering Department

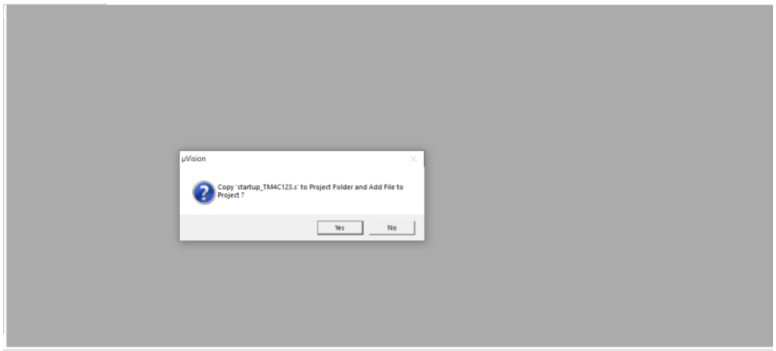
CSE 211s [Spring 2024]
Introduction to Embedded Systems

Lab 4

Introduction

To use the simulated ports of TivaC launchpad, follow the below steps:

1. Create new project.
2. Choose the target TM4C123GH6PM device.
3. Copy the start-up code of TM4C123GH6PM.

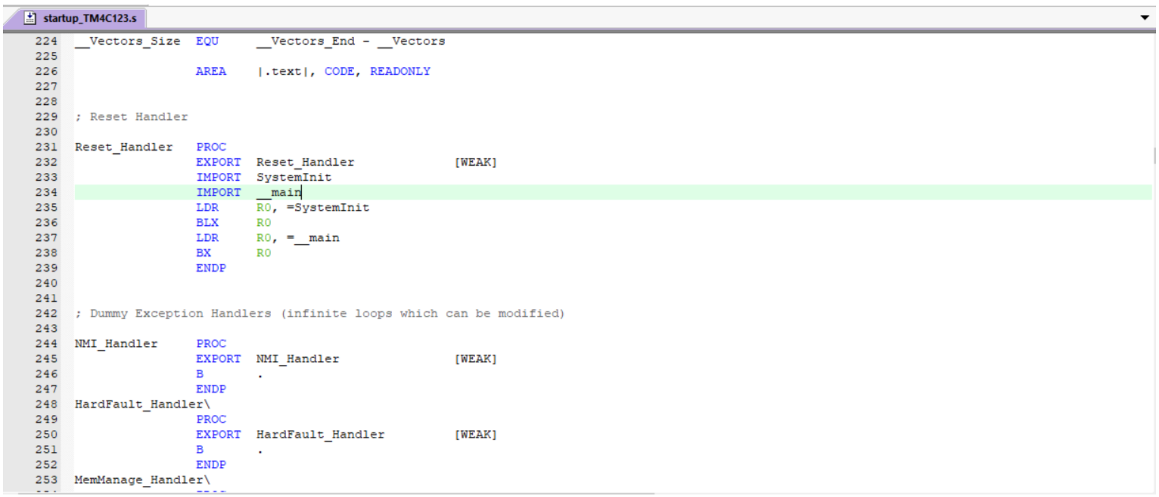


Copy 'startup_TM4C123.s' to Project Folder and Add File to Project ?

4. Remove instruction "IMPORT SystemInit"

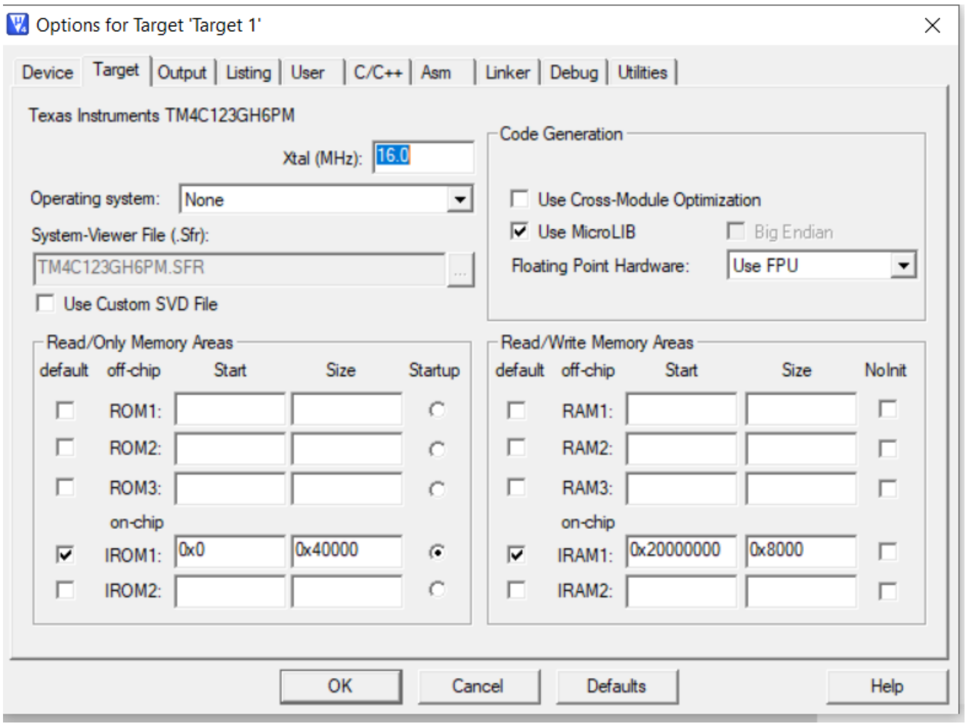
5. Remove instruction "LDR R0, =SystemInit"

6. Remove instruction "BLX R0" at line 236 as shown below



```
224  __Vectors_Size  EQU      __Vectors_End - __Vectors
225
226                  AREA     |.text|, CODE, READONLY
227
228
229  ; Reset Handler
230
231  Reset_Handler   PROC
232                  EXPORT   Reset_Handler             [WEAK]
233                  IMPORT   SystemInit
234                  IMPORT   __main
235                  LDR      R0, =SystemInit
236                  BLX      R0
237                  LDR      R0, =__main
238                  BX       R0
239                  ENDP
240
241
242  ; Dummy Exception Handlers (infinite loops which can be modified)
243
244  NMI_Handler     PROC
245                  EXPORT   NMI_Handler               [WEAK]
246                  B        .
247                  ENDP
248  HardFault_Handler\
249                  PROC
250                  EXPORT   HardFault_Handler         [WEAK]
251                  B        .
252                  ENDP
253  MemManage_Handler\
```

7. Adjust the settings of the target by checking MicroLIB field in Code Generation options.



Texas Instruments TM4C123GH6PM

Xtal (MHz): 16.0

Operating system: None

System-Viewer File (.Sfr): TM4C123GH6PM.SFR

□ Use Custom SVD File

Code Generation

□ Use Cross-Module Optimization

☑ Use MicroLIB □ Big Endian

Floating Point Hardware: Use FPU

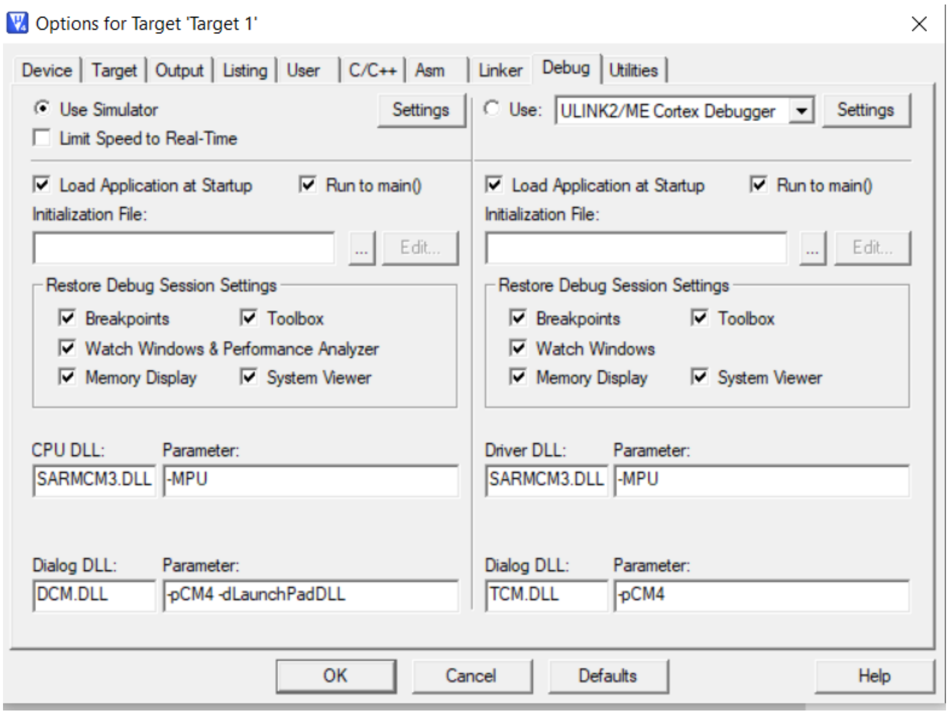
Read/Only Memory Areas

default	off-chip	Start	Size	Startup
□	ROM1:			○
□	ROM2:			○
□	ROM3:			○
	on-chip			
☑	IROM1:	0x0	0x40000	●
□	IROM2:			○

Read/Write Memory Areas

default	off-chip	Start	Size	NoInit
□	RAM1:			□
□	RAM2:			□
□	RAM3:			□
	on-chip			
☑	IRAM1:	0x20000000	0x8000	□
□	IRAM2:			□

8. Adjust the settings of the target by adding “-dLaunchPadDLL” in Parameter field to support the simulation in Keil 4.



● Use Simulator

□ Limit Speed to Real-Time

☑ Load Application at Startup ☑ Run to main()

Initialization File:

Restore Debug Session Settings

☑ Breakpoints ☑ Toolbox

☑ Watch Windows & Performance Analyzer

☑ Memory Display ☑ System Viewer

CPU DLL: SARMCM3.DLL Parameter: -MPU

Dialog DLL: DCM.DLL Parameter: -pCM4 -dLaunchPadDLL

○ Use: ULINK2/ME Cortex Debugger

☑ Load Application at Startup ☑ Run to main()

Initialization File:

Restore Debug Session Settings

☑ Breakpoints ☑ Toolbox

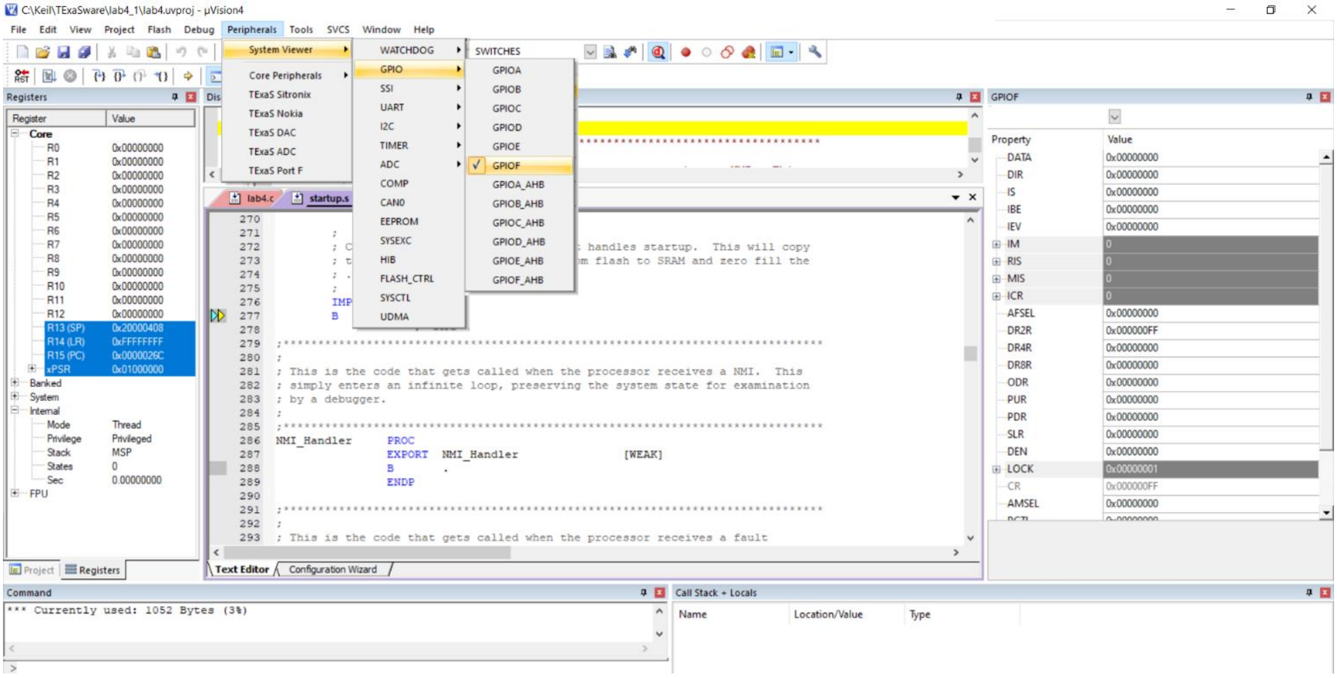
☑ Watch Windows

☑ Memory Display ☑ System Viewer

Driver DLL: SARMCM3.DLL Parameter: -MPU

Dialog DLL: TCM.DLL Parameter: -pCM4

9. You can view the values of the port F through Peripherals in tool bar -> SystemViewer-> GPIO->GPIOF.



Register	Value
Core	
R0	0x00000000
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20000408
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x0000026C
xPSR	0x01000000
Banked	
System	
Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP
States	0
Sec	0.00000000
FPU	

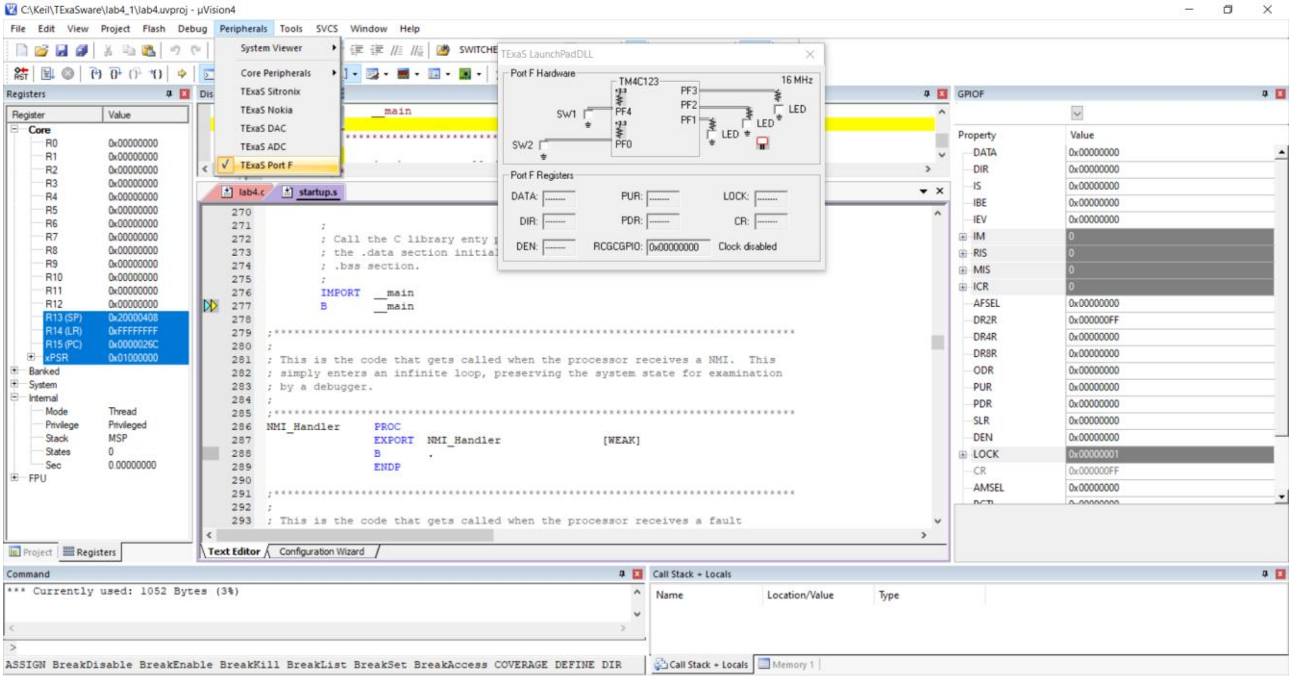
```
        ;
        ; C                                        handles startup.  This will copy
        ; t                                       m flash to SRAM and zero fill the
        ; .
        ;
        IMP
        B

;******************************************************************************
;
; This is the code that gets called when the processor receives a NMI.  This
; simply enters an infinite loop, preserving the system state for examination
; by a debugger.
;
;******************************************************************************
NMI_Handler     PROC
                EXPORT  NMI_Handler               [WEAK]
                B       .
                ENDP

;******************************************************************************
;
; This is the code that gets called when the processor receives a fault
```

Property	Value
DATA	0x00000000
DIR	0x00000000
IS	0x00000000
IBE	0x00000000
IEV	0x00000000
IM	0
RIS	0
MIS	0
ICR	0
AFSEL	0x00000000
DR2R	0x000000FF
DR4R	0x00000000
DR8R	0x00000000
ODR	0x00000000
PUR	0x00000000
PDR	0x00000000
SLR	0x00000000
DEN	0x00000000
LOCK	0x00000001
CR	0x000000FF
AMSEL	0x00000000

```
*** Currently used: 1052 Bytes (3%)
```

Name	Location/Value	Type

10. You can simulate the behavior of switches that are connected to port F in TivaC and check the behavior of the three built-in LEDs in the kit through Peripherals in tool bar -> TExaS PortF.



DATA: -------- PUR: -------- LOCK: --------
DIR: -------- PDR: -------- CR: --------
DEN: -------- RCGCGPIO: 0x00000000 Clock disabled

Register	Value
Core	
R0	0x00000000
R1	0x00000000
R2	0x00000000
R3	0x00000000
R4	0x00000000
R5	0x00000000
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x20000408
R14 (LR)	0xFFFFFFFF
R15 (PC)	0x0000026C
xPSR	0x01000000
Banked	
System	
Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP
States	0
Sec	0.00000000
FPU	

```
                                                  __main
        ;
        ; Call the C library enty p
        ; the .data section initial
        ; .bss section.
        ;
        IMPORT  __main
        B       __main

;******************************************************************************
;
; This is the code that gets called when the processor receives a NMI.  This
; simply enters an infinite loop, preserving the system state for examination
; by a debugger.
;
;******************************************************************************
NMI_Handler     PROC
                EXPORT  NMI_Handler               [WEAK]
                B       .
                ENDP

;******************************************************************************
;
; This is the code that gets called when the processor receives a fault
```

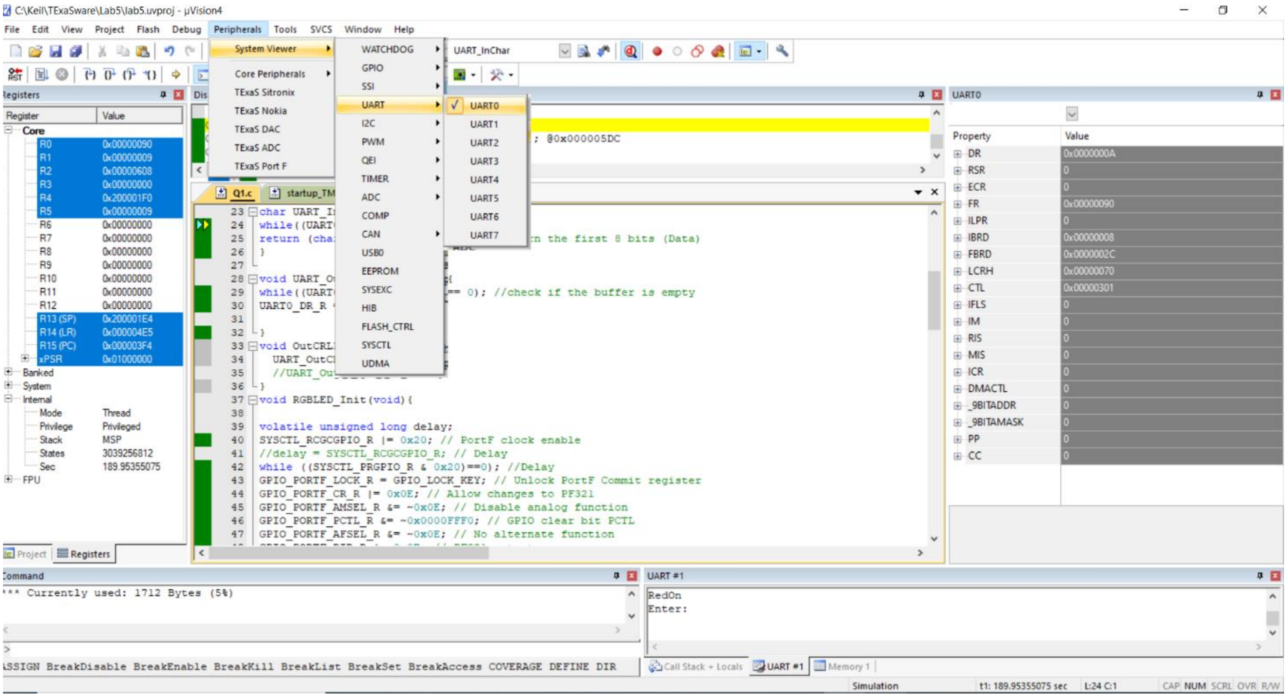
Property	Value
DATA	0x00000000
DIR	0x00000000
IS	0x00000000
IBE	0x00000000
IEV	0x00000000
IM	0
RIS	0
MIS	0
ICR	0
AFSEL	0x00000000
DR2R	0x000000FF
DR4R	0x00000000
DR8R	0x00000000
ODR	0x00000000
PUR	0x00000000
PDR	0x00000000
SLR	0x00000000
DEN	0x00000000
LOCK	0x00000001
CR	0x000000FF
AMSEL	0x00000000

```
*** Currently used: 1052 Bytes (3%)
```

```
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE DEFINE DIR
```

Name	Location/Value	Type

11. You can check the values of any UART registers through Peripherals in tool bar-> System Viewer-> UART->UART<x>.



Register	Value
Core	
R0	0x00000090
R1	0x00000009
R2	0x00000608
R3	0x00000000
R4	0x200001F0
R5	0x00000009
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x200001E4
R14 (LR)	0x000004E5
R15 (PC)	0x000003F4
xPSR	0x01000000
Banked	
System	
Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP
States	3039256812
Sec	189.95355075
FPU	

UART0

Property	Value
DR	0x0000000A
RSR	0
ECR	0
FR	0x00000090
ILPR	0
IBRD	0x00000008
FBRD	0x0000002C
LCRH	0x00000070
CTL	0x00000301
IFLS	0
IM	0
RIS	0
MIS	0
ICR	0
DMACTL	0
_9BITADDR	0
_9BITAMASK	0
PP	0
CC	0

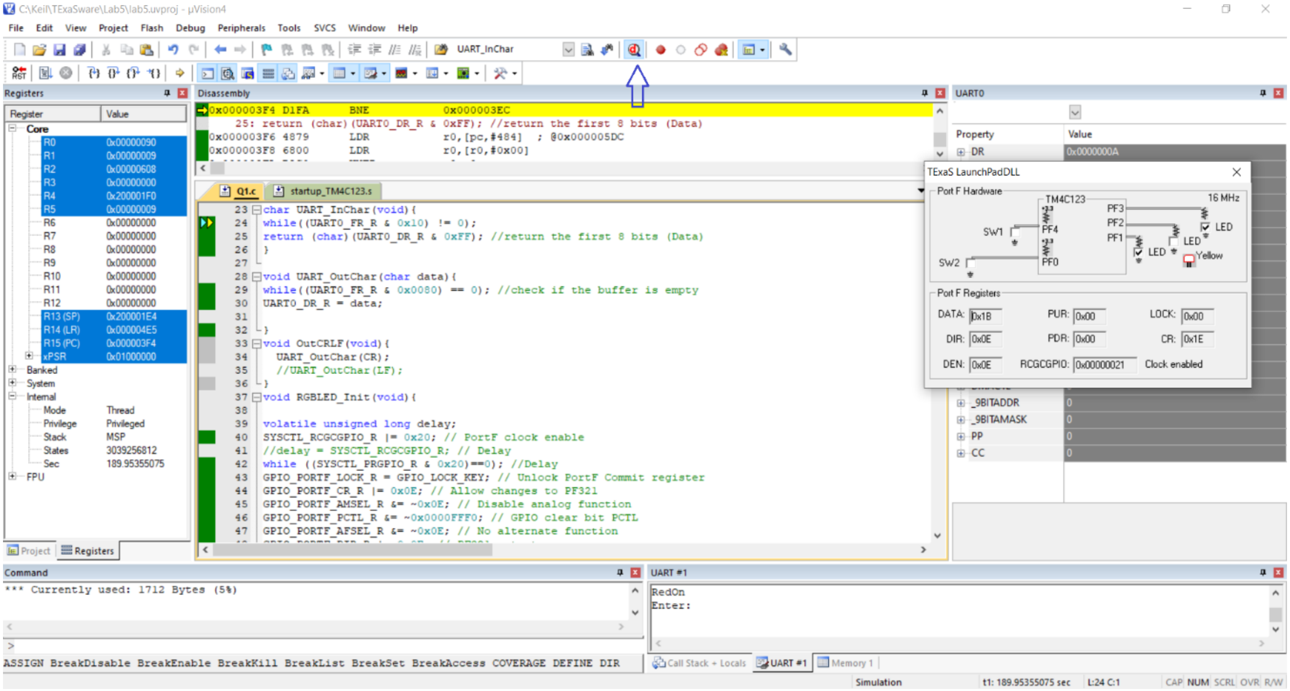
```
*** Currently used: 1712 Bytes (5%)
```

```
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE DEFINE DIR
```

UART #1

```
RedOn
Enter:
```

12. After completing your code and building it. Press on debug button in tool bar.


C:\Keil\TExaSware\Lab5\lab5.uvproj - µVision4
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
UART_InChar
Registers
Disassembly
Q1.c
startup_TM4C123.s
UART0
TExaS LaunchPadDLL
Port F Hardware
TM4C123
16 MHz
SW1
SW2
PF4
PF0
PF3
PF2
PF1
LED
LED
LED
Yellow
Project Registers
Call Stack + Locals UART #1 Memory 1
Simulation t1: 189.95355075 sec L:24 C:1 CAP NUM SCRL OVR R/W

Disassembly

```
0x000003F4 D1FA      BNE      0x000003EC
    25: return (char)(UART0_DR_R & 0xFF); //return the first 8 bits (Data)
0x000003F6 4879      LDR      r0,[pc,#484]  ; @0x000005DC
0x000003F8 6800      LDR      r0,[r0,#0x00]
```

Q1.c

```
char UART_InChar(void){
while((UART0_FR_R & 0x10) != 0);
return (char)(UART0_DR_R & 0xFF); //return the first 8 bits (Data)
}

void UART_OutChar(char data){
while((UART0_FR_R & 0x0080) == 0); //check if the buffer is empty
UART0_DR_R = data;

}
void OutCRLF(void){
  UART_OutChar(CR);
  //UART_OutChar(LF);
}
void RGBLED_Init(void){

volatile unsigned long delay;
SYSCTL_RCGCGPIO_R |= 0x20; // PortF clock enable
//delay = SYSCTL_RCGCGPIO_R; // Delay
while ((SYSCTL_PRGPIO_R & 0x20)==0); //Delay
GPIO_PORTF_LOCK_R = GPIO_LOCK_KEY; // Unlock PortF Commit register
GPIO_PORTF_CR_R |= 0x0E; // Allow changes to PF321
GPIO_PORTF_AMSEL_R &= ~0x0E; // Disable analog function
GPIO_PORTF_PCTL_R &= ~0x0000FFF0; // GPIO clear bit PCTL
GPIO_PORTF_AFSEL_R &= ~0x0E; // No alternate function
```

Port F Registers

DATA:	0x1B	PUR:	0x00	LOCK:	0x00
DIR:	0x0E	PDR:	0x00	CR:	0x1E
DEN:	0x0E	RCGCGPIO:	0x00000021	Clock enabled	

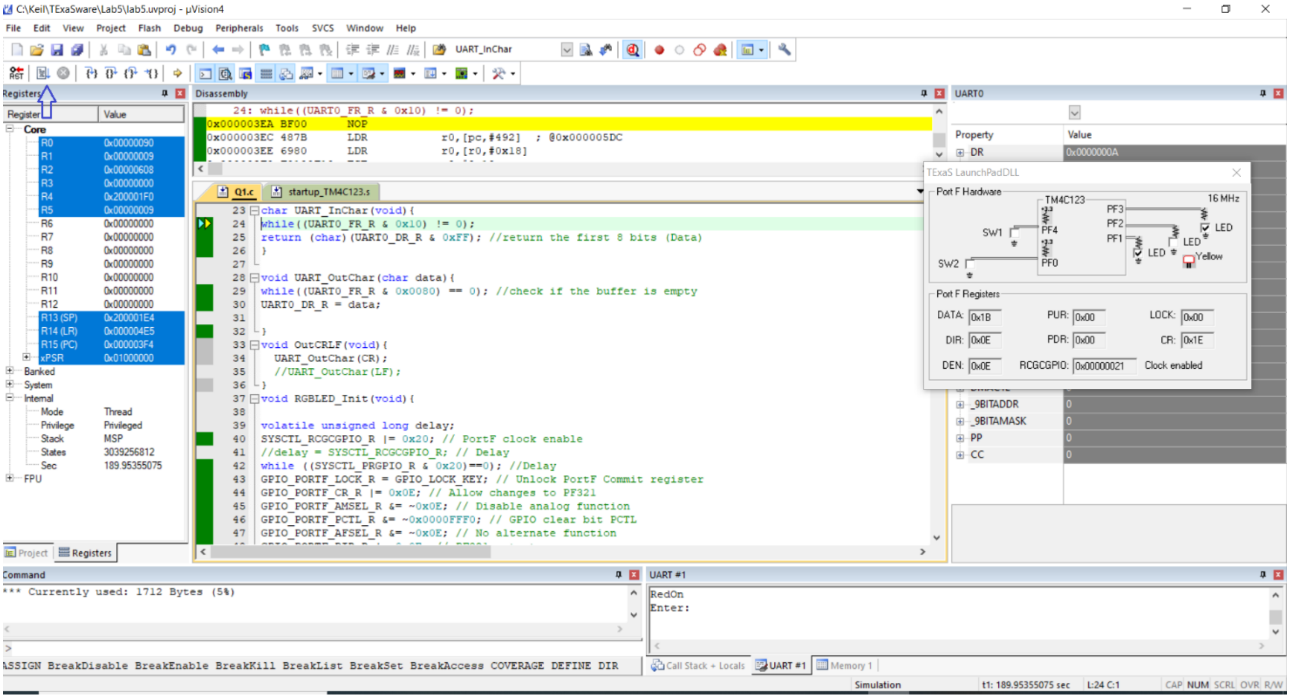
```
*** Currently used: 1712 Bytes (5%)
```

```
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE DEFINE DIR
```

UART #1

```
RedOn
Enter:
```

13. After pressing on debug button, press on run button in tool bar as shown below.



Register	Value
Core	
R0	0x00000090
R1	0x00000009
R2	0x00000608
R3	0x00000000
R4	0x200001F0
R5	0x00000009
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x200001E4
R14 (LR)	0x000004E5
R15 (PC)	0x000003F4
xPSR	0x01000000
Banked	
System	
Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP
States	3039256812
Sec	189.95355075
FPU	

```
    24: while((UART0_FR_R & 0x10) != 0);
0x000003EA BF00      NOP
0x000003EC 487B      LDR      r0,[pc,#492]  ; @0x000005DC
0x000003EE 6980      LDR      r0,[r0,#0x18]
```

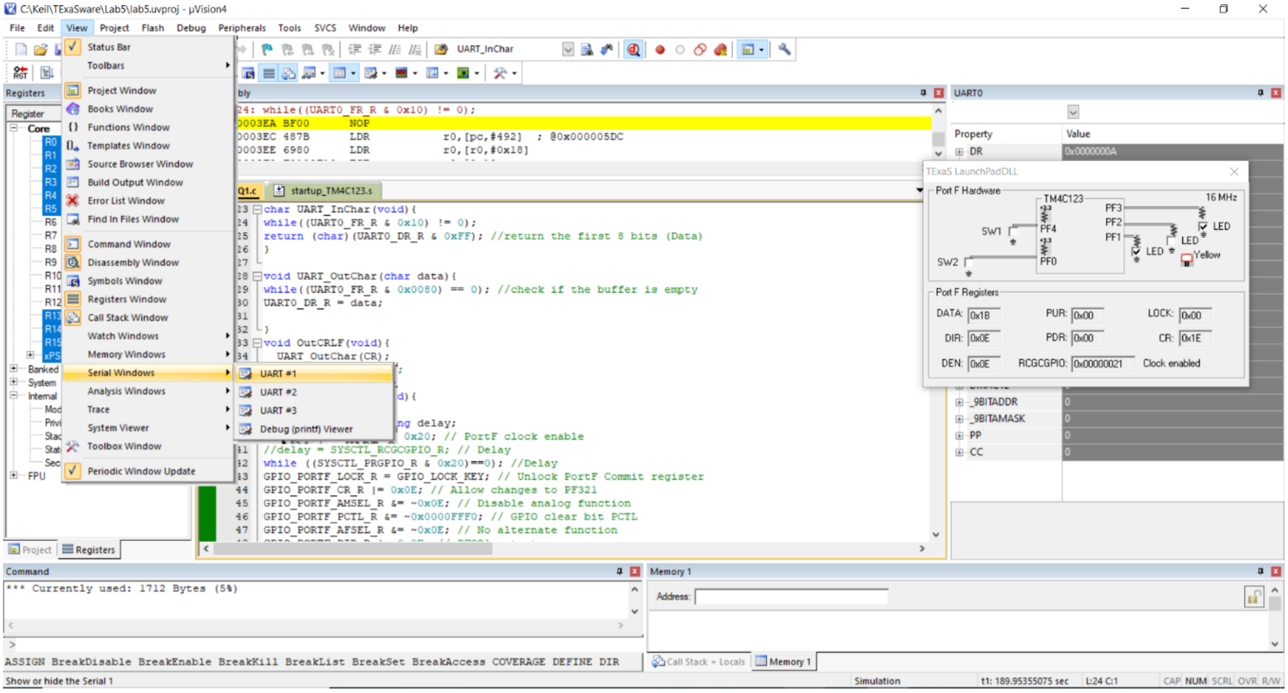
```
char UART_InChar(void){
while((UART0_FR_R & 0x10) != 0);
return (char)(UART0_DR_R & 0xFF); //return the first 8 bits (Data)
}

void UART_OutChar(char data){
while((UART0_FR_R & 0x0080) == 0); //check if the buffer is empty
UART0_DR_R = data;

}
void OutCRLF(void){
  UART_OutChar(CR);
  //UART_OutChar(LF);
}
void RGBLED_Init(void){

volatile unsigned long delay;
SYSCTL_RCGCGPIO_R |= 0x20; // PortF clock enable
//delay = SYSCTL_RCGCGPIO_R; // Delay
while ((SYSCTL_PRGPIO_R & 0x20)==0); //Delay
GPIO_PORTF_LOCK_R = GPIO_LOCK_KEY; // Unlock PortF Commit register
GPIO_PORTF_CR_R |= 0x0E; // Allow changes to PF321
GPIO_PORTF_AMSEL_R &= ~0x0E; // Disable analog function
GPIO_PORTF_PCTL_R &= ~0x0000FFF0; // GPIO clear bit PCTL
GPIO_PORTF_AFSEL_R &= ~0x0E; // No alternate function
```

```
*** Currently used: 1712 Bytes (5%)
```

```
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE DEFINE DIR
```

```
RedOn
Enter:
```

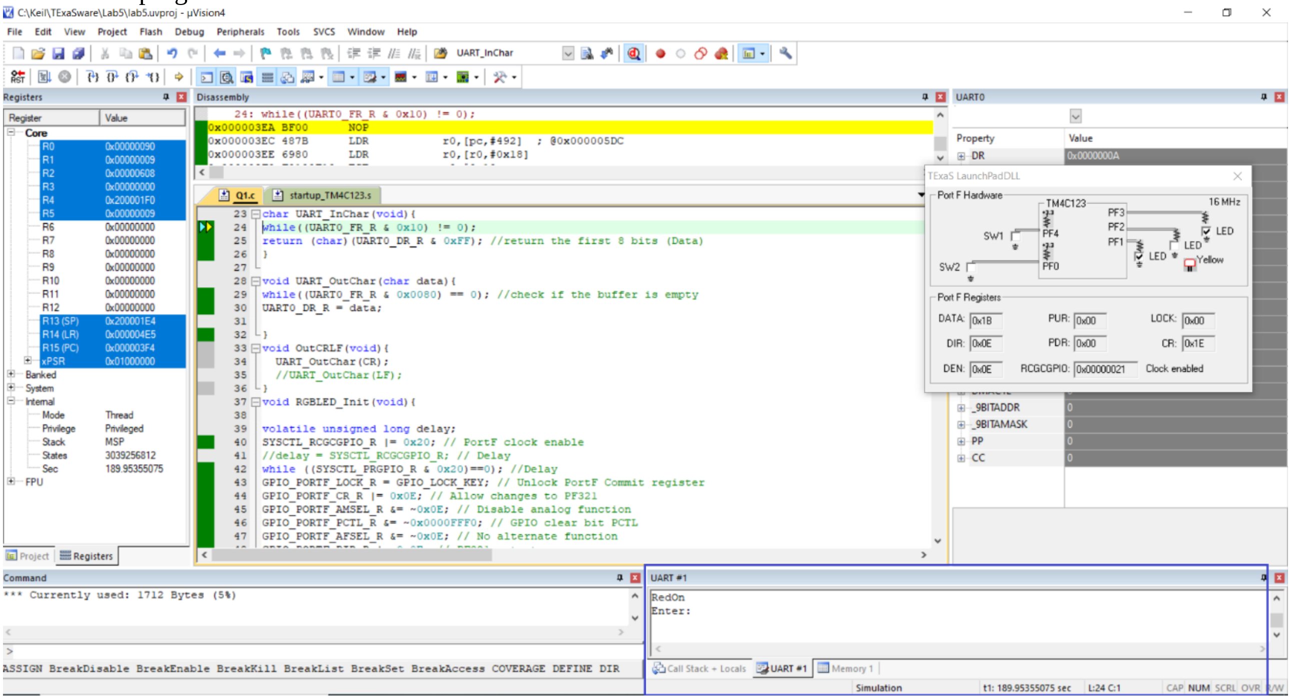
14. From view in tool bar -> choose Serial Window-> then choose UART#1 to open Keil terminal.



```
*** Currently used: 1712 Bytes (5%)
```

```
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE DEFINE DIR
```

15. The UART#1 serial window will be appeared to provide you the capability to enter your inputs during running the program.



C:\Keil\TExaSware\Lab5\lab5.uvproj - µVision4

Register	Value
Core	
R0	0x00000090
R1	0x00000009
R2	0x00000608
R3	0x00000000
R4	0x200001F0
R5	0x00000009
R6	0x00000000
R7	0x00000000
R8	0x00000000
R9	0x00000000
R10	0x00000000
R11	0x00000000
R12	0x00000000
R13 (SP)	0x200001E4
R14 (LR)	0x000004E5
R15 (PC)	0x000003F4
xPSR	0x01000000
Banked	
System	
Internal	
Mode	Thread
Privilege	Privileged
Stack	MSP
States	3039256812
Sec	189.95355075
FPU	

```
    24: while((UART0_FR_R & 0x10) != 0);
0x000003EA BF00      NOP
0x000003EC 487B      LDR      r0,[pc,#492]  ; @0x000005DC
0x000003EE 6980      LDR      r0,[r0,#0x18]
```

Q1.c   startup_TM4C123.s

```
char UART_InChar(void){
while((UART0_FR_R & 0x10) != 0);
return (char)(UART0_DR_R & 0xFF); //return the first 8 bits (Data)
}

void UART_OutChar(char data){
while((UART0_FR_R & 0x0080) == 0); //check if the buffer is empty
UART0_DR_R = data;

}
void OutCRLF(void){
  UART_OutChar(CR);
  //UART_OutChar(LF);
}
void RGBLED_Init(void){

volatile unsigned long delay;
SYSCTL_RCGCGPIO_R |= 0x20; // PortF clock enable
//delay = SYSCTL_RCGCGPIO_R; // Delay
while ((SYSCTL_PRGPIO_R & 0x20)==0); //Delay
GPIO_PORTF_LOCK_R = GPIO_LOCK_KEY; // Unlock PortF Commit register
GPIO_PORTF_CR_R |= 0x0E; // Allow changes to PF321
GPIO_PORTF_AMSEL_R &= ~0x0E; // Disable analog function
GPIO_PORTF_PCTL_R &= ~0x0000FFF0; // GPIO clear bit PCTL
GPIO_PORTF_AFSEL_R &= ~0x0E; // No alternate function
```

UART0

Property	Value
DR	0x0000000A
_9BITADDR	0
_9BITAMASK	0
PP	0
CC	0

TExaS LaunchPadDLL

Port F Hardware

Port F Registers

DATA: 0x1B   PUR: 0x00   LOCK: 0x00
DIR: 0x0E   PDR: 0x00   CR: 0x1E
DEN: 0x0E   RCGCGPIO: 0x00000021   Clock enabled

```
*** Currently used: 1712 Bytes (5%)
```

```
ASSIGN BreakDisable BreakEnable BreakKill BreakList BreakSet BreakAccess COVERAGE DEFINE DIR
```

```
RedOn
Enter:
```

16. Use the previous steps to help you out to simulate the following lab.

Lab Exercise

Write an Embedded C program that receives commands through UART communication protocol to turn the LEDs on and off.

The commands are:

1. RedOn → Turn RED LED on
2. RedOff→ Turn RED LED off
3. GreenOn → Turn Green LED on
4. GreenOff → Turn Green LED off
5. BlueOn→ Turn Blue LED on
6. BlueOff→ Turn Blue LED off
7. Anything else turn all the LEDs off

Your code should support the following assumptions:

1. Any command that turns on a specified LED should not affect the other LEDs.
2. Anything is sent except those commands should reset all the LEDs (reset the only pins that are connected to LEDs).
3. Any command that turns off a specified LED should not affect the other LEDs.
4. The new command is completed by pressing Enter.

Lab Submission

Q2. Write Embedded C program that receives commands through UART communication protocol to do the following:

1. When sending “A”, all the LEDs are turned off and the Red LED is turned on after 1 minute.
2. When sending “B”, all the LEDs are turned off and the Blue LED is turned on after 0.5 minutes.
3. When sending “D”, all the LEDs are turned off and the Green LED is turned on after 2 minutes.

Upon starting the program, all the LEDS should be turned off.

Check through the simulated Kit that the behavior of your code is correct.

For the lab submission, you should submit a pdf document contains the following.

1. Cover page that contains
    a. your name,
    b. your ID,
    c. your department
2. Place snapshots to show the state of the LEDs.
3. The snapshots must show the values of the GPIOF registers such as (DATA, DIR, AFSEL, … etc.), UART registers, and UART serial window when you verify your code on simulation level.
4. Place your code in the document.
5. Your document will be submitted on LMS.