Embedded System Series
Core 1: Microcontroller (MCU) Fundamentals with Hands-on

Dr. Mohammed Hammouda
University Lecturer

abdrabou@outlook.com

October 29, 2024

Course Objectives

- Develop a general understanding of programming fundamentals using C, the most widely used language in embedded systems.

Course Objectives

- Develop a general understanding of programming fundamentals using C, the most widely used language in embedded systems.
- Develop a comprehensive understanding on how MCU based systems work and how to program them.

Course Objectives

- Develop a general understanding of programming fundamentals using C, the most widely used language in embedded systems.
- Develop a comprehensive understanding on how MCU based systems work and how to program them.
- Explore the ARM Cortex M4 architecture in detail.

Course Objectives

- Develop a general understanding of programming fundamentals using C, the most widely used language in embedded systems.
- Develop a comprehensive understanding on how MCU based systems work and how to program them.
- Explore the ARM Cortex M4 architecture in detail.
- Gain a hands-on experience using tools like Eclipse IDE and Code Composer Studio (CCS) and Tiva C launchpad board.

Course Objectives

- Develop a general understanding of programming fundamentals using C, the most widely used language in embedded systems.
- Develop a comprehensive understanding on how MCU based systems work and how to program them.
- Explore the ARM Cortex M4 architecture in detail.
- Gain a hands-on experience using tools like Eclipse IDE and Code Composer Studio (CCS) and Tiva C launchpad board.
- Capable of applying studied concepts to advanced embedded systems.

References

- Isac Artzi's, "**C Essential Training**", Lynda.com (now LinkedIn Learning), 2018.
- Simon Allardice, "**Foundation of Progamming: Fundamentals**", Lynda.com (now LinkedIn Learning), 2011
- J. Valvano, R. Yerraballi "**Introduction to Embedded Systems**", `https://users.ece.utexas.edu/~valvano/Volume1/`, 2024.
- Daler N. Rakhmatov, "**Microprocessor Based Systems**", University of Victoria, 2023
- V. Hamacher, Z. Vranesic, M. Zaky, and A. Manjikian, "**Computer Organization and Embedded Systems**", McGraw Hill, 2011.
- Microcontrollers Lab. "*Tiva Launchpad tutorials and projects*", `https://microcontrollerslab.com/category/tiva-launchpad/`, 2024.

Table of Contents

Table of Contents

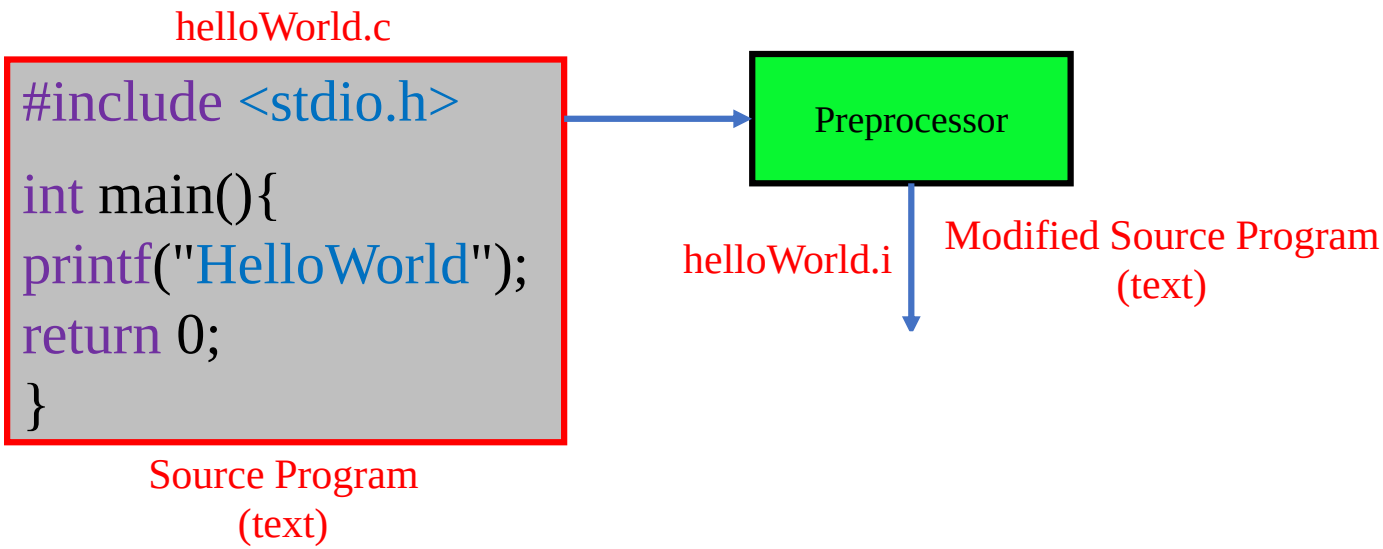
Compilation Process

helloWorld.c

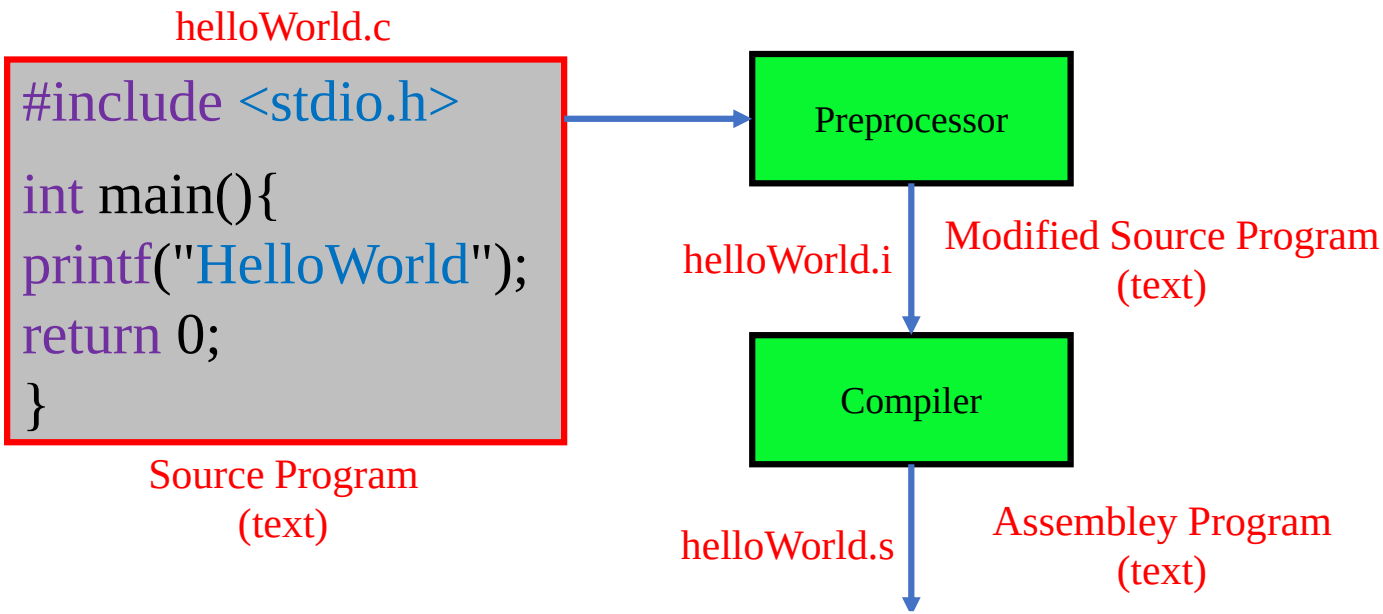
```
#include <stdio.h>

int main(){
printf("HelloWorld");
return 0;
}
```

Source Program
(text)

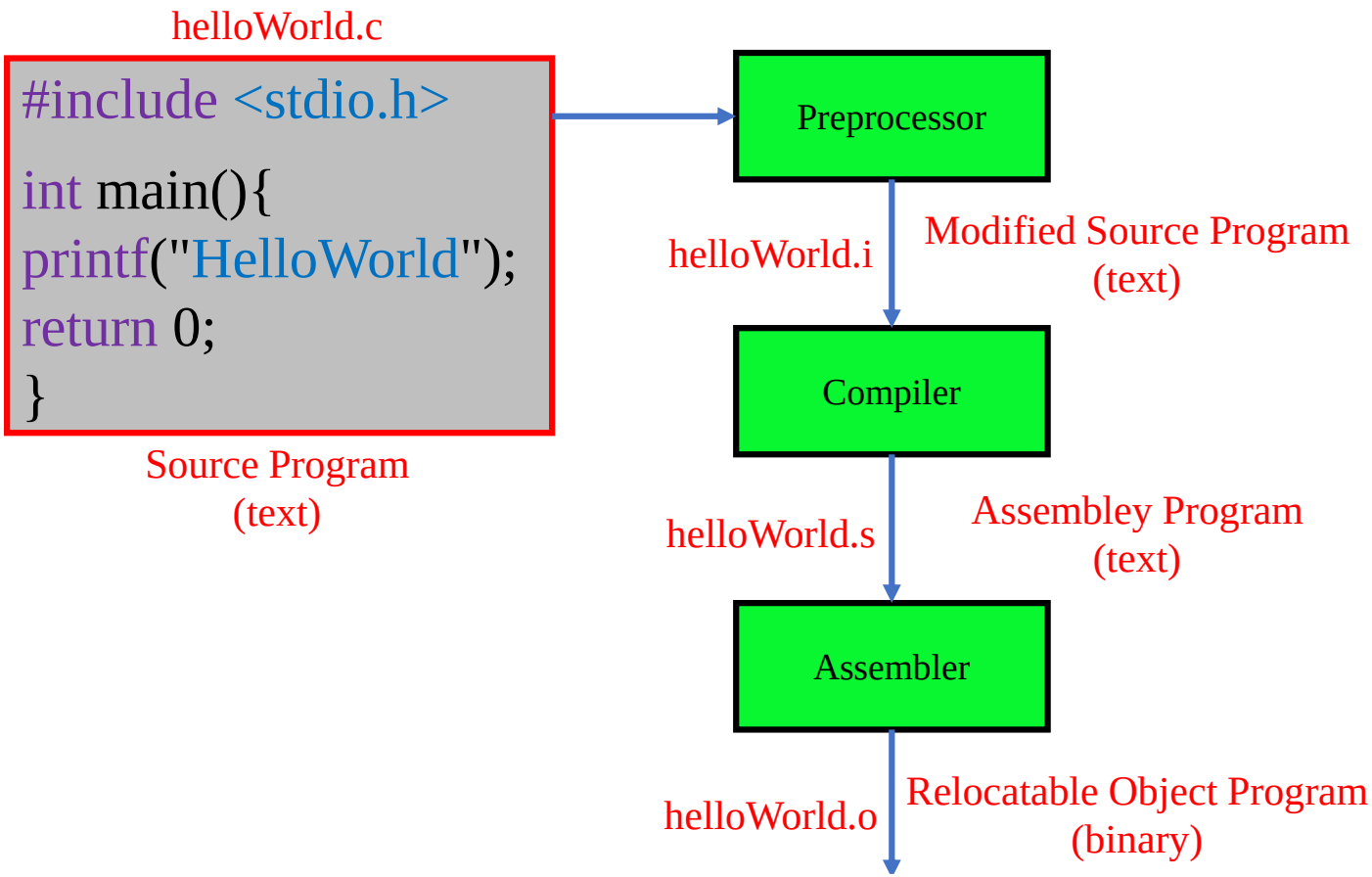
Compilation Process



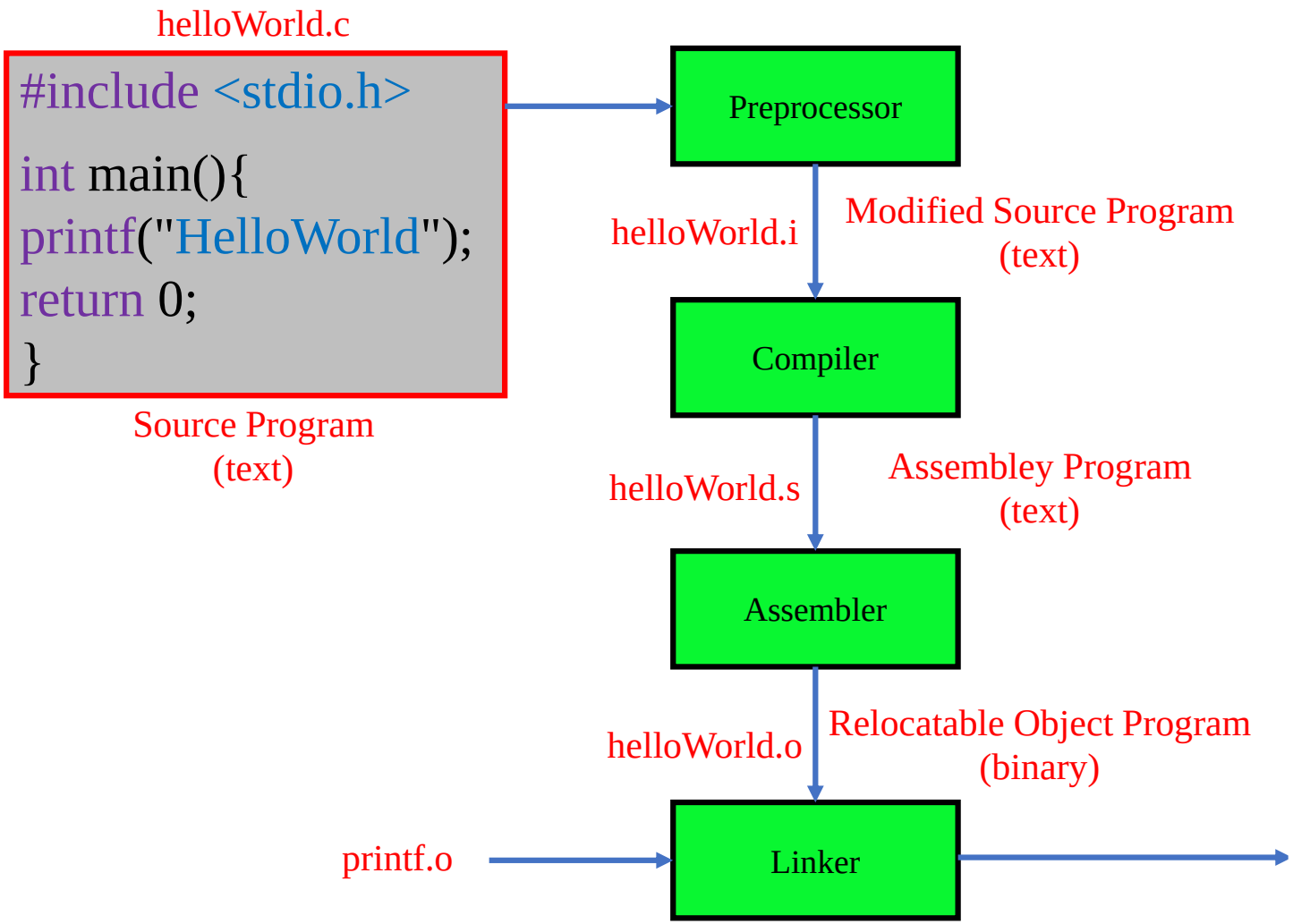
Compilation Process



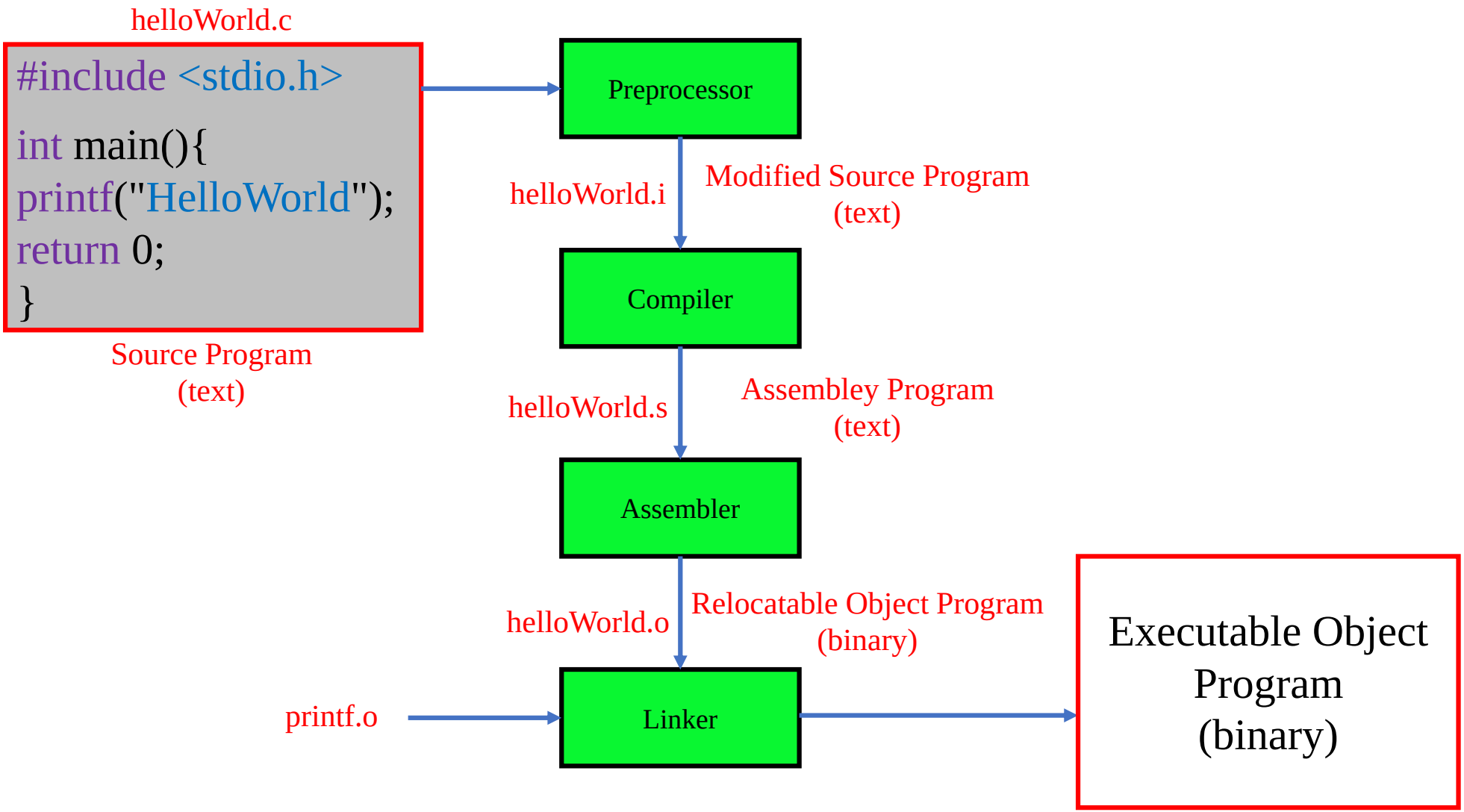
Compilation Process



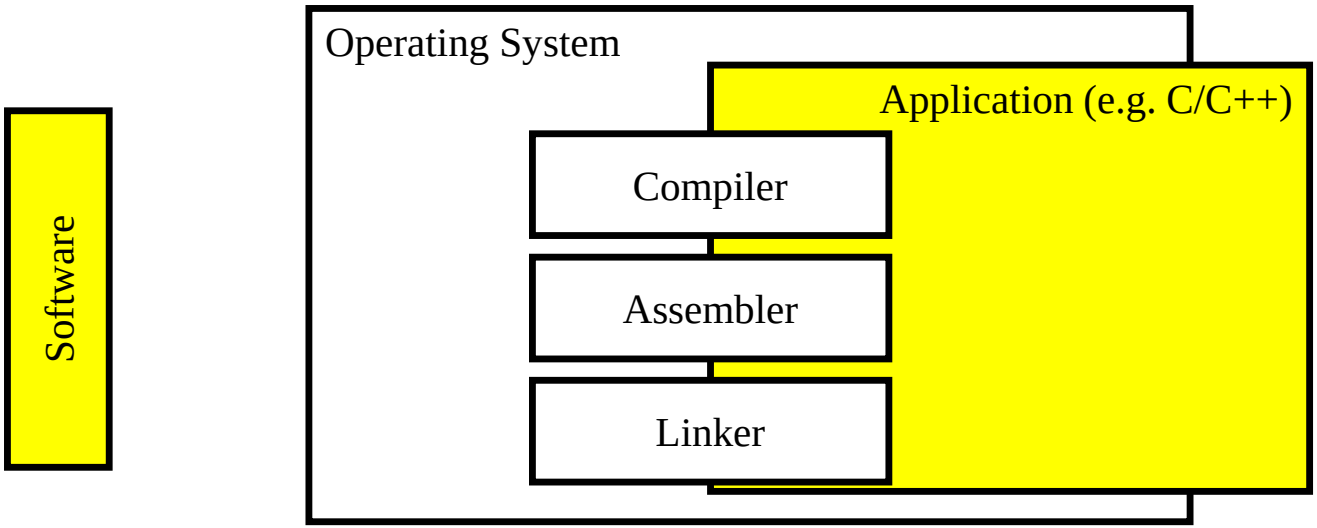
Compilation Process



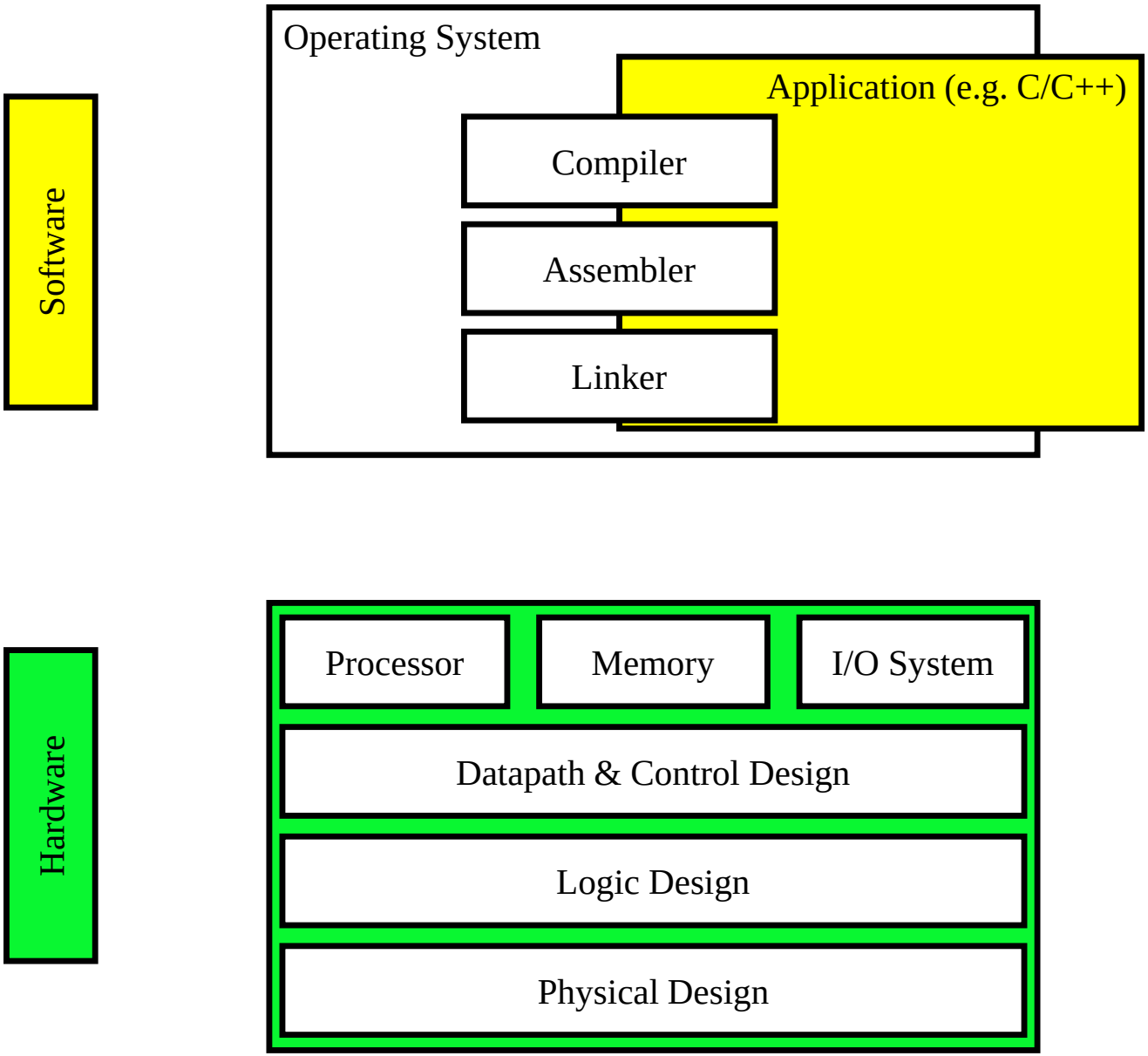
Compilation Process



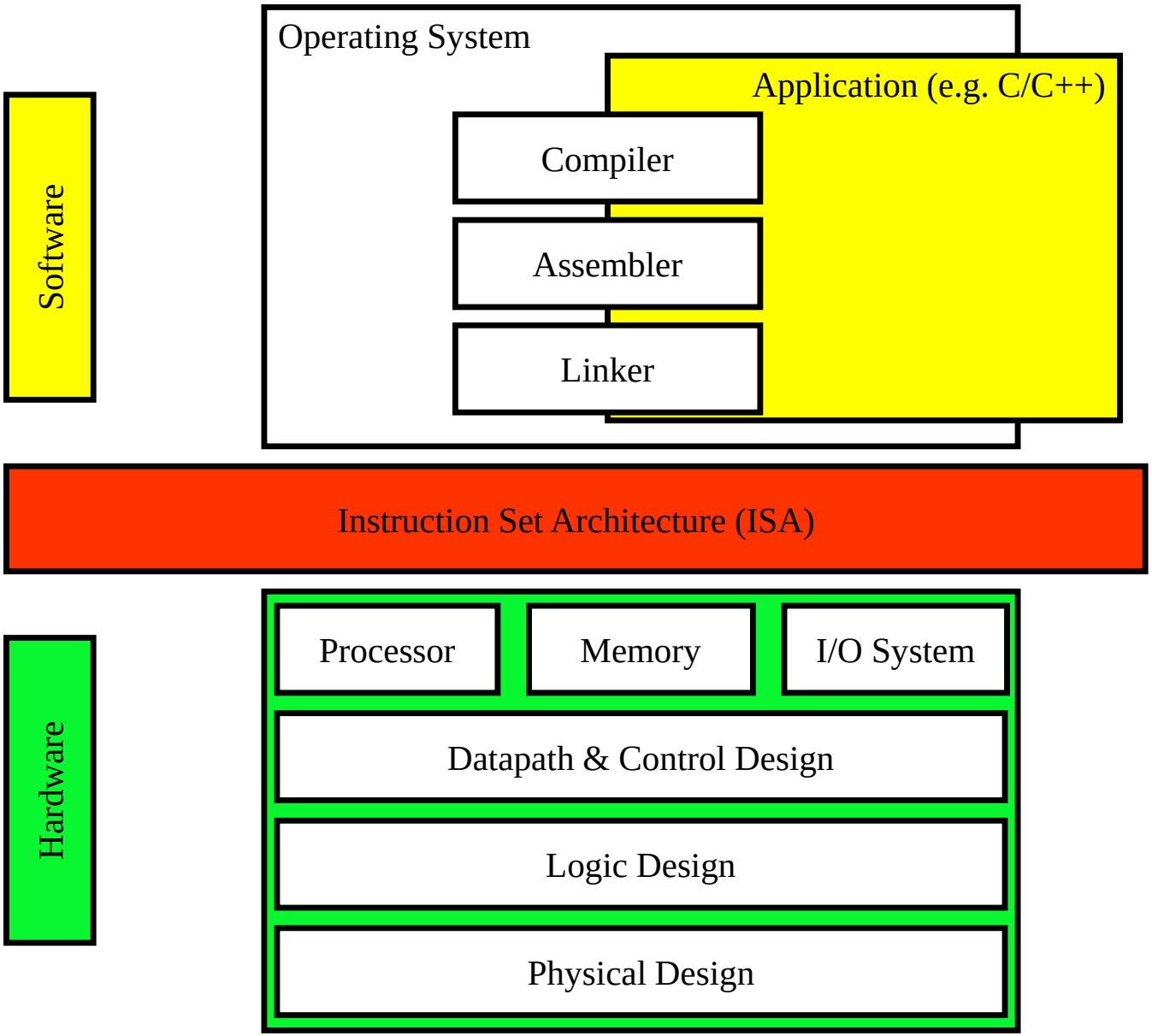
Instruction Set Architecture (ISA)



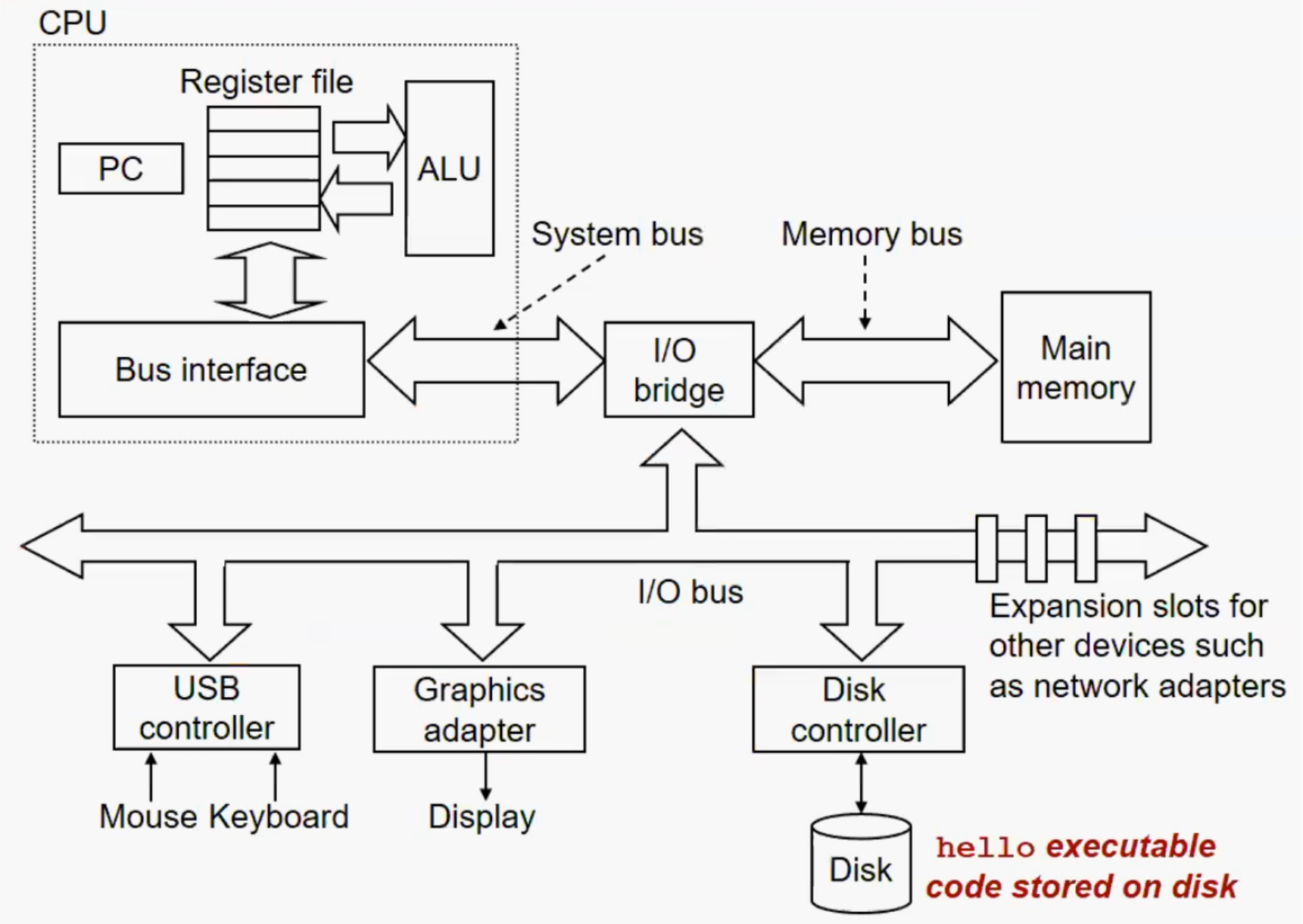
Instruction Set Architecture (ISA)



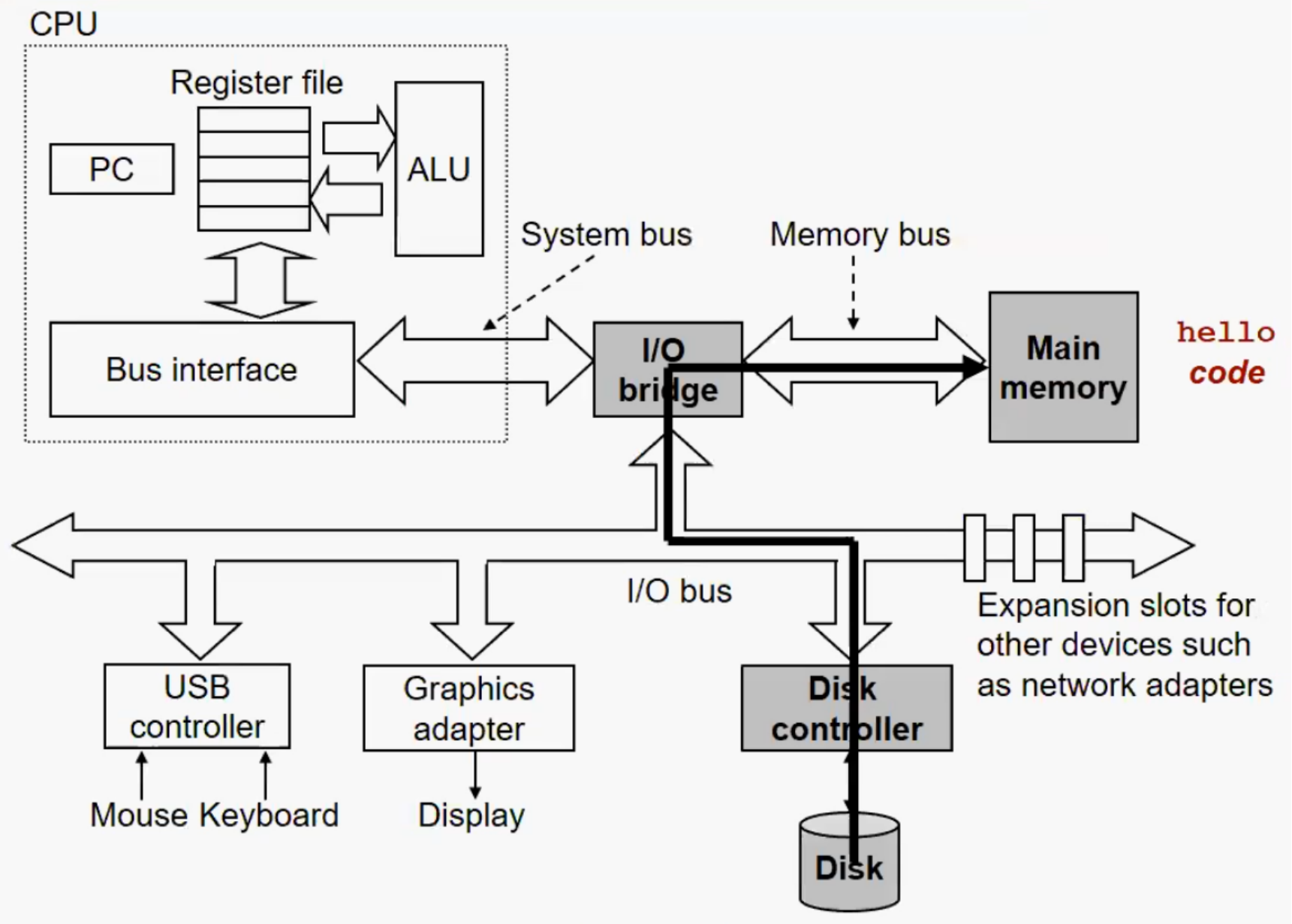
Instruction Set Architecture (ISA)



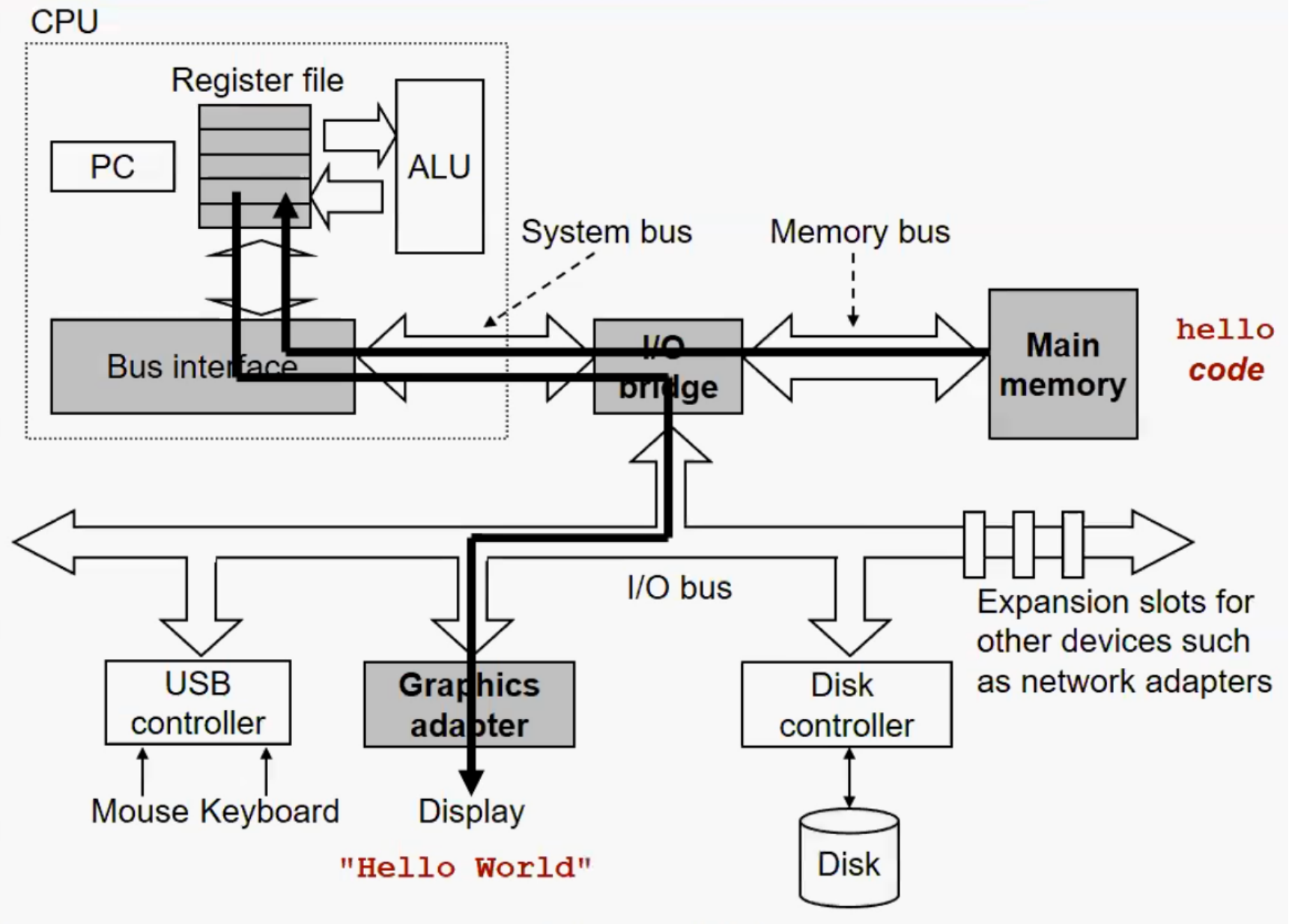
General-Purpose Computers



General-Purpose Computers (Loading Executable)



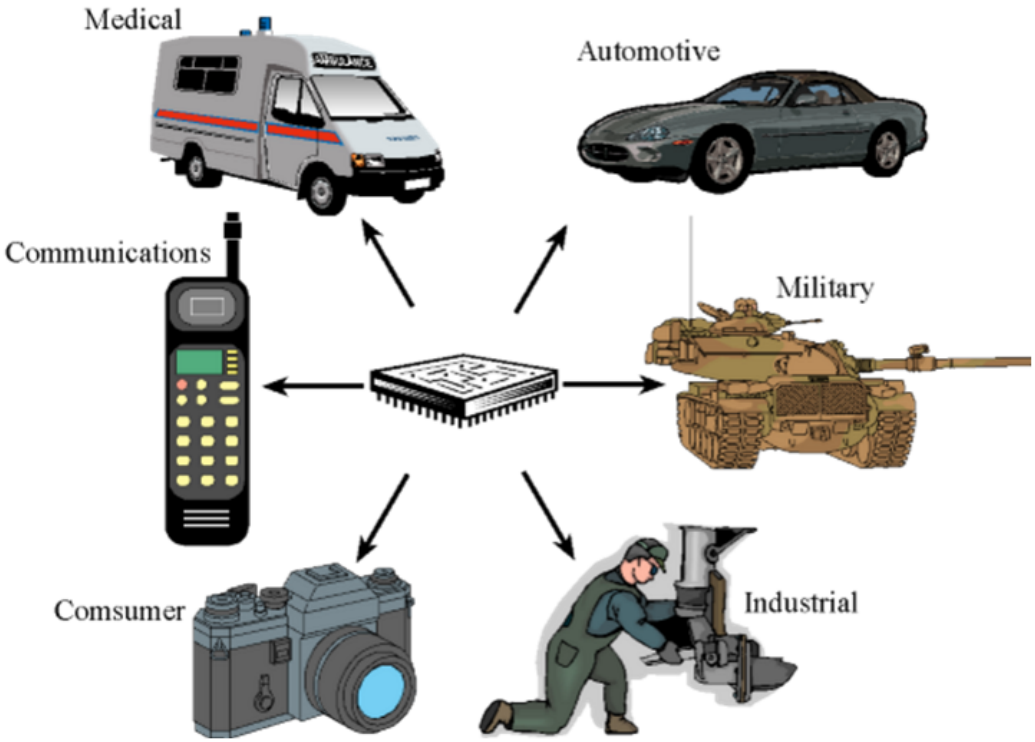
General-Purpose Computers (Execution, Writing O/P)



General-Purpose Computer Vs Embedded System

- Features:
  - Tightly constrained
  - Function-specific
  - Real-time
  - Low power consumption
  - Size

VS



- Markets
  - Consumer Electronics
  - Industrial automation
  - Medical Equipment
  - Communication
  - Military

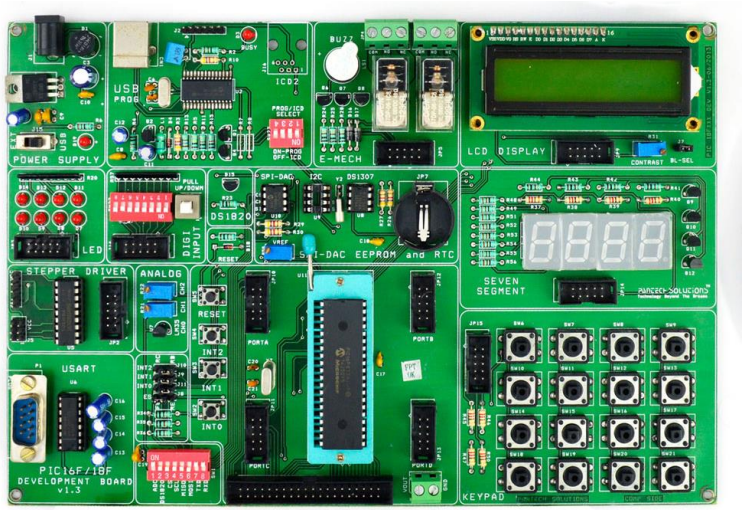
Embedded Systems Technologies

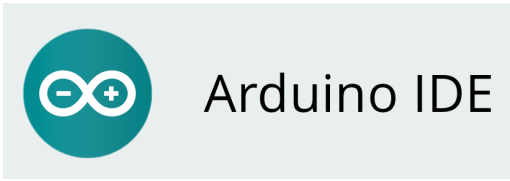
Three key technologies for embedded systems:

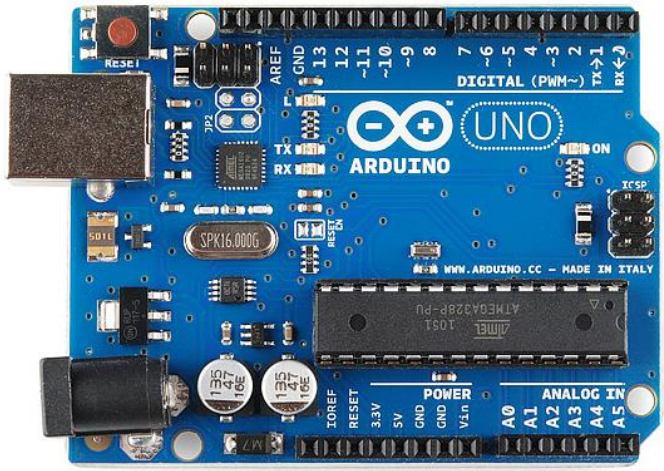
- **Processor technology**: The **architecture** of the computational engine used to implement a system's desired functionality
- **IC technology**: The manner in which an **implementation** is realized as an integrated circuit (IC)
- **Design technology**: The manner in which we perform a **mapping** of our desired system functionality to an implementation

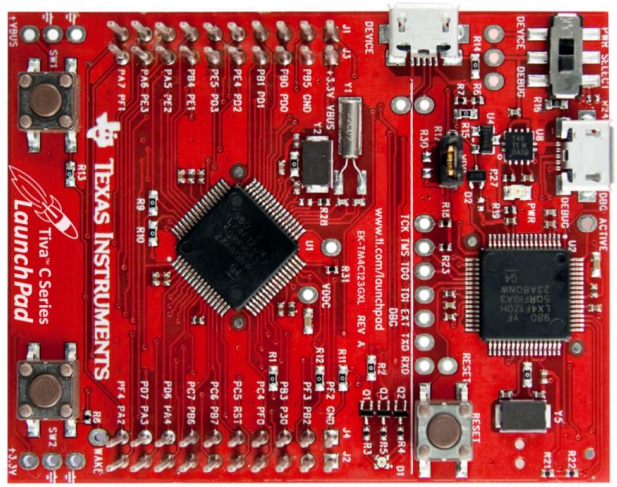
Embedded Systems Platforms

- Microcontrollers (MCUs)



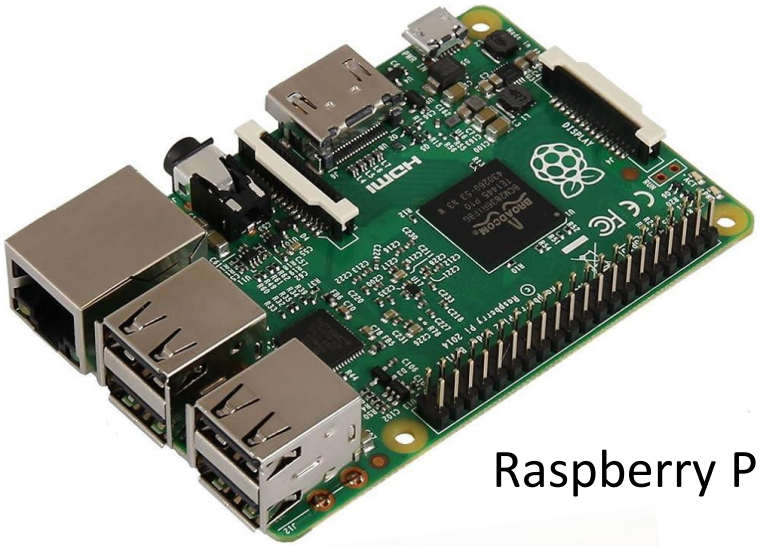




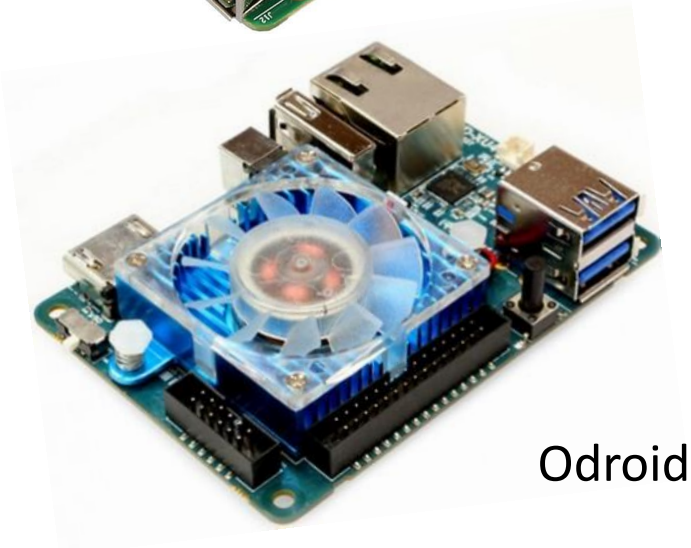


Embedded Systems Platforms

- Single-Board Computers (SBCs)



Raspberry Pi



Odroid

Embedded Systems Platforms

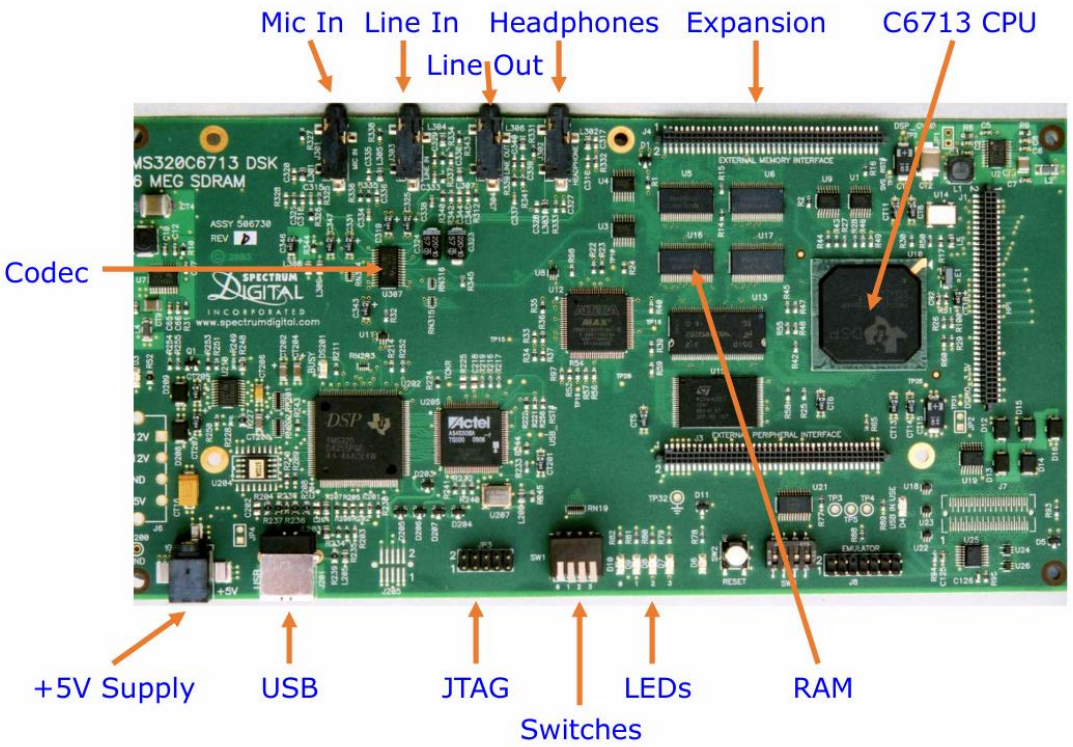
- Field-Programmable Gate Arrays (FPGA)

VHDL
VERILOG

MATLAB & SIMULINK
HDL Coder

Embedded Systems Platforms

- Digital Signal Processors (DSP)



TMS320C6713
VLIW (Very Long Instruction Word) Architecture

Embedded Systems Platforms

- Software-Defined Radio (SDR)

Ettus Research™
A National Instruments Brand

NATIONAL INSTRUMENTS™

GNU Radio

USRP

LabVIEW

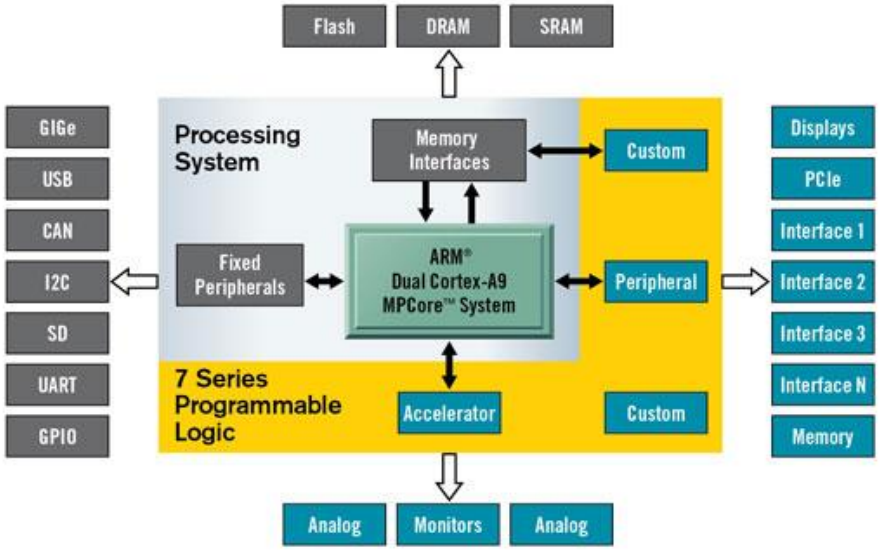
python™

MATLAB® & SIMULINK®

C/C++ PROGRAMMING

Embedded Systems Platforms

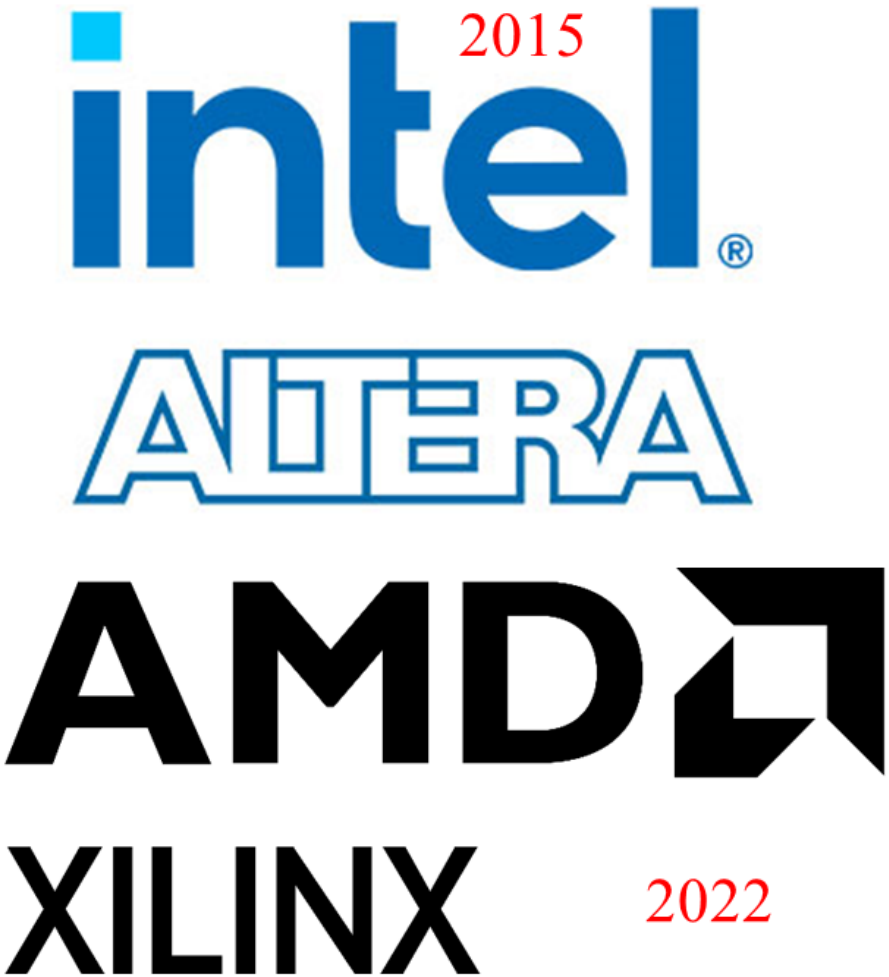
- System-on-Chip (SoC)



VHDL &
VERILOG

Embedded Systems Platforms

- Era of Adaptive and Heterogeneous Computing



Embedded Systems Platforms

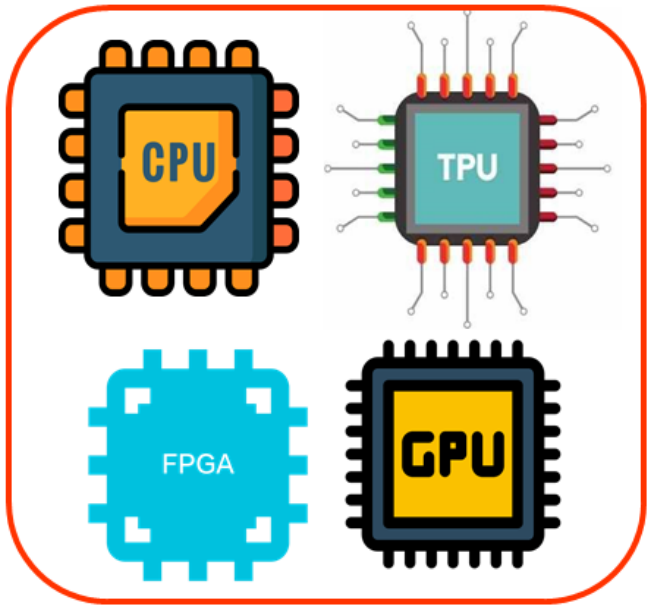
- Era of Adaptive and Heterogeneous Computing

2015 intel®

ALTERA

AMD

XILINX 2022



Embedded Systems Platforms

- Era of Adaptive and Heterogeneous Computing

2015

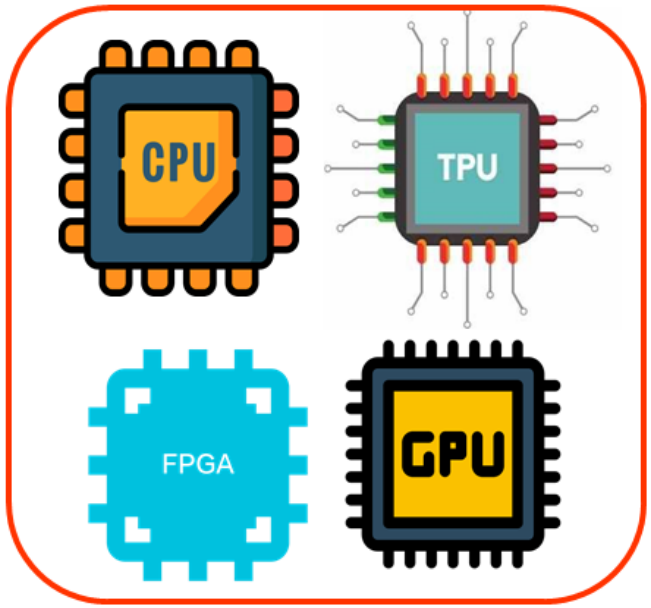
intel

ALTERA

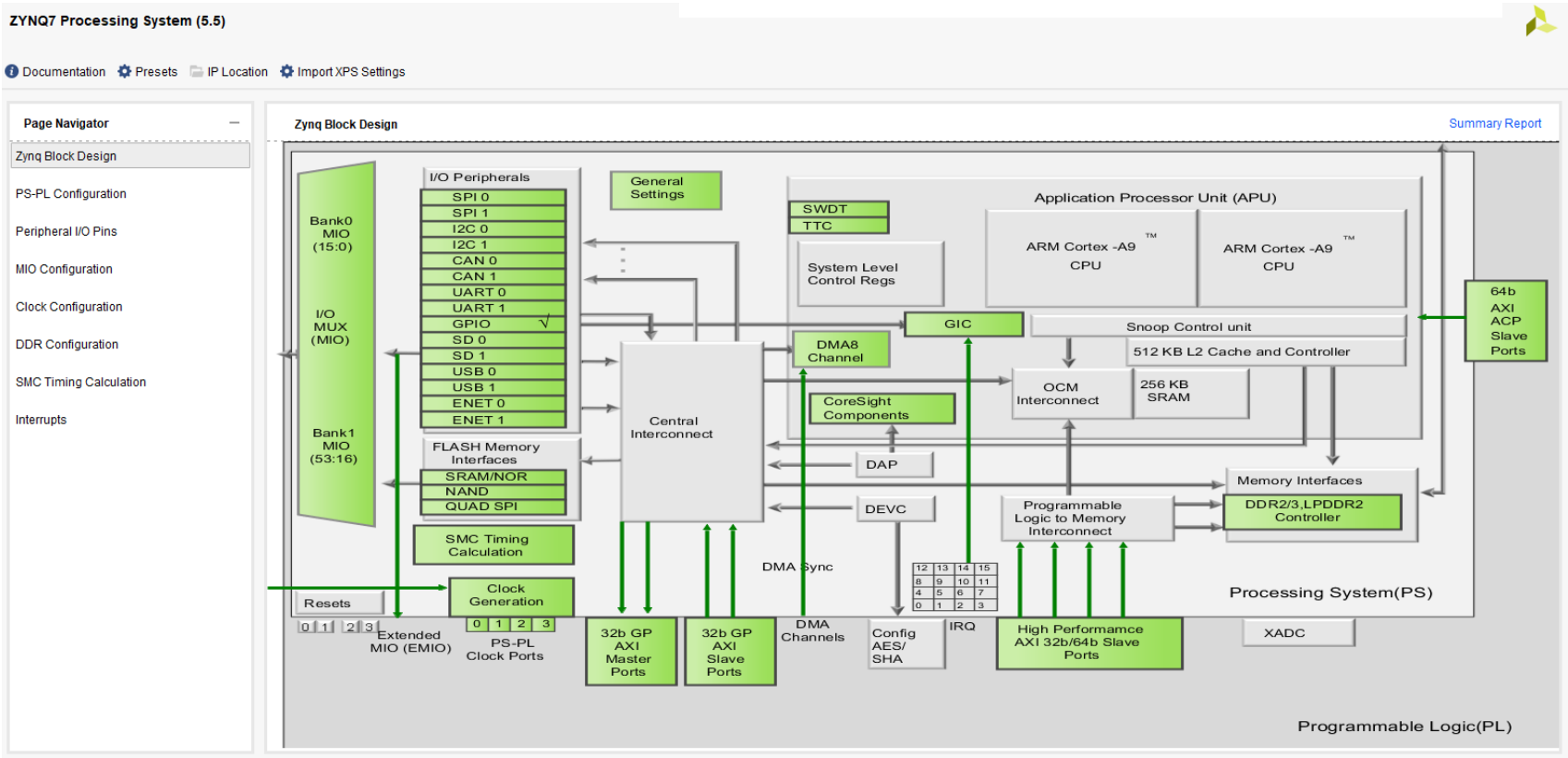
AMD

XILINX

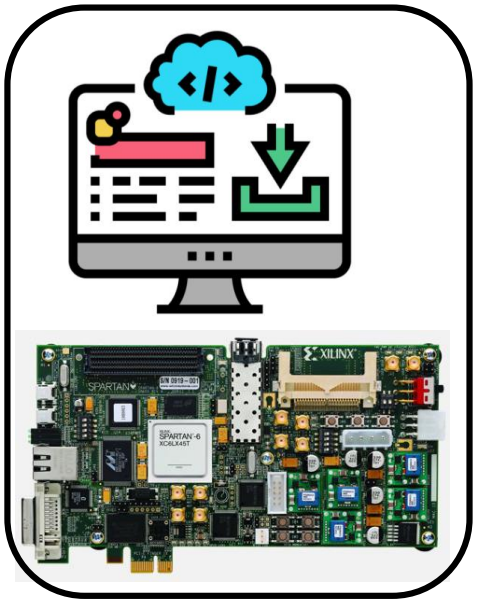
2022



MCU in Advanced Systems

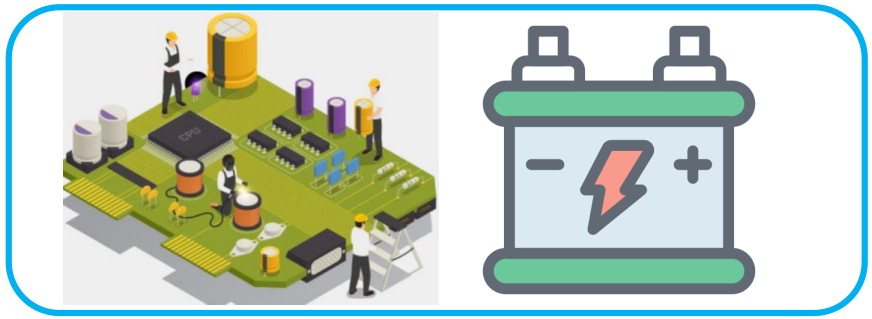


Transition from Evaluation Board to Final Product!

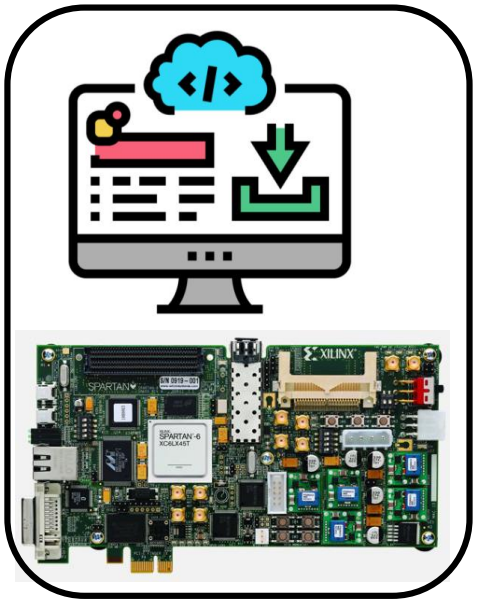


Embedded design

Transition from Evaluation Board to Final Product!

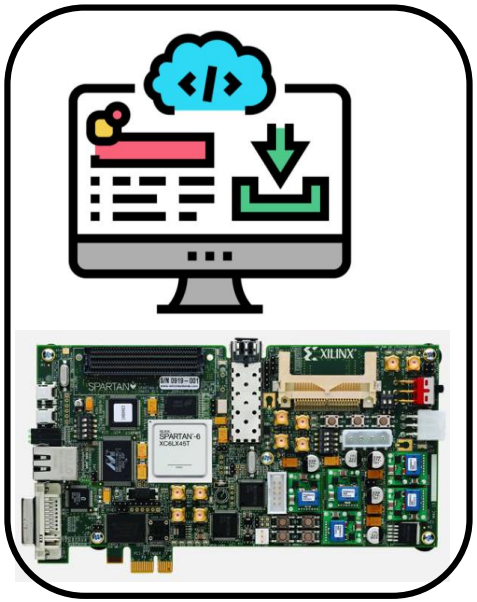


PCB design and PS

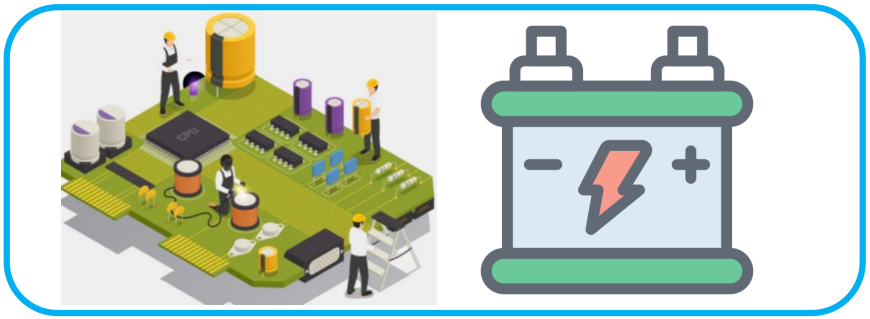


Embedded design

Transition from Evaluation Board to Final Product!



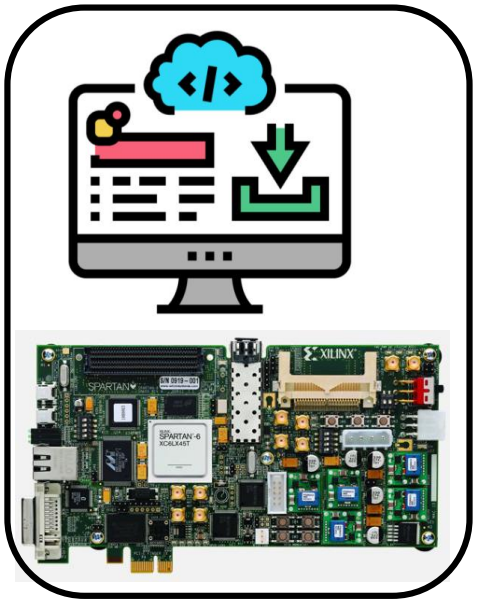
Embedded design



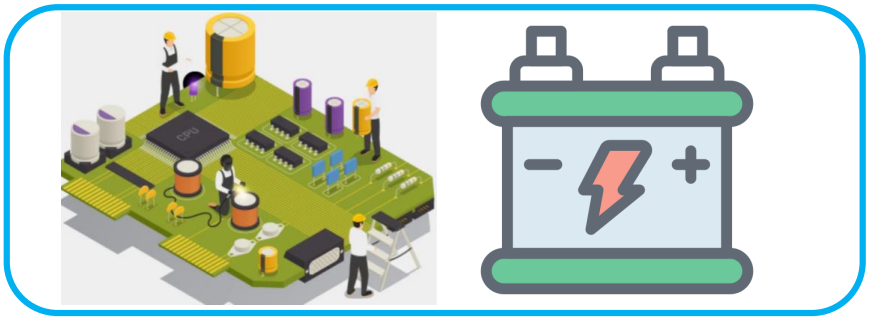
PCB design and PS

User interface

Transition from Evaluation Board to Final Product!

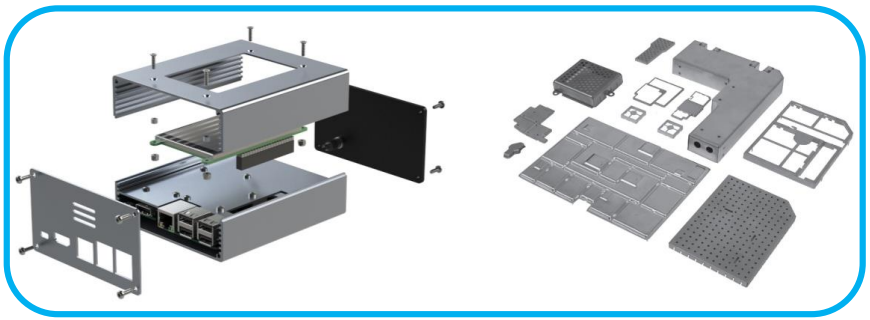


Embedded design



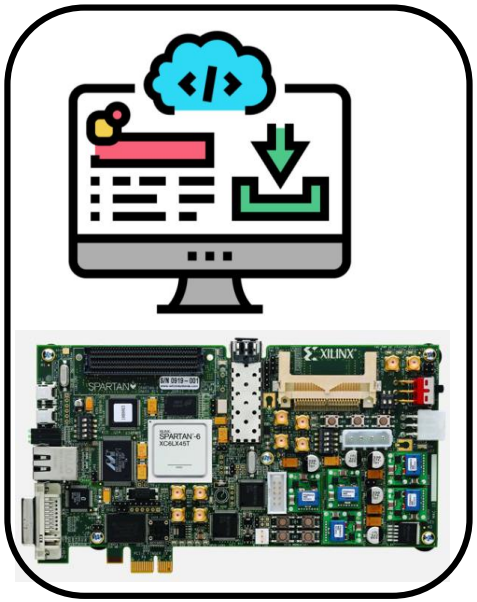
PCB design and PS

User interface

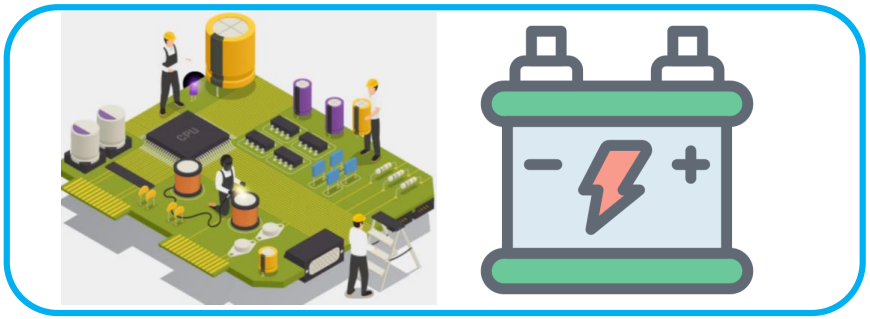


Enclosure design and shielding

Transition from Evaluation Board to Final Product!

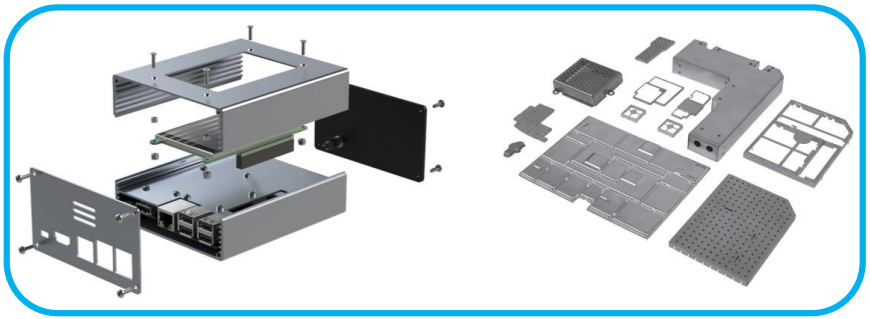


Embedded design



PCB design and PS

User interface



Enclosure design and shielding

Final Product

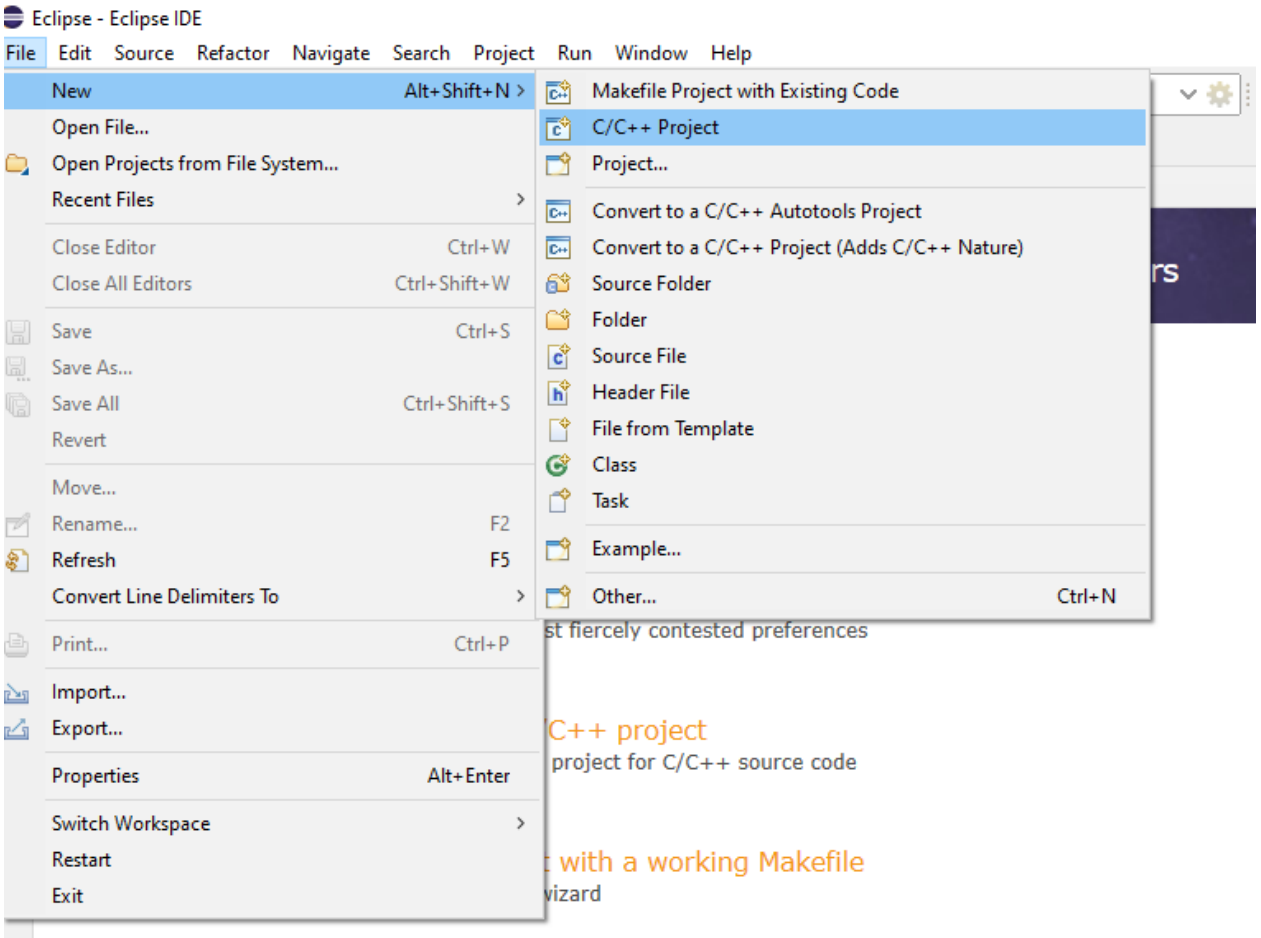
Table of Contents

Installing Eclipse

- Eclipse is an integrated development environment (IDE) that is widely used for programming in various languages, including C and C++.
- Download and install Eclipse IDE for C/C++ Developers from:
  `https://www.eclipse.org/downloads/packages/release/2024-06/r/eclipse-ide-cc-developers`
- Download Mingw64 (MinGW-W64 GCC-8.1.0/x86_64-posix-seh) from:
  `https://sourceforge.net/projects/mingw-w64/files/`
- Extract Mingw64 and copy to C:\
- Add Mingw64 to system path: C:\Mingw64 \bin

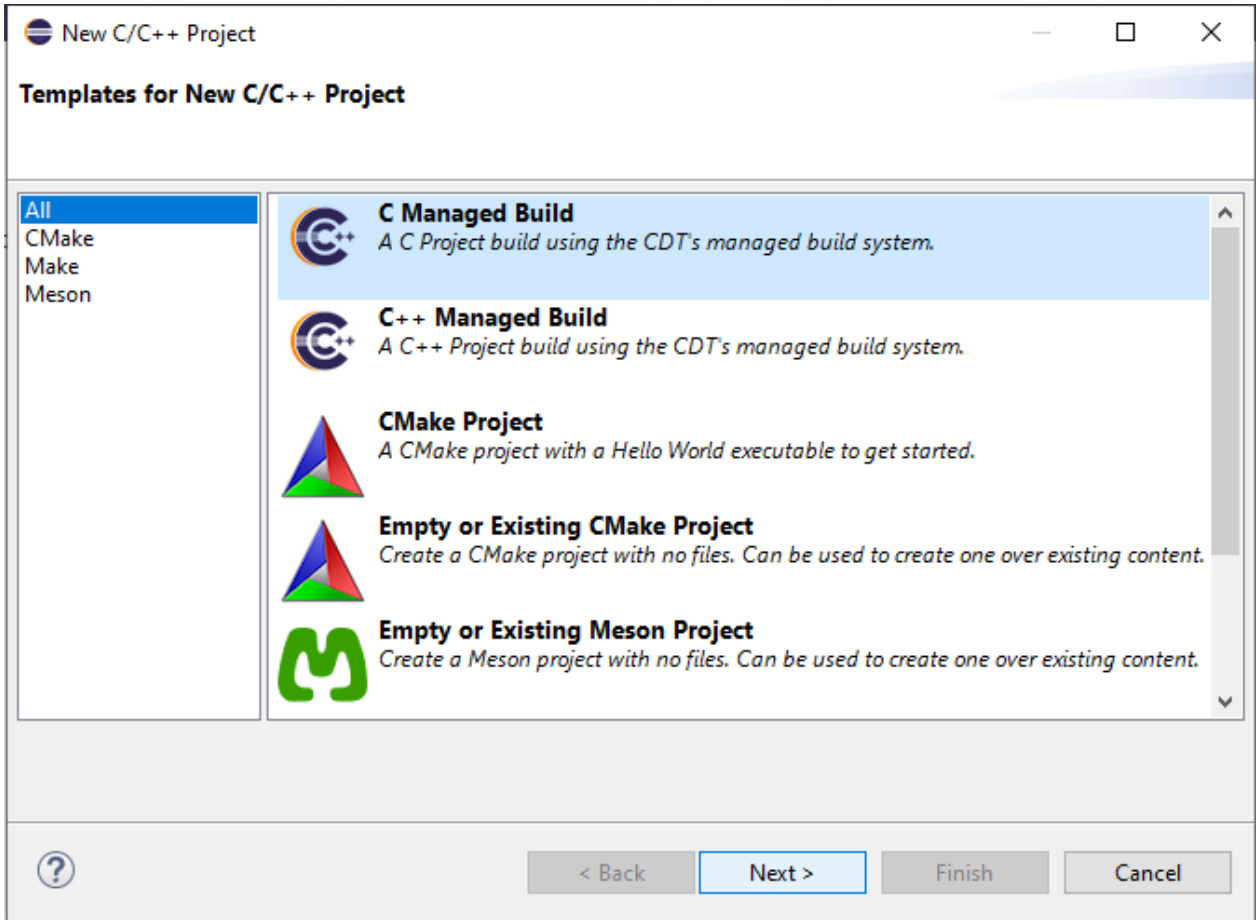
Create your first project in eclipse

- Create new C/C++ Project:



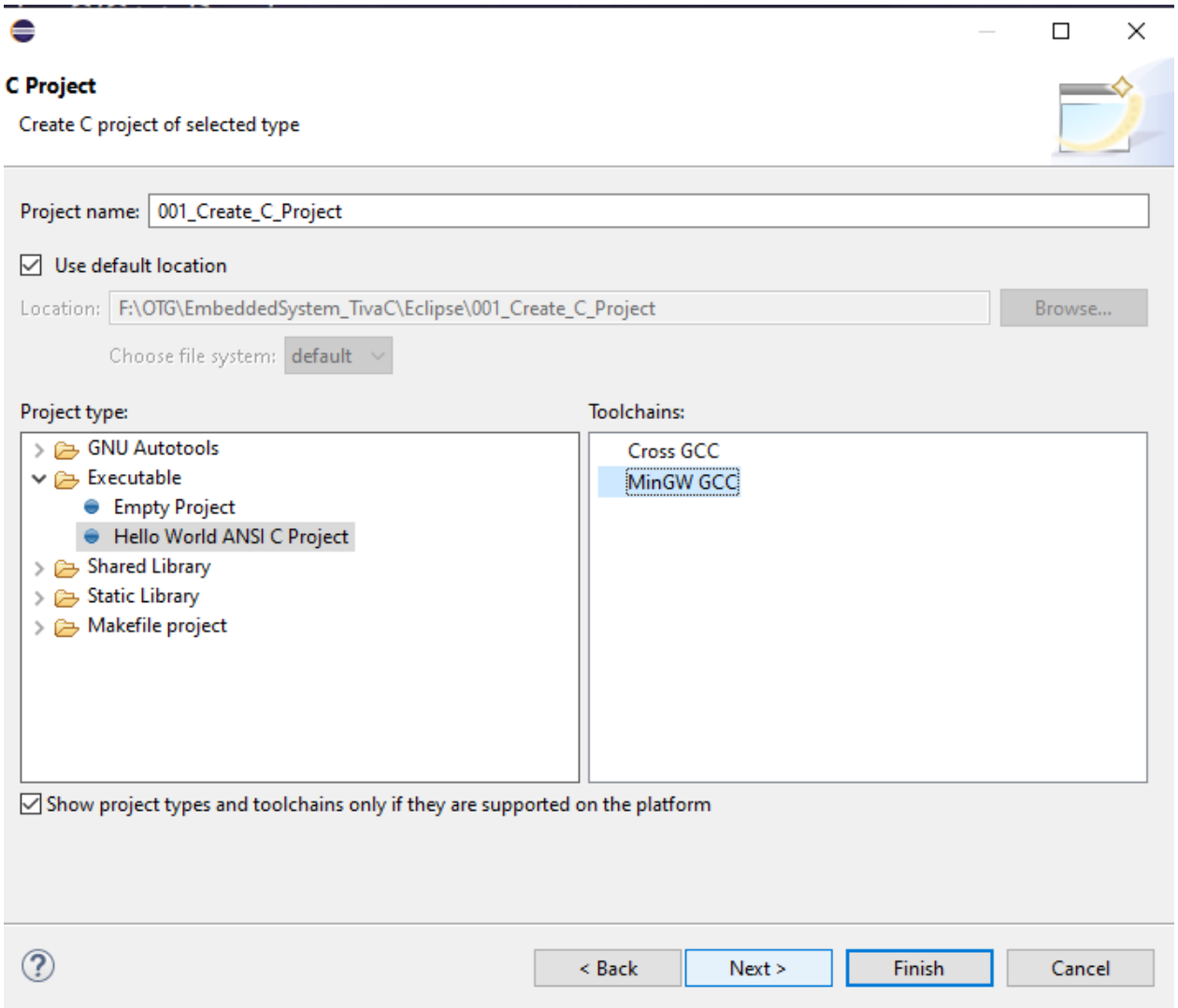
Create your first project in eclipse

- C managed build C/C++ Project:



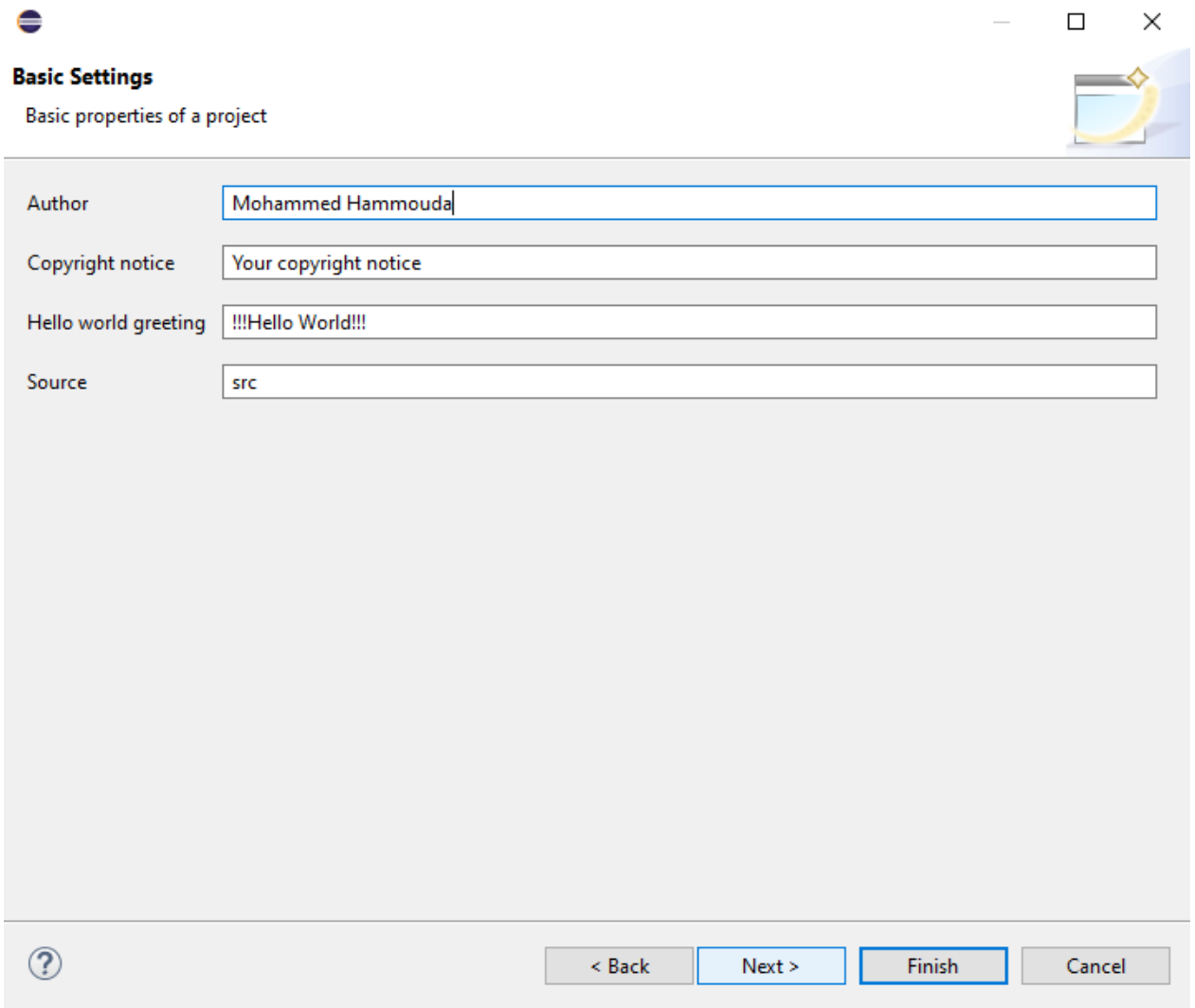
Create your first project in eclipse

- Project name, project type, and toolchains:



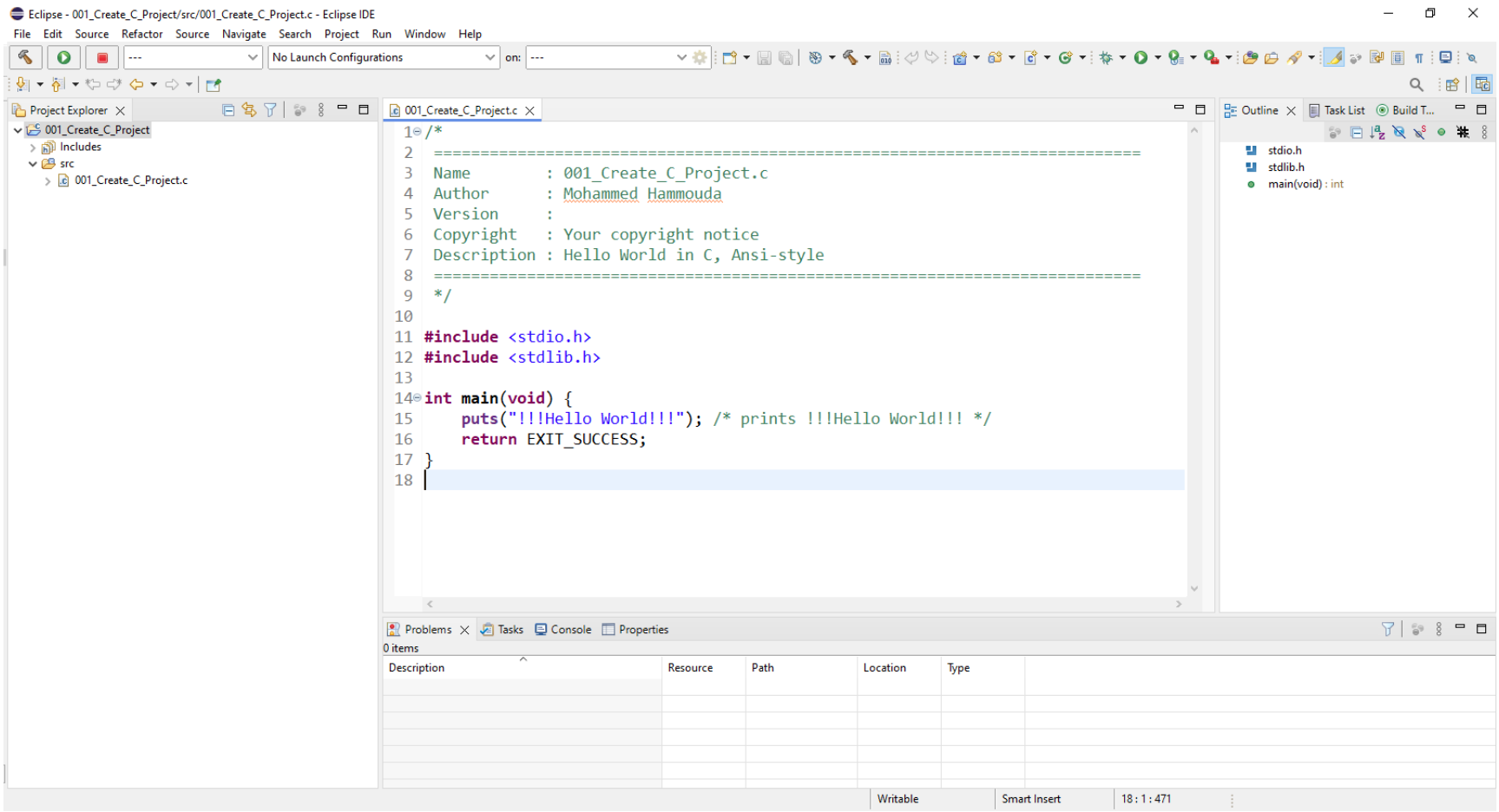
Create your first project in eclipse

- Basic setting:



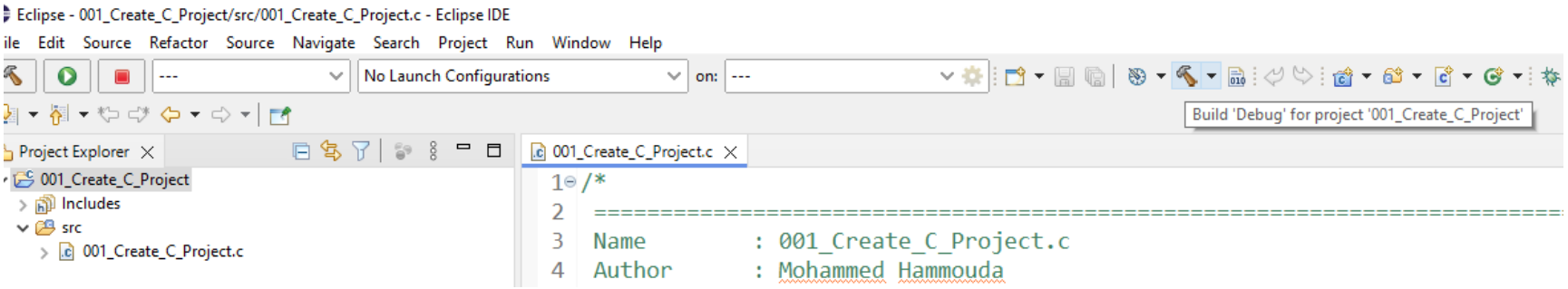
Create your first project in eclipse

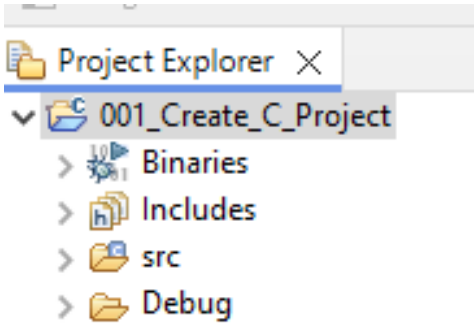
- Congratulation! Created C project:

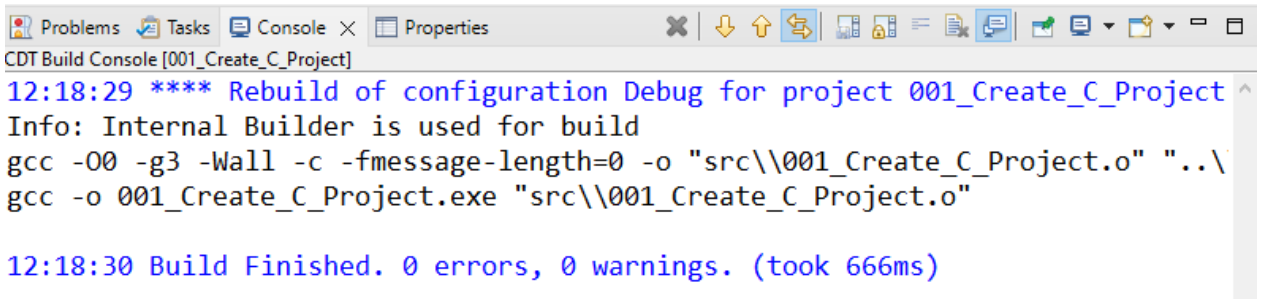


Create your first project in eclipse

- Build your project: Translates your source code into an executable.
- Check project explore and console window.

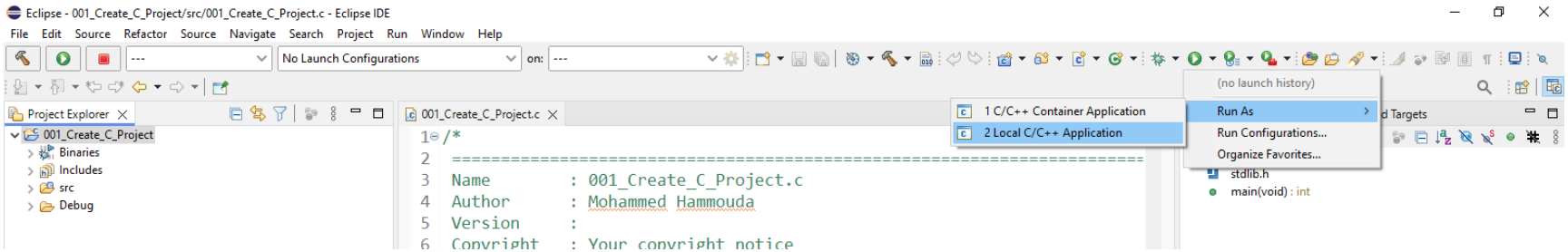


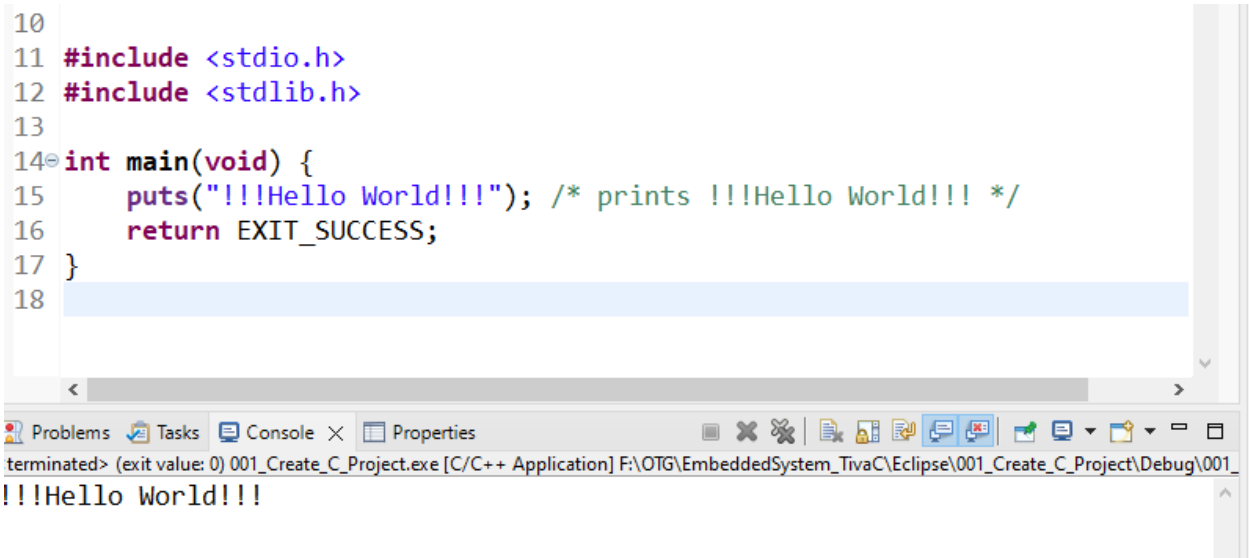




Create your first project in eclipse

- Run your project: Executes the compiled program.
- Check console window for executed output.





Preprocessor Commands in C Language

- Preprocessor: Prepare the code for the compiler using directive instruction.
  - Simplifies code management.
  - Makes code more readable and maintainable.
- Directives

```
#include
#define
#ifdef, #ifndef, #endif
#undef
#pragma
```

Preprocessor Commands in C Language

- #include directive: To include header files in the program.

```
#include <stdio.h>       \\System Header Files
#include "myheader.h"    \\User-Defined Header Files
```

- #define directive: To define constants and macros.

```
#define pi 3.14
#define square(x) ((x) * (x))
```

- #ifdef, #ifndef, #endif directive: Used to conditionally compile code.

```
   #define debug
#ifdef debug
 printf("Debugging mode is ON\n");
#endif
```

- #undef and #pragma directive: To undefine a previously defined macro and to issue special commands to the compiler.

```
#define pi 3.14159
#undef pi
#pragma once
```

Preprocessor Commands in C Language

- Code Example (main.c):

```c
#include <stdio.h>
#include "myheader.h" //First inclusion of header file
#include "myheader.h" //Second inclusion of same header file
#define debug
#ifdef debug
#define LOG(msg) printf("DEBUG: %s\n", msg)   \\macros
#else
#define LOG(msg)
#endif
int main() {
 LOG("Starting the program");
 myFunction(); // Calling the function from the header file
 printf("Hello, World!\n");
 LOG("Ending the program");
 return 0;
}
```

```c
// myheader.h
#pragma once
void myFunction() {
 printf("This function is from the header file.\n");
}
```

Data types

- Programs have to deal with data (memory $->$ **variables**).
- Data types specify the type of the variables stored in certain memory location, which specify the amount of memory a variable required.
- **Declaration** for variables before use. Check **Format specifier!** and **type casting!**

```c
#include <stdio.h>
#include <stdlib.h>
int main(void) {
 float discount = 0.5;
 int age = 998;
 char a = 'N';
 printf("discount: %f\n",discount);
 printf("y: %d\n",age);
 printf("a: %c\n",a);
 printf("float size: %lu bytes\n", (unsigned long)sizeof(discount));
 printf("int size: %lu bytes\n", (unsigned long)sizeof(age));
 printf("short size: %lu bytes\n", (unsigned long)sizeof(a));
 return 0;
}
```

Data types

- In C, the **typedef** keyword is used to create new type names for existing data types. This can make code more readable and easier to manage.
- The typedef keyword allows you to define a new type name for an existing type.

```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

typedef int Bool;
#define True 1
#define False 0

int main(void) {
   Bool aBooleanVariable;
   aBooleanVariable = True;
   printf("The value of aBooleanVariable is %d\n",aBooleanVariable);
   return 0;
 }
```

Pointer

- A pointer is a variable that stores the memory address of another variable. Allows to manage memory directly.
- The (&) operator is used to get the address of the variable size. This address is then assigned to the pointer variable pointerInt. Now, pointerInt points to the memory location where size is stored.
- The (*) operator, known as the dereference operator, is used to access the value at the address stored in pointer variable.

```c
#include <stdio.h>

int main() {
// Declaration of a variable and a pointer variable
//===============================================
//'size' is a regular integer variable
int size = 15;
//'pointerInt' is a pointer variable holding the address of an integer
int *pointerInt;

//Assign the address of the variable 'size' to the pointer 'pointerInt'
pointerInt = &size;
```

Pointer

```
// Output the value of 'size' and its memory address
 printf("Value of size: %d\n", size);   //Output: 15
 printf("Address of size: %p\n", &size);//Output: Address of 'size'

// Output the value stored in 'pointerInt' and the value it points to
 printf("Value of pointerInt (address of size): %p\n", pointerInt);
 // Output: Same as &size
 printf("Value pointed to by pointerInt: %d\n", *pointerInt);
 // Output: 15

// Modify the value of 'size' through the pointer
 *pointerInt = 25;
 printf("New value of size after modification through pointer: %d\n",
     size);  // Output: 25

 return 0;

}
```

Statements and Expressions

- Expression: Contain variables, constants, and operators.

```
25 + 12
X^2
L * W
```

- Statements: Execution for an action.

```
Your Age = TodayDate - BirthdayDate;
if(age>18)  Drive = TRUE;
```

- Code Example:

```
#include <stdio.h>
int main(void) {
 int Year_Days = 12 * 30;
 double L = 15.87;
 double W = 12.3;
 double A = L * W;
 printf("Year Days: %d \n",Year_Days);
 printf("Area: %f",A);
 return 0;
}
```

Arithmetic Operator

- C provides several basic arithmetic operators for performing mathematical operations.

```c
#include <stdio.h>
int main() {int a = 15, b = 4, add_assign = 10, sub_assign = 100,
    mul_assign = 5,div_assign = 30,mod_assign = 49;
 //Basic Operators
 int sum = a + b;                  //addition
 int difference = a - b;           //subtraction
 int product = a * b;              //multiplication
 int quotient = a / b;             //divition
 int remainder = a % b;            //Modulus
 //Unary Operators
 int increment =a++;               //increment by 1
 int decrement =a--;               //decrement by 1
 //Compound Assignment Operators: +=, -=, *=, /=, %=
 add_assign += 10;                 //add and assign
 sub_assign -= 50;     //subtract and assign
 mul_assign *=7;                   //multiply and assign
 div_assign /=6;                   //divide and assign
 mod_assign *=8;                   //mod and assign
 printf("Result of a + b = %d\n", quotient);
 return 0;}
```

Relational (Logical) Operator

- Logical operators are used to compare two values or expressions.
- The result is always Boolean value: 1 (true) if the comparison is true or 0(false) if it is not.
- Logical operators are essential in decision-making processes, such as in if statements, loops, and conditional expressions.

```
#include <stdio.h>
int main(void) { int x = 5;  int y = 7;
 printf("(x == y) is %d\n",(x==y)); //== is-equal?
 printf("(x != y) is %d\n",(x!=y)); //!= is-not-equal?
 printf("(x  > y) is %d\n",(x>y));  //>  bigger than?
 printf("(x  < y) is %d\n",(x<y));  //<  smaller than?
 printf("(x >= y) is %d\n",(x>=y)); //>= bigger-or-equal than?
 printf("(x <= y) is %d\n",(x<=y)); //<= smaller-or-equal than?
 int max = (x > y) ? x : y; //Assigns the larger of x and y to max
 printf("The maximum value is %d\n", max);
 x = 1;   y = 0;
 printf("(x && y) is %d\n",(x && y));   //&& logical and
 printf("(x || y) is %d\n",(x || y));   //|| logical or
 printf("   !x    is %d\n",!x);         //!  not
 return 0; }
```

Bit-level operators

- Used to perform operations on the individual bits of integer values.
- Useful for tasks like setting, clearing, or toggling specific bits.

```
#include <stdio.h>
int main(void) {

 int a = 12; // Binary: 00001100
 int b = 7;  // Binary: 00000111
 int c = 5;  // Binary: 00000101
 int d = 20; // Binary: 00010100
 int result_and = a & b; // Binary: 00000100 (Decimal: 4)
 int result_or  = a | b; // Binary: 00001111 (Decimal: 15)
 int result_xor = a ^ b; // Binary: 00001011 (Decimal: 11)
    int result_not = ~a;    // Binary: 11110011 (Decimal: -13 for a 8-
        bit signed integer)
 int result_shift_left  = c << 2; // Binary: 00010100 (Decimal: 20)
 int result_shift_right = d >> 2; // Binary: 00000101 (Decimal: 5)
 printf("Result %d\n",result_xor);
 printf("Result %u\n",result_not);// format specifier %d %u
 return 0;
}
```

Making Code Modular and usage of External Files

- Breaking Up Code Into Blocks (functions).
  - Built-in functions
  - Programmer-defined functions
- Enable the use of functions defined in external files
  - Functions are defined in .h file
  - Functions are implemented in .c file

```
// main.c
#include <stdio.h>
#include "specialTasks.h"
//Function prototypes
void task_1(void);
void task_2(void);
void task_3(void);
int main(void) {
 printf("%s\n","Hello, World!");  //built-in function in C
 task_1();
 task_2();
 specialTask_1();
 task_3();
 specialTask_3();
 return 0;   }
```

Making Code Modular and usage of External Files

```
void task_1(void){
 specialTask_1();
}
void task_2(void){
 specialTask_2();
}
void task_3(void){
 printf("%s\n","task_3 was called");
}
```

```
// specialTasks.h
#ifndef SPECIALTASKS_H_
#define SPECIALTASKS_H_
void specialTask_1(void);
void specialTask_2(void);
void specialTask_3(void);
#endif
```

```
// specialTasks.c
#include <stdio.h>
void specialTask_1(void){
printf("%s\n","specialTask_1 was called");
}
void specialTask_2(void){
printf("%s\n","specialTask_2 was called");
}
void specialTask_3(void){
 printf("%s\n","specialTask_3 was called");
}
```

Function Passing Arguments and Returning Value

- Functions can be used in three different ways depending on how parameters are passed and how the function returns values.
  - Function Without Passing Arguments and Without Returning a Value
  - Function With Passing Arguments But Without Returning a Value
  - Function With Passing Arguments and Returning a Value

```c
#include <stdio.h>

//Function prototypes
void function1();
void function2(int a, int b);
int  function3(int a, int b);

int main() {
 function1(); //Call function without pass any arguments
 function2(5,6); // Pass arguments to the function
 int result = function3(6, 7);//Pass argument & store return value
 printf("The product is: %d, printed inside main \n", result);

 return 0;
}
```

Function Passing Arguments and Returning Value

```c
//Function Definition
void function1() {
 printf("Hello, World!\n");
}
void function2(int a, int b) {
 int sum = a + b;
 printf("The sum is: %d, printed inside function2 \n", sum);
}
int function3(int a, int b) {
 return a * b;
}
```

Local Vs Global variable

- Global and local variables differ in their scope and how they are accessed and modified within a program.

```c
#include <stdio.h>

int globalVar = 10; // Global variable
void printGlobalVar() {
 printf("Global variable inside function: %d\n", globalVar);
}
void modifyGlobalVar() {
 //int globalVar = 20; // Local variable with the same name
 globalVar = 20;   // Update Global variable
 printf("Local variable inside function: %d\n", globalVar);
}

int main() {
 printf("Global variable in main: %d\n", globalVar);
 printGlobalVar(); // Prints the global variable
 modifyGlobalVar(); //Prints local or updated global variable
 printf("Global variable after function call: %d\n", globalVar);
 return 0;
}
```

Conditional Statements (decision-making)

- Decision-making is performed using conditional statements such as if, if-else, if- else if - else, and switch. These statements allow you to execute different blocks of code based on certain conditions.

```c
#include <stdio.h>
int main() {
 int num_1 = 10;
 int num_2=-2;
 int num_3 = 0;

 // if statment
 if (num_1 > 0) {
  printf("The number is positive.\n");
 }

 // if-else statment
 if (num_2 > 0) {
  printf("The number is positive.\n");
 } else {
  printf("The number is not positive.\n");
 }
```

Conditional Statements (decision-making)

```
    // if-else if-else statment
    if (num_3 > 0) {
     printf("The number is positive.\n");
    } else if (num_3 < 0) {
     printf("The number is negative.\n");
    } else {
     printf("The number is zero.\n");
    }
    return 0;
}
```

Loop (Executing Statements Repeatedly)

- Loops allow you to execute a block of code repeatedly based on certain conditions. There are three primary types of loops: for, while, and do-while.

```c
#include <stdio.h>
int main() {
 //for loop
 for (int i = 0; i < 5; i++) {
  printf("Iteration %d\n", i);
  if (i == 3) {
   break; // Exit the loop when i equals 3
  }
 }
 //while loop
 int j = 0;
 while (j < 5) {
  printf("Iteration %d\n", j);
  if (j == 3) {
   break; // Exit the loop when i equals 3
  }
  j++;
 }
```

Loop (Executing Statements Repeatedly)

```
    // do-while loop
    int k = 0;
    do {
     printf("Iteration %d\n", k);
     if (k == 3) {
      break; // Exit the loop when i equals 3
     }
     k++;
    } while (k < 5);
    return 0;
}
```

Manual Debugging

- Do everything from scratch.

```
#include <stdio.h>

//Constants
#define DEBUG_ON  1
#define DEBUG_OFF 0
#define DEBUG_ALERT "----->"

// Globals
int debugFlag = DEBUG_ON;

void Function1(int,int);
void Function2(float,float);

int main(void) {
 Function1(3,5);
 Function2(2.0,3.0);

 return 0;
}
```

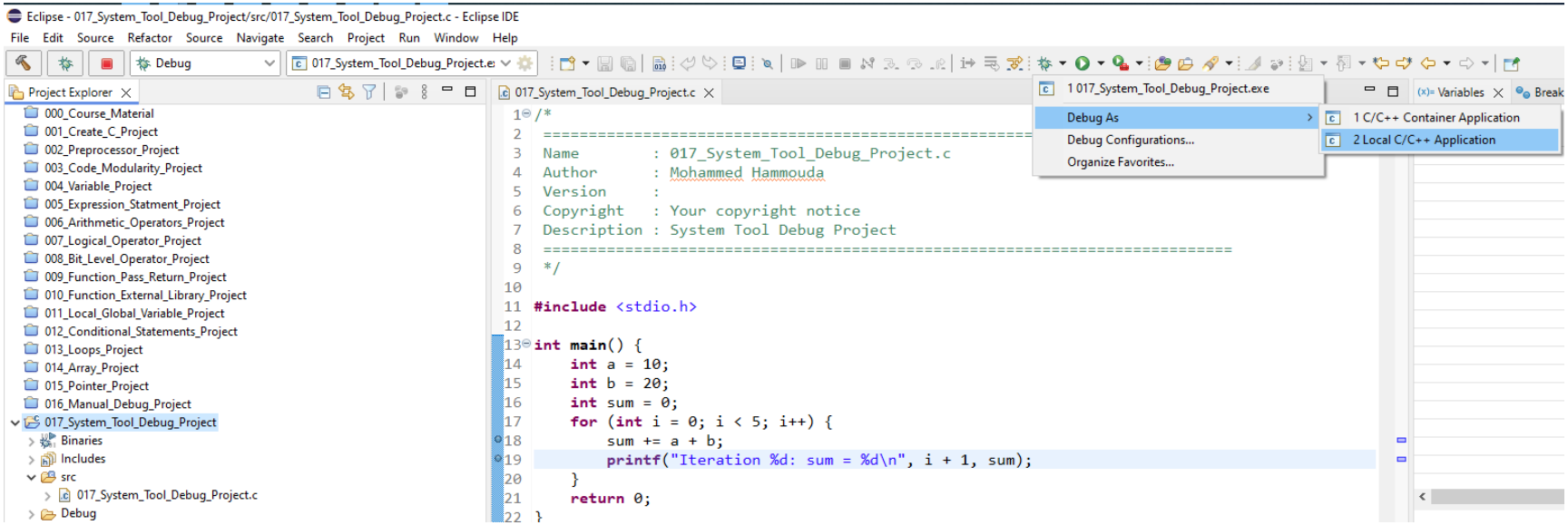
Manual Debugging

```
void Function1(int a, int b){
 if(debugFlag)
 printf ("%s In Function1, received a=%d, b=%d\n",DEBUG_ALERT, a,b);
 printf("    Function1 is executing...\n");
 printf("    Function1 is still doing something...\n");
 printf("    Function1 keeps going...\n");
 //Rest of function code
}

void Function2(float a, float b){
 if(debugFlag)
 printf ("%s In Function2, received a=%f, b=%f\n",DEBUG_ALERT,a,b);

 printf("    Function2 is executing...\n");
 int k = 4; int j = 8; int z = -4;
 double m = (double)k * (double)k /((double) j + (double) z);
 if(debugFlag)
 printf ("%s In Function2, m=%lf at line %d\n",DEBUG_ALERT,m,
     __LINE__);
 //Rest of function code
}
```

System Tool Debugging

```c
#include <stdio.h>

int main() {
 int a = 10;
 int b = 20;
 int sum = 0;

 for (int i = 0; i < 5; i++) {
  sum += a + b;
  printf("Iteration %d: sum = %d\n", i + 1, sum);
 }

 return 0;
}
```

System Tool Debugging



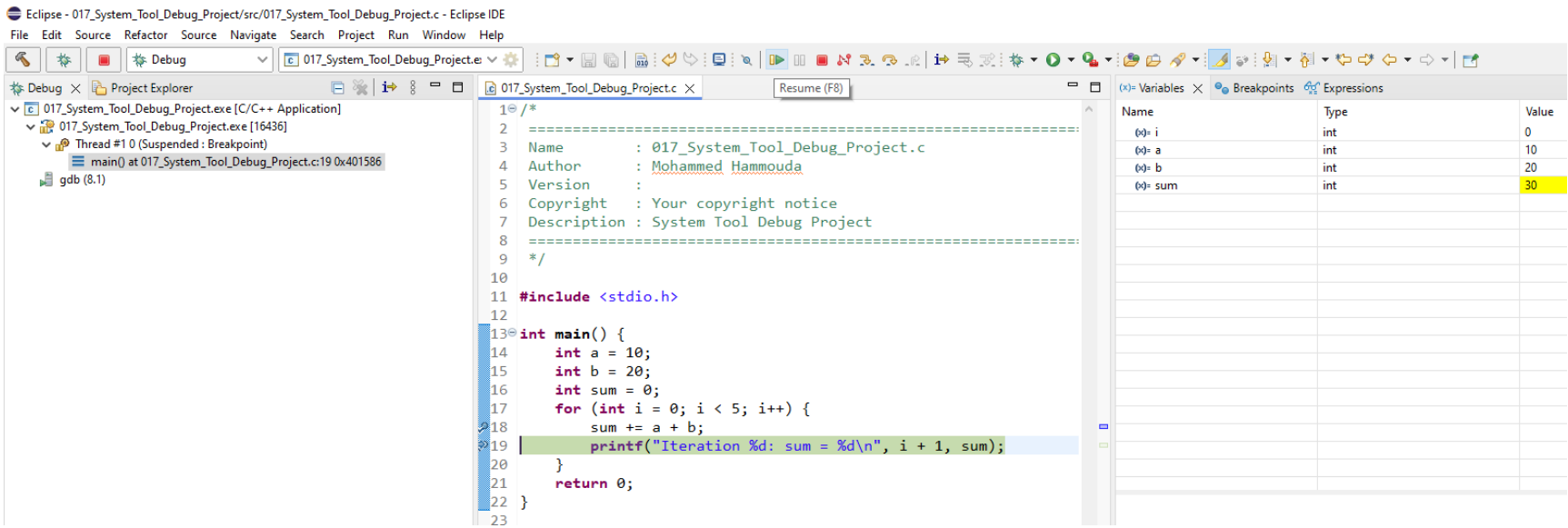


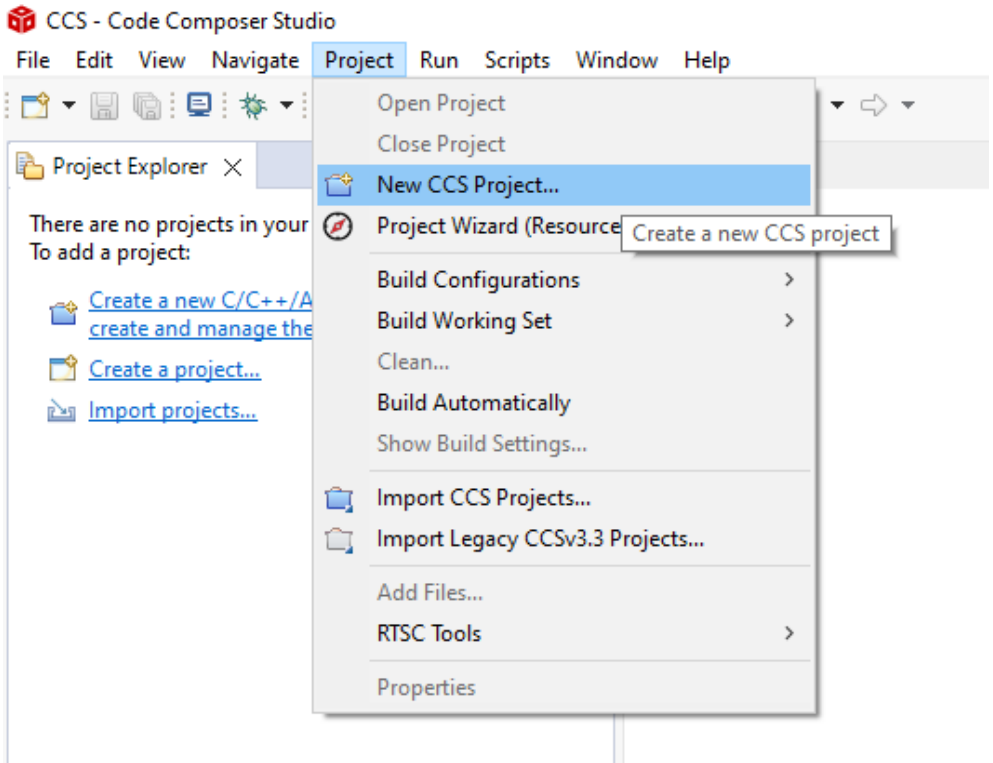
Table of Contents

Download and install CCS IDE & TivaWare

- There are many options available to write programs for Tiva C Launchpad such as Energia IDE, IAR Embedded Workbench for Arm, Code Composer Studio (CCS), and Keil.
- The IDE that we will use for programming is CCS.
- Download and install the latest version CCS :
  `https://www.ti.com/tool/CCSTUDIO`
- Download and install the latest version of TivaWare C SDK: Driver library for all device peripherals.
  `https://www.ti.com/tool/SW-TM4C`

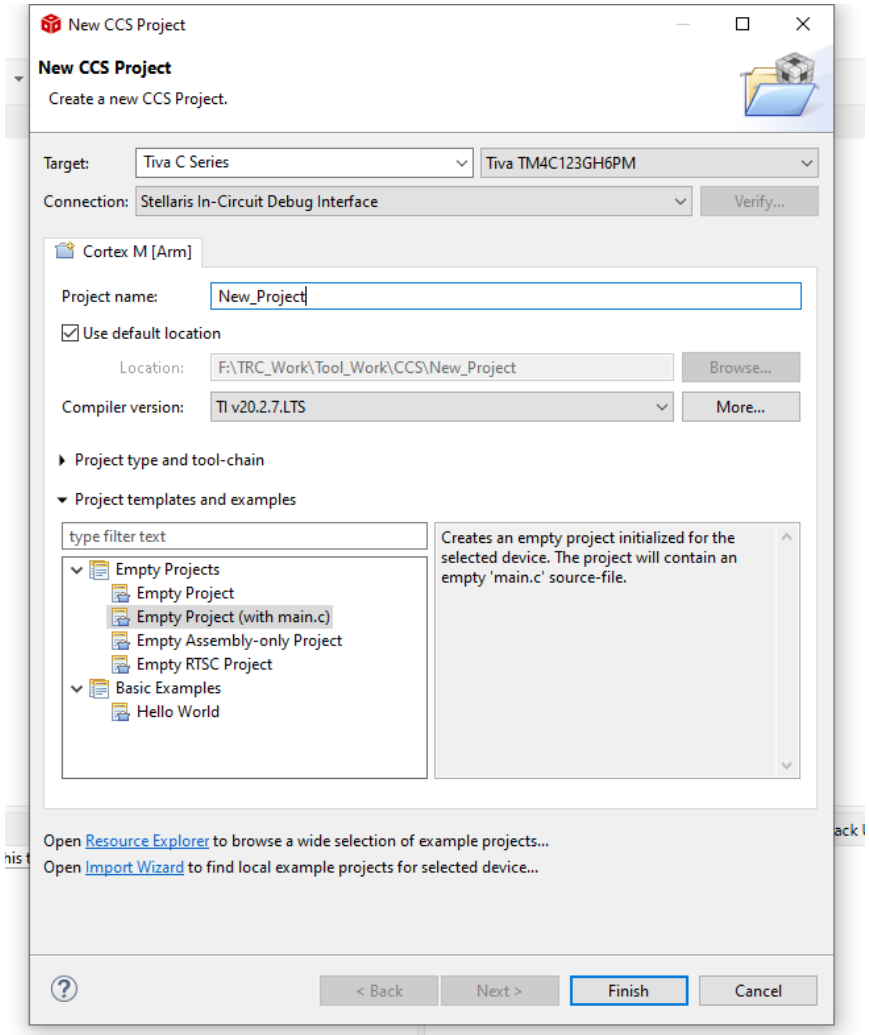
Configuration for CCS with TivaWare C

- Copy this header file ”tm4c123gh6pm.h”
  From: C:\ti\TivaWare_C_Series-2.2.0.295\inc
  To:C:\ti\cs1271\ccs\tools\compiler\ti-cgt-arm_20.2.7.LTS\include
- Create new CCS project:



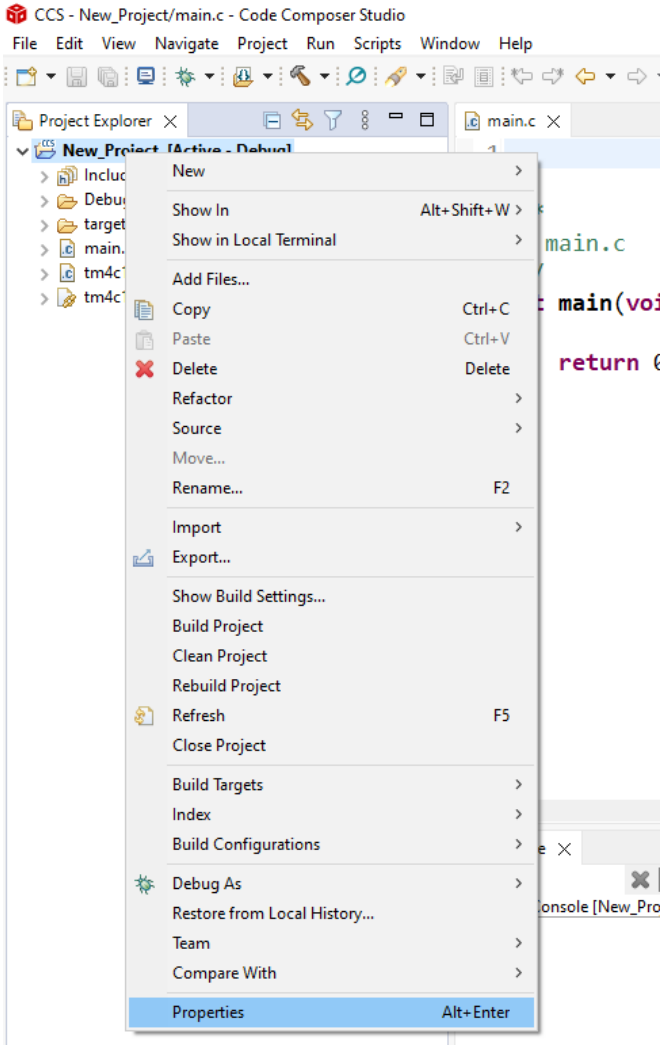
Configuration for CCS with TivaWare C

- Select the targeted board, connection, and project name:



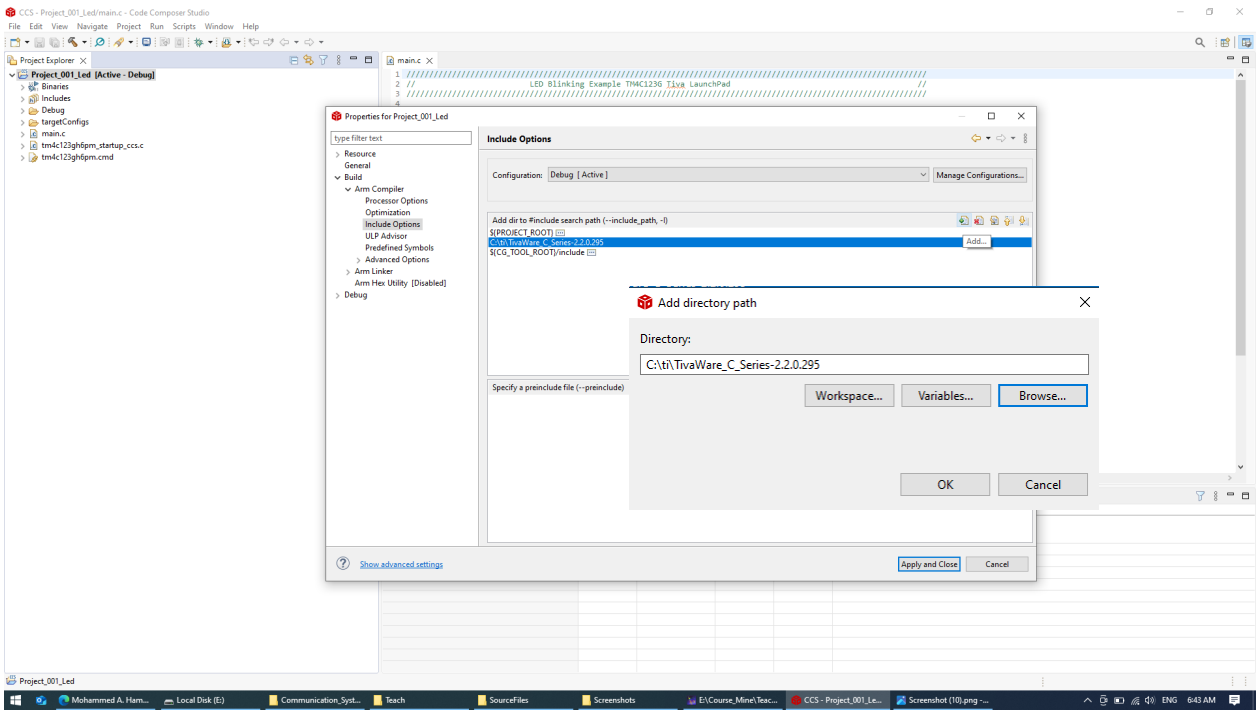
Configuration for CCS with TivaWare C

- Open the properties for the created project:



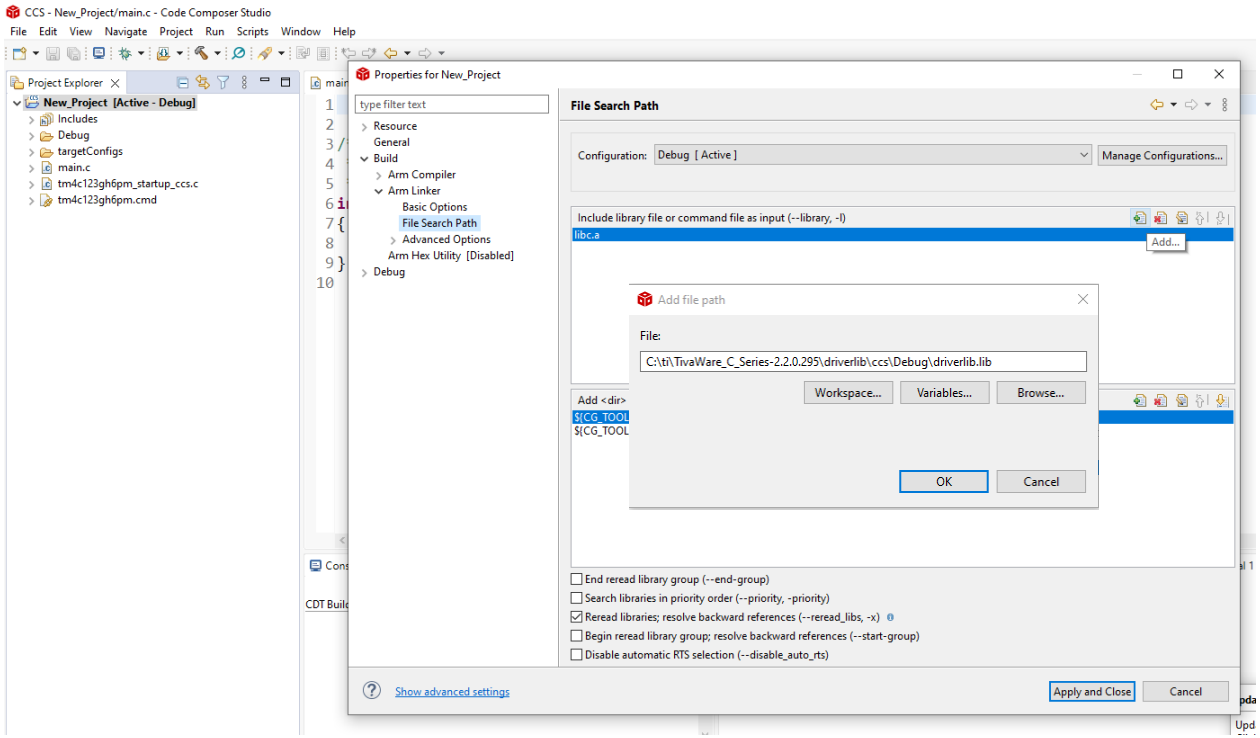
Configuration for CCS with TivaWare C

- Build− >Arm Compiler− >Include Options− >Add:



Configuration for CCS with TivaWare C

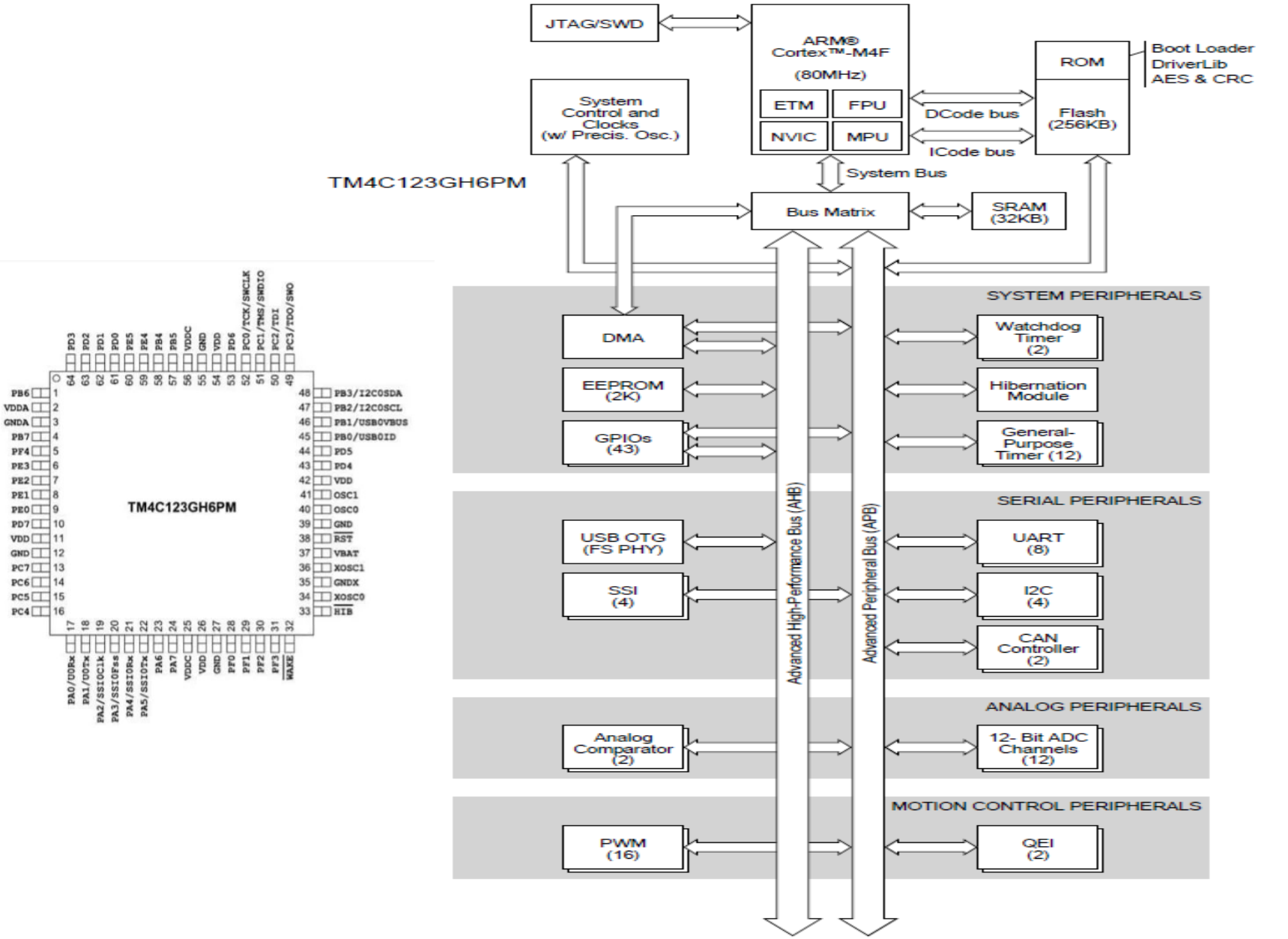
- Build− >Arm Linker− >File Search Path− >Add:



- Now your CCS configuration is ready to implement your first project

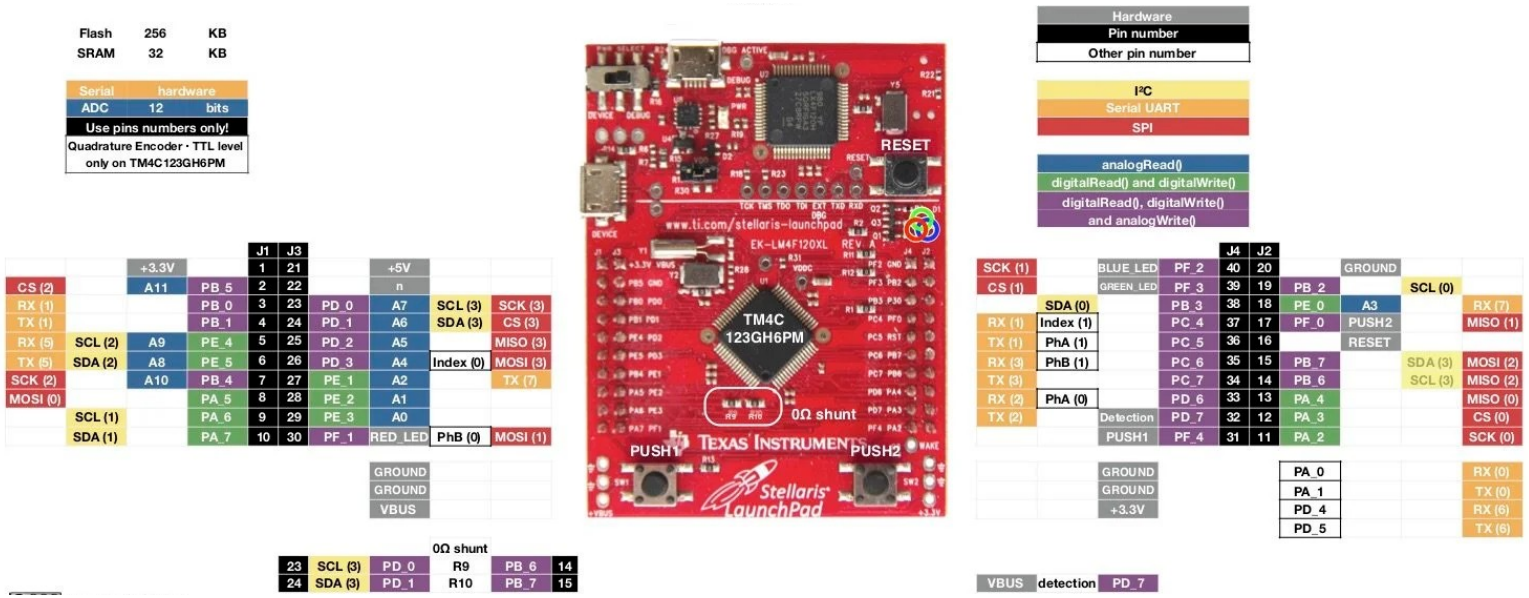
Table of Contents

TM4C123GH6PM MCU Block Diagram P.48



TM4C123GH6PM MCU

- **TM4C123GH6PM** is a member of the class of high-performance 32-bit ARM cortex M4 MCU with a broad set of peripherals developed by Texas Instrumentals.
- The pins of the MCU are internally connected to the output onboard (Tiva C LaunchPad) ports



TM4C123G Pinout Diagram

Access Memory Mapped Peripherals Reg. of MCU

- For ARM Cortex M4 32-bit (address bus width) MCU, its **addressable memory space** is 2^{32}.
- **Memory-mapped peripheral** registers, which are mapped to a specific memory addresses, are used to **control (either to read and write data from this address) and interact with the various peripherals** (such as GPIOs, timers, UARTs, ADCs, etc.) of the MCU.
- Each peripheral typically has multiple registers, these **registers can be accessed using pointers**.

Access Memory Mapped Peripherals Reg. of MCU

- ARM Cortex M4 Different Memory Regions
  - (0x0000_0000 - 0x1FFF_FFFF): is used to store program/code.
  - (0x2000_0000 - 0x3FFF_FFFF): SRAM, It is used to store temporary data such as heap, stack and temporary function variables.
  - (0x4000_0000 – 0x5FFF_FFFF): **Peripheral Registers, such as GPIO ports, UART, I2C, etc.**
  - (0x6000_0000 – 0x9FFF_FFFF): External RAM, this region is used to map external memory devices such as SD card, external Flash.
- Our main target is dealing with peripheral memory regions such as GPIO, ADC, Timers, UART, SPI, etc.

Access Memory Mapped Peripherals Reg. of MCU

- Peripheral Memory Region TM4C123GH6PM MCU
  - In this table, **offset address** defines how far away is the specific register from the **base address** of that PORT.
  - For example, if you want to find the address of GPIOADATA register, you can simply add the base address of GPIOA (0x4000_4000) with offset address of DATA register (0x000), you will get the physical address of PORTADATA register.

Port name	Lower address	Upper address
GPIO port A	0x40004000	0x40004FFF
GPIO port B	0x40005000	0x40005FFF
GPIO port C	0x40006000	0x40006FFF
GPIO port D	0x40007000	0x40007FFF
GPIO port E	0x40024000	0x40024FFF
GPIO port F	0x40025000	0x40025FFF

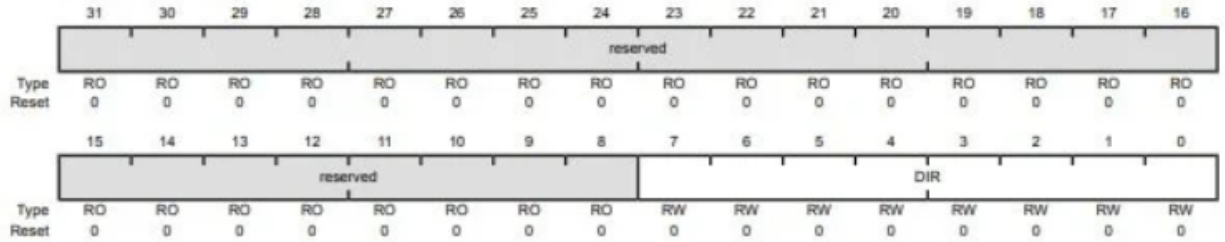
Offset	Register Name
0xFFC	GPIOPCellID3
0xFF8	GPIOPCellID2
0x41C	GPIOICR
0x418	GPIOMIS
0x414	GPIORIS
0x410	GPIOIM
0x40C	GPIOIEV
0x408	GPIOIBE
0x404	GPIOIS
0x400	GPIODIR
0x000	GPIODATA

Access Memory Mapped Peripherals Reg. of MCU

- How to access Peripheral Registers? (Data sheet P.658)
  - Direction control register of PORTF used to configure the pins of PORTF either input or output.

GPIO Direction (GPIODIR)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
GPIO Port F (APB) base: 0x4002.5000
GPIO Port F (AHB) base: 0x4005.D000
Offset 0x400
Type RW, reset 0x0000.0000



Bit/Field	Name	Type	Reset	Description
31:8	reserved	RO	0x0000.00	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
7:0	DIR	RW	0x00	GPIO Data Direction

Value	Description
0	Corresponding pin is an input.
1	Corresponding pins is an output.

Access Memory Mapped Peripherals Reg. of MCU

- How to access Peripheral Registers? (Data sheet P.658)
  - First, let's find the physical address of the direction control register **GPIODIR** of PORTF.

```
Base address of PORTF = 0x40025000
Offset address of GPIODIR = 0x400 //P. 663 datasheet
GPIOFDIR Physical address = 0x40025000+0x400 = 0x4002_5400
```

  - Pointers are used to access Peripheral Registers
  - Let's say we want to set the first four pins as DIN pins and last four pins as DOUT pins. Directly dereferencing memory

```
#define GPIO_PORTF_DIR_R(*((volatile unsigned int*)0x40025400)
GPIO_PORTF_DIR_R = 0xF0;
data =  GPIO_PORTF_DIR_R;
```

TM4C123GH6PM Clock Gating

- Clock gating is a power-saving feature that enables or disables the clock to specific peripherals.
- By default, many peripherals are disabled to save power, and you must enable them using the corresponding clock gating register.

GPIO Alternate Function

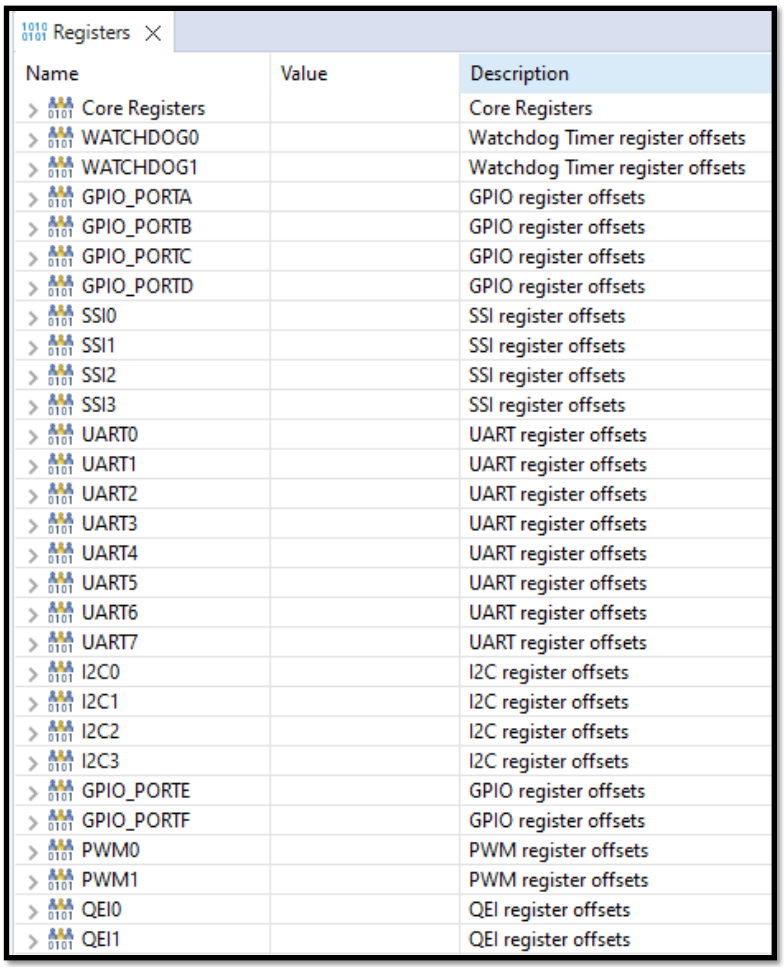
- Most of the GPIO pins can be configured for an alternate hardware function.
- Only 1 of the alternate functions can be configured at a given time.
- The user program can switch among different alternate functionality during execution.

Table 23-5. GPIO Pins and Alternate Functions

IO	Pin	Analog Function	Digital Function (GPIOPCTL PMCx Bit Field Encoding)[a]										
			1	2	3	4	5	6	7	8	9	14	15
PA0	17	-	U0Rx	-	-	-	-	-	-	CAN1Rx	-	-	-
PA1	18	-	U0Tx	-	-	-	-	-	-	CAN1Tx	-	-	-
PA2	19	-	-	SSI0Clk	-	-	-	-	-	-	-	-	-
PA3	20	-	-	SSI0Fss	-	-	-	-	-	-	-	-	-
PA4	21	-	-	SSI0Rx	-	-	-	-	-	-	-	-	-
PA5	22	-	-	SSI0Tx	-	-	-	-	-	-	-	-	-
PA6	23	-	-	-	I2C1SCL	-	M1PWM2	-	-	-	-	-	-
PA7	24	-	-	-	I2C1SDA	-	M1PWM3	-	-	-	-	-	-
PB0	45	USB0ID	U1Rx	-	-	-	-	-	T2CCP0	-	-	-	-
PB1	46	USB0VBUS	U1Tx	-	-	-	-	-	T2CCP1	-	-	-	-
PB2	47	-	-	-	I2C0SCL	-	-	-	T3CCP0	-	-	-	-
PB3	48	-	-	-	I2C0SDA	-	-	-	T3CCP1	-	-	-	-
PB4	58	AIN10	-	SSI2Clk	-	M0PWM2	-	-	T1CCP0	CAN0Rx	-	-	-
PB5	57	AIN11	-	SSI2Fss	-	M0PWM3	-	-	T1CCP1	CAN0Tx	-	-	-
PB6	1	-	-	SSI2Rx	-	M0PWM0	-	-	T0CCP0	-	-	-	-
PB7	4	-	-	SSI2Tx	-	M0PWM1	-	-	T0CCP1	-	-	-	-
PC0	52	-	TCK SWCLK	-	-	-	-	-	T4CCP0	-	-	-	-

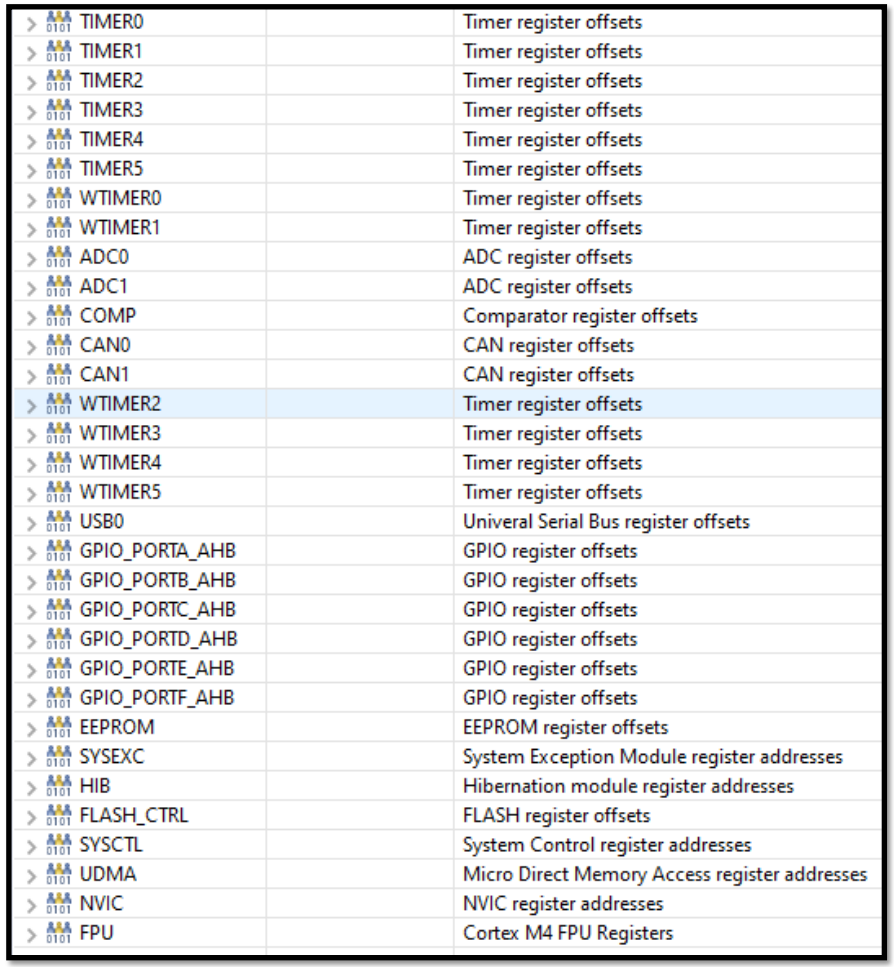
MCU Register Map

- Register map can be accessed using specific addresses to configure and controlling the MCU's peripherals (such as timers, UARTs, ADCs, GPIOs).



Registers

Name	Value	Description
Core Registers		Core Registers
WATCHDOG0		Watchdog Timer register offsets
WATCHDOG1		Watchdog Timer register offsets
GPIO_PORTA		GPIO register offsets
GPIO_PORTB		GPIO register offsets
GPIO_PORTC		GPIO register offsets
GPIO_PORTD		GPIO register offsets
SSI0		SSI register offsets
SSI1		SSI register offsets
SSI2		SSI register offsets
SSI3		SSI register offsets
UART0		UART register offsets
UART1		UART register offsets
UART2		UART register offsets
UART3		UART register offsets
UART4		UART register offsets
UART5		UART register offsets
UART6		UART register offsets
UART7		UART register offsets
I2C0		I2C register offsets
I2C1		I2C register offsets
I2C2		I2C register offsets
I2C3		I2C register offsets
GPIO_PORTE		GPIO register offsets
GPIO_PORTF		GPIO register offsets
PWM0		PWM register offsets
PWM1		PWM register offsets
QEI0		QEI register offsets
QEI1		QEI register offsets



TIMER0		Timer register offsets
TIMER1		Timer register offsets
TIMER2		Timer register offsets
TIMER3		Timer register offsets
TIMER4		Timer register offsets
TIMER5		Timer register offsets
WTIMER0		Timer register offsets
WTIMER1		Timer register offsets
ADC0		ADC register offsets
ADC1		ADC register offsets
COMP		Comparator register offsets
CAN0		CAN register offsets
CAN1		CAN register offsets
WTIMER2		Timer register offsets
WTIMER3		Timer register offsets
WTIMER4		Timer register offsets
WTIMER5		Timer register offsets
USB0		Univeral Serial Bus register offsets
GPIO_PORTA_AHB		GPIO register offsets
GPIO_PORTB_AHB		GPIO register offsets
GPIO_PORTC_AHB		GPIO register offsets
GPIO_PORTD_AHB		GPIO register offsets
GPIO_PORTE_AHB		GPIO register offsets
GPIO_PORTF_AHB		GPIO register offsets
EEPROM		EEPROM register offsets
SYSEXC		System Exception Module register addresses
HIB		Hibernation module register addresses
FLASH_CTRL		FLASH register offsets
SYSCTL		System Control register addresses
UDMA		Micro Direct Memory Access register addresses
NVIC		NVIC register addresses
FPU		Cortex M4 FPU Registers

MCU Register Map

- Registers often have individual bits or groups of bits that control different aspects of the peripheral or function (e.g., setting digital input pin, enabling a timer).

Registers

Name	Value	Description
GPIO_PORTF		GPIO register offsets
GPIO_DATA	0x00000000	GPIO Data [Memory Mapped]
GPIO_DIR	0x00000000	GPIO Direction [Memory Mapped]
GPIO_IS	0x00000000	GPIO Interrupt Sense [Memory Mapped]
GPIO_IBE	0x00000000	GPIO Interrupt Both Edges [Memory Mapped]
GPIO_IEV	0x00000000	GPIO Interrupt Event [Memory Mapped]
GPIO_IM	0x00000000	GPIO Interrupt Mask [Memory Mapped]
GPIO_RIS	0x00000000	GPIO Raw Interrupt Status [Memory Mapped]
GPIO_MIS	0x00000000	GPIO Masked Interrupt Status [Memory Mapped]
GPIO_ICR	0x00000000	GPIO Interrupt Clear [Memory Mapped]
GPIO_AFSEL	0x00000000	GPIO Alternate Function Select [Memory Mapped]
GPIO_DR2R	0x000000FF	GPIO 2-mA Drive Select [Memory Mapped]
GPIO_DR4R	0x00000000	GPIO 4-mA Drive Select [Memory Mapped]
GPIO_DR8R	0x00000000	GPIO 8-mA Drive Select [Memory Mapped]
GPIO_ODR	0x00000000	GPIO Open Drain Select [Memory Mapped]
GPIO_PUR	0x00000000	GPIO Pull-Up Select [Memory Mapped]
GPIO_PDR	0x00000000	GPIO Pull-Down Select [Memory Mapped]
GPIO_SLR	0x00000000	GPIO Slew Rate Control Select [Memory Mapped]
GPIO_DEN	0x00000000	GPIO Digital Enable [Memory Mapped]
GPIO_LOCK	0x00000001	GPIO Lock [Memory Mapped]
GPIO_LOCK	0x00000001 (Hex) -	GPIO Lock
GPIO_CR	0x000000FE	GPIO Commit [Memory Mapped]
GPIO_AMSEL	0x00000000	GPIO Analog Mode Select [Memory Mapped]
GPIO_PCTL	0x00000000	GPIO Port Control [Memory Mapped]
GPIO_ADCCTL	0x00000000	GPIO ADC Control [Memory Mapped]
GPIO_DMACTL	0x00000000	GPIO DMA Control [Memory Mapped]

Table of Contents

Tiva C Launchpad Examples

- Cortex-M4F core registers (no peripherals).
- Traffic light control system (GPIO and LED).
- Controlling LEDs with a push button (GPIO, Push Buttons, and LED).
- Serial communication between board and PC (GPIO and UART).
- External Interrupts (GPIO, Push Buttons, and LED).
- Project
  - Part 1:Using ADC through GPIO on to Measure Potentiometer Values (GPIO and ADC).
  - Part 2:Integrate ADC and UART (GPIO, ADC, and UART)

Cortex-M4F Core Registers (no peripherals)

- These registers are key to controlling the flow of program execution, managing data, handling interrupts, and performing arithmetic operations, including floating-point calculations when applicable.

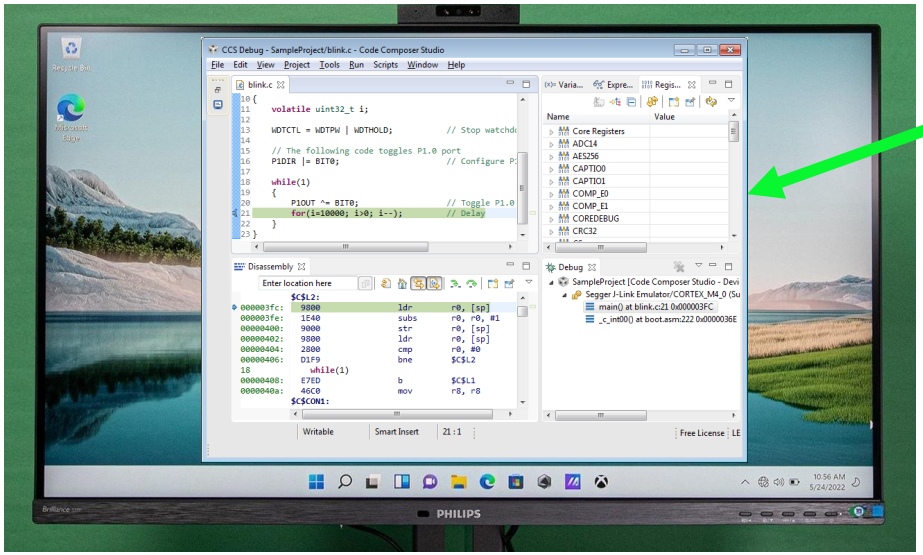
Registers

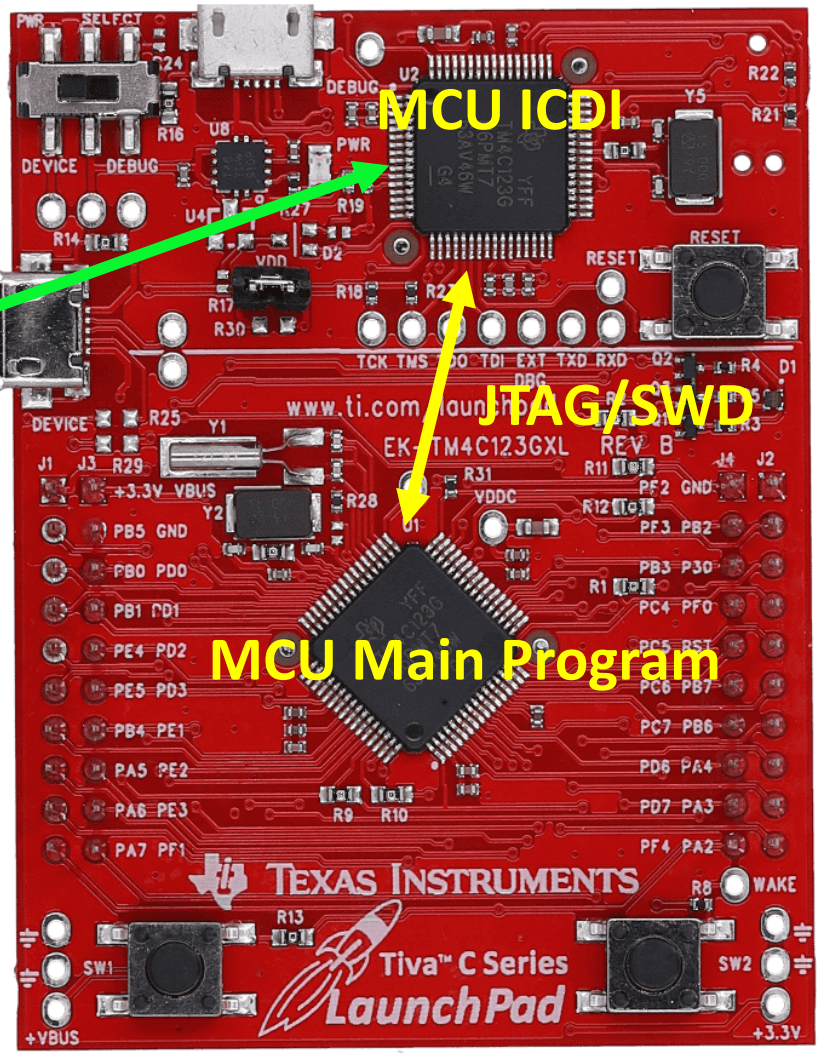
Name	Value	Description
Core Registers		Core Registers
PC	0x00000272	Program Counter [Core]
SP	0x20000200	General Purpose Register 13 - Stack Pointer [Core]
LR	0x0000032F	General Purpose Register 14 - Link Register [Core]
xPSR	0x01000000	Stores the status of interrupt enables and critical processor status signals [Core]
R0	0x00000020 (Hex)	General Purpose Register 0 [Core]
R1	0x400FE608	General Purpose Register 1 [Core]
R2	0x00000000	General Purpose Register 2 [Core]
R3	0x00000000	General Purpose Register 3 [Core]
R4	0x00000000	General Purpose Register 4 [Core]
R5	0x00000000	General Purpose Register 5 [Core]
R6	0x00000000	General Purpose Register 6 [Core]
R7	0x00000000	General Purpose Register 7 [Core]
R8	0x00000000	General Purpose Register 8 [Core]
R9	0x00000000	General Purpose Register 9 [Core]
R10	0x00000000	General Purpose Register 10 [Core]
R11	0x00000000	General Purpose Register 11 [Core]
R12	0x00000000	General Purpose Register 12 [Core]
R13	0x20000200	General Purpose Register 13 [Core]
R14	0x0000032F	General Purpose Register 14 [Core]
MSP	0x20000200	MSP Register [Core]
PSP	0x00000000	PSP Register [Core]
DSP	0x00000000	DSP Register [Core]
CTRL_FAULT_BASE	0x00000000	CM4 Special Registers [Core]

Cortex-M4F Core Registers (no peripherals)

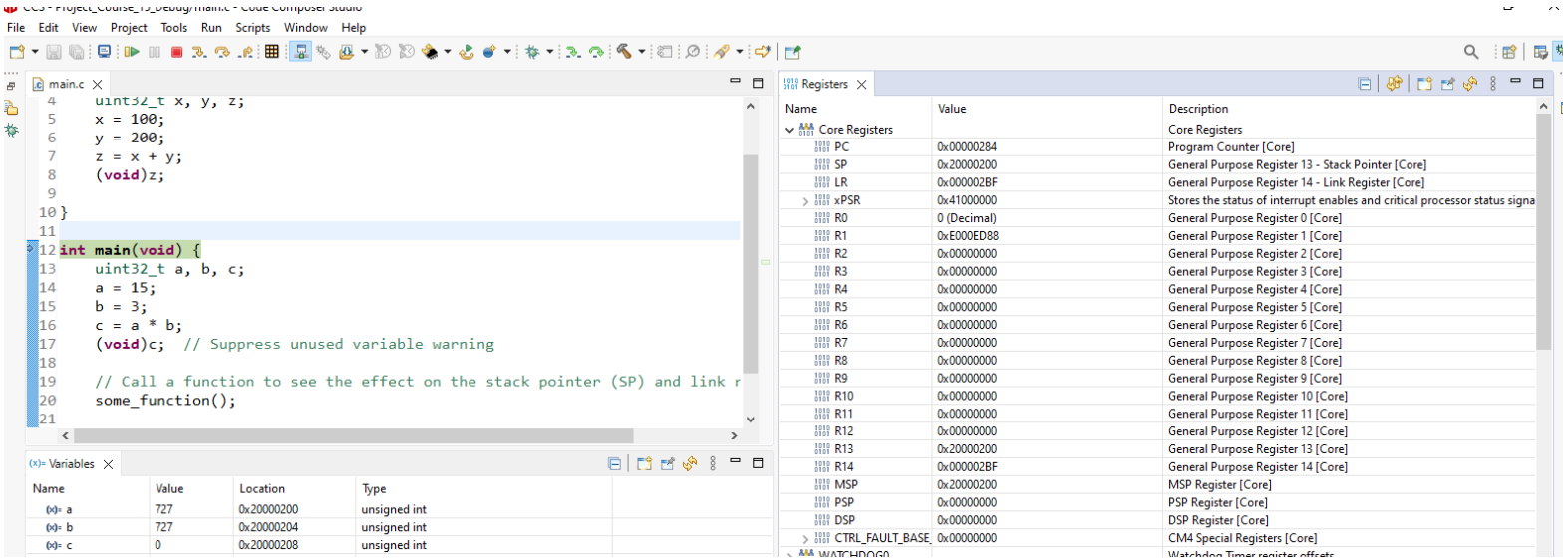
```c
#include <stdint.h>

void some_function(void) {
 uint32_t x, y, z;
 x = 100;
 y = 200;
 z = x + y;
}

int main(void) {
 uint32_t a, b, c;
 a = 15;
 b = 3;
 c = a * b;

 some_function();

 return 0;
}
```

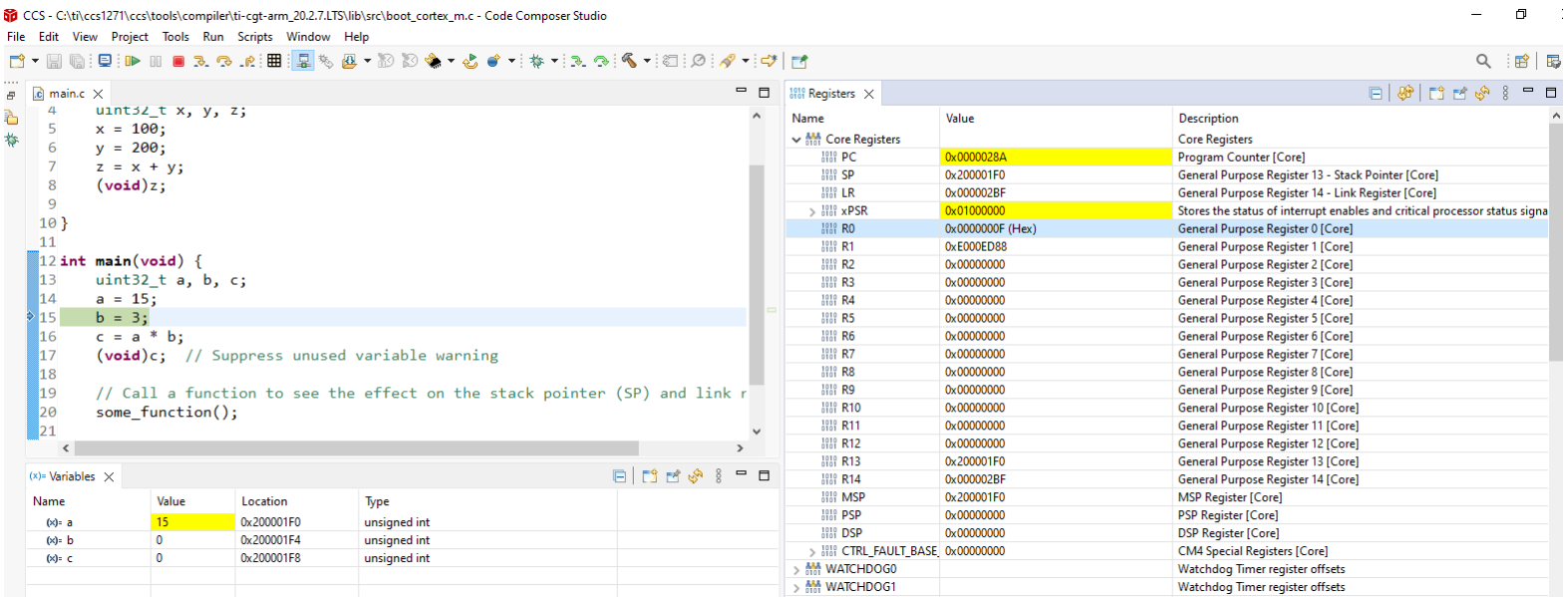
CCS In-Circuit Debug Interface (ICDI)



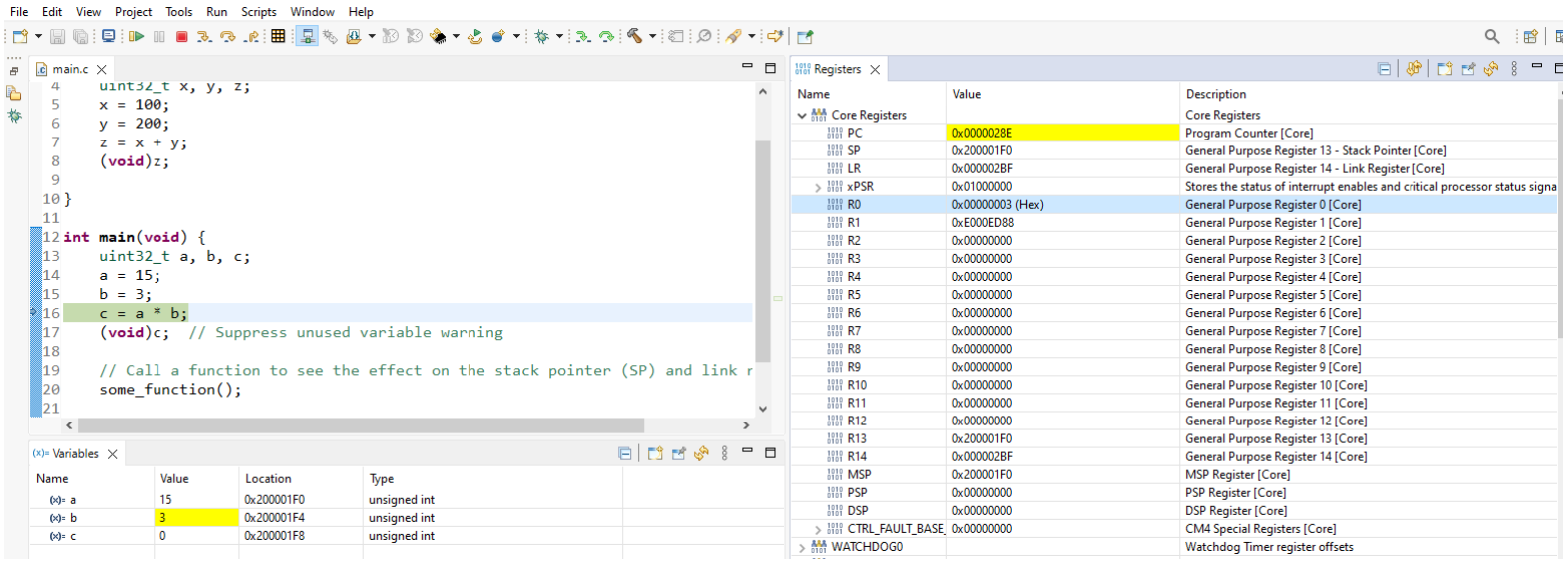


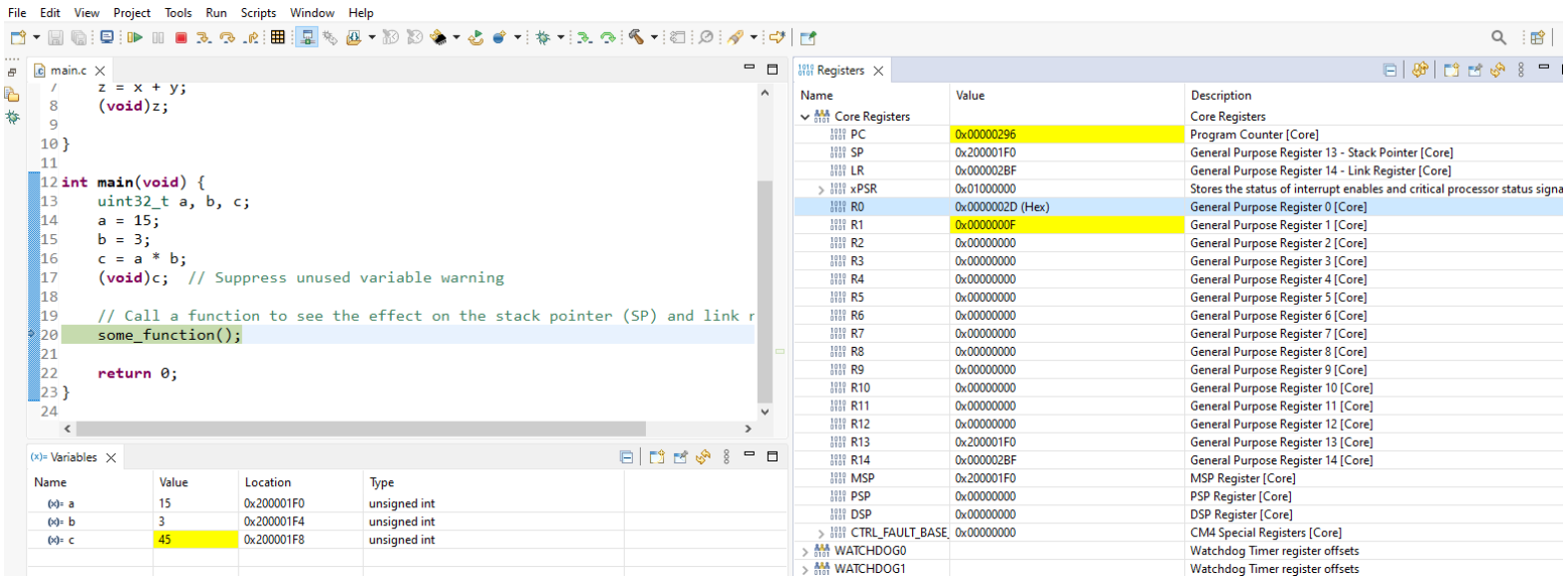
Cortex-M4F Core Registers (no peripherals)



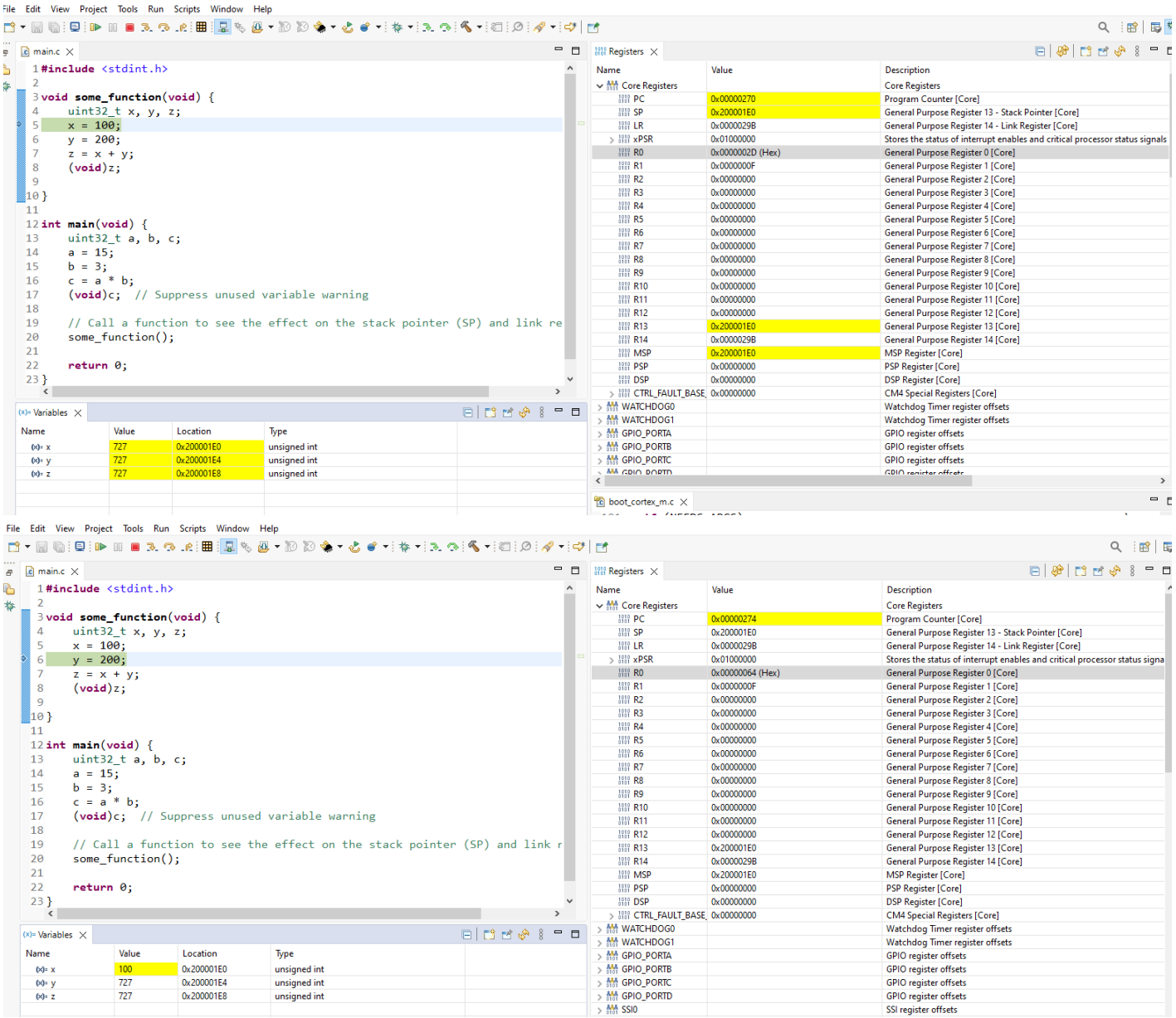


Cortex-M4F Core Registers (no peripherals)

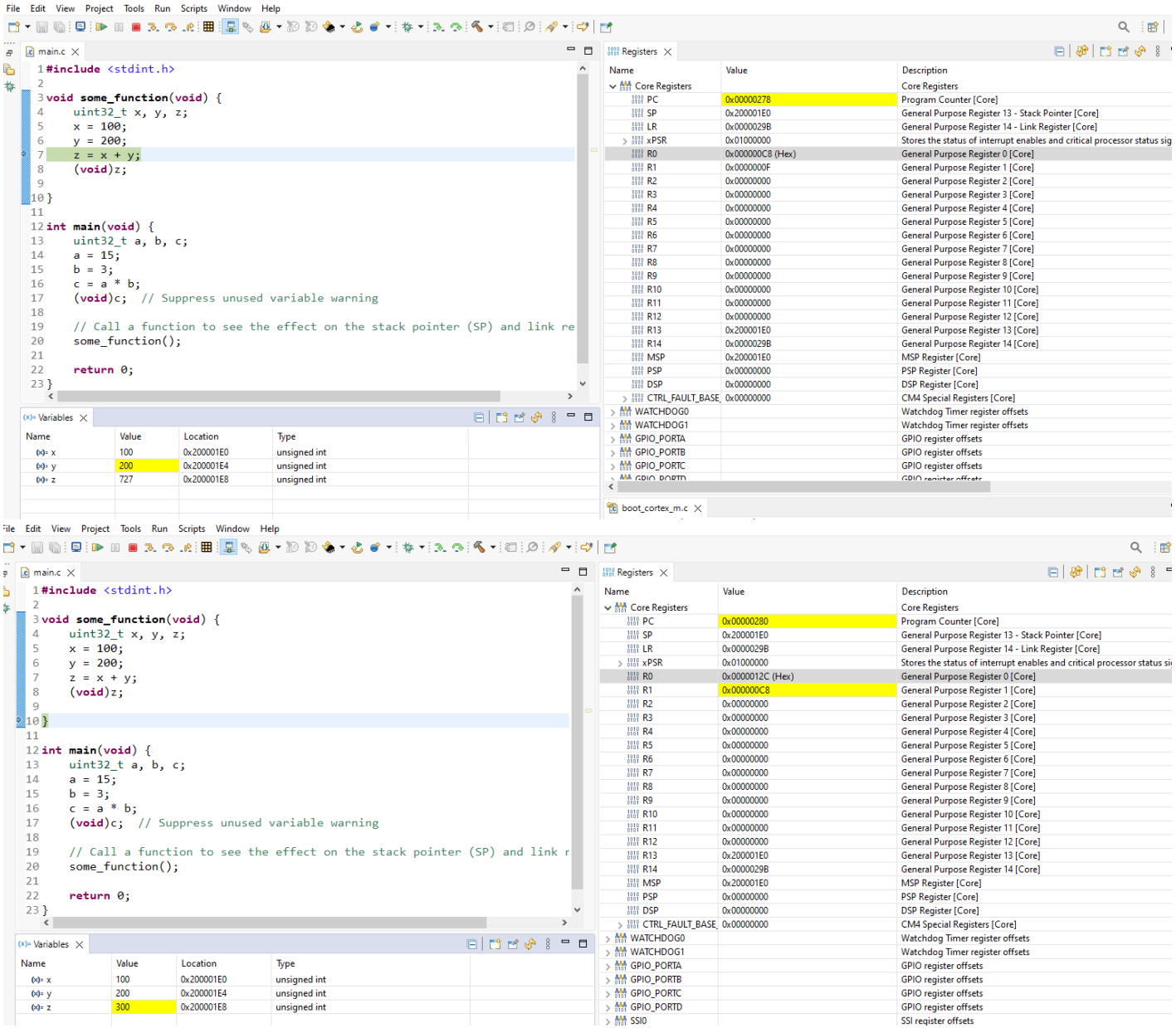




Cortex-M4F Core Registers (no peripherals)



Cortex-M4F Core Registers (no peripherals)



Traffic Light Control System

- Use GPIO for port F to control built-in LED of the TIVA C launchpad
- Used Registers, don't forget clock gating alternate function.

GPIO_PORTF		GPIO register offsets
GPIO_DATA	0x00000000	GPIO Data [Memory Mapped]
GPIO_DIR	0x00000000	GPIO Direction [Memory Mapped]
GPIO_IS	0x00000000	GPIO Interrupt Sense [Memory Mapped]
GPIO_IBE	0x00000000	GPIO Interrupt Both Edges [Memory Mapped]
GPIO_IEV	0x00000000	GPIO Interrupt Event [Memory Mapped]
GPIO_IM	0x00000000	GPIO Interrupt Mask [Memory Mapped]
GPIO_RIS	0x00000000	GPIO Raw Interrupt Status [Memory Mapped]
GPIO_MIS	0x00000000	GPIO Masked Interrupt Status [Memory Mapped]
GPIO_ICR	0x00000000	GPIO Interrupt Clear [Memory Mapped]
GPIO_AFSEL	0x00000000	GPIO Alternate Function Select [Memory Mapped]
GPIO_DR2R	0x000000FF	GPIO 2-mA Drive Select [Memory Mapped]
GPIO_DR4R	0x00000000	GPIO 4-mA Drive Select [Memory Mapped]
GPIO_DR8R	0x00000000	GPIO 8-mA Drive Select [Memory Mapped]
GPIO_ODR	0x00000000	GPIO Open Drain Select [Memory Mapped]
GPIO_PUR	0x00000000	GPIO Pull-Up Select [Memory Mapped]
GPIO_PDR	0x00000000	GPIO Pull-Down Select [Memory Mapped]
GPIO_SLR	0x00000000	GPIO Slew Rate Control Select [Memory Mapped]
GPIO_DEN	0x00000000	GPIO Digital Enable [Memory Mapped]
GPIO_LOCK	0x00000001	GPIO Lock [Memory Mapped]
GPIO_LOCK	0x00000001 (Hex) -	GPIO Lock
GPIO_CR	0x000000FE	GPIO Commit [Memory Mapped]
GPIO_AMSEL	0x00000000	GPIO Analog Mode Select [Memory Mapped]
GPIO_PCTL	0x00000000	GPIO Port Control [Memory Mapped]
GPIO_ADCCTL	0x00000000	GPIO ADC Control [Memory Mapped]
GPIO_DMACTL	0x00000000	GPIO DMA Control [Memory Mapped]

Traffic Light Control System

- GPIO Run-Mode Clock Gating Control (**RCGCGPIO**): GPIO pins clock enable register.
- GPIO Data (**GPIODATA**): The data register.
- GPIO Direction (**GPIODIR**): Data direction control register.
- GPIO Digital Enable (**GPIODEN**): Digital enable register.

General-Purpose Input/Output Run Mode Clock Gating Control (RCGCGPIO)

Base 0x400F.E000
Offset 0x608
Type RW, reset 0x0000.0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															
RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved										R5	R4	R3	R2	R1	R0
RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW	RW	RW
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit/Field	Name	Type	Reset	Description
31:6	reserved	RO	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
5	R5	RW	0	GPIO Port F Run Mode Clock Gating Control

Value	Description
0	GPIO Port F is disabled.
1	Enable and provide a clock to GPIO Port F in Run mode.

tm4c123gh6pm.h Data sheet

Traffic Light Control System

- Preprocessor Macros:

```
# define SYSCTL_RCGCGPIO_R (*(( volatile unsigned long *)0x400FE608))

# define GPIO_PORTF_DEN_R (*(( volatile unsigned long *)0x4002551C))

# define GPIO_PORTF_DIR_R (*(( volatile unsigned long *)0x40025400))

#define GPIO_PORTF_DATA_R (*(( volatile unsigned long *)0x40025038))
```

- GPIODATA Register: This register is virtually mapped to **256 locations** in the address space. The data read from and written to the registers are masked by the eight address lines [9:2]. See “Data Register Operation” P.654.
  The whole space:
  $[9:2] --> [PF7, PF6, PF5, PF4, PF3, PF2, PF1, PF0]$

```
GPIO_PORTF_DATA_R is 0x400253FC --> [9:2]
```

  To enable pin1, 2, and 3 of PORTF without affecting the other pins

```
0x038  from 0x3FC
```

Traffic Light Control System

- LED Blinking Code 1:

```
#include <stdint.h>

#define SYSCTL_RCGCGPIO_R (*((volatile uint32_t *)0x400FE608))
#define GPIO_PORTF_DEN_R  (*((volatile uint32_t *)0x4002551C))
#define GPIO_PORTF_DIR_R  (*((volatile uint32_t *)0x40025400))
#define GPIO_PORTF_DATA_R (*(( volatile unsigned long *)0x40025038 ))

int main(void)
{
 SYSCTL_RCGCGPIO_R = 0x20;//Enable clock for PORTF
 GPIO_PORTF_DEN_R = 0x0E;//Enable PORTF Pin1, 2 and 3 as a dig. pins
 GPIO_PORTF_DIR_R = 0x0E;//Conf. PORTF Pin1, 2 and 3 dig. O/P pins
 while (1)
 {
  GPIO_PORTF_DATA_R |= 0x02; // turn on red LED
 }
}
```

Traffic Light Control System

- LED Blinking Code 2:

```
#include <stdint.h>
#include"tm4c123gh6pm.h"

int main(void)
{
 SYSCTL_RCGCGPIO_R = 0x20;//Enable clock for PORTF
 GPIO_PORTF_DEN_R = 0x0E;//Enable PORTF Pin1, 2 and 3 as a dig. pins
 GPIO_PORTF_DIR_R = 0x0E;//Conf. PORTF Pin1, 2 and 3 dig. O/P pins
 while (1)
 {
  GPIO_PORTF_DATA_R |= 0x02; // turn on red LED
 }
}
```

Traffic Light Control System

- LED Blinking Code 3:

```
#include <stdint.h>
#include"tm4c123gh6pm.h"

int main(void)
{
 SYSCTL_RCGCGPIO_R = 0x20;//Enable clock for PORTF
 GPIO_PORTF_DEN_R = 0x0E;//Enable PORTF Pin1, 2 and 3 as a dig. pins
 GPIO_PORTF_DIR_R = 0x0E;//Conf. PORTF Pin1, 2 and 3 dig. O/P pins
 while (1)
 {
  GPIO_PORTF_DATA_R |= 0x02; // turn on red LED
  __delay_cycles(8000000);   // delay 1 sec
  GPIO_PORTF_DATA_R &= 0x00; // turn off red LED
  __delay_cycles(8000000);   // delay 1 sec
 }
}
```

Traffic Light Control System

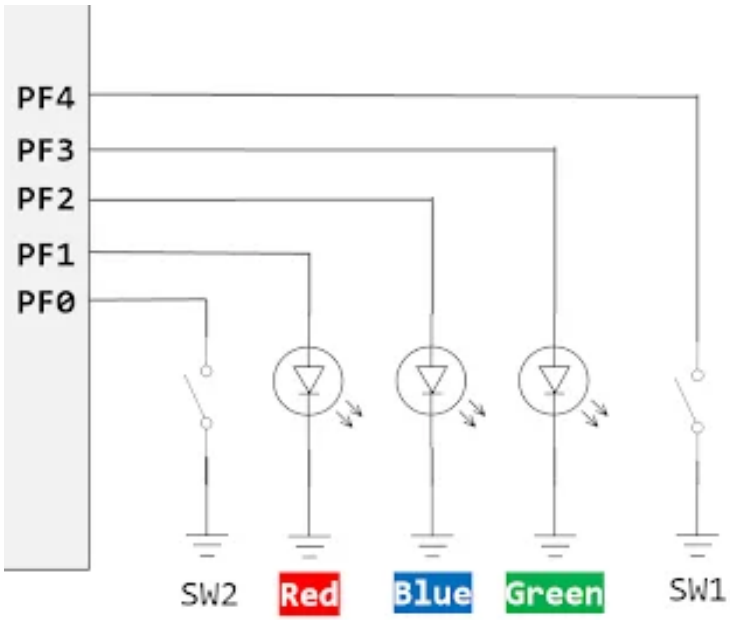
- LED Blinking Code 4:

```
  ...
while (1)
 {
 GPIO_PORTF_DATA_R |= 0x02; // turn on red LED
 __delay_cycles(8000000);   // delay 1 sec
 GPIO_PORTF_DATA_R &= 0x00; // turn off red LED
 __delay_cycles(8000000);   // delay 1 sec

 GPIO_PORTF_DATA_R |= 0x04; // turn on blue LED
 __delay_cycles(8000000);   // delay 1 sec
 GPIO_PORTF_DATA_R &= 0x00; // turn off blue LED
 __delay_cycles(8000000);   // delay 1 sec

 GPIO_PORTF_DATA_R |= 0x08; // turn on green LED
 __delay_cycles(8000000);   // delay 1 sec
 GPIO_PORTF_DATA_R &= 0x00; // turn off green LED
 __delay_cycles(8000000);   // delay 1 sec
 }
  ...
```

Controlling LED with a Push Button

- Use GPIO for port F to control built-in LED using Push Button of the TIVA C launchpad.
- Used Registers, don't forget clock gating alternate function.

User Manual

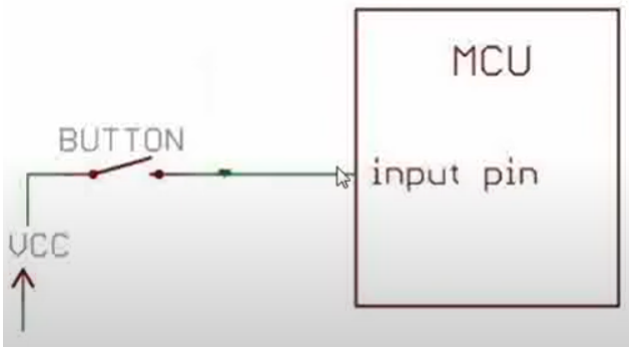


Registers

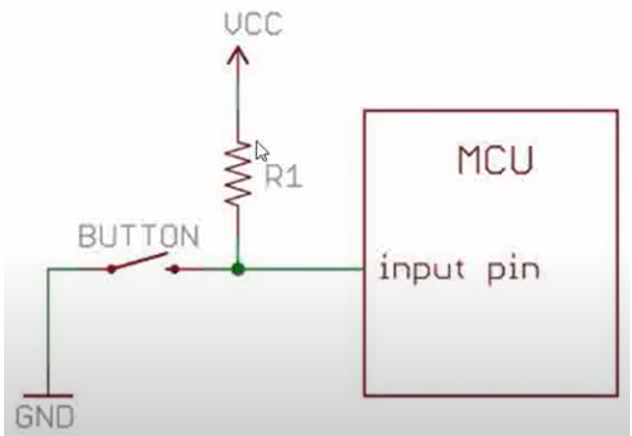
Name	Value	Description
GPIO_PORTF		GPIO register offsets
GPIO_DATA	0x00000000	GPIO Data [Memory Mapped]
GPIO_DIR	0x00000000	GPIO Direction [Memory Mapped]
GPIO_IS	0x00000000	GPIO Interrupt Sense [Memory Mapped]
GPIO_IBE	0x00000000	GPIO Interrupt Both Edges [Memory Mapped]
GPIO_IEV	0x00000000	GPIO Interrupt Event [Memory Mapped]
GPIO_IM	0x00000000	GPIO Interrupt Mask [Memory Mapped]
GPIO_RIS	0x00000000	GPIO Raw Interrupt Status [Memory Mapped]
GPIO_MIS	0x00000000	GPIO Masked Interrupt Status [Memory Mapped]
GPIO_ICR	0x00000000	GPIO Interrupt Clear [Memory Mapped]
GPIO_AFSEL	0x00000000	GPIO Alternate Function Select [Memory Mapped]
GPIO_DR2R	0x000000FF	GPIO 2-mA Drive Select [Memory Mapped]
GPIO_DR4R	0x00000000	GPIO 4-mA Drive Select [Memory Mapped]
GPIO_DR8R	0x00000000	GPIO 8-mA Drive Select [Memory Mapped]
GPIO_ODR	0x00000000	GPIO Open Drain Select [Memory Mapped]
GPIO_PUR	0x00000000	GPIO Pull-Up Select [Memory Mapped]
GPIO_PDR	0x00000000	GPIO Pull-Down Select [Memory Mapped]
GPIO_SLR	0x00000000	GPIO Slew Rate Control Select [Memory Mapped]
GPIO_DEN	0x00000000	GPIO Digital Enable [Memory Mapped]
GPIO_LOCK	0x00000001	GPIO Lock [Memory Mapped]
GPIO_LOCK	0x00000001 (Hex) -	GPIO Lock
GPIO_CR	0x000000FE	GPIO Commit [Memory Mapped]
GPIO_AMSEL	0x00000000	GPIO Analog Mode Select [Memory Mapped]
GPIO_PCTL	0x00000000	GPIO Port Control [Memory Mapped]
GPIO_ADCCTL	0x00000000	GPIO ADC Control [Memory Mapped]
GPIO_DMACTL	0x00000000	GPIO DMA Control [Memory Mapped]

Pull-up and Pull-down Resistor

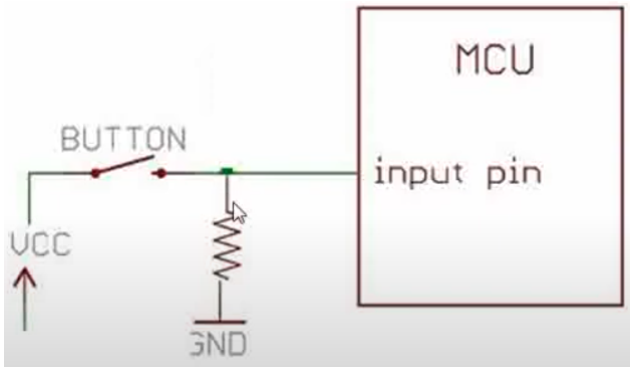
- The tm4c123gh6pm MCU has internal pull-up and pull-down registers associated with each port.



Floating Inputs



Pull-up Resistor



Pull-down Resistor

Controlling LED with a Push Button

- GPIO Run-Mode Clock Gating Control (**RCGCGPIO**): GPIO pins clock enable register.
- GPIO Data (**GPIODATA**): The data register.
- GPIO Direction (**GPIODIR**): Data direction control register.
- GPIO Digital Enable (**GPIODEN**): Digital enable register.

Controlling LED with a Push Button

- GPIO Run-Mode Clock Gating Control (**RCGCGPIO**): GPIO pins clock enable register.
- GPIO Data (**GPIODATA**): The data register.
- GPIO Direction (**GPIODIR**): Data direction control register.
- GPIO Digital Enable (**GPIODEN**): Digital enable register.
- GPIO Pull-Up Select (**GPIOPUR**).

Controlling LED with a Push Button

- GPIO Run-Mode Clock Gating Control (**RCGCGPIO**): GPIO pins clock enable register.
- GPIO Data (**GPIODATA**): The data register.
- GPIO Direction (**GPIODIR**): Data direction control register.
- GPIO Digital Enable (**GPIODEN**): Digital enable register.
- GPIO Pull-Up Select (**GPIOPUR**).
- GPIO Lock (**GPIOLOCK**): Enables write access to the GPIOCR register. Writing 0x4C4F434B unlocks the GPIOCR register. LOCKED $->$ 0x00000001 and UNLOCKED $->$ 0x00000000.

Controlling LED with a Push Button

- GPIO Run-Mode Clock Gating Control (**RCGCGPIO**): GPIO pins clock enable register.
- GPIO Data (**GPIODATA**): The data register.
- GPIO Direction (**GPIODIR**): Data direction control register.
- GPIO Digital Enable (**GPIODEN**): Digital enable register.
- GPIO Pull-Up Select (**GPIOPUR**).
- GPIO Lock (**GPIOLOCK**): Enables write access to the GPIOCR register. Writing 0x4C4F434B unlocks the GPIOCR register. LOCKED $->$ 0x00000001 and UNLOCKED $->$ 0x00000000.
- GPIO Commit (**GPIOCR**): Determines which bits of the GPIOAFSEL, GPIOPUR, GPIOPDR, and GPIODEN registers are committed when write to these registers.

Controlling LED with a Push Button

- Preprocessor Macros:

```
# define SYSCTL_RCGCGPIO_R (*(( volatile unsigned long *)0x400FE608))

# define GPIO_PORTF_DEN_R (*(( volatile unsigned long *)0x4002551C))

# define GPIO_PORTF_DIR_R (*(( volatile unsigned long *)0x40025400))

#define GPIO_PORTF_DATA_R (*(( volatile unsigned long *)0x40025038))

#define GPIO_PORTF_LOCK_R (*((volatile uint32_t *)0x40025520))

#define GPIO_PORTF_CR_R   (*((volatile uint32_t *)0x40025524))

#define GPIO_PORTF_PUR_R  (*((volatile uint32_t *)0x40025510))
```

Controlling LED with a Push Button

- Push Button Code 1:

```
#include <stdint.h>
#include"tm4c123gh6pm.h"

int main(void)
{
 unsigned int state;   // Used to hold the state of the button.
 SYSCTL_RCGCGPIO_R  |= 0x20;          // enable clock to GPIOF
 GPIO_PORTF_LOCK_R   = 0x4C4F434B;   // unlockGPIOCR register
 GPIO_PORTF_CR_R |= 0x10; //En. GPIOPUR enable to commit
 GPIO_PORTF_PUR_R   |= 0x10;    // Enable Pull Up resistor PF4
 GPIO_PORTF_DIR_R|= 0x02;//set PF1 as O/P and PF4 as I/P pin
 GPIO_PORTF_DEN_R|= 0x12;// En. PF1 and PF4 as dig. GPIO pins
 while(1)
 {
 state = GPIO_PORTF_DATA_R & 0x10; // Read PF4 (active high)
if (state) {
 GPIO_PORTF_DATA_R &= ~0x02; // Turn off red LED (PF1)
} else {
 GPIO_PORTF_DATA_R |= 0x02;  // Turn on red LED (PF1)
 }
 }
}
```

Controlling LED with a Push Button

- Push Button Code 2: (Specific address for data reg.)

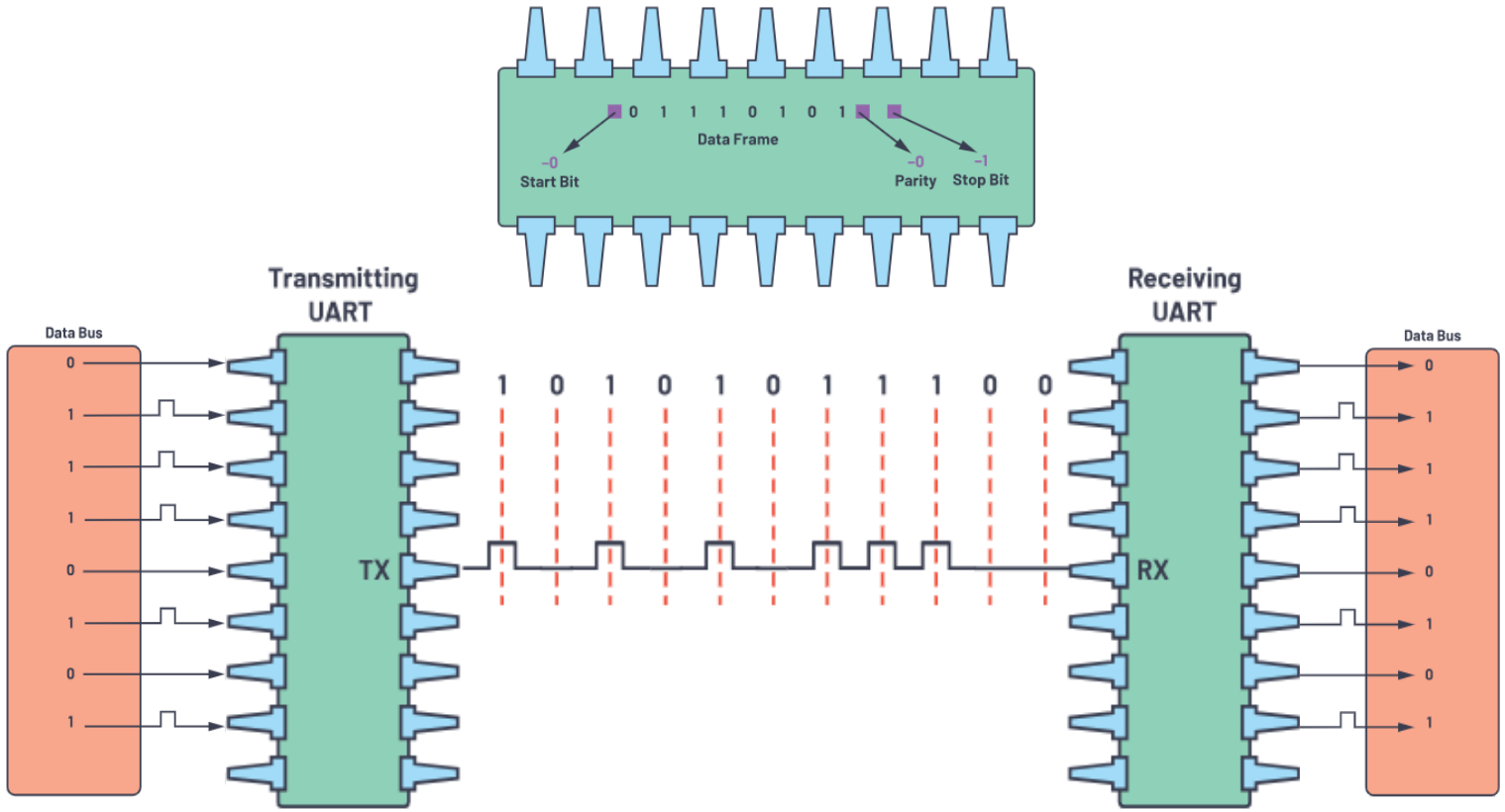
```
#include <stdint.h>
#include"tm4c123gh6pm.h"

# define GPIO_PORTF_DATA_RD (*(( volatile unsigned long *)0
    x40025040))
# define GPIO_PORTF_DATA_WR (*(( volatile unsigned long *)0
    x40025020))

int main(void) {
 static char flag   = 0;
 SYSCTL_RCGCGPIO_R |= 0x20;
 GPIO_PORTF_DEN_R  |= 0x18;
 GPIO_PORTF_DIR_R  |= 0x08;
 GPIO_PORTF_DIR_R  &= 0xfe;
 GPIO_PORTF_LOCK_R  = 0x4C4F434B;
 GPIO_PORTF_CR_R   |= 0x10;
 GPIO_PORTF_PUR_R  |= 0x10;
```

Controlling LED with a Push Button

```
    while(1)
    {
     if(GPIO_PORTF_DATA_RD == 0)
     {
      __delay_cycles(1000000);   // delay 1/8 sec
      if(( flag == 0) && (GPIO_PORTF_DATA_RD == 0))
      {
       GPIO_PORTF_DATA_WR ^= 0x08;
       flag = 1;
      }
     }
     else
     {
      flag = 0;
     }
    }
  }
```

Serial Communication (UART)

- Serial interfacing is widely used in embedded industry.
- We will illustrate one of the first and simplest protocols that transmit one bit at a time universal asynchronous receiver/transmitter (UART)



Serial Communication (UART)

- Use UART5 on port E to TR/RX serial data to PC.
- Used Registers, don't forget clock gating alternate function.

Registers

Name	Value	Description
UART5		UART register offsets
UART_DR	0x00000500	UART Data [Memory Mapped]
UART_RSR	0x00000005	UART Receive Status/Error Clear [Memory Mapped]
UART_ECR	0x00000005	UART Receive Status/Error Clear [Memory Mapped]
UART_FR	0x00000090	UART Flag [Memory Mapped]
UART_ILPR	0x00000000	UART IrDA Low-Power Register [Memory Mapped]
UART_IBRD	0x00000068	UART Integer Baud-Rate Divisor [Memory Mapped]
UART_FBRD	0x0000000B	UART Fractional Baud-Rate Divisor [Memory Mapped]
UART_LCRH	0x00000060	UART Line Control [Memory Mapped]
UART_CTL	0x00000301	UART Control [Memory Mapped]
UART_IFLS	0x00000012	UART Interrupt FIFO Level Select [Memory Mapped]
UART_IM	0x00000000	UART Interrupt Mask [Memory Mapped]
UART_RIS	0x000002AF	UART Raw Interrupt Status [Memory Mapped]
UART_MIS	0x00000000	UART Masked Interrupt Status [Memory Mapped]
UART_ICR	0x00000000	UART Interrupt Clear [Memory Mapped]
UART_DMACTL	0x00000000	UART DMA Control [Memory Mapped]
UART_9BITADDR	0x00000000	UART 9-Bit Self Address [Memory Mapped]
UART_9BITAMASK	0x000000FF	UART 9-Bit Self Address Mask [Memory Mapped]
UART_PP	0x00000003	UART Peripheral Properties [Memory Mapped]
UART_CC	0x00000000	UART Clock Configuration [Memory Mapped]

Serial Communication (UART)

- TM4C123GH6PM MCU supports 8 UART ports.

UART Module	Rx Pin	Tx Pin
UART0	PA0	PA1
UART1	PC4	PC5
UART2	PD6	PD7
UART3	PC6	PC7
UART4	PC4	PC5
UART5	PE4	PE5
UART6	PD4	PD5
UART7	PE0	PE1

Serial Communication (UART)

- UART Module Configuration Registers.
  - UART Clock Control Register (P.342)

```
#define SYSCTL_RCGCUART_R (*((volatile uint32_t *)0x400FE618))
```

Universal Asynchronous Receiver/Transmitter Run Mode Clock Gating Control (RCGCUART)

Base 0x400F.E000
Offset 0x618
Type RW, reset 0x0000.0000

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved								R7	R6	R5	R4	R3	R2	R1	R0
Type	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit/Field	Name	Type	Reset	Description
31:8	reserved	RO	0	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
7	R7	RW	0	UART Module 7 Run Mode Clock Gating Control

Value	Description
0	UART module 7 is disabled.
1	Enable and provide a clock to UART module 7 in Run mode.

Serial Communication (UART)

- UART Module Configuration Registers.
  - GPIO Port Control (GPIOPCTL) for PMC $->$ UART5 (P.688 & P.1352)

```
#define GPIO_PORTE_PCTL_R (*((volatile uint32_t *)0x4002452C))
```

GPIO Port Control (GPIOPCTL)

GPIO Port A (APB) base: 0x4000.4000
GPIO Port A (AHB) base: 0x4005.8000
GPIO Port B (APB) base: 0x4000.5000
GPIO Port B (AHB) base: 0x4005.9000
GPIO Port C (APB) base: 0x4000.6000
GPIO Port C (AHB) base: 0x4005.A000
GPIO Port D (APB) base: 0x4000.7000
GPIO Port D (AHB) base: 0x4005.B000
GPIO Port E (APB) base: 0x4002.4000
GPIO Port E (AHB) base: 0x4005.C000
GPIO Port F (APB) base: 0x4002.5000
GPIO Port F (AHB) base: 0x4005.D000
Offset 0x52C
Type RW, reset -

Table 23-5. GPIO Pins and Alternate Functions *(continued)*

IO	Pin	Analog Function	Digital Function (GPIOPCTL PMCx Bit Field Encoding)[a]										
			1	2	3	4	5	6	7	8	9	14	15
PE4	59	AIN9	U5Rx	-	I2C2SCL	M0PWM4	M1PWM2	-	-	CAN0Rx	-	-	-
PE5	60	AIN8	U5Tx	-	I2C2SDA	M0PWM5	M1PWM3	-	-	CAN0Tx	-	-	-

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
PMC7				PMC6				PMC5				PMC4			
Type	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Reset	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PMC3				PMC2				PMC1				PMC0			
Type	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW	RW
Reset	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-

Bit/Field	Name	Type	Reset	Description
31:28	PMC7	RW	-	Port Mux Control 7 This field controls the configuration for GPIO pin 7.
27:24	PMC6	RW	-	Port Mux Control 6 This field controls the configuration for GPIO pin 6.
23:20	PMC5	RW	-	Port Mux Control 5 This field controls the configuration for GPIO pin 5.
19:16	PMC4	RW	-	Port Mux Control 4 This field controls the configuration for GPIO pin 4.

Serial Communication (UART)

- UART Module Configuration Registers.
  - UART Control Register ((UARTCTL)) (P.918)

```
#define UART5_CTL_R (*((volatile uint32_t *)0x40011030))
```

9	RXE	RW	1	UART Receive Enable

Value	Description
0	The receive section of the UART is disabled.
1	The receive section of the UART is enabled.

If the UART is disabled in the middle of a receive, it completes the current character before stopping.

Note: To enable reception, the `UARTEN` bit must also be set.

8	TXE	RW	1	UART Transmit Enable

Value	Description
0	The transmit section of the UART is disabled.
1	The transmit section of the UART is enabled.

If the UART is disabled in the middle of a transmission, it completes the current character before stopping.

Note: To enable transmission, the `UARTEN` bit must also be set.

0	UARTEN	RW	0	UART Enable

Value	Description
0	The UART is disabled.
1	The UART is enabled.

If the UART is disabled in the middle of transmission or reception, it completes the current character before stopping.

Serial Communication (UART)

- UART Module Configuration Registers.
  - Baud Rate Configuration Register (P.896):
    UART Integer Baud-Rate Divisor (UARTIBRD) (P.914)
    UART Fractional Baud-Rate Divisor (UARTFBRD) (P.915)

$$\text{BDR} = \text{BDRI} + \text{BDFR} = \frac{\text{UARTSysClk}}{\text{ClkDiv} * \text{Baud Rate}}$$

$$\text{UARTFBRD} = \text{integer}(\text{BDFR} * 64 + 0.5)$$

```
#define UART5_IBRD_R (*((volatile uint32_t *)0x40011024))
#define UART5_FBRD_R (*((volatile uint32_t *)0x40011028))
```

Serial Communication (UART)

- UART Module Configuration Registers.
  - UART Clock Configuration (UARTCC)(P.938):

```
#define UART5_CC_R  (*((volatile uint32_t *)0x40011FC8))
```

UART Clock Configuration (UARTCC)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
UART2 base: 0x4000.E000
UART3 base: 0x4000.F000
UART4 base: 0x4001.0000
UART5 base: 0x4001.1000
UART6 base: 0x4001.2000
UART7 base: 0x4001.3000
Offset 0xFC8
Type RW, reset 0x0000.0000

Bit	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	reserved												CS			
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit/Field	Name	Type	Reset	Description
31:4	reserved	RO	0x0000.000	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
3:0	CS	RW	0	UART Baud Clock Source The following table specifies the source that generates for the UART baud clock:

Value	Description
0x0	System clock (based on clock source and divisor factor)
0x1-0x4	reserved
0x5	PIOSC
0x5-0xF	Reserved

Serial Communication (UART)

- UART Module Configuration Registers.
  - UART Line Control (UARTLCRH) (P.915):

```
#define UART5_LCRH_R (*((volatile uint32_t *)0x4001102C))
```

UART Line Control (UARTLCRH)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
UART2 base: 0x4000.E000
UART3 base: 0x4000.F000
UART4 base: 0x4001.0000
UART5 base: 0x4001.1000
UART6 base: 0x4001.2000
UART7 base: 0x4001.3000
Offset 0x02C
Type RW, reset 0x0000.0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
reserved															
Type	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO	RO
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved								SPS	WLEN		FEN	STP2	EPS	PEN	BRK
Type	RO	RO	RO	RO	RO	RO	RO	RO	RW	RW	RW	RW	RW	RW	RW	RW
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Bit/Field	Name	Type	Reset	Description
31:8	reserved	RO	0x0000.00	Software should not rely on the value of a reserved bit. To provide compatibility with future products, the value of a reserved bit should be preserved across a read-modify-write operation.
7	SPS	RW	0	UART Stick Parity Select When bits 1, 2, and 7 of **UARTLCRH** are set, the parity bit is transmitted and checked as a 0. When bits 1 and 7 are set and 2 is cleared, the parity bit is transmitted and checked as a 1. When this bit is cleared, stick parity is disabled.
6:5	WLEN	RW	0x0	UART Word Length The bits indicate the number of data bits transmitted or received in a frame as follows: Value Description 0x0 5 bits (default) 0x1 6 bits 0x2 7 bits 0x3 8 bits

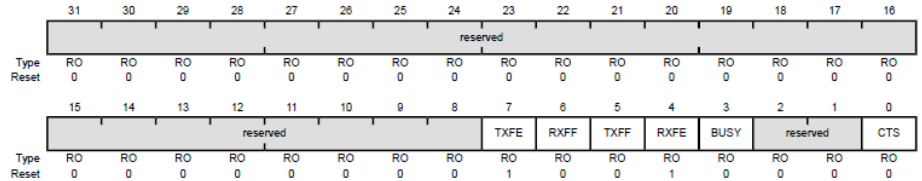
Serial Communication (UART)

- UART Module Configuration Registers.
  - UART Flag (UARTFR) (P.911)
  - UART Data (UARTDR) (P.906)

```
#define UART5_FR_R      (*((volatile uint32_t *)0x40011018))
#define UART5_DR_R      (*((volatile uint32_t *)0x40011000))
```

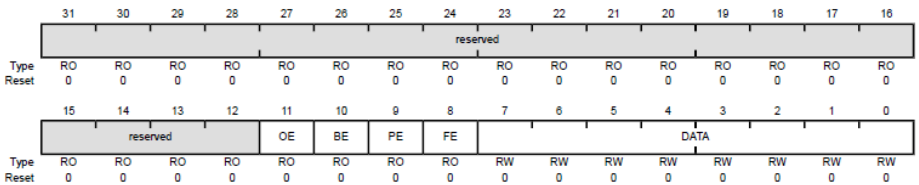
UART Flag (UARTFR)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
UART2 base: 0x4000.E000
UART3 base: 0x4000.F000
UART4 base: 0x4001.0000
UART5 base: 0x4001.1000
UART6 base: 0x4001.2000
UART7 base: 0x4001.3000
Offset 0x018
Type RO, reset 0x0000.0090



UART Data (UARTDR)

UART0 base: 0x4000.C000
UART1 base: 0x4000.D000
UART2 base: 0x4000.E000
UART3 base: 0x4000.F000
UART4 base: 0x4001.0000
UART5 base: 0x4001.1000
UART6 base: 0x4001.2000
UART7 base: 0x4001.3000
Offset 0x000
Type RW, reset 0x0000.0000



Serial Communication (UART)

```
#include <stdint.h>
#include <stdlib.h>
#include"tm4c123gh6pm.h"

void Delay(unsigned long counter);
char UART5_Receiver(void);
void UART5_Transmitter(unsigned char data);
void printstring(char *str);

int main(void)
{
 SYSCTL_RCGCUART_R |= 0x20;/* enable clock to UART5 */
 SYSCTL_RCGCGPIO_R |= 0x10;/* En Clk to PORTE for PE4/Rx - PE5/Tx*/
 Delay(1);
 /* UART5 TX5 and RX5 use PE4 and PE5. Configure them digital and
     enable alternate function */
 GPIO_PORTE_DEN_R   = 0x30;         /* set PE4 and PE5 as digital */
 GPIO_PORTE_AFSEL_R = 0x30;        /* Use PE4,PE5 alternate function
     */
 GPIO_PORTE_AMSEL_R = 0;           /* Turn off analog function */
 GPIO_PORTE_PCTL_R  = 0x00110000; /* Cong. PE4 - PE5 for UART, P.1352
      */
 /* 0x00330000  for I2C, P.1352  and  0x00440000  for PWM, P.1352 */
```

Serial Communication (UART)

```
  /* UART5 initialization */
  UART5_CTL_R = 0;      /* UART5 module disbable */
  UART5_IBRD_R = 104;   /* for 9600 baud rate, integer = 104 */
  UART5_FBRD_R = 11;    /* for 9600 baud rate, fractional = 11*/
  UART5_CC_R = 0;       /*select system clock*/
  UART5_LCRH_R = 0x60; /*data lenght 8-bit, not parity bit, no FIFO*/
  UART5_CTL_R = 0x301; /* Enable UART5 module, Rx and Tx */

  Delay(1);

  printstring("UART Testing Example");
  printstring("\n\r");

  while(1)
  {
   char c = UART5_Receiver();          /* get a character from UART5
       */
   Delay(160000);
   UART5_Transmitter(c);
   printstring("\n\r");
  }
 }
```

Serial Communication (UART)

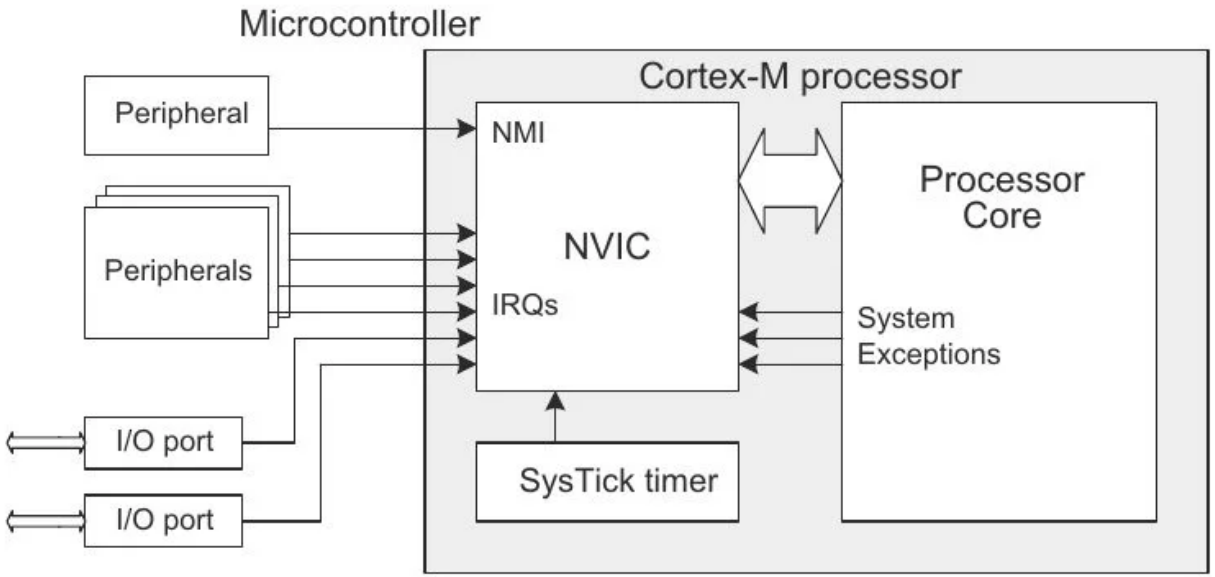
```
char UART5_Receiver(void)
{
 while((UART5_FR_R & (1<<4))!= 0);/* wait until Rx buffer is not full
      */
 return (unsigned char)(UART5_DR_R & 0xFF);/* before giving it
     another byte */
}
void UART5_Transmitter(unsigned char data)
{
 while((UART5_FR_R & (1<<5)) != 0);/*wait until Tx buffer not full */
 UART5_DR_R = data;                  /* before giving it another byte */
}
void printstring(char *str)
{
 while(*str)
 { UART5_Transmitter(*(str++)); }
}
void Delay(unsigned long counter)
{
 unsigned long i = 0;
 for(i=0; i< counter; i++);
}
```

Interrupt Driven Event in Embedded Systems

- **Polling and interrupt driven event** are two methods used to handle events or signals from peripherals (like sensors, timers, or communication modules).
- Interrupts allow **efficient CPU utilization**, but in pooling the CPU would need to continuously poll (check) the status of peripherals or events, which wastes processing power.
- **Real-Time Responsiveness**: Interrupts allow the CPU to respond to events immediately, rather than waiting for the main program to detect them.
- **Prioritization**: More critical events can preempt less critical ones.
- **Concurrency**: Interrupts allow multiple tasks to be handled concurrently.

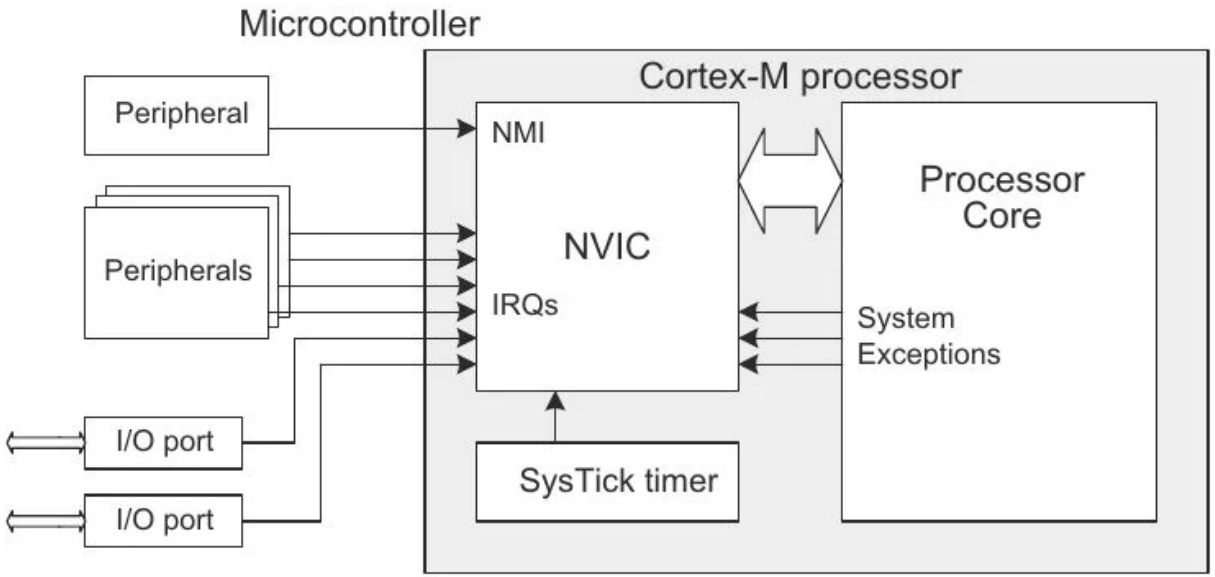
Nested Vectored Interrupt Controller (NVIC)

- **NVIC** is an on-chip controller used to handle all the exceptions and interrupts (interrupt-driven events) in ARM Cortex-M MCUs.
- ARM Cortex-M architecture supports a total of 256 interrupts:
  - System Exceptions (Interrupts 0-15): Reset - Usage Fault.
  - User Interrupts (Interrupts 16-255): GPIOs - ADCs -UARTs
  - Programmable priority level for each IRQ: $0 --> $ highest priority



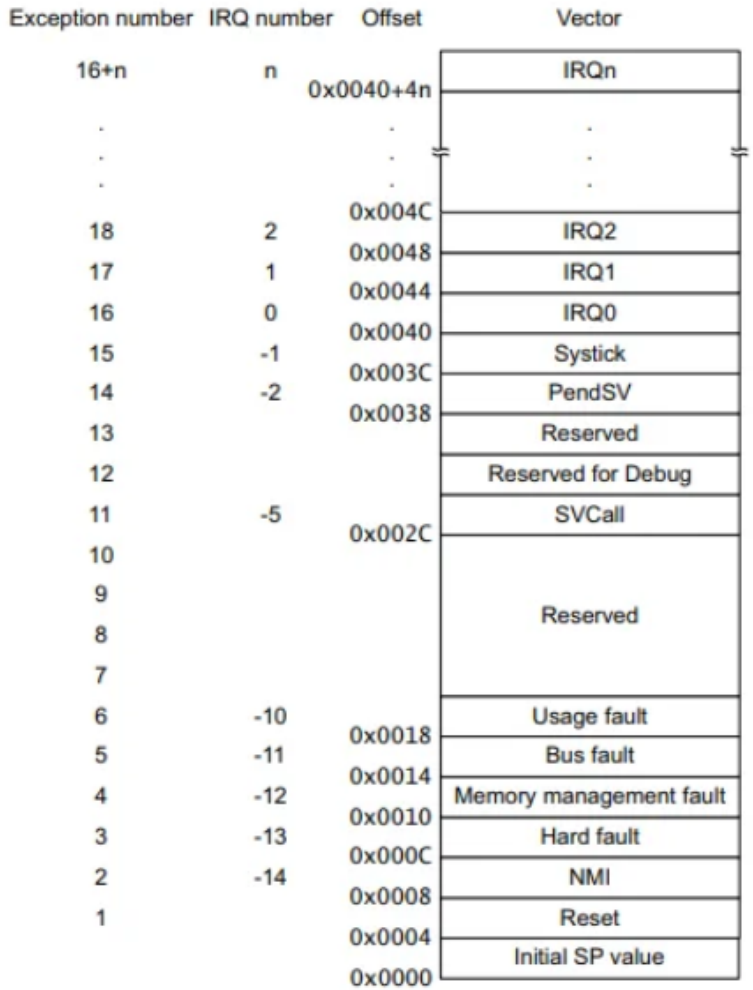
Nested Vectored Interrupt Controller (NVIC)

- SysTick timer in NVIC is a 24-bit down-counter that provides a simple way to generate periodic interrupts at fixed intervals.
- SysTick can use the system clock (CPU clock).
- Used for creating system ticks in real-time operating systems (RTOS), timing delays, or task scheduling.



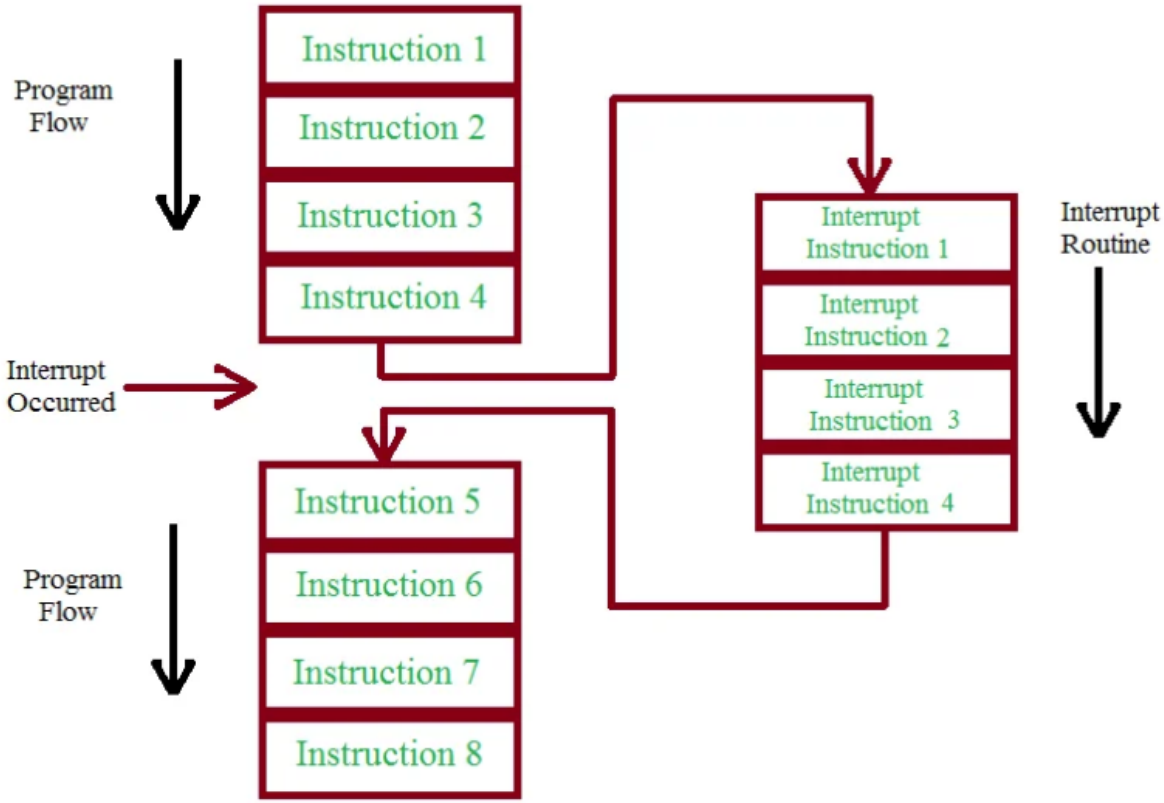
Nested Vectored Interrupt Controller (NVIC)

- Interrupt Vector Table (IVT): As interrupt x occurs, **IRQ** will be sent to NVIC. Next, NVIC find the starting address of the ISR inside IVT.
- Then NVIC uses the content of that memory address to execute the exception handler.
- CPU starts to execute the exception routine.
- IVT configuration: In embedded development using CMSIS (Cortex MCU Software Interface Standard), the vector table is usually defined in a **startup file** or automatically handled by the development environment.
- Check (tm4c123gh6pm_startup_ccs.c) file in your project.



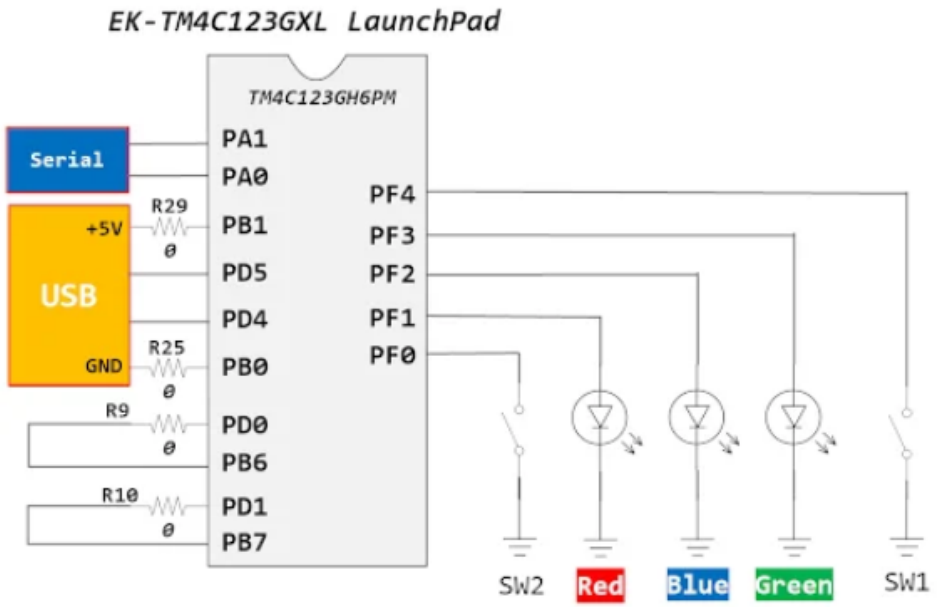
Nested Vectored Interrupt Controller (NVIC)

- Interrupts (system exception or peripheral interrupts) cause the program flow to jump to a different position.
- Simple Interrupt Processing $->$ Interrupt Service Routine (ISR) (interrupt handler))



GPIO Interrupts: External Interrupts

- Use GPIO interrupts which are also known as external interrupts.
- Configure GPIO interrupts as an edge triggered such as rising or falling edge or level triggered such as active high or active low level triggered.
- Use of external GPIO interrupts makes embedded system event driven, responsive and they make use of MCU's processing time and resources efficiently



GPIO Interrupts: External Interrupts

- How to Configure GPIO Interrupts:
  - Find GPO interrupt number: Each interrupt has a unique number assigned to it. This interrupt number is defined inside the startup file (P.104-table 2.9).

Table 2-9. Interrupts *(continued)*

Vector Number	Interrupt Number (Bit in Interrupt Registers)	Vector Address or Offset	Description
45	29	0x0000.00B4	Flash Memory Control and EEPROM Control
46	30	0x0000.00B8	GPIO Port F
47-48	31-32	-	Reserved
49	33	0x0000.00C4	UART2
50	34	0x0000.00C8	SSI1

GPIO Interrupts: External Interrupts

- How to Configure GPIO Interrupts:
  - Enable GPIO Interrupts from NVIC interrupt enable registers (NVIC_EN0_R) (P.94 P.134 P.142): First, enable the source of interrupt requests for this particular interrupt source using the NVIC interrupt control register (NVIC interrupt enable registers), then enable peripheral interrupt.

```
#define NVIC_EN0_R  (*((volatile uint32_t *)0xE000E100))
NVIC_EN0_R    |= (1<<30); /* En. IRQ30 (D30 of ISER[0]: PORTF
    Interrupt)*/
```

  - Bit0 of NVIC_EN0_R corresponds to Interrupt 0; bit31 corresponds to Interrupt 31. Bit 0 of NVIC_EN1_R corresponds to Interrupt 32; bit31 corresponds to Interrupt 63.
  - After enabling interrupt in NVIC register, enable the interrupt of the peripheral which you want to use (GPIOIM).

GPIO Interrupts: External Interrupts

- How to Configure GPIO Interrupts:
  - Enable the interrupt of the peripheral which you want to use.
  - In order to enable GPIO interrupt GPIO interrupt mask enable register is used (GPIOIM) (P.667).
  - First eight bits of GPIOIM register enable or disable interrupt functionality for each pin.

```
#define GPIO_PORTF_IM_R (*((volatile uint32_t *)0x40025410))
GPIO_PORTF_IM_R  |= (1<<4)|(1<<0);  /* unmask interrupt */
```

  - GPIO Interrupt Sense (GPIOIS) Interrupt edge or level triggered setting (P.664)

```
#define GPIO_PORTF_IS_R (*((volatile uint32_t *)0x40025404))
GPIO_PORTF_IS_R  &= ~(1<<4)|~(1<<0);/* make pins 4, 0 edge
    sensitive */
```

GPIO Interrupts: External Interrupts

- How to Configure GPIO Interrupts:
  - GPIO Interrupt Even Register (GPIOIEV): After configure GPIOIS the GPIOIEV select either positive/negative edge or positive/negative level triggered (P.666)

```
#define GPIO_PORTF_IEV_R (*((volatile uint32_t *)0x4002540C))
GPIO_PORTF_IEV_R &= ~(1<<4)|~(1<<0); /* falling edge trigger
    */
```

  - GPIO Interrupt Both Edges (GPIOIBE):Can configure GPIO pins to cause an interrupt on both edges or detect both rising and falling edges (P. 665).
  - If GPIOIBE disabled, GPIOIEV control interupt generation, otherwise GPIOIBE work regardless GPIOIEV.

```
#define GPIO_PORTF_IBE_R (*((volatile uint32_t *)0x40025408))
GPIO_PORTF_IBE_R &=~(1<<4)|~(1<<0);/*trigger is controlled by
    IEV*/
```

  - GPIO Masked Interrupt Status (GPIOMIS): The masked interrupt status register (indicate interrupt triggered or not).

GPIO Interrupts: External Interrupts

- How to Configure GPIO Interrupts:
  - GPIO Interrupt Handler Name: In the startup file (tm4c123gh6pm_startup_ccs.c), there is a vectored mapped table which contains the starting addresses of all system exceptions and peripheral interrupt service routines.

GPIO Interrupts: External Interrupts

- Inside the PORTF interrupt handler function, we will turn on the onboard green LED of TM4C123 Tiva C launchpad, if SW1 is pressed and turns off the LED if SW2 is pressed.

```
#include <stdint.h>
#include"tm4c123gh6pm.h"

void GPIOF_Handler(void);

int main(void)
{
 SYSCTL_RCGCGPIO_R |= (1<<5);/* Set bit5 of RCGCGPIO to En. Clk to
      PORTF*/

 /* PORTF0 has special function, need to unlock to modify */
 GPIO_PORTF_LOCK_R = 0x4C4F434B;   /* unlock commit register */
 GPIO_PORTF_CR_R = 0x01;           /* make PORTF0 configurable */
 GPIO_PORTF_LOCK_R = 0;            /* lock commit register */
```

GPIO Interrupts: External Interrupts

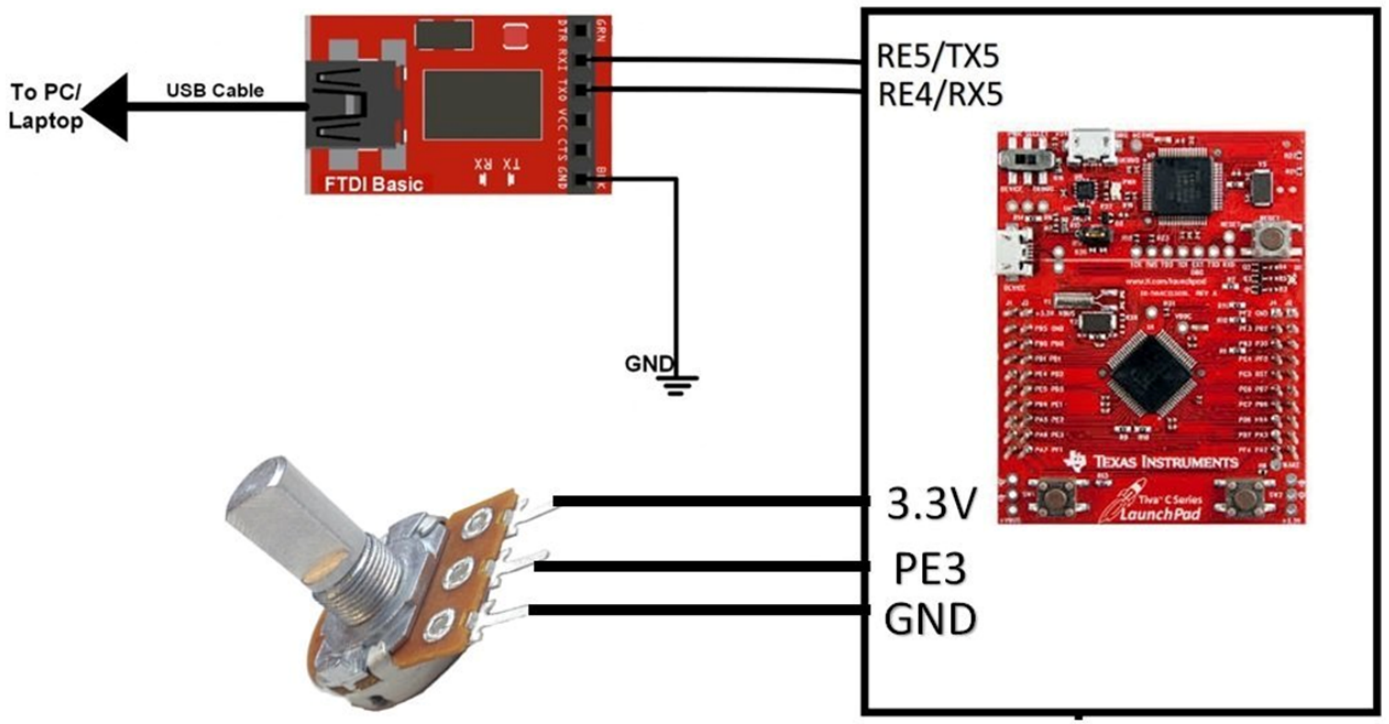
```
/*Initialize PF3 as a digital output, PF0 and PF4 as digital input
    pins*/
GPIO_PORTF_DIR_R &= ~(1<<4)|~(1<<0);/*Set PF4 and PF0 as a digital
    input pins*/
GPIO_PORTF_DIR_R |= (1<<3);/*Set PF3 as digital output to control
    green LED*/
GPIO_PORTF_DEN_R |= (1<<4)|(1<<3)|(1<<0);/*make PORTF4-0 digital pin
    */
GPIO_PORTF_PUR_R |= (1<<4)|(1<<0);/*enable pull up for PORTF4,0*/

/* configure PORTF4, 0 for falling edge trigger interrupt*/
GPIO_PORTF_IS_R  &= ~(1<<4)|~(1<<0);/* make bit 4, 0 edge sensitive */
GPIO_PORTF_IBE_R &=~(1<<4)|~(1<<0); /* trigger is controlled by IEV */
GPIO_PORTF_IEV_R &= ~(1<<4)|~(1<<0);/* falling edge trigger */
GPIO_PORTF_ICR_R |= (1<<4)|(1<<0);  /* clear any prior interrupt */
GPIO_PORTF_IM_R  |= (1<<4)|(1<<0);  /* unmask interrupt */

/* enable interrupt in NVIC and set priority to 3*/
NVIC_PRI30_R = 3 << 5;      /* set interrupt priority to 3*/
NVIC_EN0_R |= (1<<30); /*enable IRQ30 (D30 of ISER[0]) P.94 P. 134*/
while(1)
{// do nothing and wait for the interrupt to occur}
}
```

GPIO Interrupts: External Interrupts

```
/* SW1 is connected to PF4 pin, SW2 is connected to PF0. */
/* Both of them trigger PORTF falling edge interrupt */
void GPIOF_Handler(void)
{
 if (GPIO_PORTF_MIS_R & 0x10) /* check if interrupt causes by PF4/SW1
     */
 {
  GPIO_PORTF_DATA_R |= (1<<3);
  GPIO_PORTF_ICR_R |= 0x10;  /* clear the interrupt flag */
 }
 else if (GPIO_PORTF_MIS_R & 0x01) /* check if interrupt causes by PF0
     /SW2 */
 {
  GPIO_PORTF_DATA_R &= ~0x08;
  GPIO_PORTF_ICR_R |= 0x01; /* clear the interrupt flag */
 }
}
```

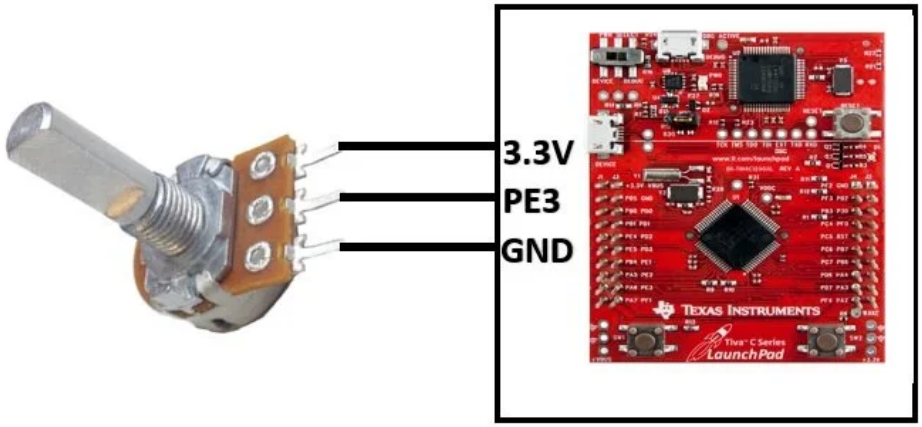
Project

- Part 1: Measure analog voltage signal by using one of the analog input pins through GPIO. Use LED through GPIO to test your measurement.
- Part 2: Send the measured voltage tp PC through UART.



ADC Module to Measure Potentiometer Values

- Measure analog voltage signal by using one of the analog input pins through GPIO.



Registers

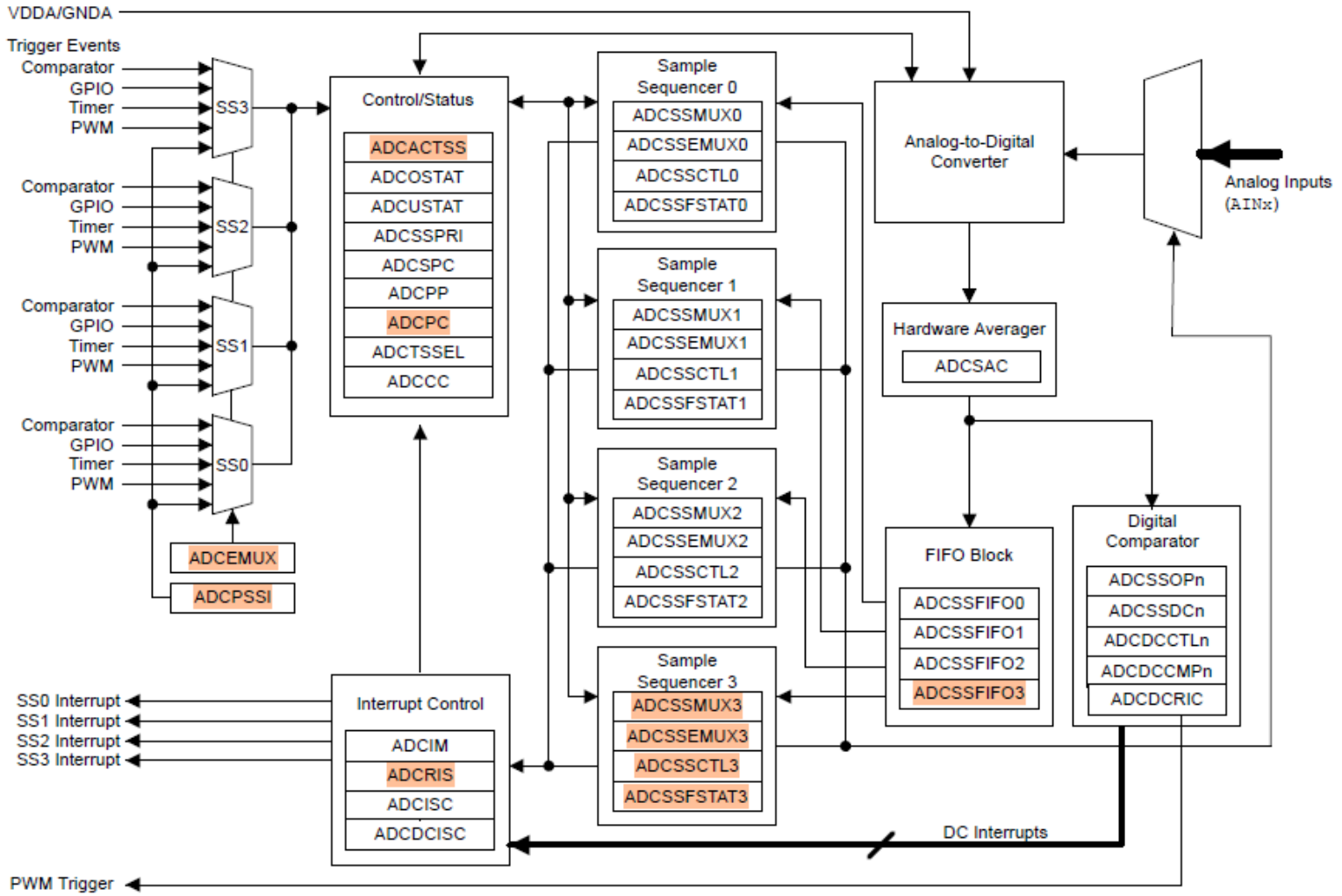
Name	Value	Description
GPIO_PORTF		GPIO register offsets
GPIO_DATA	0x00000008	GPIO Data [Memory Mapped]
GPIO_DIR	0x00000008	GPIO Direction [Memory Mapped]
GPIO_IS	0x00000000	GPIO Interrupt Sense [Memory Mapped]
GPIO_IBE	0x00000000	GPIO Interrupt Both Edges [Memory Mapped]
GPIO_IEV	0x00000000	GPIO Interrupt Event [Memory Mapped]
GPIO_IM	0x00000000	GPIO Interrupt Mask [Memory Mapped]
GPIO_RIS	0x00000000	GPIO Raw Interrupt Status [Memory Mapped]
GPIO_MIS	0x00000000	GPIO Masked Interrupt Status [Memory Mapped]
GPIO_ICR	0x00000000	GPIO Interrupt Clear [Memory Mapped]
GPIO_AFSEL	0x00000000	GPIO Alternate Function Select [Memory Mapped]
GPIO_DR2R	0x000000FF	GPIO 2-mA Drive Select [Memory Mapped]
GPIO_DR4R	0x00000000	GPIO 4-mA Drive Select [Memory Mapped]
GPIO_DR8R	0x00000000	GPIO 8-mA Drive Select [Memory Mapped]
GPIO_ODR	0x00000000	GPIO Open Drain Select [Memory Mapped]
GPIO_PUR	0x00000000	GPIO Pull-Up Select [Memory Mapped]
GPIO_PDR	0x00000000	GPIO Pull-Down Select [Memory Mapped]
GPIO_SLR	0x00000000	GPIO Slew Rate Control Select [Memory Mapped]
GPIO_DEN	0x00000008	GPIO Digital Enable [Memory Mapped]
GPIO_LOCK	0x00000001	GPIO Lock [Memory Mapped]
GPIO_LOCK	0x00000001 (Hex) -	GPIO Lock
GPIO_CR	0x000000FE	GPIO Commit [Memory Mapped]
GPIO_AMSEL	0x00000000	GPIO Analog Mode Select [Memory Mapped]
GPIO_PCTL	0x00000000	GPIO Port Control [Memory Mapped]
GPIO_ADCCTL	0x00000000	GPIO ADC Control [Memory Mapped]
GPIO_DMACTL	0x00000000	GPIO DMA Control [Memory Mapped]

Registers

Name	Value	Description
ADC0		ADC register offsets
ADC_ACTSS	0x00000008	ADC Active Sample Sequencer [Memory Mapped]
ADC_RIS	0x00000000	ADC Raw Interrupt Status [Memory Mapped]
ADC_IM	0x00000000	ADC Interrupt Mask [Memory Mapped]
ADC_ISC	0x00000000	ADC Interrupt Status and Clear [Memory Mapped]
ADC_OSTAT	0x00000000	ADC Overflow Status [Memory Mapped]
ADC_EMUX	0x00000000	ADC Event Multiplexer Select [Memory Mapped]
ADC_USTAT	0x00000000	ADC Underflow Status [Memory Mapped]
ADC_TSSEL	0x00000000	ADC Trigger Source Select [Memory Mapped]
ADC_SSPRI	0x00003210	ADC Sample Sequencer Priority [Memory Mapped]
ADC_SPC	0x00000000	ADC Sample Phase Control [Memory Mapped]
ADC_PSSI	0x00000000	ADC Processor Sample Sequence Initiate [Memory Mapped]
ADC_SSMUX3	0x00000000	ADC Sample Sequence Input Multiplexer Select 3 [Memory Mapped]
ADC_SSCTL3	0x00000006	ADC Sample Sequence Control 3 [Memory Mapped]
ADC_SSFIFO3	0x00000FFF	ADC Sample Sequence Result FIFO 3 [Memory Mapped]
ADC_PC	0x00000003	ADC Peripheral Configuration [Memory Mapped]

ADC Module to Measure Potentiometer Values

Figure 13-2. ADC Module Block Diagram



Open P.802

ADC Module to Measure Potentiometer Values

- ADC using Potentiometer Code:

```
#include <stdint.h>
#include"tm4c123gh6pm.h"
int main(void)
{
 unsigned int adc_value;
 /* Enable Clock to ADC0 and GPIO pins*/
 SYSCTL_RCGCGPIO_R = SYSCTL_RCGCGPIO_R4; /* Enable Clock to GPIOE or
      PE3/AN0 */
 SYSCTL_RCGCADC_R = SYSCTL_RCGCADC_R0;  /* AD0 clock enable*/
 /* initialize PE3 for AIN0 input*/
 GPIO_PORTE_AFSEL_R |= (1<<3); /* enable alternate function */
 GPIO_PORTE_DEN_R &= ~(1<<3);  /* disable digital function */
 GPIO_PORTE_AMSEL_R |= (1<<3); /* enable analog function */
 /* initialize sample sequencer3*/
 ADC0_ACTSS_R &= ~(1<<3);/* disable SS3 during configuration */
 ADC0_EMUX_R &= ~0xF000;    /* software trigger conversion */
 ADC0_SSMUX3_R = 0;         /* get input from channel 0 */
 ADC0_SSCTL3_R |= (1<<1)|(1<<2);        /* take one sample at a time
     , set flag at 1st sample */
 ADC0_ACTSS_R  |= (1<<3);          /* enable ADC0 sequencer 3 */
```

ADC Module to Measure Potentiometer Values

- ADC using Potentiometer Code:

```
/*Iniitialize PF3 as a digital output pin*/
SYSCTL_RCGCGPIO_R = 0x20; // turn on bus clock for GPIOF
GPIO_PORTF_DIR_R |= 0x08; //set GREEN pin as a digital output pin
GPIO_PORTF_DEN_R |= 0x08;  // Enable PF3 pin as a digital pin
while(1)
{
 ADC0_PC_R = 0x3;                /*sampling rate = 250ksps*/
 ADC0_PSSI_R |= (1<<3);          /* Enable SS3 conversion or start
     sampling data from AN0 */
 while((ADC0_RIS_R & 8) == 0) ;   /* Wait untill sample conversion
     completed*/
 adc_value = ADC0_SSFIFO3_R; /* read adc coversion result from SS3
     FIFO*/
 ADC0_ISC_R = 8;             /* clear coversion clear flag bit*/
 /*control Green PF3->LED */
 if(adc_value >= 2048)
 GPIO_PORTF_DATA_R  = 0x08; /* turn on green LED*/
 else if(adc_value < 2048)
 GPIO_PORTF_DATA_R  = 0x00; /* turn off green LED*/
 }
}
```

Integrate ADC and UART Example

```
#include <stdint.h>
#include <stdlib.h>
#include <stdio.h>
#include"tm4c123gh6pm.h"

void Delay(unsigned long counter);
void UART5_init(void);
void UART5_Transmitter(unsigned char data);
void printstring(char *str);
char mesg[1000];
float voltage;

int main(void)
{
 unsigned int adc_value;
 UART5_init();

 /* Enable Clock to ADC0 and GPIO pins*/
 SYSCTL_RCGCGPIO_R = SYSCTL_RCGCGPIO_R4;/*En Clk to GPIOE or PE3/AN0
      */
 SYSCTL_RCGCADC_R = SYSCTL_RCGCADC_R0;  /*AD0 Clk An*/
```

Integrate ADC and UART Example

```
/* initialize PE3 for AIN0 input*/
GPIO_PORTE_AFSEL_R |= (1<<3); /* enable alternate function */
GPIO_PORTE_DEN_R &= ~(1<<3);  /* disable digital function */
GPIO_PORTE_AMSEL_R |= (1<<3); /* enable analog function */

/* initialize sample sequencer3*/
ADC0_ACTSS_R &= ~(1<<3);/* disable SS3 during configuration */
ADC0_EMUX_R &= ~0xF000;     /* software trigger conversion */
ADC0_SSMUX3_R = 0;          /* get input from channel 0 */
ADC0_SSCTL3_R |= (1<<1)|(1<<2);
/* take one sample at a time, set flag at 1st sample*/
ADC0_ACTSS_R  |= (1<<3);    /* enable ADC0 sequencer 3 */

printstring("\rReading ADR Value through MCU-UART to PC-UART:\n");
printstring("\r=======================================\n");
```

Integrate ADC and UART Example

```
    while(1)
    {
     ADC0_PC_R = 0x3;                    /*sampling rate = 250ksps*/
     ADC0_PSSI_R |= (1<<3);/* Enable SS3 conversion or start sampling
         data from AN0 */
     while((ADC0_RIS_R & 8) == 0) ;/*Wait untill sample conversion
         completed*/
     adc_value = ADC0_SSFIFO3_R; /* read adc coversion result from SS3
         FIFO*/
     ADC0_ISC_R = 8;              /* clear coversion clear flag bit*/

     /* convert digital value back into voltage */
     voltage = (adc_value * 0.0008);

     sprintf(mesg, "\r\nVoltage = %0.3f Volts", voltage);
     printstring(mesg);
     Delay(1000);
    }
   }
```

Integrate ADC and UART Example

```
void UART5_init(void)
{
 SYSCTL_RCGCUART_R |= 0x20;/*En Clk to UART5*/
 SYSCTL_RCGCGPIO_R |= 0x10;/*En Clk to PORTE for PE4/Rx and PE5/Tx*/
 Delay(1);
 /* UART5 TX5 and RX5 use PE4 and PE5. Configure them digital and
     enable alternate function */
 GPIO_PORTE_DEN_R   = 0x30;      /* set PE4 and PE5 as digital */
 GPIO_PORTE_AFSEL_R = 0x30;  /* Use PE4,PE5 alternate function */
 GPIO_PORTE_AMSEL_R = 0;     /* Turn off analog function */
 GPIO_PORTE_PCTL_R  = 0x00110000;/*Conf. PE4 & PE5 for UART,P.1352
     */
 /* 0x00330000  for I2C, P.1352 */
 /* 0x00440000  for PWM, P.1352 */
 /* UART5 initialization */
 UART5_CTL_R = 0;          /* UART5 module disbable */
 UART5_IBRD_R = 104;       /* for 9600 baud rate, integer = 104 */
 UART5_FBRD_R = 11;        /* for 9600 baud rate, fractional = 11*/
 UART5_CC_R = 0;           /*select system clock*/
 UART5_LCRH_R = 0x60;/* data lenght 8-bit, not parity bit, no FIFO
     */
 UART5_CTL_R = 0x301;      /* Enable UART5 module, Rx and Tx */
}
```

Integrate ADC and UART Example

```
void UART5_Transmitter(unsigned char data)
{
 while((UART5_FR_R & (1<<5)) != 0);/*wait until Tx buffer not full
     */
 UART5_DR_R = data;                  /*before giving it another byte */
}

void printstring(char *str)
{
 while(*str)
 {
  UART5_Transmitter(*(str++));
 }
}

void Delay(unsigned long counter)
{
 unsigned long i = 0;

 for(i=0; i< counter*1000; i++);
}
```


Thank You!