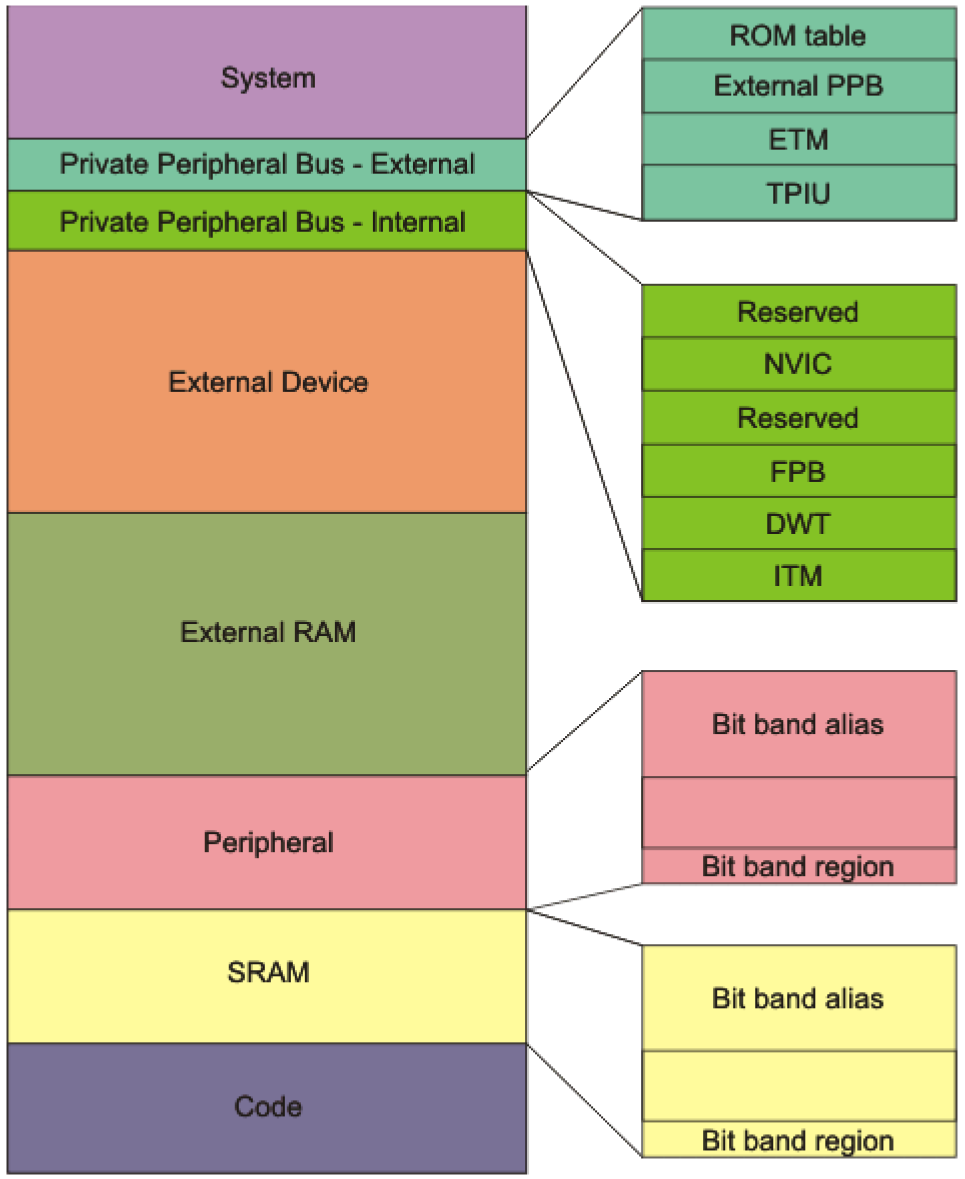
Embedded Applications Development Memory Management and Direct Memory Access (DMA)

Contents

- Understanding memory architecture in micro-controllers.
- Stack and heap memory management.
- Pointers and their usage in embedded C.
- Storing data in Local Memory, Main Memory and Permanent Memory

Microcontroller Memory Architecture

- Flash Memory
  - Stores the program code (firmware) and constant data.
- SRAM (Static RAM)
  - **Program Memory:**
  - Temporary storage used during execution, holding variables, the stack, and the heap.
  - **Data Memory:**
  - Used for dynamic data storage, including global variables, static variables, and temporary data.
- DRAM (Dynamic RAM)
  - Typically used in more complex microcontrollers or embedded systems where larger data storage is needed.
  - Stores variable data, buffers, and other dynamic data structures.



Memory Architecture Overview:

- Harvard Architecture: Many microcontrollers use this architecture, where program and data memories are separate, allowing simultaneous access to both.
- Von Neumann Architecture: Some microcontrollers use this unified memory architecture, where program and data are stored in the same memory space, but this can introduce bottlenecks since program and data fetches compete for the same bus.

Pointers in Embedded C

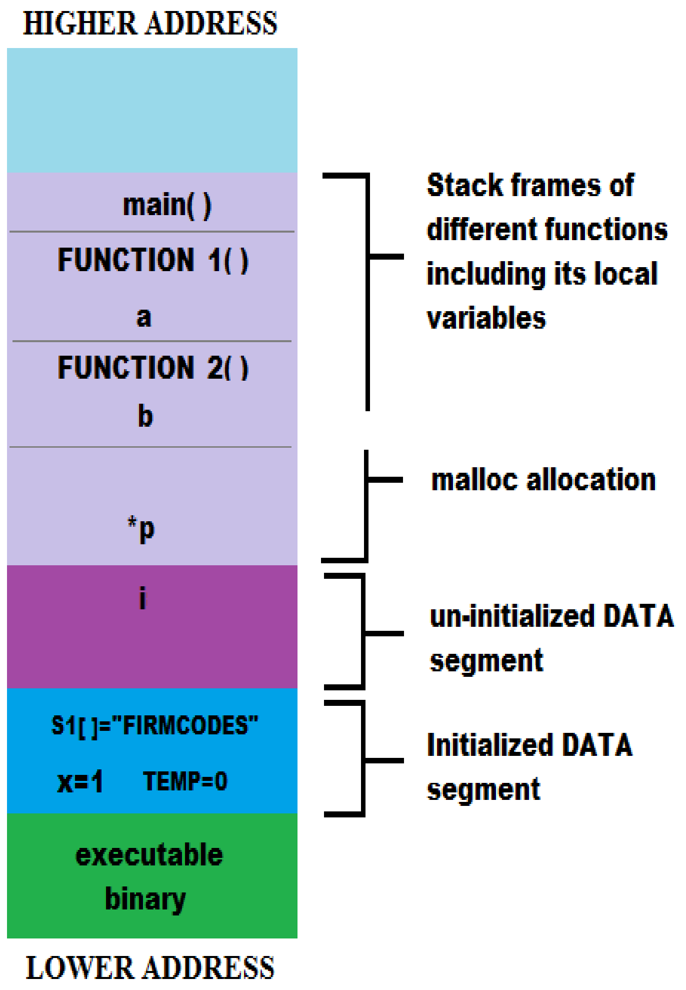
- Pointers in C are variables that store the memory address of another variable. In embedded C, pointers are particularly important because they provide a way to directly access and manipulate hardware registers, memory locations, and peripheral devices.
- Direct Hardware Access: Pointers allow you to interact with specific memory-mapped registers and peripherals.
- Memory Management: They are used for dynamic memory allocation, accessing arrays, and structures efficiently.
- Function Arguments: Pointers can be passed to functions to modify variables or return multiple values.

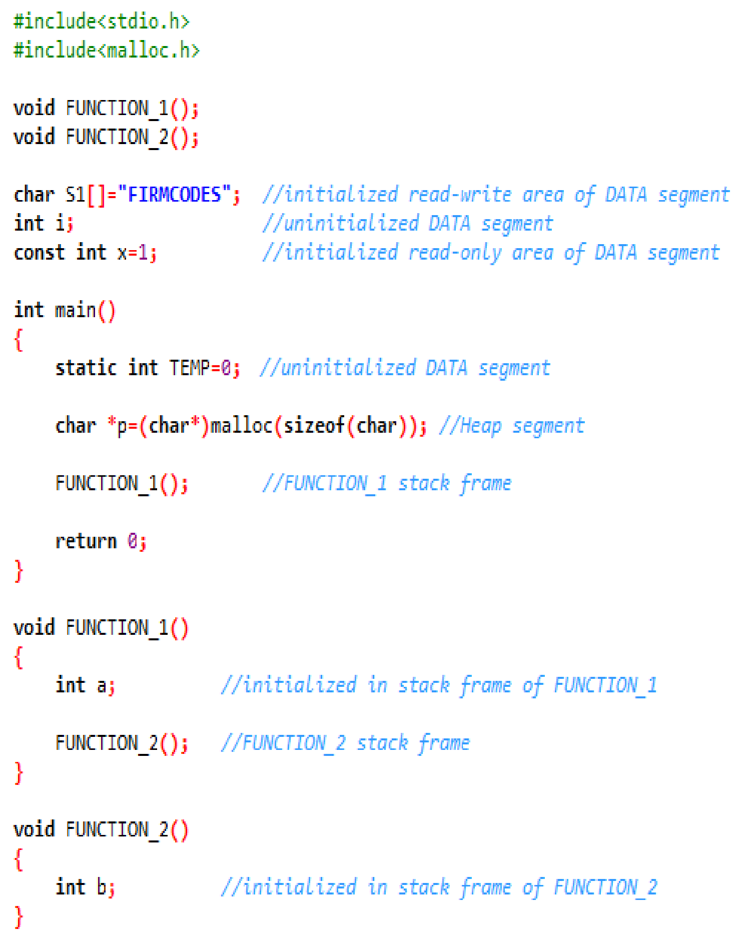
```
int a = 10;
int *ptr = &a;  // ptr is a pointer that stores the address of variable 'a'
// Access the value of 'a' using the pointer
printf("Value of a: %d\n", *ptr);  // Outputs 10
// Modify the value of 'a' using the pointer
*ptr = 20;
printf("New value of a: %d\n", a);  // Outputs 20


int arr[5] = {1, 2, 3, 4, 5};
int *ptr = arr;  // Points to the first element of the array
// Accessing array elements using the pointer
for(int i = 0; i < 5; i++) {
   printf("arr[%d] = %d\n", i, *(ptr + i));
}
```

```
void myFunction(int x) {
    printf("Value is: %d\n", x);
}


int main() {
    void (*funcPtr)(int) = myFunction;  // Declare a function pointer
    funcPtr(10);  // Call the function using the pointer
    return 0;
}


#define GPIO_PORTA_BASE 0x40004000
volatile unsigned int *gpioData = (volatile unsigned int *)(GPIO_PORTA_BASE + 0x3FC);
*gpioData = 0xFF;  // Set all bits of PORTA to 1
```

Program Memory

- Stack and Heap: Typically located in SRAM.
- Global/Static Variables: Located in SRAM.
- Constant Data: Located in Flash.




```
#include<stdio.h>
#include<malloc.h>

void FUNCTION_1();
void FUNCTION_2();

char S1[]="FIRMCODES";  //initialized read-write area of DATA segment
int i;                  //uninitialized DATA segment
const int x=1;          //initialized read-only area of DATA segment

int main()
{
    static int TEMP=0;  //uninitialized DATA segment

    char *p=(char*)malloc(sizeof(char)); //Heap segment

    FUNCTION_1();       //FUNCTION_1 stack frame

    return 0;
}

void FUNCTION_1()
{
    int a;          //initialized in stack frame of FUNCTION_1

    FUNCTION_2();   //FUNCTION_2 stack frame
}

void FUNCTION_2()
{
    int b;          //initialized in stack frame of FUNCTION_2
}
```

Generate Instruction and Data Memory

- riscv32-unknown-elf-gcc -march=rv32i -S -o riscv.s ./code.c
- riscv32-unknown-elf-as -march=rv32i -S -o riscv.o ./riscv.s
- riscv32-unknown-elf-as -march=rv32i -o riscv.o ./riscv.s
- riscv32-unknown-elf-ld -o riscv ./riscv.o
- riscv32-unknown-elf-objcopy -O binary --only-section=.text riscv instr.mem
- riscv32-unknown-elf-objcopy -O binary --only-section=.data riscv data.mem
- riscv32-unknown-elf-objdump -D -b binary -m riscv:rv32i instr.mem

Managing Local Memory (SRAM)

```
#include <stdio.h>
void printArray() {
    // Local array stored in stack memory (SRAM)
    int a[10] = {1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
    for(int i = 0; i < 10; i++) {
        printf("%d ", a[i]);
    }
    printf("\n");
}

int main() {
    printArray();
    return 0;
}
```

Main Memory

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

int main() {
    // Allocate memory for an array of 10 integers in the heap
    int *a = (int *)malloc(10 * sizeof(int));

    // Check if memory allocation was successful
    if (a == NULL) {
        printf("Memory allocation failed\n");
        return 1;
    }

    // Initialize the array with values
    for(int i = 0; i < 10; i++) {
        a[i] = i + 1;
    }

    // Print the array values
    for(int i = 0; i < 10; i++) {
        printf("%d ", a[i]);
    }
    printf("\n");
    // Free the allocated memory
    free(a);
    return 0;
}
```