./

Learning Report – MBSE

Course Code: <CODE>



Version Number:

Team Members :

Team No:

Module: Model Based System Engineering

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Ver. Rel. No.** | **Release Date** | **Prepared. By** | **Reviewed By** | **Approved By** | **Remarks/Revision Details** |
|  | 9-03-2021 | Shiva Kumar Naga Vankadhara |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

**Document History**

# 

# CONTENTS

**Activity1:** Create a MAKEFILE……………………………………………………………………………………………………………………

**Activity2:** Create a Startup File…………………………………………………………………………………………………………………..

**Activity3:** Create a Linker Script………………………………………………………………………………………………………………..

**Activity4:** Debugging Technique……………………………………………………………………………………………………..

**Activity5:** Implementation of GPIO,EXTI,ADC,SPI and UART using STM IDE…………………………………………..

**ACTIVITY1:**

Create a Makeup File:

Make file is a tool to simplify or to organize code for compilation. Make file is a set of commands with variable

names and targets to create object file and to remove them. In a single make file, we can create multiple

targets to compile and to object, binary files.

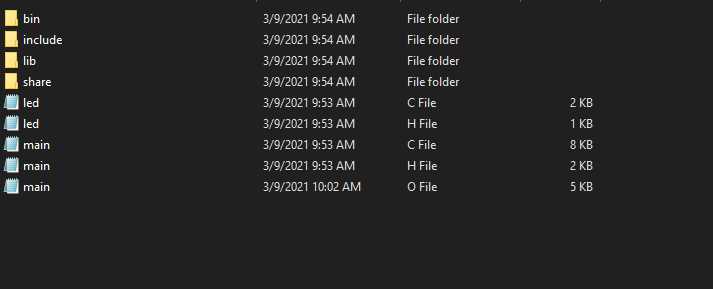


Fig1. Created an Object file

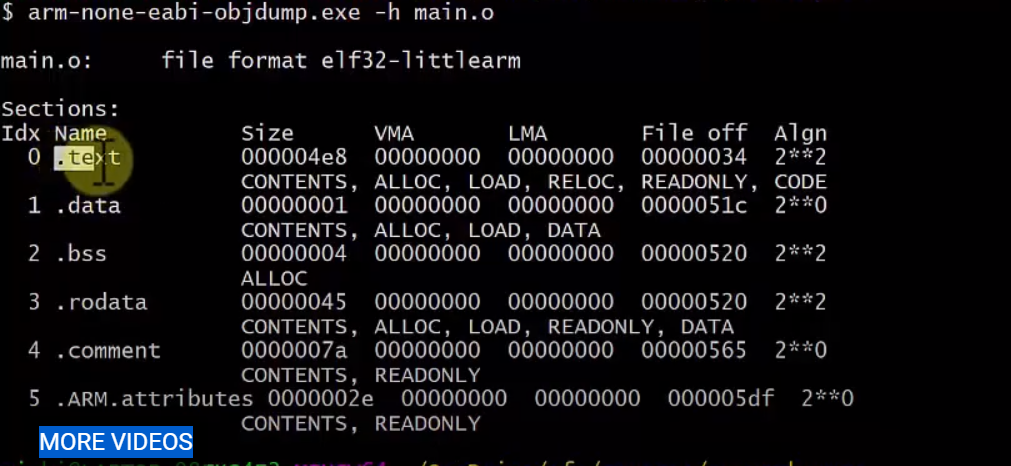


Fig2. Created a Makeup File

**ACTIVITY2:**

Create a Startup File:

The C startup code is used to set up data memory such as global data variables. It also zero initializes part of

the data memory for variables that are uninitialized at load time. After this initialization, the C startup code

branches to the beginning of the main () program.

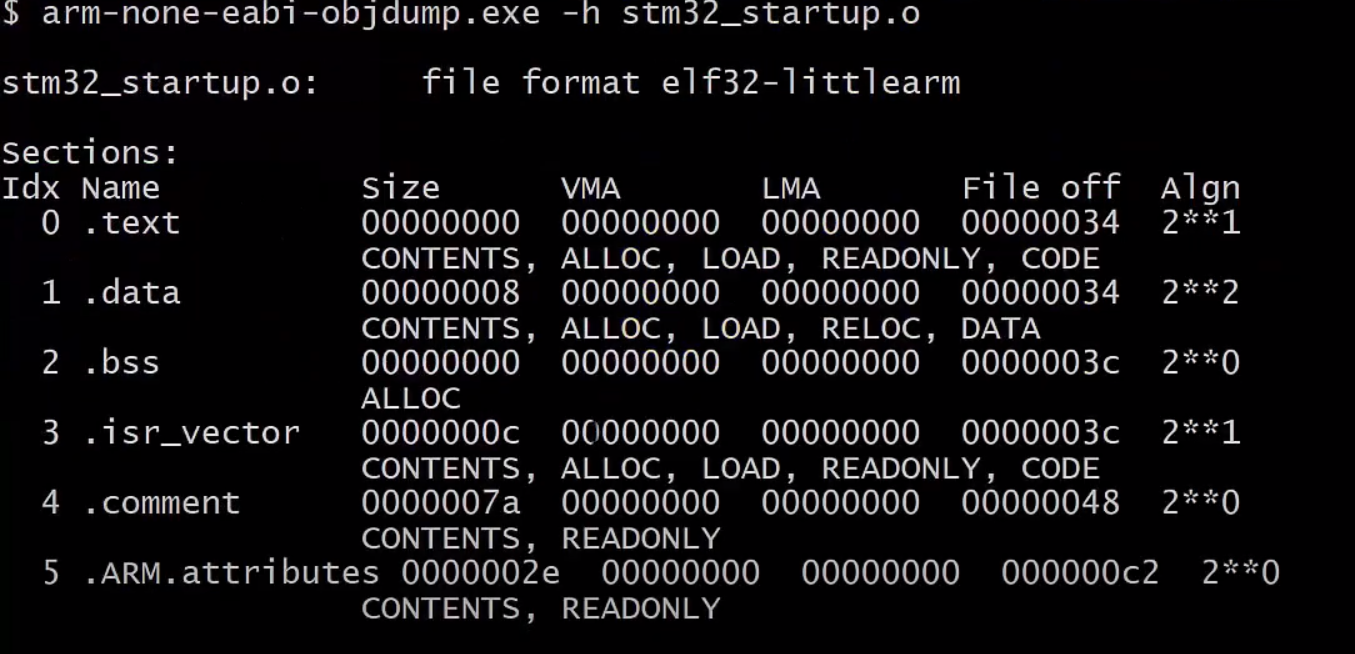


Fig3. Creating a Start Up file

**ACTIVITY3:**

Create a Linker script:

We need to define some basic concepts and vocabulary in order to describe the linker script language. 

The linker combines input files into a single output file. The output file and each input file are in a special data format known as an object file format. Each file is called an object file. The output file is often called an executable, but for our purposes we will also call it an object file. Each object file has, among other things, a list of sections. We sometimes refer to a section in an input file as an input section; similarly, a section in the output file is an output section. Each section in an object file has a name and a size. Most sections also have an associated block of data, known as the section contents. A section may be marked as loadable, meaning that the contents should be loaded into memory when the output file is run. A section with no contents may be allocatable, which means that an area in memory should be set aside, but nothing in particular should be loaded there (in some cases this memory must be zeroed out). A section, which is neither loadable nor allocatable, typically contains some sort of debugging information Every loadable or allocatable output section has two addresses. The first is the VMA, or virtual memory address. This is the address the section will have when the output file is run. The second is the LMA, or load memory address. This is the address at which the section will be loaded. In most cases the two addresses will be the same. An example of when they might be different is when a data section is loaded into ROM, and then copied into RAM when the program starts up (this technique is often used to initialize global variables in a ROM based system). In this case the ROM address would be the LMA, and the RAM address would be the VMA.You can see the sections in an object file by using the ` objdump ' program with the ` -h ' option.

Every object file also has a list of symbols , known as the symbol table . A symbol may be defined or undefined. Each symbol has a name, and each defined symbol has an address, among other information. If you compile a C or C++ program into an object file, you will get a defined symbol for every defined function and global or static variable. Every undefined function or global variable, which is referenced in the input file, will become an undefined symbol. You can see the symbols in an object file by using the ` nm ' program, or by using the ` objdump ' program with the ` -t ' option.

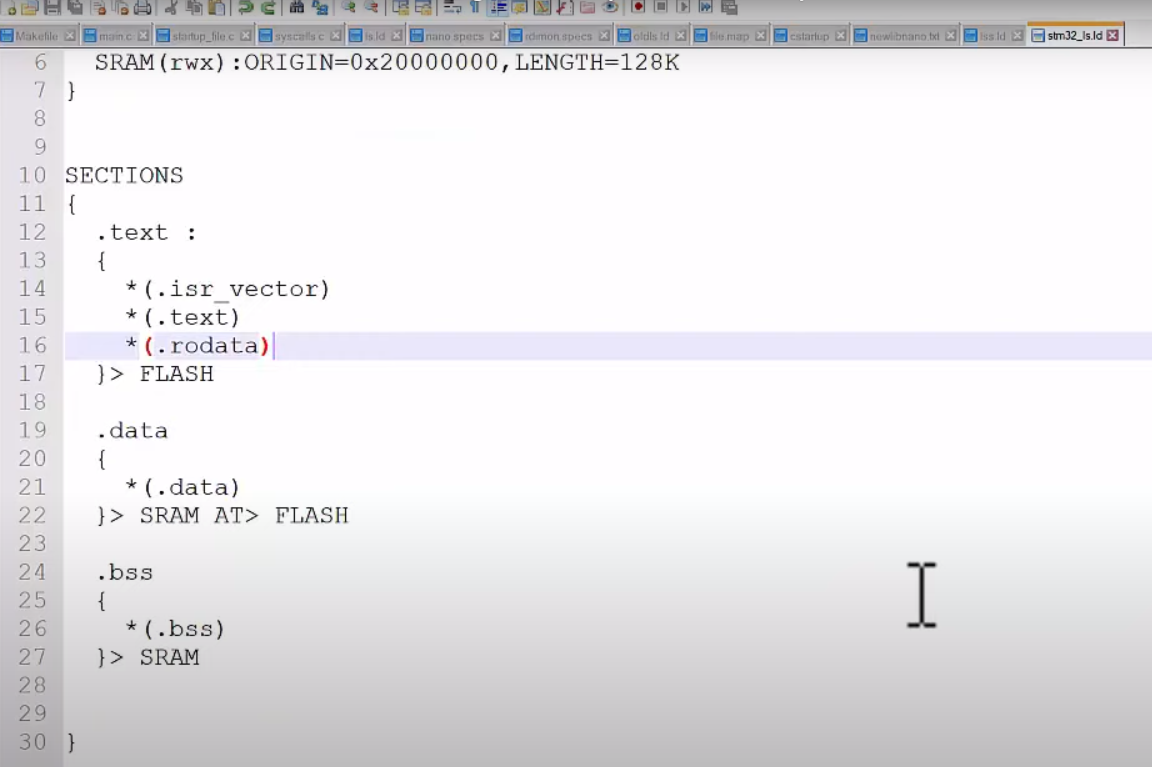


FIG4. Create a Linker Script

**ACTIVITY4:**

Debugging Techniques

**ACTIVITY5:**

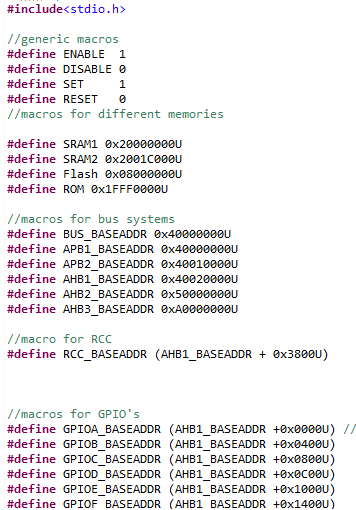
Implementation of GPIO, EXTI, ADC, SPI and UART using STMIDE:

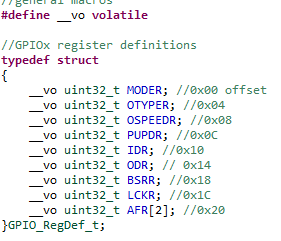
Implementation of GPIO:

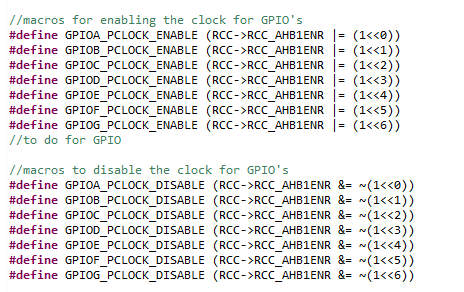
Created the typedef structures for the GPIO’s and included the specific functions of each GPIO’s and created

API’s, .H-files, .c-files for LED Blinking and tested the code for the working of LED-BLINKING by using the

base address of buses of the ARM cortex processor.







Implementation of EXTI, ADC, SPI and UART: