

Booting sequence report

Export the PATH of ARM-Cross-toolchain “arm-none-eabi-gcc.exe” to make things ez

```
MINGW64:/c/embbbed/LAB_1
ismail@LAPTOP-VO8R1D4K MINGW64 /c/embbbed/LAB_1
$ export path=../ARM/bin/:$path
ismail@LAPTOP-VO8R1D4K MINGW64 /c/embbbed/LAB_1
$ ls
app.c  uart.c  uart.h
ismail@LAPTOP-VO8R1D4K MINGW64 /c/embbbed/LAB_1
$ ls *.c
app.c  uart.c
ismail@LAPTOP-VO8R1D4K MINGW64 /c/embbbed/LAB_1
$ arm-none-eabi-g
bash: arm-none-eabi-g: command not found
ismail@LAPTOP-VO8R1D4K MINGW64 /c/embbbed/LAB_1
$ arm-none-eabi-gcc
arm-none-eabi-gcc.exe: fatal error: no input files
compilation terminated.
ismail@LAPTOP-VO8R1D4K MINGW64 /c/embbbed/LAB_1
$ |
```

Then generate out file using touch command then use the command “arm-none-eabi-gcc.exe --help” to see what u want the output to be or what at what stage do you want to stop and give it the architecture of the MCP.

```
MINGW64:/c/embbbed/LAB_1
c:/arm-none-eabi/10.2.1/../../../../arm-none-eabi/bin/ld.exe: c:/program files (x86)/gnu arm embedded toolchain/10 2020-q4-major/bin/./lib/gcc/arm-none-eabi/10.2.1/../../../../arm-none-eabi/lib/libg.a(lib_a-exit.o): in function `exit':
exit.c:(.text.exit+0x2c): undefined reference to `_exit'
c:/program files (x86)/gnu arm embedded toolchain/10 2020-q4-major/bin/./lib/gcc/arm-none-eabi/10.2.1/../../../../arm-none-eabi/bin/ld.exe: C:\Users\ismail\AppData\Local\Temp\cc9xmJSR.o: in function `main':
C:\embbbed\LAB_1/app.c:5: undefined reference to `UART_SEND_STRING'
collect2.exe: error: ld returned 1 exit status

ismail@LAPTOP-VO8R1D4K MINGW64 /c/embbbed/LAB_1
$ arm-none-eabi-gcc.exe -c -g -I . -mcpu=arm926ej-s app.c -o app.o

ismail@LAPTOP-VO8R1D4K MINGW64 /c/embbbed/LAB_1
$ arm-none-eabi-gcc.exe -c -g -I . -mcpu=arm926ej-s uart.c -o uart.o

ismail@LAPTOP-VO8R1D4K MINGW64 /c/embbbed/LAB_1
$ ls *.o
app.o  uart.o
ismail@LAPTOP-VO8R1D4K MINGW64 /c/embbbed/LAB_1
```

Object file is a relocatable Binary which means that it has not a physical address on the system bus, and it can not to run on processor.

- To know the Sections of App.c just use “arm-none-eabi-objdump.exe”.

```
MINGW64:/c/embbbed/LAB_1
ismail@LAPTOP-VO8R1D4K MINGW64 /c/embbbed/LAB_1
$ arm-none-eabi-objdump.exe -h app.o

app.o:          file format elf32-littlearm

Sections:
Idx Name              Size      VMA          LMA          File off  Algn
 0 .text               0000001c  00000000  00000000  00000034  2**2
   CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
 1 .data               00000064  00000000  00000000  00000050  2**2
   CONTENTS, ALLOC, LOAD, DATA
 2 .bss                00000000  00000000  00000000  000000b4  2**0
   ALLOC
 3 .debug_info         00000068  00000000  00000000  000000b4  2**0
   CONTENTS, RELOC, READONLY, DEBUGGING, OCTETS
 4 .debug_abbrev       0000005e  00000000  00000000  0000011c  2**0
   CONTENTS, READONLY, DEBUGGING, OCTETS
 5 .debug_aranges     00000020  00000000  00000000  0000017a  2**0
   CONTENTS, RELOC, READONLY, DEBUGGING, OCTETS
 6 .debug_line        0000003b  00000000  00000000  0000019a  2**0
   CONTENTS, RELOC, READONLY, DEBUGGING, OCTETS
 7 .debug_str         000000a1  00000000  00000000  000001d5  2**0
   CONTENTS, READONLY, DEBUGGING, OCTETS
 8 .comment            0000004e  00000000  00000000  00000276  2**0
   CONTENTS, READONLY
 9 .debug_frame       0000002c  00000000  00000000  000002c4  2**2
   CONTENTS, RELOC, READONLY, DEBUGGING, OCTETS
```

If u don't want the debug section to be created because the debug section takes more space from memory just in creating the object file remove -g.

```
MINGW64:/c/embbbed/LAB_1
ismail@LAPTOP-VO8R1D4K MINGW64 /c/embbbed/LAB_1
$ arm-none-eabi-gcc.exe -c -I . -mcpu=arm926ej-s uart.c -o uart.o

ismail@LAPTOP-VO8R1D4K MINGW64 /c/embbbed/LAB_1
$ arm-none-eabi-gcc.exe -c -I . -mcpu=arm926ej-s app.c -o app.o

ismail@LAPTOP-VO8R1D4K MINGW64 /c/embbbed/LAB_1
$ arm-none-eabi-objdump.exe -h app.o

app.o:          file format elf32-littlearm

Sections:
Idx Name              Size      VMA          LMA          File off  Algn
 0 .text               0000001c  00000000  00000000  00000034  2**2
   CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
 1 .data               00000064  00000000  00000000  00000050  2**2
   CONTENTS, ALLOC, LOAD, DATA
 2 .bss                00000000  00000000  00000000  000000b4  2**0
   ALLOC
 3 .comment            0000004e  00000000  00000000  000000b4  2**0
   CONTENTS, READONLY
 4 .ARM.attributes     0000002c  00000000  00000000  00000102  2**0
   CONTENTS, READONLY
```

You will need a startup file to know the initialize the stack pointer to reserve the Area from stack in RAM and to know the start point that the processor will execute it first thing when it's on.

```
ismaïl@LAPTOP-VO8R1D4K MINGW64 /c/embbbed/LAB_1
$ arm-none-eabi-gcc.exe -c -I . -mcpu=arm926ej-s startup.s -o startup.o
startup.s: Assembler messages:
startup.s: Warning: end of file not at end of a line; newline inserted

ismaïl@LAPTOP-VO8R1D4K MINGW64 /c/embbbed/LAB_1
$ arm-none-eabi-objdump.exe -h startup.o

startup.o:          file format elf32-littlearm

Sections:
Idx Name              Size      VMA           LMA           File off  Algn
  0 .text              00000010  00000000  00000000  00000034  2**2
    CONTENTS, ALLOC, LOAD, RELOC, READONLY, CODE
  1 .data              00000000  00000000  00000000  00000044  2**0
    CONTENTS, ALLOC, LOAD, DATA
  2 .bss               00000000  00000000  00000000  00000044  2**0
    ALLOC
  3 .ARM.attributes    0000001c  00000000  00000000  00000044  2**0
    CONTENTS, READONLY

ismaïl@LAPTOP-VO8R1D4K MINGW64 /c/embbbed/LAB_1
$ |
```

Finally, you will need a linker script to define the volume of RAM and ROM and pass this information to the linker to link the different object files and give the variable we used a physical address in Memory and the beginning of the reset Section According to the specs of CPU entry point then put it at this address and we can use it by this command “arm-none-eabi-ld.exe” and then we need the binary file that will run on the specific MCP so we use this command “arm-none-eabi-objcopy.exe”.

```
ismaïl@LAPTOP-VO8R1D4K MINGW64 /c/embbbed/LAB_1
$ arm-none-eabi-ld.exe -T linker_script.ld app.o uart.o -o learn-in-depth.elf

ismaïl@LAPTOP-VO8R1D4K MINGW64 /c/embbbed/LAB_1
$ arm-none-eabi-objcopy.exe -O binary learn-in-depth.elf learn-in-depth.bin
```

To run the code making sure that it works on the target we use qemu.

```
ismaïl@LAPTOP-VO8R1D4K MINGW64 /c/embbbed/LAB_1
$ ../qemu/qemu-system-arm -M versatilepb -m 128M -nographic -kernel learn-in-dep
th.bin
learn-in-depth:ismaïl|
```