Embedded Systems
Tools part 2

AMIT
Beyond Education

Prepared By: AMIT Learning

Getting Started …

Please make sure not to exert so much effort studying this material as its main goal not to deliver information, but to evaluate your comprehensive ability through the assessment exam held after that.

It also helps us determining your ability to self study some topics that gives alot of aid determining the course execution nature that suits you the best.

Osama Shaaban- Embedded Systems Training Manager

Simulator

•Simulator: is used to simulate the behaviour of the hardware without the existence of the hardware itself.
•It is a software on the PC in order to expect the behaviour of the MC.

•The simulator can understand the assembly instruction of the MC and it acts as if it was the MC and start changing the contents of RAM, Registers according to the code.

•The simulator can’t simulate the real time property of the embedded system.

Simulator – Cont’

- Simulator can be used to calculate the time consumbed by a part of code by calculating the number of cycles consumed to execute this code, and using the info of the frequency of the system, we could calculate a expected time for the code.

Emulator

- Emulator is a certain hardware for a specific family of controllers.

- It contains the max RAM & ROM size and all the prepherals and registers the could exist in a microcontroller in this family.

- It is FPGA that implement the core of the MC.

- It premits the programmer to make a hardware breakpoint and halt the hardware registers, so the programmer simply could debug the constraints in realtime manners.

In Circuit Emulator

- Simply it is the emulator connected to the board as if it is the MC itself and at this time I could emulate the I/O ports and other functionalities in my system.

- It is done by make the board without the MC then connect the emulateor in the MC place.

Debugger

- It is a way to see what the controller is doing and it is a hardware, but it is different from the emulator that it can't emulate the MC.

- The debugger is connected to a complete board and the user only able to read the changes done inside the registers and the memory of the controller.

- It communicate with a software on the PC through any kind of protocols like JTAG or serial.

Debugging Info

The Debug Info helps debuggers to analyze the internal layout of the debugged application. In particular, it helps the debugger to locate addresses of variables and functions, display values of variables (including complex structures and classes with nontrivial binary layout), and map raw addresses in the executable to the lines of the source code.

Debugging Info - Example

•Functions and variables :

For every function or variable, debug information stores its location and name.

•Source file and line information :

This kind of information maps every line of every source file to the corresponding location in the executable. (Of course it is possible that a source line does not have the corresponding location at all (e.g. a comment line). Such lines are not present in debug information).

Debugging Info - Example

Type information :

For every function or variable, debug information can store additional information about its type. For a variable or a function parameter, this information will tell the debugger whether it is an integer, or a string, or a user defined type, and so on. For a function, it will tell the number of parameters, calling convention, and the type of the function's return value.

Programmer/Flasher

It is hardware used in order to Download the code to the Microcontroller ROM(Flash/EEPROM).

Software Developing Environment (IDE)

What is an IDE?

An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development.

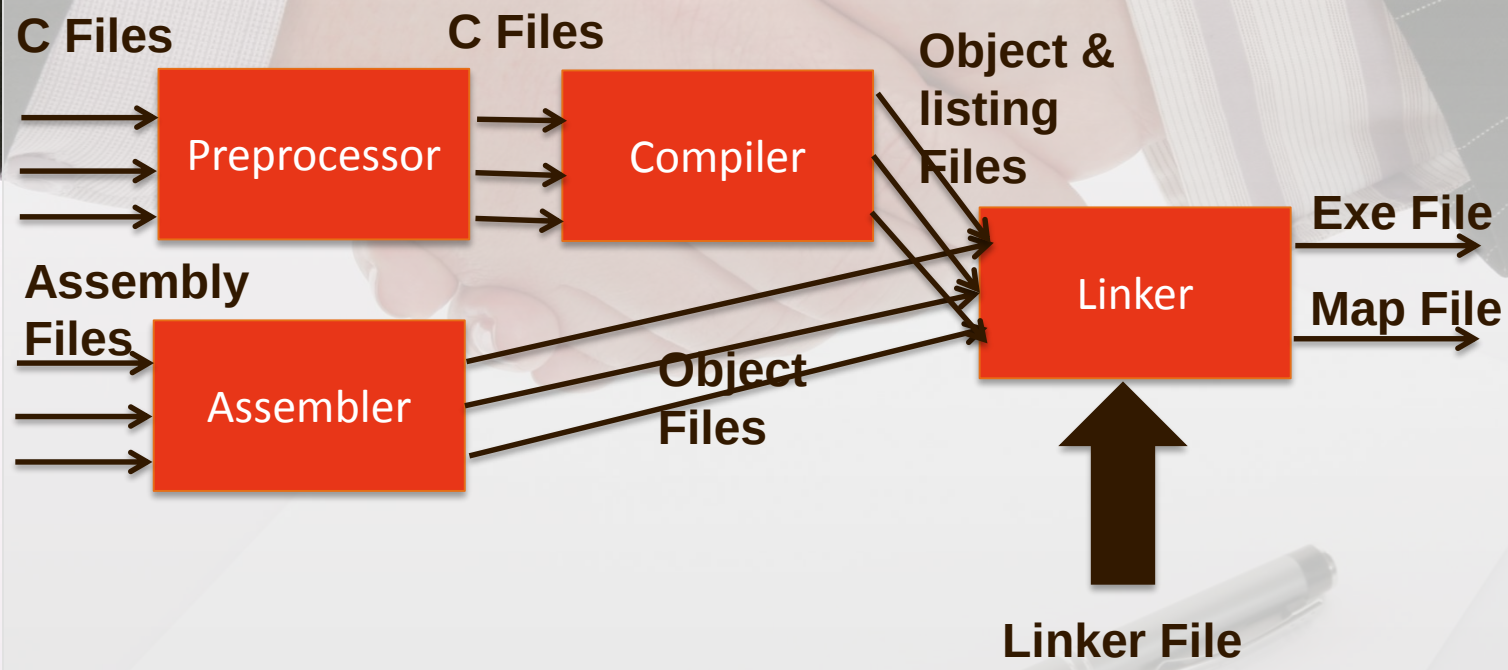
- Eclipse
- NetBeans
- MonoDevelop
- CodeWarrior

Components of IDE

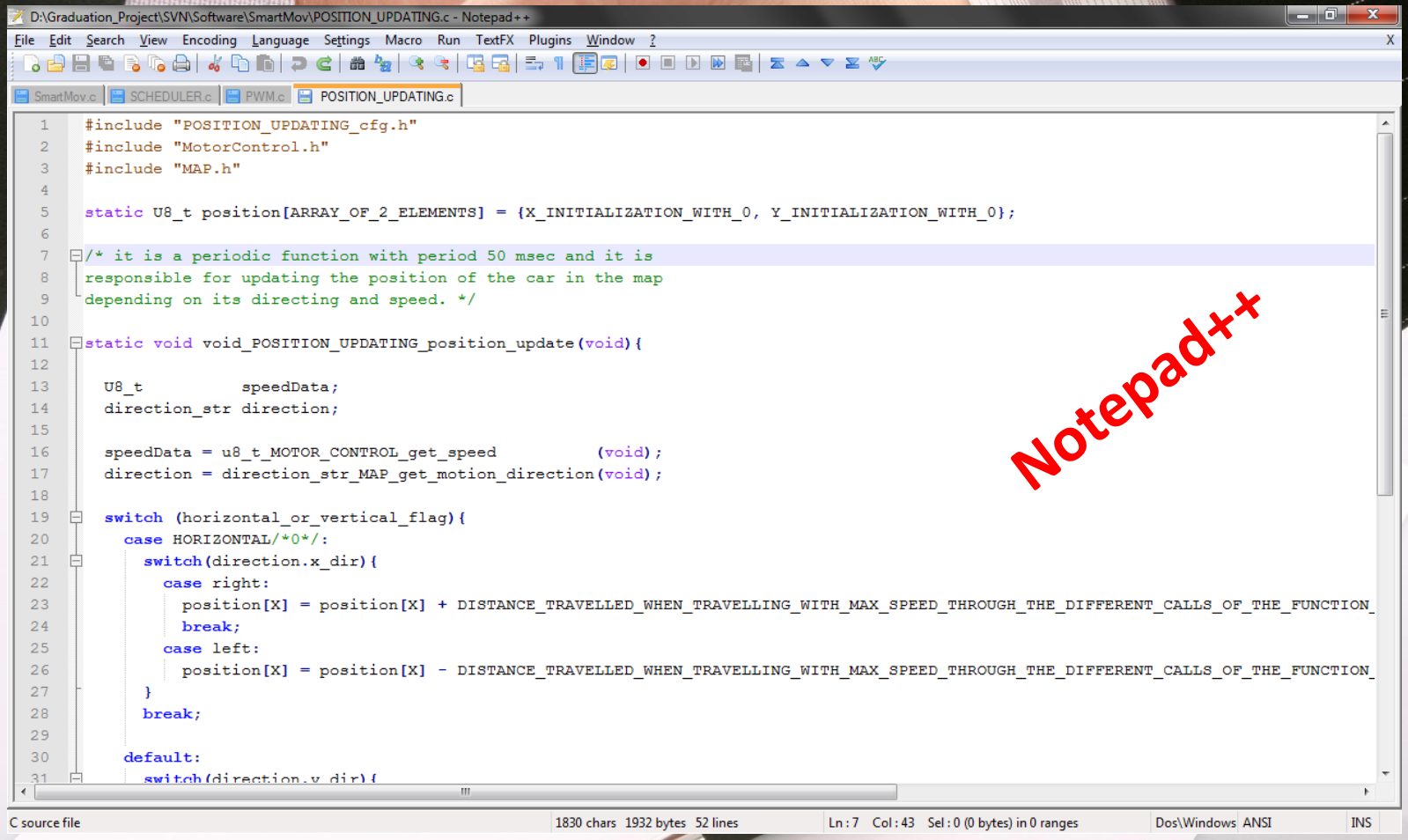
An IDE normally consists of:

- Source code editor
- Compiler and/or Interpreter
- Build automation tools
- Debugger

Software Life Cycle



Source code editor



Prepared by: AMIT Learning

Preprocessor

Preprocessor

- The input to this phase is the .c File, .h Files
- The preprocess process the preprocessor keywords like #define, #ifdef.... and generate a new .c File after the text replacement process.
- The output of this phase is a .c File without any preprocessor keyword.

Compiler

- A compiler is a computer program (or set of programs) that transforms source code written in a programming language (the source language) into another computer language (the target language).

- The most common reason for wanting to transform source code is to create an executable program.

Compiler (Cont..)

- Compiler
  - The input to this phase .c File
  - The compiler parse the code, and check the syntax correctness of the file.
  - Convert the .c Code into optimized machine language code.
  - The output of this phase is object file (.o File) and list file (.lss file).
- List File:

Contain the corresponding assembly code for each line.

Compiler (Cont..)

- A Cross compiler is a compiler capable of creating executable code for a platform other than the one on which the compiler is running.
- Cross compiler tools are used to generate executables for embedded system or multiple platforms. It is used to compile for a platform upon which it is not feasible to do the compiling, like microcontrollers that don't support an operating system

Compiler (Cont..)

GCC Compiler:

- GNU C Compiler, Started at 1987 as a part of the GNU project
- Renamed to GNU Compiler Collection
- GCC is not only a Native Compiler but also it's a Cross Compiler
- GCC is ported to a wide variety of processor architectures and is also available for most embedded platforms

Compiler (Cont..)

Compilation Options:

gcc h2d.c –o h2d.exe	..quick and easy
gcc h2d.c –Wall –o h2d.exe	..get compiler warnings
gcc h2d.c –std=c99 –o h2d.exe	..use C99 standard

Note: options in any order, but " –o h2d.exe" is one option

Example: **gcc** –o h2d.exe –std=c99 h2d.c

Compiler (Cont..)

gcc –E h2d.c

Produce preprocessed file(.i file) of h2d.c

gcc –S h2d.c

produce assembly code in h2d.s

gcc –c h2d.s

produce object code in h2d.o

gcc –o final.exe main.c aux1.s aux2.o

compile, assemble and link three files into one executable

gcc h2d.c –O2 –o h2d.exe

-O0, -O1, -O2, -O3 are optimization levels

Assembler

- An assembler creates object code by translating assembly instruction mnemonics into opcodes, and by resolving symbolic names for memory locations and other entities.

- Most assemblers include macro facilities for performing textual substitution

Assembler (Cont..)

Assembler

- The input to this phase .asm File
- The Assembler coverts the assembly code to the corresponding machine language code.
- The output of this phase is object file (.o File).

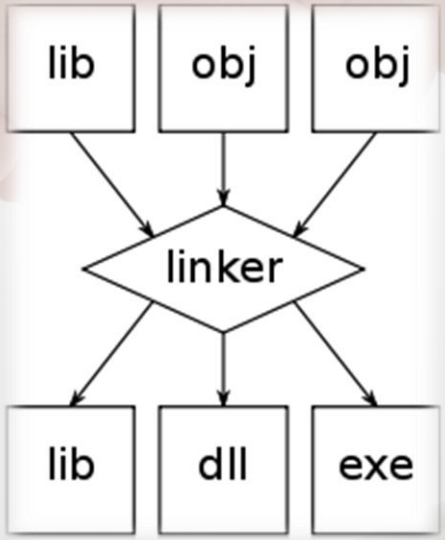
Assembler (Cont..)

- GAS : GNU Assembler
- The executable name of GAS is “as”
- It is the default back-end of GCC
- We use this command to generate the object file out of the assembly file

```
as -o object_file.o code_file.s
```

Linker

A Linker is a program that takes one or more objects generated by a compiler and combines them into a single executable program



Linker (Cont..)

- Linker
  - The input to this phase multiple .O File
  - The Linker merges different object files and library files.
  - The linker allocate target memory (RAM, ROM, Stack)
  - The linker produce the debug info
  - The output of this phase is the executable file and the map file.

The Map file format is mainly dependent on the linker, but in general it contains the allocation of different object files in the different memory segments.

Linker (Cont..)

LD :

- Is a GNU linker
- Is a part of GCC “GNU Compiler Collection”
- Is responsible of doing the linking operation.
- Is called automatically within the compilation process.

Build Automation Tool

Build automation

- Build automation is the act of scripting or automating a wide variety of tasks that software developers do in their day-to-day activities including things like:
  - compiling computer source code into binary code.
  - packaging binary code.
  - running tests.
  - deployment to production systems.
  - creating documentation and/or release notes.

Build automation (Cont..)

- One specific form of build automation is the automatic generation of Makefiles.
- This is accomplished by tools like
  - GNU Automake
  - CMake
  - Imake
  - qmake
  - nmake
  - wmake
  - OpenMake Meister

Build automation (Cont..)

Why makefiles ?

- Let's say that we have the following files "main.c", "file1.c" and "file2.c" to compile these files we should type the following line in the the shell :

```
gcc -std=c99 -wall -o final.exe main.c file1.c file2.c
```

- We do the same thing every time we compile the files and that may be time consuming so we recompile the changed files only.

- So a tool to save the compilation time and decide which files to be recompiled and which to be not, is strongly required.

Build automation (Cont..)

GNU Automake

- GNU Automake is a programming tool that produces portable makefiles for use by the make program, used in compiling software.
- It is made by the Free Software Foundation as one of GNU programs, and is part of the GNU build system.
- a script file called makefile is responsible of guiding make.
- The script file should be named “makefile”, “Makefile” or “GNUmakefile”.
- The C file is recompiled if it has been changed or any of it’s dependencies is changed

Build automation (Cont..)

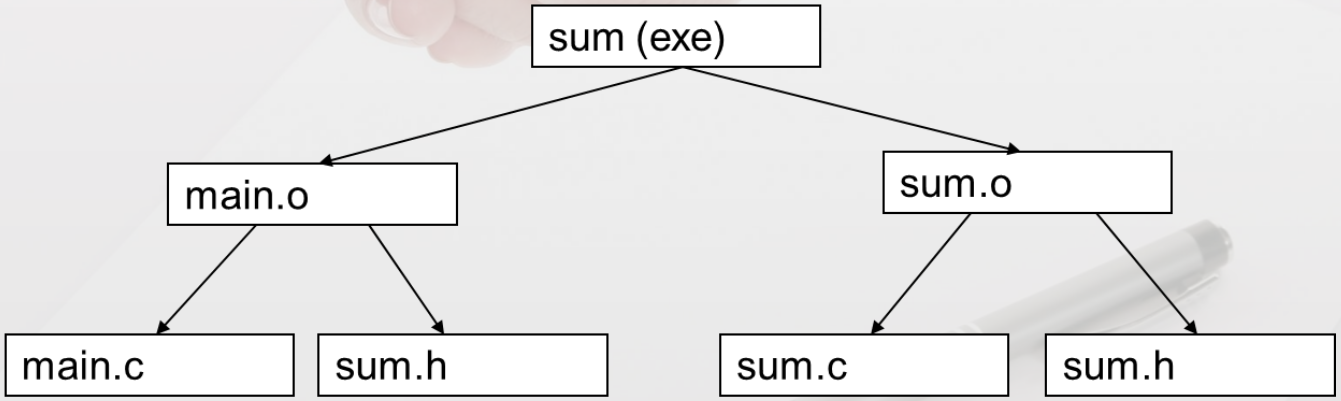
Project Compilation

- Done by the Makefile mechanism
- A makefile is a file (script) containing:
  - Project structure (files, dependencies)
  - Instructions for files creation
  - The **make** command reads a makefile, understands the project structure and makes up the executable
- Note that the Makefile mechanism is not limited to C programs

Build automation (Cont..)

Project structure

- Program contains 3 files
- main.c., sum.c, sum.h
- sum.h included in both .c files
- Executable should be the file sum

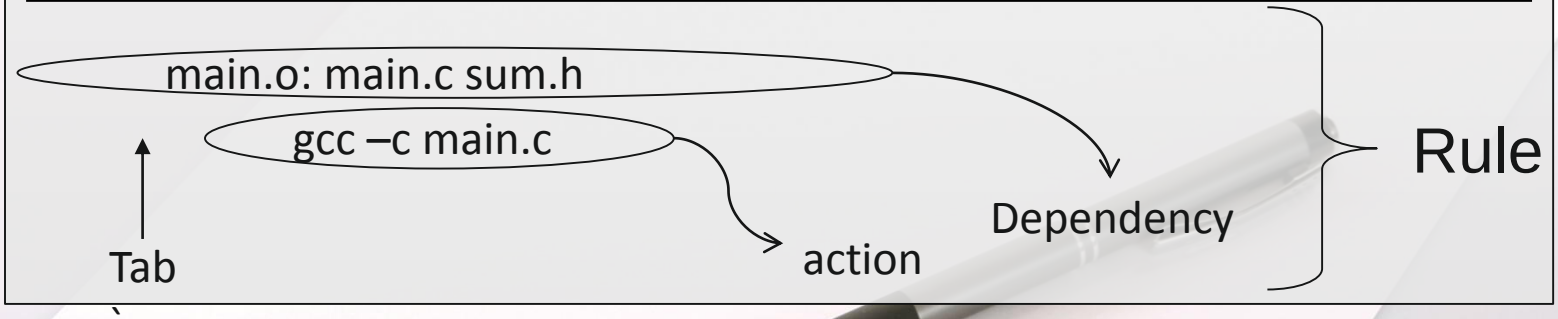


Build automation (Cont..)

Group of lines

- **Target**: the file you want to create
- **Dependencies**: the files on which this file depends
- **Command**: what to execute to create the file (after a TAB)

```
Testmath :  testmath.o mymath.o
    gcc –o testmath testmath.o mymath.o
```



Build automation (Cont..)

MakeFile

```
sum: main.o sum.o
   gcc –o sum main.o sum.o

main.o: main.c sum.h
   gcc –c main.c

sum.o: sum.c sum.h
   gcc –c sum.c
```

Build automation (Cont..)

MakeFile

.o depends (by default) on corresponding .c file. Therefore, equivalent makefile is:

```
sum: main.o sum.o
        gcc –o sum main.o sum.o
main.o: sum.h
        gcc –c main.c
sum.o: sum.h
        gcc –c sum.c
```

Build automation (Cont..)

Make Operation

- Project dependencies tree is constructed
- Target of first rule should be created
- We go down the tree to see if there is a target that should be recreated. This is the case when the target file is older than one of its dependencies
- In this case we recreate the target file according to the action specified, on our way up the tree. Consequently, more files may need to be recreated
- If something is changed, linking is usually necessary

Build automation (Cont..)

Make Operation

- make operation ensures minimum compilation, when the project structure is written properly.

Do not write something like

```
prog: main.c sum1.c sum2.c
        gcc –o prog main.c sum1.c sum2.c
```

which requires compilation of all project when something is changed

Build automation (Cont..)

Using Makefile

Example

Build automation (Cont..)

Targets

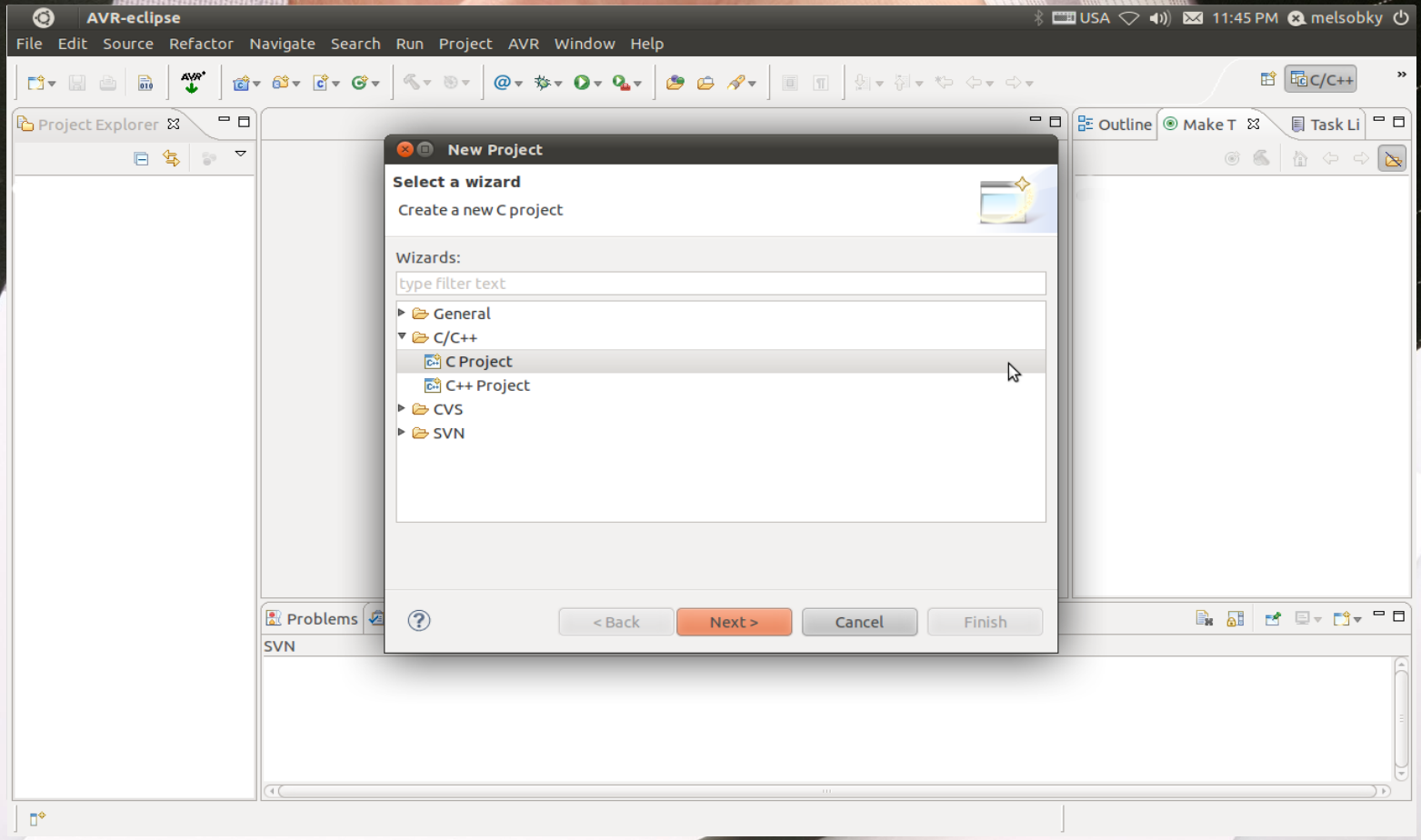
- We can define multiple targets in a makefile
- Target clean – has an empty set of dependencies. Used to clean intermediate files.
- make
  - Will execute the first target in make file
- make **TARGET_NAME**
  - Will execute the action of this target
- make **clean**
  - Will remove intermediate files(like .o Files)

Eclipse IDE Integration

Eclipse IDE Integration

- Create a New Project by selecting File >> New >> Project
- Select the C Project Option and press Next

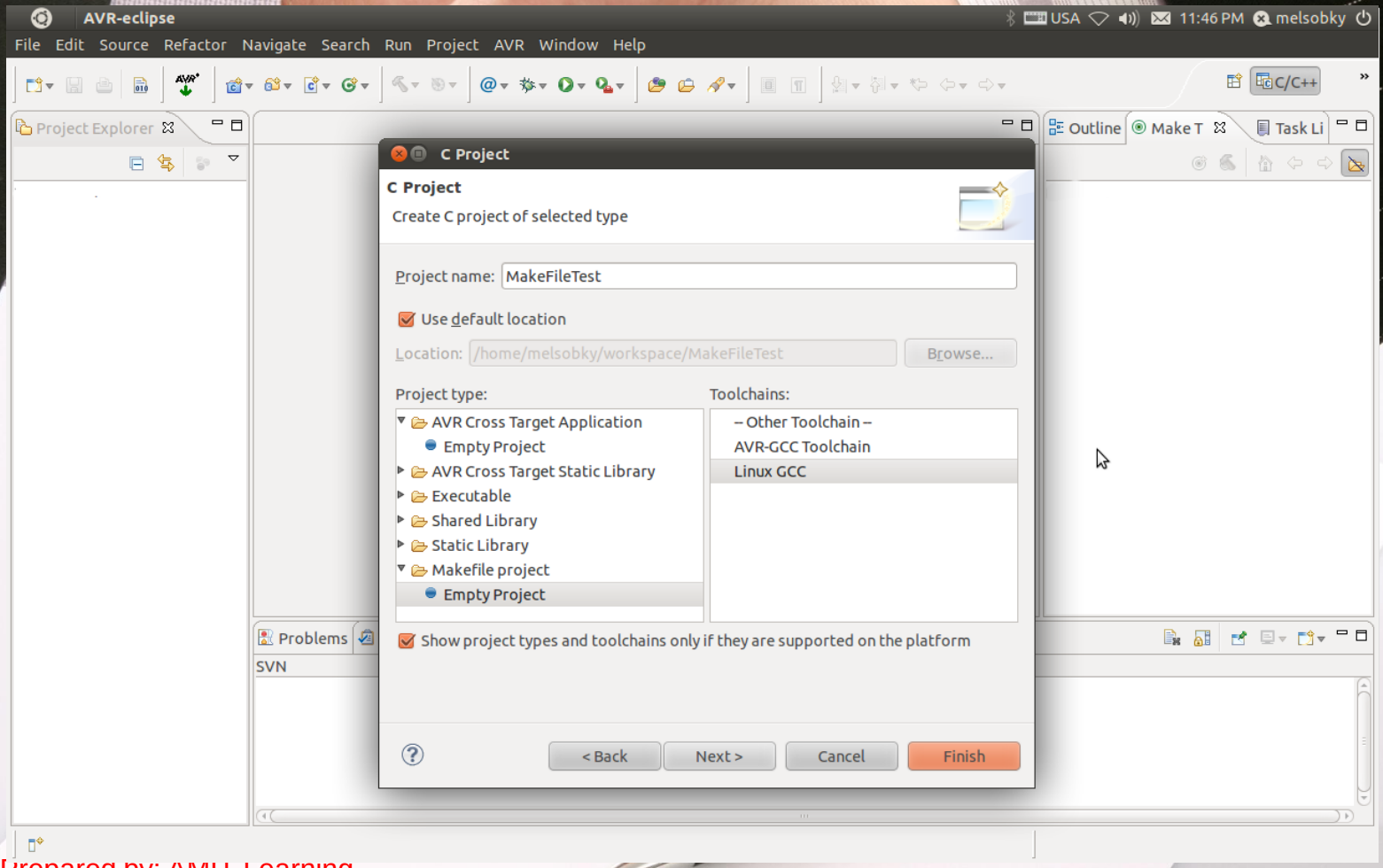
Eclipse IDE Integration (Cont..)



Eclipse IDE Integration (Cont..)

- From the project type menu select Makefile Project
- From the Tool chain menu select the compiler

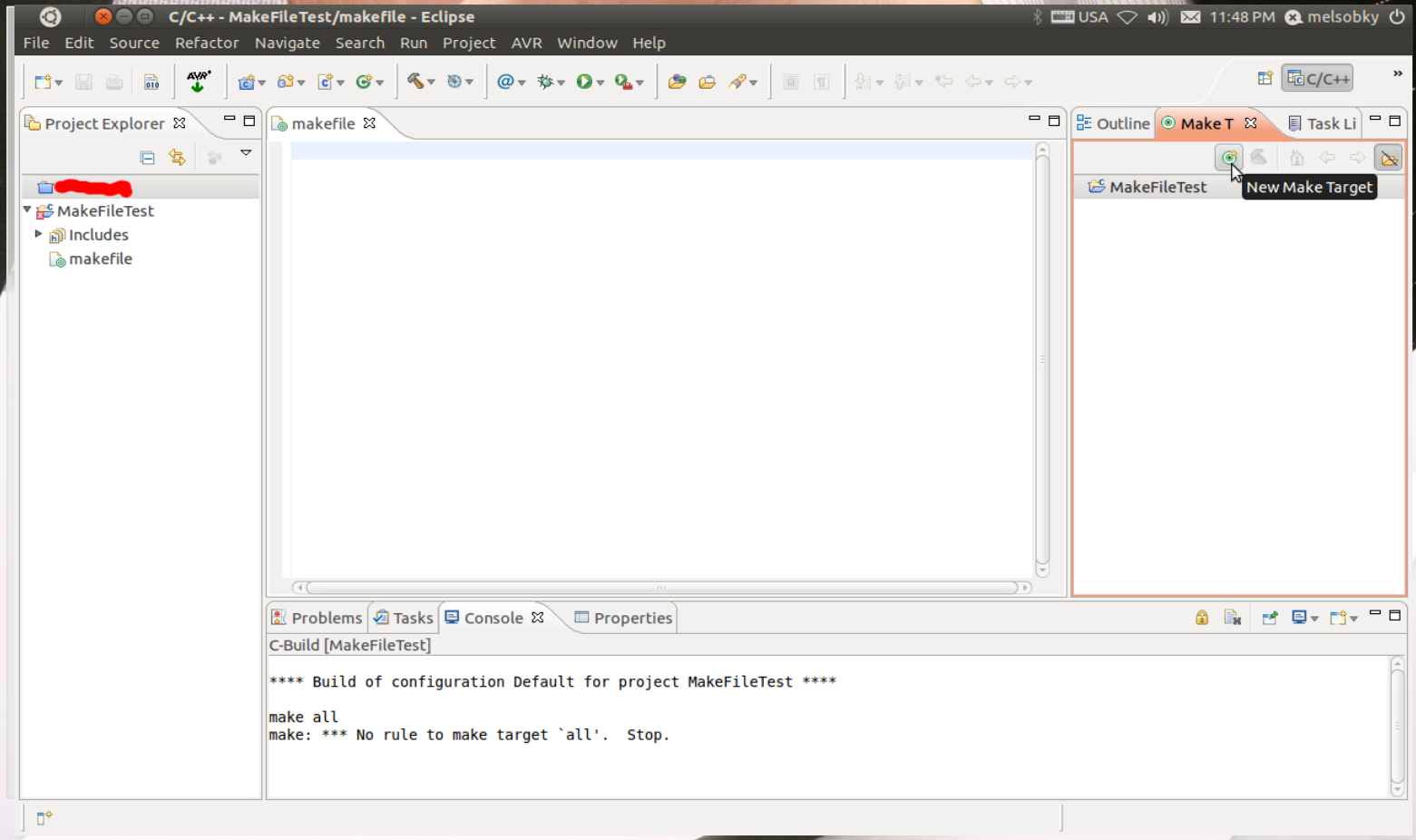
Eclipse IDE Integration (Cont..)



Eclipse IDE Integration (Cont..)

- Add a new empty file to the project and name it makefile
- Form the Make Target Section select your project then click new make target

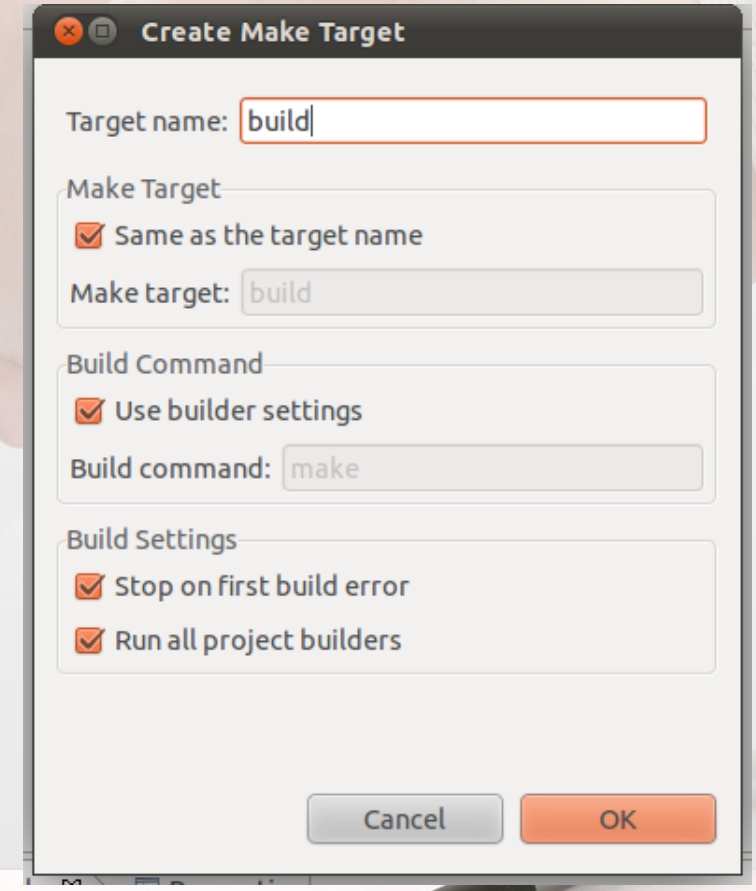
Eclipse IDE Integration (Cont..)



Eclipse IDE Integration (Cont..)

- Select the Taget name to be displayed in the Make Target Section
- Select the target name as it's written in the makefile

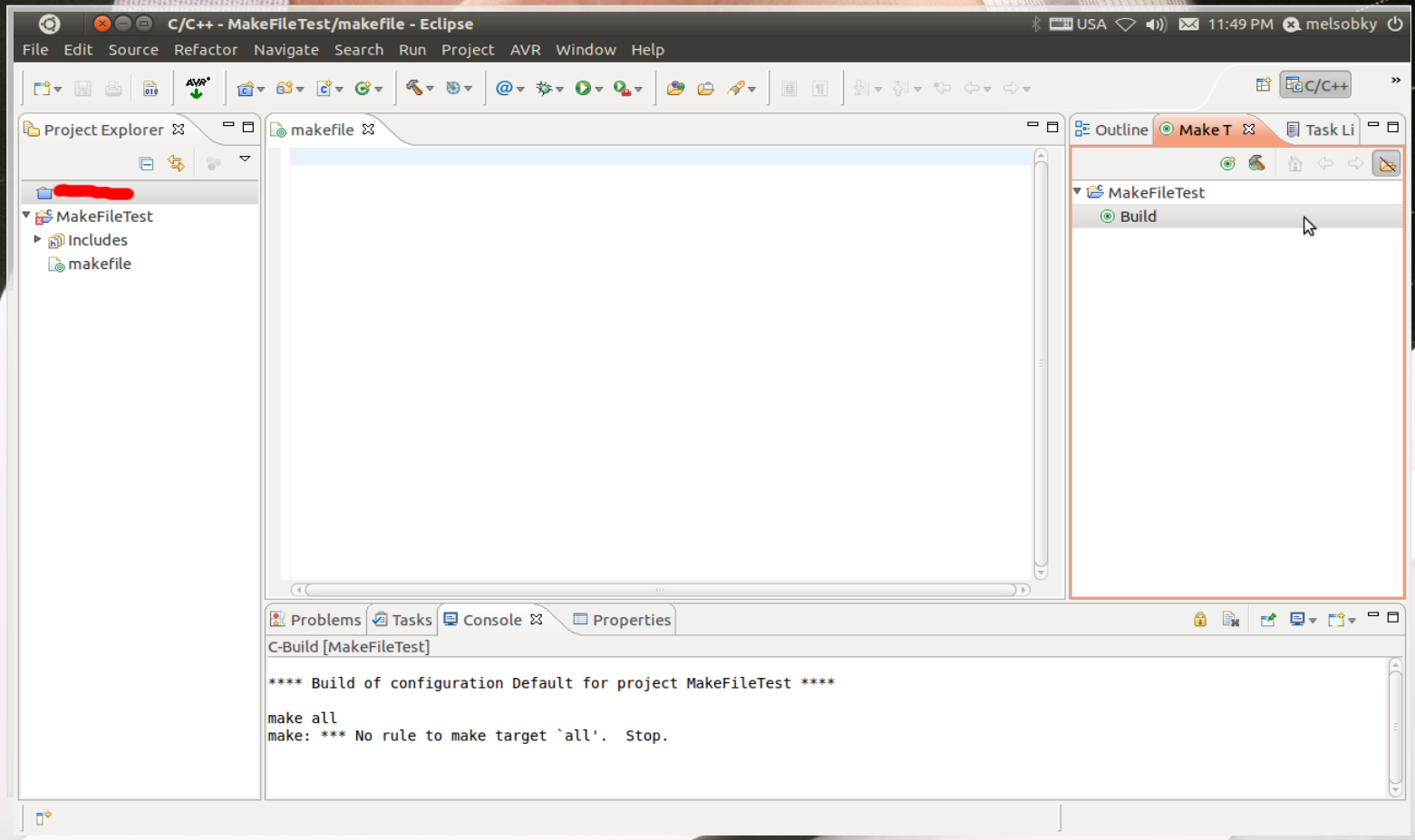
Eclipse IDE Integration (Cont..)



Eclipse IDE Integration (Cont..)

- Now the make target is ready, double click on it to start the build

Eclipse IDE Integration (Cont..)



THANK YOU!