Microprocessor Systems Laboratory

Developing tools - Simulators

What is an Simulator ?

The Simulator is a **program** that supports microcontroller's opcodes and peripherals to let you debug the program in a safe, crash-proof environment. Running on PC with **no additional hardware**. Simulator is ideal for testing entire programs before the target hardware is ready. On-chip peripherals are fully simulated, while off-chip peripherals can be simulated by files or keypresses.
Simulator often comes as a part of an Integrated Development Environment (IDE) for simulation of the embedded microcontroller applications. The simulator allows to switch between source mode and disassembly mode debugging as required. You can choose between disassembled code and original assembler or High Level Language (HLL) source code. Disassembly mode debugging lets you focus on the critical sections of your application, and provides you with precise control over the hardware. You can execute the program as an assembler instruction at a time, and display the registers and memory or change their contents.
The simulator makes it easy to test hardware defects and critical situations which are difficult to debug with real hardware.

Example of an IDE

The example of an Integrated Development Environment is µVision2. The µVision2 IDE from Keil Software, combines project management, make facilities, source code editing, program debugging, and complete simulation in one environment. µVision2 helps you get programs working fast while providing an easy-to-use development platform. The editor and debugger are integrated into a single application and provide a seamless embedded project development environment.

Steps to use the µVision2 IDE

Project

First step is to create the project. A **Project** is the collection of all the source files as well as the compiler, assembler, and linker settings required to compile and link a program. The Project menu provides access to all dialogs for project management including:

- **New Project**. which creates a new project.
- **Targets, Groups, Files**. which add components to a project. The Local menu in the Project window allows you to add files to the project.
- **Open Project**. which opens an existing project.

When you create a new project, you select the chip you will use and µVision2 automatically sets the necessary assembler, compiler, and linker options.

Program

Second step is to write the program. You can use included editor. Color syntax highlighting and text indentation are optimized for editing C source code. Most editor functions may be

quickly accessed from the toolbar. The editor is available while debugging your program. This gives a natural debugging environment that lets correct errors in source code.

Compilation

As a next step the program must be compiled. µVision2 includes an integrated **make** facility to compile, assemble, and link the program. After compilation and assembling the source files, status information as well as errors and warnings appear in the Output Window.

Debugging

When the program is free of compilation errors it can be simulated for debugging. In the µVision2 debugger, you run your target program by clicking on the Run button on the toolbar. The Run button executes code until a breakpoint is reached. In addition to simply running your program, you may use the buttons on the toolbar to step through your application program one line of code at a time.
When Trace Recording is enabled, the Show Trace Records button lists the last 1024 instructions that were executed. Trace recording lets you analyze the program flow prior to a breakpoint.

The debugger allows you to select a line of code at which execution is halted. This is called a Simple (or Execution) Breakpoint. While debugging your program, right-click on a line of code to set an execution breakpoint. Then, when that line is reached, the debugger halts program execution and allows you to examine memory, registers, variables, and so on. In addition to Simple Breakpoints, the µVision2 debugger allows you to set breakpoints on conditional expressions and even on different types of memory accesses. Breakpoints may include a count which decreases until the breakpoint is triggered. Commands may be executed when the breakpoint is triggered or, if no command is specified, program execution stops. Use the **Breakpoint** command from the Debug menu to define complex breakpoints.

Program optimizing

The IDE offers tools for optimizing the program to take less time to execute. The built-in performance analyzer in the µVision2 debugger records and displays execution times for functions and program blocks you select. Bar graphs display the amount of CPU time spent in each part of your program. You may use the information gathered by the performance analyzer to determine execution hot-spots in your application. Then, you can concentrate your efforts on making that part of your program faster.

Based on the Keil IDE description. http://www.keil.com/uvision2/

Exercise description

During the exercise students are to:

- Create the project in µVision2
- Write the program in C (topic given by the supervisor)
- Compile the program
- Observe program execution in the simulator watching parameter passing to the procedures
- Analyze execution time of program blocks
- Optimize the most time-consuming blocks using assembler

The optimization can be done in three ways:

First way is to create the assembler source file instead of object file during compilation process. Such assembler file can be optimized manually and assembled. To compile the C source to the assembler source file you have to use `#pragma SRC` directive at the top of the file. The compiler will create the *.src file that you should rename to *.a51.

Second way is to write some part of the code in assembler inside the C file (inline assembly). To do it you have to:

- use `#pragma SRC` directive at the top of the file
- use the pair of directives inside the file:

```
#pragma asm
    // add inline asm instructions here
#pragma endasm
```

- turn off the checkbox "Include in Target build" in the options for C source file.

Third way is to write one module in C, other - more time consuming module - in assembler. To write the assembler function called from C you can use the template created as in first example.

Report

The report should contain:

- Title page
- Topic of the program
- Source code of the program **with comments**
- Description of passing the parameters to the procedures
- Performance analyze
- Result of optimization (changed blocks of disassembled program compared to original compiled version)
- Conclusions