

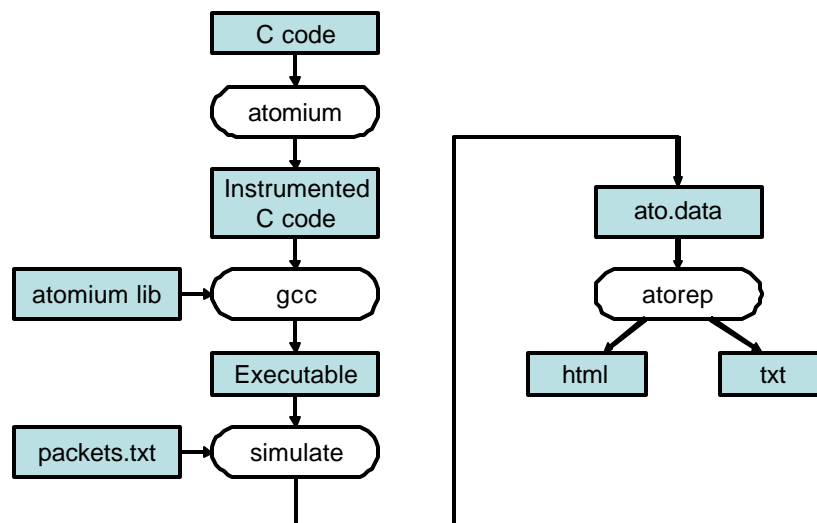
EE201A - Homework 3

Memory Analysis of the Embedded Webserver

After the introductory steps of compilation and testbench design, the webserver code holds no more secrets for us. We can now start considering the issues for embedded implementation. The overall target for the webserver design is to optimize for performance, relative to the initial mapping from Homework 1.

As with so many problems in embedded system design, optimizations at a high level of abstraction have far more impact than detailed peephole optimizations. For the case of the webserver in this project, ‘high level’ means: *target-independent C source code* level. The next two phases in the project will consider optimization of the memory architecture of the embedded webserver. At high level, the memory architecture of your design is tangible in the way the code works with arrays. If we can modify the source code such that the size of the arrays shrinks for example, we will need fewer background memory locations.

First, we will perform analysis of array accesses, and following the analysis we will apply optimizing transformations to our code. We will use the ATOMIUM tool to help us analyze and transform our C code. ATOMIUM is a tool that allows you to find out the number of writes and the number of reads of any background (array) variable of your C program. ATOMIUM works largely automatic and does not require you to do extensive rework of your C source code.



The design flow will follow with ATOMIUM is shown in the figure above. We start with C code for the web server (`uip_hw3`). This code is processed and transformed by ATOMIUM into *instrumented C code*. The instrumentation adds extra code to your original program that executes transparently, but at the same time tracks and counts access to arrays. The transformed program can be compiled and run as you normally would do with

the original code. We will use the same testing approach for `uip_hw3` as we did with Homework 2 (`uip_hw2`). That is, we prepare a text file `packets.txt` that contains hexdumps of the packets that are received by the webserver. After the simulation, a file `ato.data` is created that contains profiling information on the array accesses in the program during simulation. This file can be postprocessed by a utility `atorep`, which creates a text report or a browsable (html) report. The kind of questions you can answer using this report are for example: How many reads are done from the array `uip_buf` and at which places in the source code? What is the peak memory usage of arrays during the entire simulation?

To complete Homework 3, take the following steps:

1. Log into your SEAS account. The atomium tools are installed on seasnet (solaris) under `/u/ee/class/ee201a/cbin/atomium`. Everybody will work on the same platform for this homework. If you have no SEAS account yet, go ahead and request one using `http://www.seas.ucla.edu/acctapp`. Keep in mind it takes at least one day to activate an account. Do not wait until last minute. The manuals for ATOMIUM are found in `/w/class.01/ee/ee201a/cbin/atomium/v1.2.2/doc`. You don't need to worry about command line options and so on - the package `uip_hw3` that you will use has a makefile that runs all tools in the design flow for you.
2. Download the package `uip_hw3` from the project homepage. Unzip the package under your SEAS account. There are two directories in this package: (a) `atomium/`: in this directory you run `atomium` (b) `packetgen/`: in this directory you create a test set of packets (see 3.). When you type 'make' in `atomium/`, the webserver code is compiled according to the design flow shown earlier. When you type 'make report', the simulation is run to create the profile data. The code in `uip_hw3` is functionally identical to that of `uip_hw2`, but some modifications have been done to comply with the requirements of ATOMIUM towards C code. The modifications are all related to the use of macro's and the include file structure. If you are familiar with the C files in `uip_hw2`, you will readily find your way around in `uip_hw3`.
3. The stimuli file `packets.txt` is not yet created for `uip_hw3`. The program in `packetgen` can help you to create the required `packets.txt`. In this directory, type 'make' to create an executable `packetgen`. Create the file `packets.txt` by running:

```
packetgen > packets.txt
```

By making modifications to `packetgen.c`, you will be able to create different traces. We will require the following two traces for the ATOMIUM simulations. All packets will come from a host 128.97.0.1 (ethernet AA:BB:CC:AA:BB:CC) and will be directed to the webserver at 128.97.88.190 (ethernet 00:BD:3B:33:05:71). The two traces to use are:

- (a) A series of 1000 ping-request packets, addressed to the webserver. The program `packetgen.c` as found in `uip_hw3` creates this sequence for you.
- (b) A series of 1000 webpage downloads from the webpage (slashdot your device!). The sequence of packets for a 'webpage download' is: SYN, (server replies SYNACK), ACK, (server replies ACK), 'GET /', (server replies several ACK) and 4 more ACK

from which the last one also resets the TCP connection, i.e. it is a ACK-RST packet. Each webpage download has to come from a different source port, starting at 0x8000, then 0x8001 and so on.

You will need to modify packetgen.c such that it creates a trace of packets like (b) instead of (a).

4. Run atomium ('make report') for each of the packet sequences created with step 3. Use output redirection to simulate faster ('make report >output'). For each report, take a close look at the html output using your browser and locate the following three pieces of information
 - (a) The three array variables with the most accesses.
 - (b) The five functions with the most *local* accesses to background variables.
 - (c) The peak memory usage (counted in array locations)
5. You're done !

What to turn in

*> Deadline: Thursday April 24, 2003, 12 noon at Marilyn's desk.

*> 1 page containing your name, and contact E-mail. In addition, also provide a link to your EE201A project webpage and post the following items: The two packet traces for 'ping' (3a) and 'web server access' (3b).

In the report, include the results of (4) as follows:

TABLE 1. Three Most accessed array variables

Variable	Elements	Writes	Reads
uip_buf	xxx	yyy	zzz
...

TABLE 2. Five Most access-intensive functions

Function	Activations	Local Writes	Local Reads
chksum	xxx	yyy	zzz
...

also include Peak Memory Usage = xxxx

*> Good Luck, use the Class Bulletin Board for any questions!