Computer Systems 414

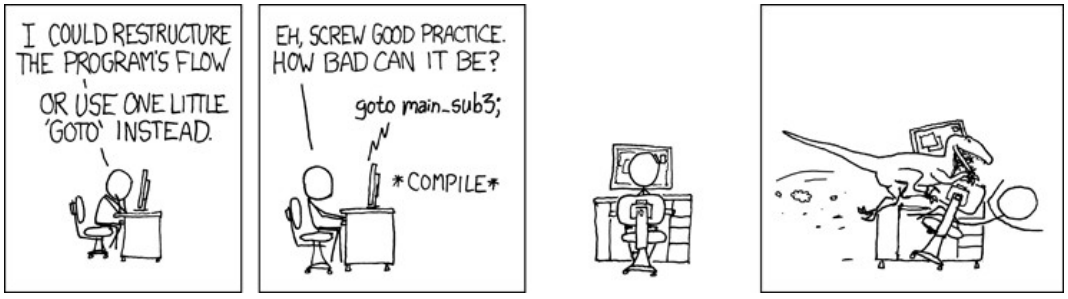
Practical 3

2021

Aim of Practical 4:

a) To investigate effects of software on performance of peripherals

b) To get experience of an ARM cache memory, specifically Raspberry Pi (R-PI) and Beagle Bone Black (BBB).



Rules of Engagement

1. Please see the study guide for regulations regarding attendance. Part 3A of the practical to be done in the lab. Attendance is not required for part 3B.

2. Work in groups of 2 to 3 .

3. Obtain all relevant supporting documentation on learn.sun.ac.za

4. The report is to be submitted on learn.sun.ac.za. The deadline for the reports are 16:00 0n Wednesday 7 April 2021 for Assignment 3A and 13:00 on Wednesday 14 April 2021 for Assignment 3B. **No late submissions will be accepted.**

5. Do not forget to cite and give credit for any information reported which is not your property.

6. Google is your friend. Any information not given is left out on purpose. Search your solution and on the internet or relevant manuals and documentation.

Assignment 4A

Get the BBB and R-Pi hardware from the store. Use *cacheLine.c* as source code (from learn.sun.ac.za):

a. Compile and run it on R-Pi. Repeat the run a few times to get average values and note your results. Refer to Practical 1 for setup details!

b. Compile it on the BBB. Set the BBB's CPU to the following frequencies and run the application for each frequency (repeat a few times for averages):

   i. 300MHz, 600MHz, 800MHz, 1000MHz
   ONLY on BBB!! To change frequency, use the command:
   cpufreq-set -f <VAL_ABOVE_WITH_MHz>
   Note your results.

c. Explain your results in terms of the effect that the usage and size of the cache has on the execution of the code. Use cache memory information from the BBB and Raspberry data sheets.

Assignment 4B

Refer to the Beaglebone Black system diagram. When using a specific peripheral, such as a UART, you have to be careful how you write your code, since the program can go into an ineffective "wait" state. Waiting is simply when the CPU is waiting for communication from a peripheral instead of executing any important work. Most peripherals have several flags which the CPU uses to determine when it can send data to it or receive data from it. However, it is important to realise that many of these peripherals preform work independently from the CPU.

Task (Refer to Practical 2 and use the uploaded project for the Keil IDE.)

You are a highly trained American Marine who has just been insulted over the internet. While sending a threatening message to the perpetrator you also wish to trace his IP. The tracing is done using an algorithm while you use a UART (your satellite phone's serial connection) to send your newfound arch nemesis a message. You need to decide between two methods of using the UART to reach an optimal balance between tracing (CPU work) and threatening (peripheral work).

a. First, run the *BusyWaitPrint* method along with the generic algorithm. Use the *Performance Analyser* as well as UART memory window to analyse the method.

b. Next, run the *FlagControlledPrint* method along with the generic algorithm. Analyse it as well.

c. What is different between the two methods and how does it affect the effectiveness of the overall system?

d. What would an ever better method be to handle the appropriate system?

e. Show three diagrams which compare the execution times of *generic_algorithm.c*, *print_to_uart.c* and *serial.c*.

Note: In order to make the practical work you may need to start uVision in administrator mode and update the STM32F1xxDFP pack. To do this, use the Pack Installer and go to:

Devices > STM32F1 Series
Packs > Keil::STM32F1xx_DFP and click Install.