Computer and Digital System Architecture

EE/CpE-517-A

Bruce McNair
bmcnair@stevens.edu
Course project description
Project requirements

• As part of this course, each student will be responsible for an independent individual or small group (2-3 members) design project.

• Choose from the potential project ideas on the following slides, or feel free to define another project of comparable complexity – all of the potential projects listed or other student defined projects will vary in their level of challenge, which will be factored into grading.

• Research and document reasonable system requirements to define the system scope

• Define the system architecture and create a design to satisfy the system requirements
Project requirements

• The design should be created and demonstrated using appropriate design or simulation tools. For some of the potential projects, a complete simulation or emulation will be required. For others, a detailed system design will suffice. If a working demonstration of the end result is possible, it will be worth extra credit.

• A final written report will include a system description, block diagrams, calculations of relevant parameters and simulation. Grading criteria:
  – Definition of problem 10%
  – Assumptions: 10%
  – System specification 10%
  – System architecture 15%
  – Detailed design 15%
  – Design tradeoffs 20%
  – Demonstration of results through simulation or design documentation 10%
  – References and proper citations 10%
Project requirements

• The completed design project will be due at the end of the course. It is expected that the final report will be at least 10-15 pages.

• A project presentation will be due before the 13th week of the course. In class presentation will be held during Class 13 and 14. On-line presentations for WebCampus sections will be held during Week 13. The presentation should describe the high-level approach, results and insights gained through the project. The presentation should be approximately 15 minutes.
Potential projects

1. Implement an emulator for a MUxx processor in a high-level language. C, Pascal, Java, even Matlab or Fortran 99, are viable languages that an emulator could be written in. The MU00 machine from Furber, Chapter 1 with the enhancement of a hardware multiply instruction and left/right shift instructions should be sufficient.

   – Use the emulator to load a set of data from a text file into the processor memory, run a MUxx assembly language program to process the data, storing the results back in memory, then have the emulator write the memory contents back to a text file. The input data file will be provided, but will be a few hundred integer values between 0 and 511 that represent samples of a chirp waveform. The assembly language program will filter the waveform with a set of FIR filter coefficients that represent a bandpass filter and will be provided.
Potential projects

2. Design an arithmetic logic unit that might be used in a microcontroller. It should include at least circuitry to add, subtract, multiply, divide, perform logical and arithmetic shifts of arbitrary amounts, and logic operations like AND, OR, NOT and NOR. For extra credit, the ALU should perform functions like sqrt(), 1/x, sin(), cos(), tan(), log() and exp().
   - The design could be a digital design verified with a hardware simulator or can be implemented through software emulation. The design must be performed at the gate/register level.
   - The correctness of the design must be demonstrated with a set of test vectors, e.g., define a set of input values and use the ALU design to calculate the correct output values. These test vectors should exercise all of the ALU functions under a small set of different situations.
Potential projects

3. Design an embedded system using an ARM controller. You may choose an ARM controller (e.g., like the Atmel SAM7X or similar microcontroller from Freescale, Intel, TI, etc.), or you may assume the use of an ARM core in a System-on-a-chip
   - The system designed should be of low to medium complexity, for instance, a traffic light controller for a 4-way intersection and pedestrian crossing. Extra credit will be given for more complex systems, e.g., a real-time high speed router. (My first serious embedded software design was for a Morse Code tutor. If you are interested in implementing this, I can discuss my algorithm)
   - The design should be demonstrated using ARM PC-based emulation tools
Potential projects

4. Define a representative application for an ARM-like controller, such as a cellular phone, a home automation system, a home entertainment system, a building security system, etc.
   – Consider the memory and processing requirements for such a system, including I/O buffering, real-time and background processing, etc.
   – Define the memory speed/size tradeoffs and design an appropriate memory architecture (e.g., register memory, cache, RAM, and mass storage).
   – Define memory usage patterns for this application (e.g., real-time sample rates, processing packet sizes, etc.) and simulate the memory performance.
   – Results should include register/cache hit rates for a set of memory size alternatives.
   – (optional) Investigate a variety of caching strategies (e.g., LRU, LFU) for your simulation to see how they might influence performance.
Potential projects

5. Interrupts are an integral part of any system that has to perform a set of independent tasks, particularly where some of these tasks have real-time performance requirements.
   – Identify a multitasking or real-time application where an ARM-like controller might be an appropriate solution
   – Identify a set of tasks, along with their priorities, that this system may be called upon to perform.
   – Define a scheduling process and interrupt structure to ensure tasks receive adequate resources. Verify that the system meets the performance requirements via simulation.
Potential projects

6. Most embedded systems will have a variety of memory-related resources and a variety of memory devices. For instance, RAM is needed for the stack as well as buffer and variable storage. Memory-mapped I/O devices require their own memory addresses. ROM, EPROM, flash memory, or battery-backed RAM is needed for vectored interrupt addresses, program storage and boot loader code.

- Define a representative application for an embedded system and define the memory requirements.
- Define a “memory map,” locating various memory addressed devices.
- Design a scheme to efficiently access the various memory devices using minimal hardware and incurring minimal delays. Your design should be flexible enough to deal with changes in system requirements that are certain to develop after system deployment without requiring expensive hardware changes.