# EE432 Advanced Digital Design with HDL
# Term Project

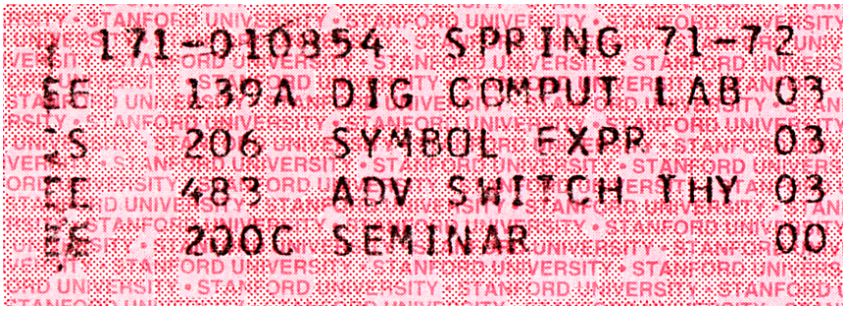
Spring 2013
Instructor — Tom Almy

I (the instructor) am always looking for ways to make the courses interesting, exciting, and educational. I've found that projects are a great way to learn a subject. This year we will have a team project to implement the PDP-8 Minicomputer. Don't worry, the PDP-8 is no match in performance to an Intel i7, but it represented low cost ($18,000) computing power in the 1960's. We are going to build a subset of a full PDP-8 system.

I have had experience building a computer in an academic setting, in 1972. When I was in college I took a course where 10 students designed and built a computer out of many hundreds of integrated circuits over the course of 10 weeks. Each student was assigned a component of the system and had to work out with other students how the components would interface together. The components were assembled on breadboards in suitcases, which were brought together near the end of the term to get working together. A program was written by one student that calculated prime numbers. It was a 16 bit computer which, because of its very slow design, took 24 hours to find all the prime numbers less than 65535.

We have both advantages and disadvantages compared to that class. The major advantages are:

- The architecture is defined
- We can simulate operation to uncover errors early in the design process.
- 40 years of technical advances mean we can design and "wire" the system in a program, rather than mounting and wiring hundreds of parts.
- You get four credits for this (although there will be some other assignments) and I only got 3.



One potential problem that we are going to handle from the beginning is a student's failure to complete their assigned component. Luckily, this didn't happen in 1972, when if any student didn't get their component working the whole project would have failed. To avoid that problem, all components will have two people assigned who will work independently. Their components should share the same interfaces so that they can be interchanged. We will call these teams “A” and “B”. We should end up with two PDP-8 systems, one consisting of team “A” components and the other consisting of team “B” components. Note that there is expected to be no correlation between team names and final grades.

Some terms we will use:
- *Designer* - a student in EE 432
- *Design* - A PDP-8 computer "clone".
- *Component* - a part of the design done by a single, or sometimes a pair, of designers.
- *Team* - All of the designers working on a single design. We will call these teams “A”, “B”, and in the case of large enrollment “C”.

If a student, say a team “A” designer, drops the course or otherwise fails to complete their component in a timely manner, the team of “A” designers can use a component from another team, a “B” component. The team could also decide to redivide the work load.

A tentative set of interfaces for each component will be provided. If a designer feels they need to change the interface, they must consult will all other designers in all teams that use that interface to reach consensus. In addition, it will be necessary for the team to combine components for testing, which will necessitate collaboration during the design process.

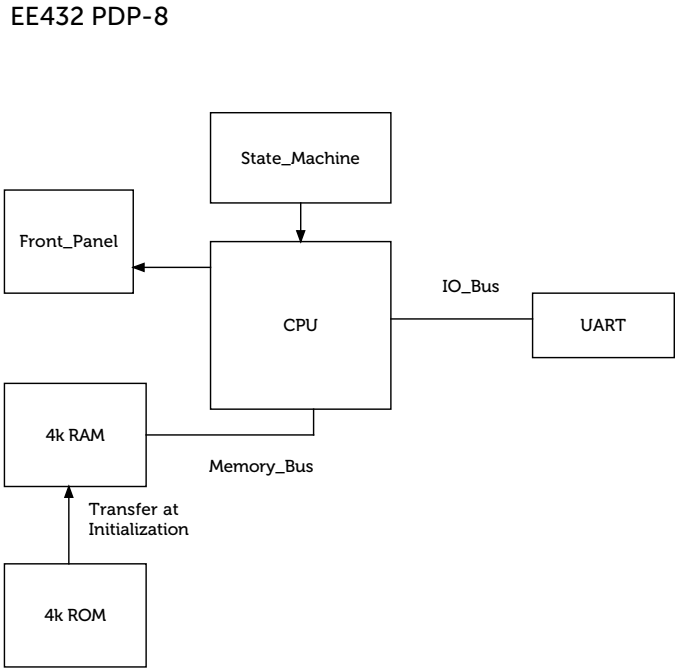
So lets have a quick look at the project.

# The Project

A number of documents are on the distribution DVD which pertain to the PDP-8 design.

- **PDP-8 - Wikipedia.pdf** The PDP-8 page on Wikipedia as of July 11, 2008. Because Wikipedia is a moving target, I’ve extracted the page here so we all see the same thing. This is a good overview of a PDP-8
- **PDP-8 Memory Reference Instructions.pdf** The description of the operation of the memory reference instructions.
- **PDP-8 Microcoded Instructions.pdf** The description of the remaining instructions, except for the Extended Arithmetic Unit, which does multiplies, divides, and extended shifts.

- **PDP-8 Console Teletype Instructions.pdf** Documentation of the IOT instructions for operating the teletype.
- **InstructionSet.txt** Another explanation of the PDP-8 instruction set.
- **68HCS12Text_Partial.pdf** Chapters in my microcontroller textbook on binary arithmetic and how a microcontroller executes instructions (hint— the PDP8 works basically the same way!) If you know nothing about computer architecture, read this early on.
- **UART_Design.pdf** Instructions on creating a UART, which was done by all DIgital Systems students for a couple of years before the PDP-8 assignment. The UART Designer may find this useful.
- **Implementation_Details.pdf** — Suggested interfaces between modules and some additional hints and suggestions for the various designers. Start here in your implementation.
- **Multiply_and_Divide.txt** — The EAE multiply and divide instructions.
- **InstructionSet.txt** — Another view of the instruction set
- **Small_Computer_Handbook** — basically a PDP-8 users manual, dating from, coincidentally, 1972. It can get dense at times, but it is the definitive reference. Only sections useful to this project have been included. The only documentation for the Extended Arithmetic Unit is here.
- **PDP-8 Memory.JPG** and **PDP-8 Block Diagram.JPG** Included for reference, we won't architect our PDP-8 like this! Besides, we don't have any magnetic core memory.
- **Adept IO Expansion rd.pdf** to supply our front panel controls.
- **Programming Tools** — The student who does the software task will be particularly interested in the contents of this folder. It contains the assembler program, a simulator, the binary tape loader programs, and two example, functional programs that calculate prime numbers. The student will have to rewrite at both of these in PDP-8 assembler. One of them uses the Extended Arithmetic Unit. The other calculates primes and displays there values without doing any multiplies or divides, but is a more complicated algorithm.




Our PDP-8 will have 4096 12-bit words, which simplifies its design. When documentation refers to changes in the PDP-8 design, we will assume we are implementing a design prior to the PDP-8/E (such as a PDP-8/I). However our

design will not have interrupt support or Direct Memory Access (which Digital Equipment Corporation called “Data Break”). We will simulate most of a front panel using the Adept I/O Expansion facility. All computers used to have front panels to operate them manually. No computers do anymore. We need the front panel to observe operation during debugging. Now days these tasks, and more, are performed by the operating system.

# The Goal

The team will be successful if they can do the following with their design:

1. With the PDP-8 design loaded into a Nexys 3 board, and the Nexys 3 board connected to a PC via the RS232 (COM) port, start the RIM loader program initially in the PDP-8 memory via the front panel.
2. Send the PDP-8 prime number program in RIM format from the PC's terminal program to the PDP-8.
3. When the program has been transferred, start the program running from the front panel of the PDP-8.
4. All prime number less than 4095 are calculated and printed on the PC terminal program display.

ASM12 (from the Microcontroller class) or other program can be used as the terminal program.

# The Components

The first evening of the class students will pick the component they wish to work on and the team. A table showing each students component assignment will be on the forum site, **http://oitclass.com/forums**. The forum can be used for communication among team members, although individual teams may set up other methods if they wish. Students may decide to swap their components (or teams). Swapping may occur at any time up to the end of the second class session.

All designers are expected to finish their component designs and have VHDL that compiles without error but otherwise untested by week 6 (unless otherwise specified). Why so early? Because debugging is a major time consumer. All components should be tested individually and in small combinations (such as CPU and state machine) by week 8. This will give two weeks to combine all the components, wring out the remaining bugs, and demonstrate operation with the prime number program.

The sole reason teams in past years haven't completed the project on time has been procrastination. For this reason, the week 6 items must be turned by the week 6 class meeting, and week 8 items must be turned in by the week 8 class meeting. Please submit any pertinent VHDL module files, test benches, Design summaries (they are HTML files in the project folder), and screenshots of simulations in a ZIP archive. There is no partial credit for these items.

Designers for the CPU and state machine need to be able to work together outside of class time as necessary since their components interact strongly with each other. All designers must work together in the final design debugging to ensure project success!

There is one unique component assignment. It requires no hardware design for the project, but still has deadlines for weeks 6 and 8. In return, for the right student this is probably the easiest task, and if successful will result in an A grade for the project.

## 1 Software Component

A student requesting the software component must either have some assembler programming experience (such as having successfully taken the Microcontroller sequence) or a strong desire to self-teach. Note that this student will still have to complete the VHDL homework assignments, so there is no getting out of digital system design. This position is so critical that if the student successfully completes all tasks an A grade for the project is assured even if the the final design doesn't function correctly.

The Software Component Designer has the following tasks:

1. Write a C program which converts PDP-8 assembler output into a block memory initialization file (".coe"). This will allow running PDP-8 code without having to load from the front panel or "Teletype".
2. Using the C program, preload the RIM loader so that it doesn't need to be loaded from the front panel. The RIM loader will be used to load programs from "paper tape", in our case the download function of a terminal program.
3. In the course of learning the PDP-8 instruction set, write some small programs that exercise all of the instructions and all of the memory addressing modes. Verify operation in the simulator. The CPU and State Machine designers will need these programs for testing. Contact the designers so that at least one program is available when they are ready to start debugging their designs. Items 1-3 are due at week 6.
4. Convert one or both of the prime number programs to PDP-8 assembler. Test the programs using the PDP-8 simulator. There must be a working program by the time it is needed in week 8. Supplying both programs increases the chances that the team will be able to get at least one working program on the hardware.

In order to be successful, the Software Component Designer must successfully pass the programs to the teams so that they can be used.

## 2 CPU Component

The CPU component consists of the registers, arithmetic and logic functions, and data paths of the processor. The architecture has been defined. The CPU component is to include the components necessary to implement the extended

arithmetic unit functionality.  Each team will have a designer for the CPU component. The tasks are:

1. Write VHDL for the component. This must successfully compile by week 6.
2. Write a test bench that executes all the data paths by week 6. The test bench must successfully execute by week 8.
3. Must collaborate with the State Machine Designer for the team in writing a test bench which has the CPU, state machine, and a behavioral model for the memory. The test bench will be used by the State Machine Designer.
4. Must collaborate with front panel designer as to the interface design between the CPU and front panel.
5. Must collaborate with the EAE component designer in providing the necessary CPU features for the EAE component.
6. Must change the CPU if necessary to meet any unforeseen needs of the State Machine and Extended Arithmetic Unit Designers.

### 3 State Machine (CPU controller) Component

The state machine component consists of a state machine which controls the CPU. The Designer may wish to divide the state machine into two or more state machines to accomplish the tasks. Each team will have a State Machine designer. The interface to the CPU is defined, but might need to be changed to complete the tasks. The tasks are:

1. Must implement the entire instruction set — the memory and microcoded instructions.
2. Must write a test bench to check the basic operation of the state machine by week 6.
3. Must collaborate with the CPU designer for the team in writing a test bench which has the CPU, state machine, and a behavioral model for the memory. This is due by week 8.
4. Must use the test bench of task 3, in combination with tools and test programs provided by the Software Designer, to test for successful operation of each instruction. Should work with CPU designer in correcting any errors in the CPU, but it is the CPU Designer's responsibility to maintain the CPU VHDL code.
5. Must collaborate with front panel designer for run/stop control of CPU.
6. Must collaborate with the EAE component designer, as necessary.

### 4 EAE Component

The Extended Arithmetic Element Unit exists as added registers in the CPU and a state machine, perhaps an additional state machine, to implement the instructions.

The designer of the EAE has perhaps the most challenging task. The designer must make sure the CPU component is sufficiently robust to support the EAE operations. The designer must also implement a state machine, or contribute to the CPU

controller state machine effort to implement the instructions. As a perhaps worst case, the EAE designer may want a separate module with the additional registers and control circuits, and working with the CPU components designers define an interface between their respective components. The goal (the EAE instructions) are well defined, but the approach to achieving the goal is completely up to the EAE component designer.

Note that only the instructions that multiply, divide, and access the EAE registers (to allow use of the multiply and divide instructions) need be implemented. Specifically these are the CLA, MQA, MQL, SWP (MQA and MQL combined), MUL, and DVI instructions. Only mode A is supported.

The EAE component designer has no specific deliverables, but relies on the submissions of the CPU and state machine designers. (This can be viewed as a blessing or a curse.)

If there are six persons on a team, the EAE component design tasks will be done by the two CPU designers.

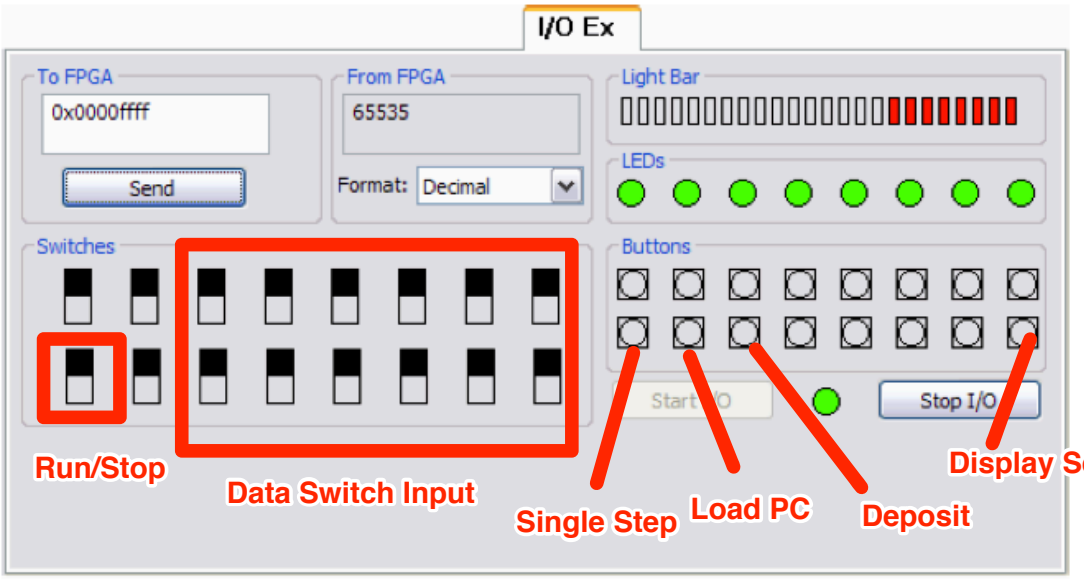
## 5 UART Component

The UART (Universal Asynchronous Receiver/Transmitter) is the "serial port" that provides a user interface as well as a way to load programs from a connected personal computer. Each team will have a UART designer. The interface to the UART is defined, but might need to be changed to complete the tasks. The tasks are:

1. The interface is to operate at 9600 bits/second, with 8 data bits and no parity.
2. Must write a test bench to check the basic operation of the state machine by week 6. The test bench needs to model the IO bus interface to the CPU as well as the serial port connection.
3. Must realize the serial port in the FPGA, with a state machine to "loop back" characters from the receiver to transmitter using a state machine to manipulate the control signals which would come from the CPU module. The implementation is to be tested by connecting to a PC to verify operation. This is due by week 8.
4. Must assist as necessary with integration and testing with the project as a whole. It is the UART designer's responsibility to maintain the UART VHDL code.

## 6 Front Panel Component

The front panel provides the only physical interface to the computer. At one time all computers had a front panel which was used for loading programs by hand, and for debugging. We can be a bit more advanced since we can load programs initially in the ISE by providing an initial value to the RAM contents. We are also limited in the number of switches and lights provided on the Nexys 3 board so we will use a virtual front panel provided by the Adept IO Expansion module.




I propose the following, which can be modified by the designers if they see a better approach:

- 7-segment LED display shows the content of a selected register as an octal value. One of the pushbuttons cycles through the registers.
  - Accumulator with link bit displayed as decimal point
  - MQ register
  - Program counter
  - Memory contents at the address set in the switch register
- 4 LEDs indicated the register selected on the 7 segment LED display.
- 1 LED to indicate the CPU is running.
- 12 Virtual slide switches for data input.
- One virtual switch for run/stop
- One virtual button to execute a single instruction
- One virtual button to store the data switch register value into the program counter to set the starting address, called “Deposit”.
- One virtual button to store the data switch register value at the location specified by the Program Counter, and then increment the Program Counter.

The tasks are:

1. Must design front panel interface.
2. Must collaborate with CPU and state machine designers as appropriate.
3. Must have front panel interface with “dummied” CPU and state machine interfaces running by week 6. This is actually the earliest due date for designs running in the Nexys board.

4. Must be ready for testing with other components by week 8, will all interface details worked out.

### 7 Memory System Designer and Top Level Integrator

The memory system component consists of the external RAM memory, which is to be used by the PDP-8 as it's 4k x 12 RAM, and an internal Block Memory configured as ROM. When the PDP-8 is initialized, the contents of the Block Memory is to be transferred to the external RAM memory.

As the Top Level Integrator, the tasks are mainly managerial: keep in contact with all designers to make sure all interfaces (ports) are consistent, merge all component VHDL files into a single Xilinx Project, and turn in the all the design files at the end of the term.

The tasks are:
1. Successfully read and write to the external RAM by week 6. The designer will need to devise logic in the FPGA to test the operation or perform a timing simulation of the RAM in a test bench. Running on the hardware is preferred.
2. Must collaborate with the CPU and state machine designers on an interface. Handshaking may be needed between the components, if not between the designers.
3. Complete the initialization from Block Memory and the interface to the CPU/ State Machine by week 8.
4. Assist as necessary with the final integration.

### One Extra Component

An additional component, which implements the IO "bus" posed a problem. Was it part of the CPU, the UART, or the Top Level? I solved the problem by providing it myself. It's the module *IOT_Distributor.vhd*.