Lecture 1

1.
   1. Moore’s law predicts the increase in the number of transistors increases (that it would double every 18-24 months) whereas Dennard scaling predicts the decrease in the size of individual transistors (that they would shrink by 30% every technology generation leading to a 50% decrease in area). Moore’s law currently holds, but Dennard scaling stopped holding in around 2006.

   2. In accordance with Moore’s law transistor density is increasing. Therefore, to reduce power consumption, the number of transistors that are switching needs to be constrained. The “power wall” is the issue of having more transistors than in the previous technology generation, but having the same power budget, so still only being able to have the same number of switches.

      One way to address this is with accelerators – hardware optimised for specific tasks which can be turned off when not in use (e.g., for video decoding). Another way to help the problem is having multiple types of processor core: some small efficient cores and some larger cores for heavy workloads. This would reduce overhead power consumption when the workload is light.

      Furthermore, active power is equal to capacitance times voltage^2 times clock frequency times activity factor. If capacitance and activity factor must remain constant, there is an option to reduce power consumption by reducing the operating voltage or the clock frequency, both of which will result in a subsequent decrease in performance.

   3. Moore’s law does indeed hold for non-volatile (flash) memories.

Lecture 2

2.
   1. Timing analysis allows the designer to ensure that all the components in their circuit will work properly together. For example, it can tell you whether the output of one component will be available and stable by the time it needs to be used as an input for another component. It is also used to give a range of conditions in which the circuit will be usable such as a range of clock frequencies, temperatures, etc. It also helps measure the effects of clock skew.

      It can determine the path through the circuit with the greatest delay (critical path) by finding the longest combinatorial path in which the output needs to be stable before the start of the next clock cycle.

      Using this delay and a detailed physical model, timing analysis can tell you whether a given clock frequency is safe, and so by gradually increasing it and analysing again, you an determine the maximum safe clock frequency of the circuit.

2. The SystemVerilog source code is first converted into a gate-level netlist. At this stage it undergoes Boolean optimisation, optimal state assignment, and retiming for timing closure. Each component is then allocated a physical location on the FPGA (Place & Route) in accordance with area and timing constraints. Timing analysis and gate-level simulations are performed to ensure that the design still works with these gate-level components in their allocated locations.

3. Synchronisation is important because it allows each component to only have to worry about complying with the clock restraints, rather than complying with the timing constraints of every component around it. This is because if the circuit is synchronised and if all the components comply with the clock restraints, then they must all by necessity comply with each other's timing restraints.

   Furthermore, synchronisation makes timing analysis possible to confirm the above.

Lecture 3

3.
  1.
    1. Designing for Moore's law allows designers to make more efficient use of the large number of transistors available to them. Assuming that Moore's law will continue to hold also allows for a well-structured development protocol such as Intel's "tick tock" model in which the "tock" phase relied on the assumption that there would be improvements in silicon technology.

    2. Using abstraction brings the act of designing a system closer to the cognitive model that a human designer would have when thinking about the design. That is to say that a designer might think about the big-picture operation of the system, and then separately think about the lower-level implementation details. Abstraction allows the design itself to be separated in the same way.

       Abstraction also increases designer efficiency because large pre-built elements can be composed at the higher-level.

       Abstraction also reduces bugs, as you can design small safe protocols at the low level and then combine them with less risk at the high level.

    3. Making the common case fast reduces overall/average execution time of a program much more than an equal or even greater improvement in an uncommon operation. It is important to bear this in mind while designing because it can help shift your focus to the task which will result in the best overall improvement to the system.

       For example, it can be easy to expend a lot of effort and time vastly improving the efficiency of an operation which will be used very rarely, when more would be gained by spending that same time and effort on a lesser improvement to a more common operation.

4. Parallelism is important because it allows for more efficient use of the computer’s resources.

   For example, instruction-level parallelism allows for more of the hardware to be used at once to execute multiple instructions (although this only applies when it can create the illusion of sequential operation to the program).

   Thread-level parallelism allows for a new programming paradigm resulting in software which is optimised for concurrency.

   Data-level parallelism allows for vectorised instructions which can vastly reduce the overhead which comes with loops, iterations, and control flow.

5. Pipelining is similar to instruction-level parallelism in that the fetch-decode-execute cycle is split up into sections, so that the processor does not have to wait for one instruction to finish before starting to process the next one.

6. Prediction allows for a performance gain by making sure that data which will likely be needed soon is nearby (making it faster to access), predicting that certain values in memory will not change, predicting which instructions will be executed next, etc. Depending on how accurate these predictions are, pipelining can become more efficient.

7. You can only have so much memory close to the processor, and memory far from the processor can take many clock cycles. As such, a hierarchy of memories allows the computer to make some memories closer to the processor (for data which is currently being worked on) and some further away (which is accessed infrequently or likely won’t be accessed soon). This is an effective trade-off between memory size and access speed.

8. Redundancy allows for data to be recovered if slightly corrupted or lost (e.g., error-correcting codes). This means that if a CD get’s scratched or there is interference on a wire, the data stored/transferred can be reconstructed when needed.

   Redundancy is also used for important operations, e.g., instructions can be run on multiple different processors in case some of them fail.

   Furthermore, safety-critical sections of code can be written multiple times independently by different teams to the same specification, just in case one implementation contains bugs.

Lecture 4

4.

1. RISC is a free ISA designed to fill the gap in the market for free/open alternatives to the proprietary x86 and ARM ISAs. It is designed to be a good compilation target (hence reduced instruction set) which is simple to decode. It is modular but extensible with several standard extensions. It is also designed to be stable, making it a good candidate for the standardisation of ISAs.

2. RISC-V is an instruction set specification, not a processor. It acts as an interface between the processor and the software.

3. A pseudo instruction is an assembly instruction which isn't mapped directly to its own unique machine code instruction but is instead just shorthand for an existing (more complicated) conventional assembly instruction which might be more difficult for the programmer to remember or just clunkier to use.

4. R-type instructions contain fields funct3 and funct7 which encode the instruction function. Since these can have many values each, there can be many different R-type instructions per opcode. Whereas, with U-type instructions there is only one per opcode.

5. A load/store architecture is when an ISA's instructions are separated into two distinct categories – load/store instructions which use both the memory and the registers, and ALU operations which only use the registers.

6. A function calling convention is a standard way in which different registers are used by the compiler and linker when a subroutine is called. Since RISC tries to keep all of the registers general purpose, calling conventions give additional names to the registers which the programmer can use in assembly, and which represent how the register is going to be used (e.g., x1 is also called ra to indicate that it is used as the return address by the ret pseudoinstruction)

Additional Questions

1.
   a.
      i. In accordance with Moore's law, transistors became exponentially cheaper, and therefore more numerous on chips. This meant that instructions could take fewer clock cycles by instead using more transistors.

      ii. The phenomenon of Dennard scaling meant that individual transistors were getting smaller by 30% in each direction every generation, which reduced the delay across them by 30% and so increased their operating frequency by 40% every generation. The fact that transistors were able to operate faster allowed for an increase in clock speed.

   b. These annual gains slowed down due to power limitations, the limits of instruction-level parallelism, and memory latency constraints.

2.
   a. An FPGA is a programmable circuit made up of configurable components. These are supposed to be configured by a designer after manufacturing in order to make the circuit perform a given task.

   b. A general-purpose processor can fall victim to the "Turing tax". They have many transistors which are used to simply move data around, and relatively few for actually doing computation.

FPGAs give the designer much lower-level control which can mean cutting down on the overhead required for general-purpose computation.

c. I believe that this statement only holds true for simple algorithms (which, admittedly, is what FPGAs would tend to be used for) but once the designs get complicated (e.g., requiring buffered external inputs, persistent data storage on an external device, complicated mathematical operations) the sheer number of slower components you would need on an FPGA would outweigh the overhead gained by using a general-purpose processor.

3. Small memories can be closer to the processor which means it takes less time for the signals to travel between the processor and the memory.

4. Having complex addressing modes can allow you to encode more complicated functions in fewer instructions. For example, adding two registers and applying a shift can be done in a single ARM instruction whereas it would take multiple RISC instructions. This is particularly useful when compiling object-oriented code. However complex addressing modes can lead to register update side-effects.

5. This makes it clear whose responsibility it is to store the state of the registers. For example, when a programmer is writing a subroutine, they know that they do not need to include instructions to save the state of the temporaries. For these registers, that is appropriate because the caller may not even need to remember the state of the temporaries, and so it should be up to the caller to save them if necessary. However, for other registers such as the saved registers, it more likely that the caller will need to remember their state, and unclear whether the callee will even modify them, so it should be up to the callee to save their state if they’re going to be modified.