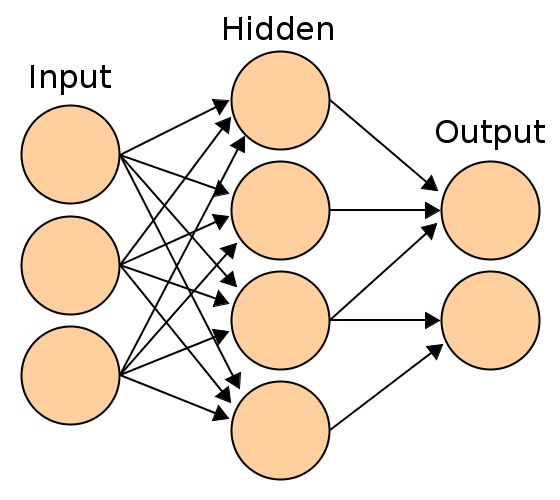
**FPGA Systems Speed analysis**

**Reason for not having so much analysis in this area:**

              In a normal industrial situation when using an FPGA, all operations are coded, simulated, and tested prior to choosing an FPGA model. However, due to our time constraints, coding and testing before designing the circuit board is not feasible. Hence, we are shooting for the best and using the resources we have to as best as possible, guesstimate the needs of our operations on this FPGA.

**Analysis:**

Traditionally, the term neural network had been used to refer to a network or circuit of biological neurons. The modern usage of the term often refers to artificial neural networks, which are composed of artificial neurons or nodes. Artificial neural networks are made up of interconnecting artificial neurons (programming constructs that mimic the properties of biological neurons). Artificial neural networks may either be used to gain an understanding of biological neural networks, or for solving artificial intelligence problems without necessarily creating a model of a real biological system. The real, biological nervous system is highly complex and includes some features that may seem superfluous based on an understanding of artificial networks.  The cognitive modelling field involves the physical or mathematical modeling of the behaviour of neural systems; ranging from the individual neural level (e.g. modelling the spike response curves of neurons to a stimulus), through the neural cluster level (e.g. modelling the release and effects of dopamine in the basal ganglia) to the complete organism (e.g. behavioural modelling of the organism's response to stimuli). For more detailed info about neural networks please see external sources such as Wikipedias or GCCIS Faculty.

        This design shown below in figure 1 was implemented on a Digilent Basys Spartan 3E-100 developement board. It currently performs the function of XOR, however has no heuristic coding to help out. Instead it uses the theory touched on above to learn acceptable and unacceptable responses to input. This is not a simple design.  Results of this network is outputs with in 10% of the goal values for "high" and "low". These values are relatable to digital logic values in hardware and can be used as such. The results of the implementation in VHDL using the Xilinx WebPack ISE are shown below in table 1.

|  |  |  |  |
| --- | --- | --- | --- |
| Total resources used: 19% | **Device Utilization Summary** |  |  |
| Logic Utilization | Used | Available | Utilization |
| Number of Slice Latches | 22 | 1,920 | 1% |
| Occupied Slices | 197 | 960 | 20% |
| 4 input LUTs | 360 | 1,920 | 18% |
|  | Logic | 328 | 17% |
|  | Route-thru | 32 | 3% |
| Number of bonded IOBs | 10 | 108 | 9% |
| MULT18X18SIOs | 4 | 4 | 100% |

Table 1: Device Utilization table for Spartan 3E-100

|  |  |  |
| --- | --- | --- |
| **Node** | **Levels** | **Time** |
| Data In | 29 | 13ns |
| Data Out | 2 | 5ns |

Table 2: Data speeds

Looking at the data from Table 1 we can see how little resources this design took up a little less than 19% overall. This is not much considering the complexity of the design and the simplicity of the Spartan 3E-100 FPGA. For example the last row of table 1 shows that only 4 built in hardware optimized 18x18 bit multipliers exist in this device and all are used. For sure this design does a substantial amount of math to calculate the weights on the connections between neurons and more ALUs will need to be created using general purpose slices as can be seen in the usage of slices being about 20%.

This design was very fast and was able to process changes in inputs very quickly. We can see that 29 levels of logic were needed to be traversed from the input to the end of the processing pipeline. However this only takes 13ns, we can calculate the frequency of this to be 76.92MHz. This number implies that we can handle 76 million changes of inputs per second on each pin that has input to this logic design. From this point in the design, to get to the output is only 2 levels and takes 5ns (or a speed of 200MHz). This is quite speedy on a device that we released to be a low price, slowest device in the product line in 2005. There are not many people that are still using computers considered "hot items" in 2005 (think first dual core processors, Celeron D, ...). If the internal logic was simpler, or more pipelined we would be able to reduce these speeds in final design.

Correlation to this project:

    Unfortunately we have not been able to simulate this design on a Spatan 6LX75T core. We have been having problems getting the Xilinx software setup. However we can attempt to build a ratio between the two devices. For example, the table below compares directly, the resources that both devices have available. As you can see there is a large increase on "on die" resources available. This does not mean that we can do the same task with less resources, just that the design will take up less space on this model. However this model is built using a different process of creating transistors, and thus will be able to run faster since the length of distance between individual elements is smaller on the same chip (up to a limit). We can see that the standard clock speed of the spartan 6 is 2.5 times that of the tested unit. This will directly correlate to the speed of the device running from input to output. However this is not a 1:1 ratio, we cannot say that because the clock speed is higher it will run at 2.5x that of the Spartan3. But we can say that that speed available with the specific sequential calculations will be higher than that to some degree above 1 and below 2.5x. There are various factors to consider in this calculation, including the amount of actual processing that will be done (currently unknown), the clock speed of the other components in the design (memory, SATA, etc.). However, due to the amount of resources available and the low-end of the speed spectrum we know (250MHz), we can estimate that the speed will be closer to the 2.5x.

For example, say we are processing a pixel and we need to do X amount of math that takes Y seconds. Lets say that Y is longer than 1/30th of a seconds (IR camera picture rate), this will cause a problem that we cannot process pictures fast enough. We will solve this not by making the FPGA faster, or have a higher clock speed, but instead by parallelizing the math done in X. This will reduce the time Y needed to process the pixel. Now, we realize that only so much can be done in parallel, and that we will not be utilizing all of the resources of this large FPGA. We will solve this next problem by parallel processing of multiple pixels simultaneously. Theres, no reason why we wouldnt be able to just say copy the image pipeline above (A) and create another one called B. From this we can say after the pixel data is recieved for the next image, and pipeline A is not done yet, we can start the processing in B. This technique is very scalable, so the amount of processing we do is directly proportional to the number of parallel pipelines we will need to process all of the data in "real time".

