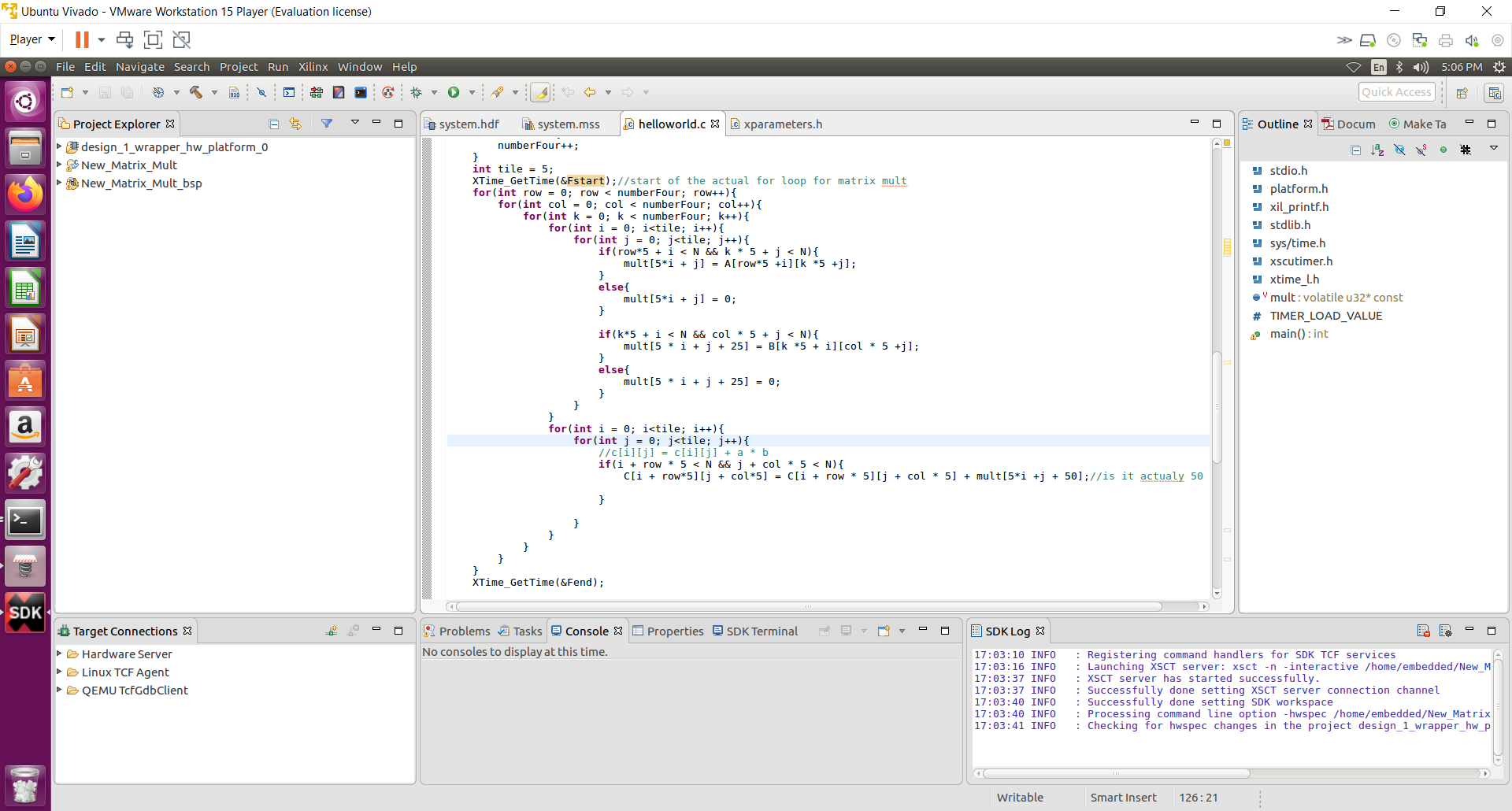
Jonathan Barnes

Project 4 report

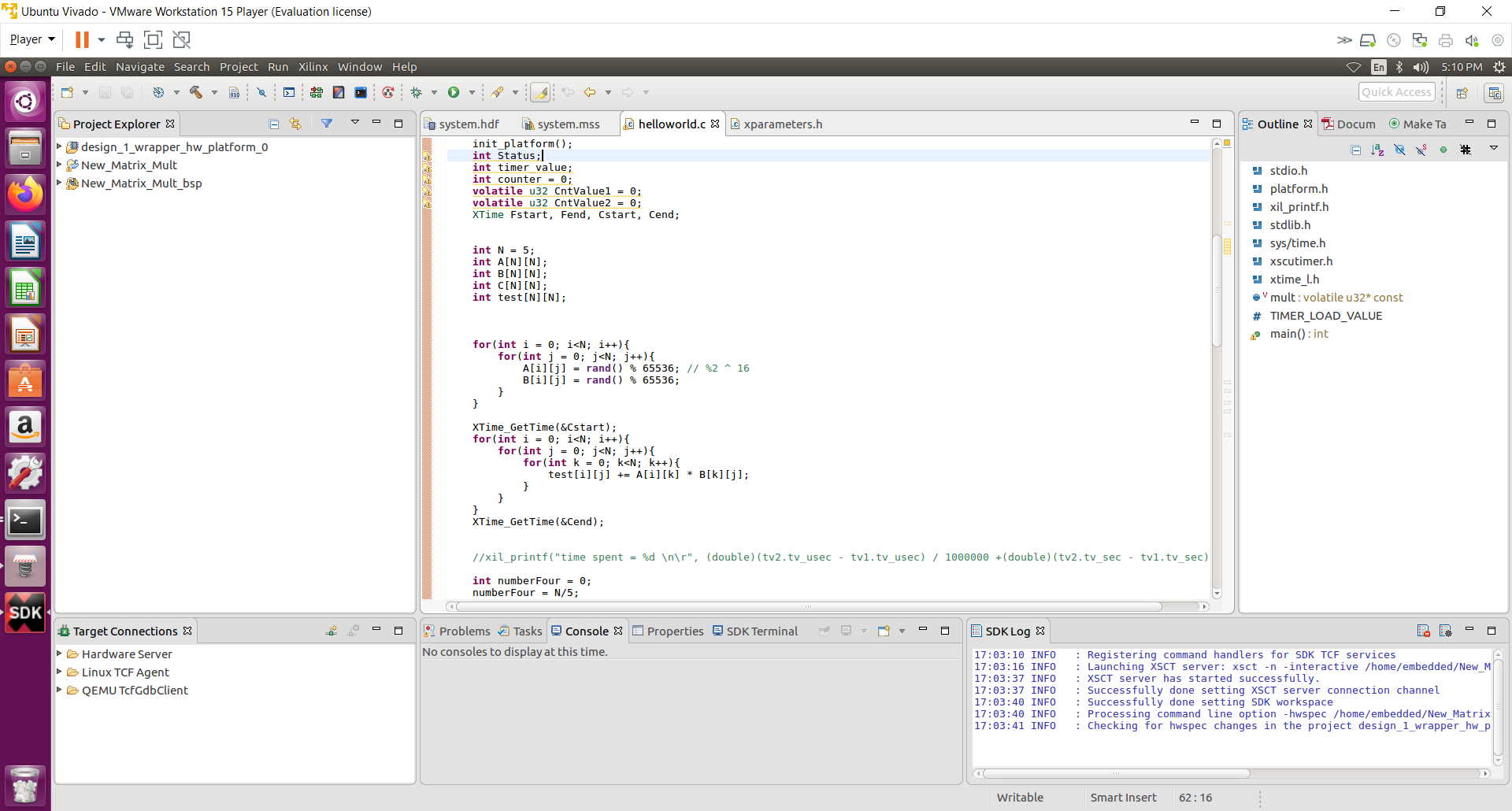
**Design Summary:**

This project involved 2 different implementations of matrix multiplication. One implementation in c cod that ran on the cpu, and one implementation that ran on the FPGA. The Software implementation was very simple, I simply created three for loops that went over the N elements of matrices A and B, multiplied the row of A with the column of B and stored the value in array temp(picture of code below). I then started on my hardware implementation of the matrix multiplication. The method that I ended up using was the tiling method. The tiling method breaks up the arrays A and B into square tiles and does matrix multiplication on each of those tiles, once a multiplication is done the tile on A shifts one to the right and the tile on B shifts one down, the result of this multiplication is summed with the previous one, at the end you have completed one tile of the output matrix C. For instances when the matrix being sent in didn’t fit perfectly with my tile size I sent in 0s to fill the empty parts of the matrix since those don’t affect the output. (picture of the code for this below) The way I programmed the matrix multiplication in in vhdl was fairly bruit forced, I just sent in the full 2 matrices that were being multiplied, and typed out all the multiplications and additions for each part of the product matrix sent out. My final implementation of the hardware matrix multiplication multiplies 5 by 5 matrices but with tiling can do matrices of any size. The main bottleneck I faced in the hardware implementation was LUTs. In the first implementation I attempted, I was using the 30 by 30 matrix multiplication that professor talked about in her announcement, with sending in one row of A, and all of matrix B. But my implementation wound up using about 150000 LUT’s. I then changed to using the tiling method and send in both matrix A, and B. I started with a 4 by 4 which used about half of my LUTs, and I knew that by increasing the matrix size from 4 to 5 I would increase the number of multiplications from 16 to 25, which would result in nearly doubling the number of LUTs used, but I had enough and wound up using 5by5 for my final implementation.

VHDL Implemented:



CPU implementation below:



**Results Display:**

The C code is implemented in a way that allows for tiling. It finds the number of tiles that will be needed in each direction. then does a double for loop over a double for loop that loops over each of the large tiles. This double for loop is around another double for loop that sends each of the data elements from 5 by 5 tile of A and B we are looking at into the FPGA to be multiplied. If the tile doesn’t quite fit the size N then the program sends in 0s to the FPGA where the tile goes beyond matrix A and B. The output array is then read back and summed with the output tile we are looking at for the final array. After the multiplication the results are printed, and a check is performed to see if the matrix multiplication was correct, and if it was faster on the hardware implementation. Unfortunately as you can see from our results below the hardware implementation never actually got to be faster than the software implementation, this could be due to a variety of reasons. One reason is that the hardware needs to perform multiplication on larger matrices in order to begin out performing the software, I think this is believable looking at the graph of our timing, the hardware was getting closer to the software up to the 5 by 5 matrix size we implemented. Another reason for it being slower could be due to my tiling implementation, wen doing tiling you have to do a lot mor writing back and forth between the fpga and the cpu and that takes time.

2by2, 3by3, 4by4:

A screenshot of a computer screen

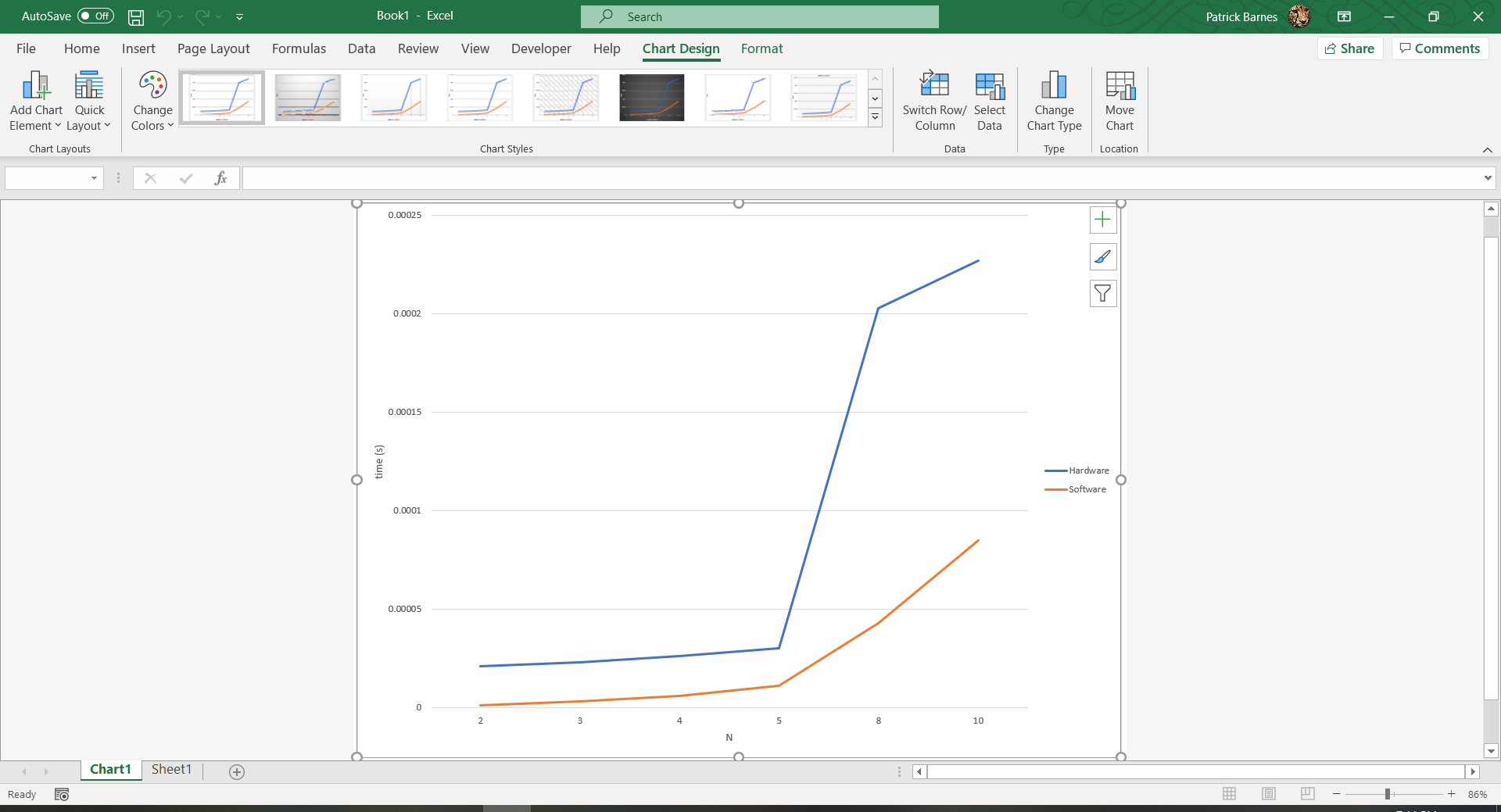
Description automatically generated

8by8, 5by5:

A screenshot of a computer screen

Description automatically generated

Graph of Data:



**Design Implementation Report:**

1. Hardware utilization

|  |  |
| --- | --- |
| VHDL Design |  |
| LUT | 1595 |
| LUTRAM | 60 |
| FF | 2206 |
| DSP | 5 |
| BUFG | 1 |

1. Power

Dynamic power = 1.41W 92%

* Sig <1%
* Logic <1%
* Clocks <1%
* DSP <1%
* PS7 96%

Static power = .117W 8%

1. Timing report

clock period = 20 ns

WNS: 1.2ns

Critical path = 20ns – 1.2ns = 18.8ns

1. Design Schematic

