

## CMPE 260 Laboratory Exercise 2

### Register File

Glenn Vodra  
Performed: February 2, 2022  
Submitted: February 16, 2022

Lab Section: 3  
Instructor Sutradhar  
TA: Ian Chasse  
Cole Johnson  
Robbie Riviere  
Jonathan Russo  
Colin Vo

Lecture Section: 1  
Professor Cliver

By submitting this report, you attest that you neither have given nor have received any assistance (including writing, collecting data, plotting figures, tables or graphs, or using previous student reports as a reference), and you further, acknowledge that giving or receiving such assistance will result in a failing grade for this course.



# Abstract

The purpose of this exercise was to design a two-read, one-write register file. This register file can be configured for the desired number of registers, as well as the desired bit depth, or size of each register. This register file also has a write enable that is active high. This register file is designed to write on the falling clock edge, and read asynchronously. The first register, register zero, cannot be written to. In addition to this design file, a self-checking test bench was also written. The test bench contained a record of twenty-six tests that tried to read and write data to every register in the design. This design was simulated using two tools, Vivado and Modelsim. Each of these tools produced the expected results and therefore it was concluded that the exercise was successful.

## Design Methodology

There are two aspects of the register file that were considered for this design. The first of these was the register module. This was generated using a data type called `mem_type`. `Mem_type` is a custom type that consists of an array of arrays. The first array contains two to the power of (`log_port_depth`). `Log_port_depth` is an integer that defines the number of bits available to address the registers. To test this design, a value of three was used. Two to the power of three provides eight addressable registers. The second array, contained within each of the eight addressable registers is the amount of data stored in each register. For this exercise, a value of eight was used, which gives each of the eight registers, eight bits of data. An instance of this type was then created and initialized to all zero;

The second aspect of the register was the register file functionality. The read functionality happens asynchronously so the data of the register can be assigned outside of a process. This is done by taking the address provided by the `Addr1` and `Addr2` signals and finding their data contents from the two-dimensional array described above. For the write functionality, a process was created that contained the steps needed. The first of these is to wait for the falling edge of the clock signal. The following logic is then executed if it is the falling edge. First check the one-bit signal `WE` (write enable, active high), and second check the address that is being written. The address, `Addr3`, must not be zero, because the zero register must always return the value zero. If all of these conditions are met, then the value of `WD` (write data) is put into the array at the address specified by `Addr3`.

## Results and Analysis

To ensure this design works, twenty-six test cases were created to test every register in the design and ensure the zero registers could not be overwritten. The first two simulations, the behavioral and post-implementation were run in Xilinx's Vivado, while the third, was run in software outside of Vivado; this being ModelSim. The following figure, figure one shows the simulation results of the behavioral simulation.



Once the behavioral simulation was run, a Post-Implementation simulation was run. Figure three, below, shows the timing for this simulation using the same test cases found in figure one.



The timing diagram displays the following signals and their values over time:

Signal	Initial Value	Value 1	Value 2	Value 3	Value 4	Value 5	Value 6	Value 7	Value 8	Value 9	Value 10	Value 11	Value 12	Value 13	Value 14	Value 15	Value 16	Value 17	Value 18	Value 19	Value 20	Value 21	Value 22	Value 23	Value 24	Value 25	Value 26	Value 27	Value 28	Value 29	Value 30	Value 31	Value 32	Value 33	Value 34	Value 35	Value 36	Value 37	Value 38	Value 39	Value 40	Value 41	Value 42	Value 43	Value 44	Value 45	Value 46	Value 47	Value 48	Value 49	Value 50	Value 51	Value 52	Value 53	Value 54	Value 55	Value 56	Value 57	Value 58	Value 59	Value 60	Value 61	Value 62	Value 63	Value 64	Value 65	Value 66	Value 67	Value 68	Value 69	Value 70	Value 71	Value 72	Value 73	Value 74	Value 75	Value 76	Value 77	Value 78	Value 79	Value 80	Value 81	Value 82	Value 83	Value 84	Value 85	Value 86	Value 87	Value 88	Value 89	Value 90	Value 91	Value 92	Value 93	Value 94	Value 95	Value 96	Value 97	Value 98	Value 99	Value 100	Value 101	Value 102	Value 103	Value 104	Value 105	Value 106	Value 107	Value 108	Value 109	Value 110	Value 111	Value 112	Value 113	Value 114	Value 115	Value 116	Value 117	Value 118	Value 119	Value 120	Value 121	Value 122	Value 123	Value 124	Value 125	Value 126	Value 127	Value 128	Value 129	Value 130	Value 131	Value 132	Value 133	Value 134	Value 135	Value 136	Value 137	Value 138	Value 139	Value 140	Value 141	Value 142	Value 143	Value 144	Value 145	Value 146	Value 147	Value 148	Value 149	Value 150	Value 151	Value 152	Value 153	Value 154	Value 155	Value 156	Value 157	Value 158	Value 159	Value 160	Value 161	Value 162	Value 163	Value 164	Value 165	Value 166	Value 167	Value 168	Value 169	Value 170	Value 171	Value 172	Value 173	Value 174	Value 175	Value 176	Value 177	Value 178	Value 179	Value 180	Value 181	Value 182	Value 183	Value 184	Value 185	Value 186	Value 187	Value 188	Value 189	Value 190	Value 191	Value 192	Value 193	Value 194	Value 195	Value 196	Value 197	Value 198	Value 199	Value 200	Value 201	Value 202	Value 203	Value 204	Value 205	Value 206	Value 207	Value 208	Value 209	Value 210	Value 211	Value 212	Value 213	Value 214	Value 215	Value 216	Value 217	Value 218	Value 219	Value 220	Value 221	Value 222	Value 223	Value 224	Value 225	Value 226	Value 227	Value 228	Value 229	Value 230	Value 231	Value 232	Value 233	Value 234	Value 235	Value 236	Value 237	Value 238	Value 239	Value 240	Value 241	Value 242	Value 243	Value 244	Value 245	Value 246	Value 247	Value 248	Value 249	Value 250	Value 251	Value 252	Value 253	Value 254	Value 255	Value 256	Value 257	Value 258	Value 259	Value 260	Value 261	Value 262	Value 263	Value 264	Value 265	Value 266	Value 267	Value 268	Value 269	Value 270	Value 271	Value 272	Value 273	Value 274	Value 275	Value 276	Value 277	Value 278	Value 279	Value 280	Value 281	Value 282	Value 283	Value 284	Value 285	Value 286	Value 287	Value 288	Value 289	Value 290	Value 291	Value 292	Value 293	Value 294	Value 295	Value 296	Value 297	Value 298	Value 299	Value 300	Value 301	Value 302	Value 303	Value 304	Value 305	Value 306	Value 307	Value 308	Value 309	Value 310	Value 311	Value 312	Value 313	Value 314	Value 315	Value 316	Value 317	Value 318	Value 319	Value 320	Value 321	Value 322	Value 323	Value 324	Value 325	Value 326	Value 327	Value 328	Value 329	Value 330	Value 331	Value 332	Value 333	Value 334	Value 335	Value 336	Value 337	Value 338	Value 339	Value 340	Value 341	Value 342	Value 343	Value 344	Value 345	Value 346	Value 347	Value 348	Value 349	Value 350	Value 351	Value 352	Value 353	Value 354	Value 355	Value 356	Value 357	Value 358	Value 359	Value 360	Value 361	Value 362	Value 363	Value 364	Value 365	Value 366	Value 367	Value 368	Value 369
--------	---------------	---------	---------	---------	---------	---------	---------	---------	---------	---------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------	-----------

Figure 4: *Modelsim Simulation*

Note, that instead of using EDA playground, Modelsim was used. This was communicated to be an acceptable method of simulation for this exercise. To ensure the results are consistent with those from Vivado, a snippet of figure four is used.

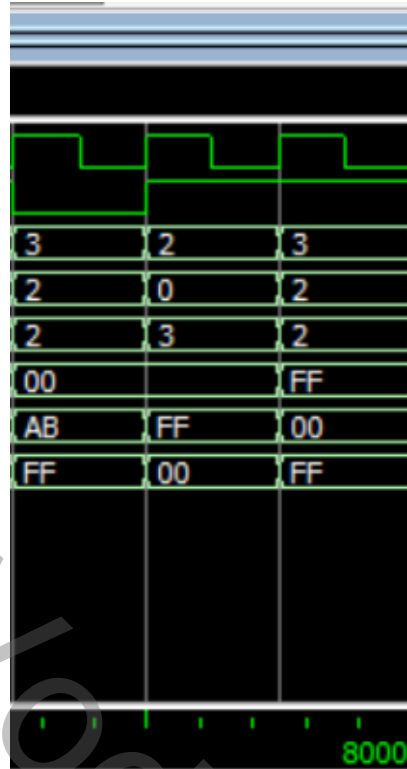


Figure 5: *Modelsim Example*

Figure five shows a different test, than the one shown in figure two, however, it produces consistent behavior for the operation of the register file. Note, the signals displayed are listed in the same order. In this test case, the write enable functionality is tested by trying to write the value “0x00” to register two when the write enable is low. In addition, the value stored in register three is shown as AB in the first period. In the next period, the value of register two is read. Register two still holds “0xFF”, which is the value from a previous write operation, and not the write that was attempted when the write functionality was not enabled. Also during this clock period write enable is set high and the value “0x00” is written to register three. In the next period, the value of two is checked again, and it is still “0xFF”. The value of register three however was changed, as the output is now “0x00”.

## Conclusion

This exercise was successful because a register file design was successfully created, and tested using multiple forms of software simulators. Registers or even broader, addressable memory are very important for a computer system; they allow data to be stored and used for operations. Ensuring that data is being accessed from the correct location and read/stored at the correct time is very important as a processor is developed over multiple exercises. As this design is expanded, it is also important that this register design can be quickly changed to work with different data sizes and a different number of data storage locations (addressable registers in this case). As demonstrated in this exercise, the read-and-write operations work as intended, no data can be stored in register zero, and the write enable works, disabling and enabling write operations.