

# EXERCISE 2 (Asynchronous FIFO)

Daniel Duggan

March 21<sup>th</sup> 2025

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Task 1</b>	<b>2</b>
<b>3</b>	<b>Task 2</b>	<b>3</b>
3.1	Logical Equations for Binary Code to Grey Code Conversion and Grey Code to Binary Code Conversion . . . . .	3
<b>4</b>	<b>Task 3</b>	<b>5</b>
4.1	Test Results from Simulation . . . . .	7
<b>5</b>	<b>Appendix</b>	<b>14</b>
5.1	Async_FIFO.vhd . . . . .	14
5.2	FIFO_write_control.vhd . . . . .	16
5.3	FIFO_read_control.vhd . . . . .	18
5.4	write_pointer_sync.vhd . . . . .	20
5.5	read_pointer_sync.vhd . . . . .	21
5.6	Async_FIFO_tb.sv . . . . .	22
5.6.1	Output Delayed by Extra Clock due to Output Registers . . . . .	26

# 1 Introduction

The purpose of this exercise is to design an asynchronous First In First Out (FIFO) buffer as shown in figure 1. Such asynchronous FIFO's are typically used in designs that have a CDC i.e. where communication is required between sections of a design that run on different clock periods. This report details the implementation of an asynchronous FIFO that can be used in such circumstances as outlined in the exercise sheet found here (task 3). Furthermore, it answers the questions asked in task 2 and task 1. All code can be found on github at url: [https://github.com/duggan9265/FPGA-Design-For-Communication-Systems/tree/master/Course\\_work/Ex\\_2\\_Async\\_Fifo](https://github.com/duggan9265/FPGA-Design-For-Communication-Systems/tree/master/Course_work/Ex_2_Async_Fifo) **Note: Updates have been made to correct the functionality of the async FIFO, which did not operate correctly originally. These updates are written in blue. Design changes are given via images from GitHub Desktop.**

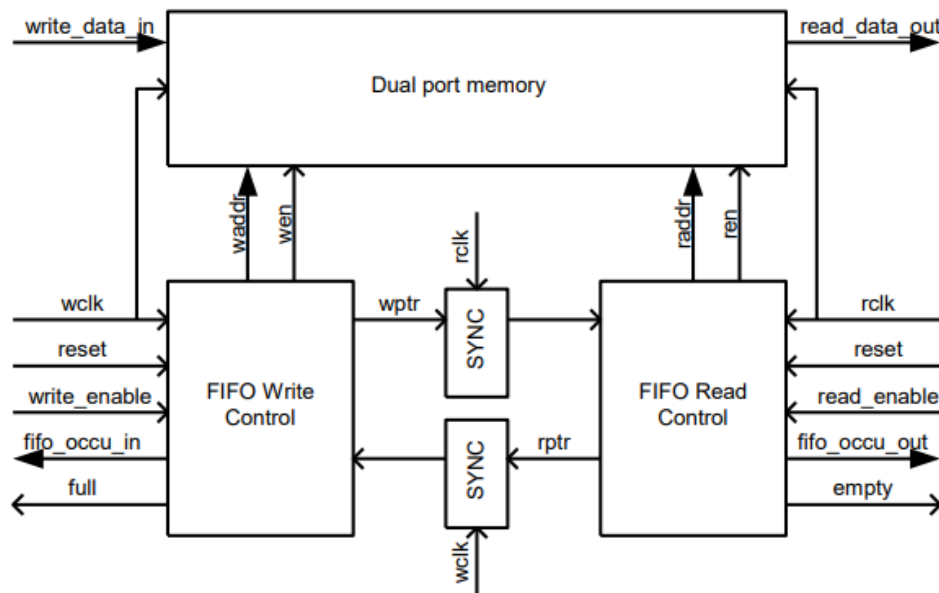


Figure 1: Block diagram of the asynchronous FIFO.

## 2 Task 1

Task 1 enquires as to why the circuit in figure 2 is insufficient for use in an asynchronous design. This circuitry is insufficient as the write pointer (wrptr), which is a multi-bit signal, can have propagation delays. As we have asynchronous sampling, a metastable state is more likely, as is partial or invalid data capture. The latter is shown in figure 3. It can be seen from this figure that the 3 bit wrptr has initial value (1 1 1). When its clock goes high, this data changes to (0 0 0) i.e. 3 bits change. There is a propagation delay. When read clock (rd\_clk) goes high, this propagation delay causes incorrect data to be captured. Instead of (0 0 0), (0 0 1) is captured instead. Thus data from an incorrect memory address may be read out. A metastable state would occur if the data change occurs on the same rising edge as the rd\_clk i.e. violating setup or hold time requirements. Adding more flip-flops will not solve the fundamental issue of incorrect multi-bit sampling. Furthermore, it is only useful for reducing

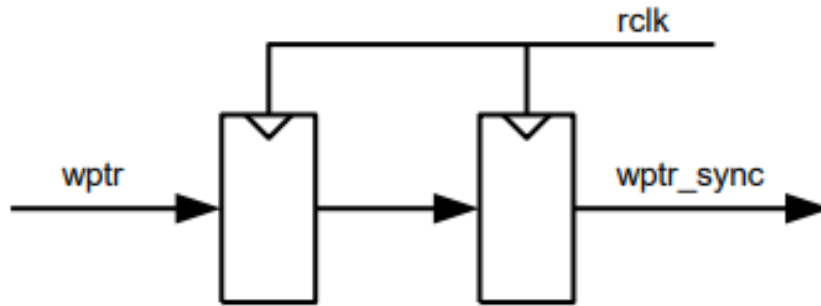


Figure 2: Circuitry that is insufficient to handle correct synchronisation in an asynchronous FIFO design.

the probability of having metastable states when transferring *single* bits between clock domains, as opposed to multiple bits as is the case here. It is required to use a circuitry that involves a binary to grey code mapping as seen in figure 4.

### 3 Task 2

The correct synchronisation circuitry is given in figure 4. Here, binary to grey code and grey code to binary code is used. The reason this circuitry is a better design is the binary to grey code ensures that only one bit changes at a time when the pointer is incremented or decremented which reduces the likelihood of incorrect data capture and/or metastable states. This circuitry introduces a delay however, which is very important. The longer the delay, the larger ones FIFO must be in order to not fill the buffer too early and cause data loss. Typically, one would impose a timing constraint on this delay in order for the tools to sufficiently implement the design. However, due to the small size of the FIFO, this delay should not be an issue here.

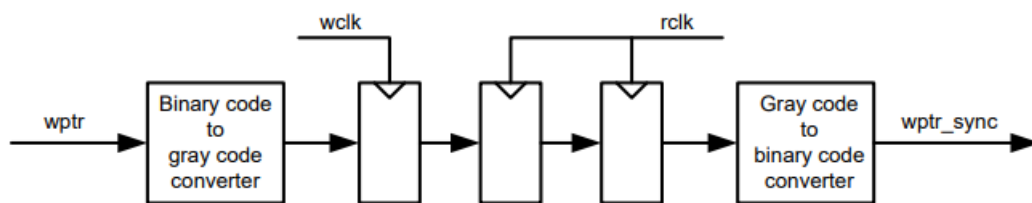


Figure 4: Synchronisation circuit for the write pointer.

#### 3.1 Logical Equations for Binary Code to Grey Code Conversion and Grey Code to Binary Code Conversion

The logical equation for binary to grey code conversion is

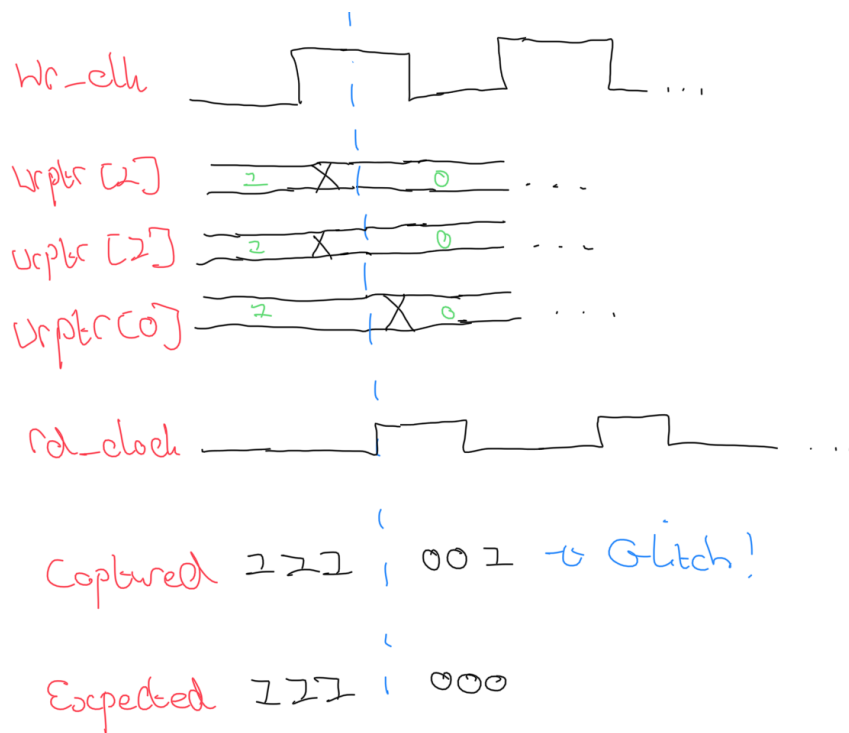


Figure 3: Example as to how circuitry in figure 2 can lead to incorrect output

$$G_{n-1} = B_{n-1}$$

$$G_i = B_{i+1} \oplus B_i$$

Here, the top equation states that the most significant bit is the same. The lower equation states that an XOR operation is conducted between bit  $i$  and bit  $i+1$  up to  $n$  where  $n$  is the number of bits. For  $n=5$  one has

$$G_4 = B_4$$

$$G_3 = B_4 \oplus B_3$$

$$G_2 = B_3 \oplus B_2$$

$$G_1 = B_2 \oplus B_1$$

$$G_0 = B_1 \oplus B_0$$

To do the reverse, and go from grey code to binary code, one uses:

$$B_{n-1} = G_{n-1}$$

$$B_i = G_i \oplus B_{i+1}$$

where the top equation states that the most significant bit is the same. For  $n=5$ , one has To do the reverse, and go from grey code to binary code, one uses:

$$B_4 = G_4$$

$$B_3 = G_3 \oplus B_4$$

$$B_2 = G_2 \oplus B_3$$

$$B_1 = G_1 \oplus B_2$$

$$B_0 = G_0 \oplus B_1$$

## 4 Task 3

For task 3, the block diagram in figure 1 is designed. A top level entity called Async\_FIFO.vhd is created. This contains the external inputs RST, WCLK, RCLK, READ\_ENABLE, WRITE\_ENABLE, WRITE\_DATA\_IN and the external output READ\_DATA\_OUT. Instantiated within the top level are the entites for the read and write control (FIFO\_read\_control.vhd and FIFO\_write\_control.vhd respectively), the dual-port memory (blk\_mem\_gen\_0 IP from AMD) and the read and write pointer synchronisation entities (read\_pointer\_sync.vhd and write\_pointer\_sync.vhd respectively.) Within the FIFO\_write\_control (FIFO\_read\_control) entity, the data is written (read) to the memory. Part of the synchronisation circuit in figure 4, namely that shown in figure 6 is coded within this entity, namely the binary code to gray code converter and first flip-flop.

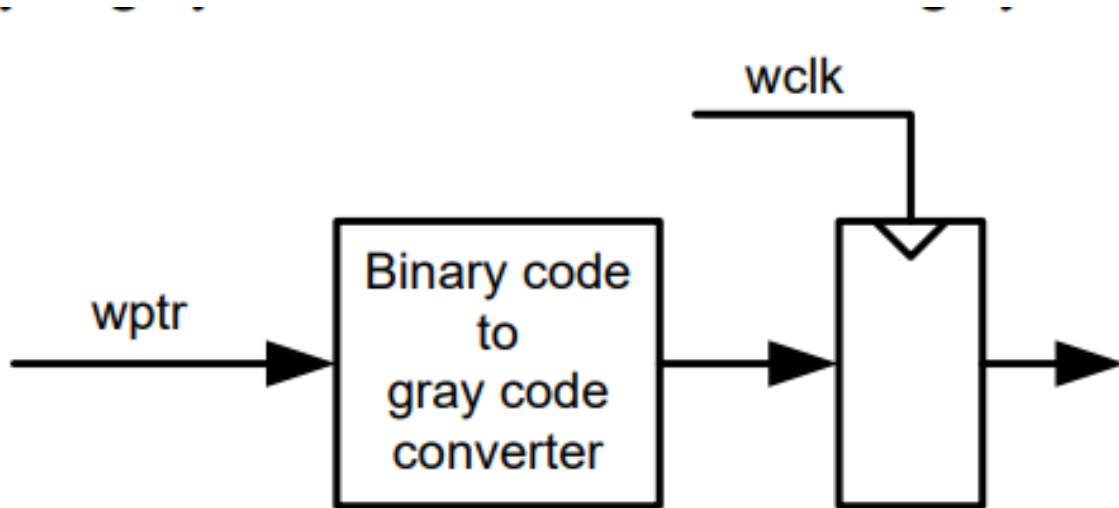


Figure 5: Partial synchronisation circuit carried out inside FIFO\_write\_control. The corresponding circuit for the read control is carried out inside FIFO\_read\_control.

```

35
36
37
38     wr_ptr_grey_code(4) <= wr_ptr_sig(4); -- Binary code to grey code
39     wr_ptr_grey_code(3) <= wr_ptr_sig(4) xor (wr_ptr_sig(3));
40     wr_ptr_grey_code(2) <= wr_ptr_sig(3) xor (wr_ptr_sig(2));
41     wr_ptr_grey_code(1) <= wr_ptr_sig(2) xor (wr_ptr_sig(1));
42     wr_ptr_grey_code(0) <= wr_ptr_sig(1) xor (wr_ptr_sig(0));
43
44     if WRITE_ENABLE(0) = '1' and full_sig = '0' then --don't write to full memory
45         wr_ptr_sig <= (wr_ptr_sig + 1); --unsigned so naturally wraps to 0
46         write_enable_sig <= (others => '1');
47     else
48         write_enable_sig <= (others => '0');
49
50     end if;
51     WPTR <= wr_ptr_grey_code; -- WPTR is now in grey code. Sent to write_pointer_sync for sync
52 end if;
53 end process;
54

```

Figure 6: Code to create the partial synchronisation circuit carried out inside FIFO\_write\_control. The corresponding code for the read control is inside FIFO\_read\_control which can be viewed in the appendix.

The rest of the synchronisation circuit is contained within write\_pointer\_sync (read\_pointer\_sync). It's architecture is shown in figure 7

```

17 architecture rtl of write_pointer_sync is
18     signal wptr_ff_1 : unsigned(4 downto 0);
19     signal wptr_ff_2 : unsigned(4 downto 0);
20     signal grey2binary : unsigned(4 downto 0);
21
22 begin
23
24     second_FF_process : process (RCLK, RST)
25     begin
26         if rst = '0' then
27             WRITE_POINTER_SYNC <= (others => '0');
28             wptr_ff_1 <= (others => '0');
29             wptr_ff_2 <= (others => '0');
30
31         elsif rising_edge(RCLK) then
32             -- wptr_ff_0 <= WPTR;
33             wptr_ff_1 <= WPTR;
34             wptr_ff_2 <= wptr_ff_1;
35
36             -- Grey code to binary code
37             grey2binary(4) <= wptr_ff_2(4);
38             grey2binary(3) <= wptr_ff_2(3) xor grey2binary(4);
39             grey2binary(2) <= wptr_ff_2(2) xor grey2binary(3);
40             grey2binary(1) <= wptr_ff_2(1) xor grey2binary(2);
41             grey2binary(0) <= wptr_ff_2(0) xor grey2binary(1);
42         end if;
43         WRITE_POINTER_SYNC <= grey2binary;
44     end process;
45     --WRITE_POINTER_SYNC <= grey2binary;
46 end architecture rtl;

```

Figure 7: Architecture of write\_pointer\_sync that completes the block diagram of figure 4 i.e. the synchronisation of the write pointer between clock domains. The corresponding code for the read pointer synchronisation is inside read\_pointer\_sync which can be viewed in the appendix. [This design has been updated. Please see section 4.1.](#)

## 4.1 Test Results from Simulation

The created entity Async\_FIFO is tested in simulation using the Async\_FIFO\_tb.sv script. This script can be viewed in the appendix. The write operation and full flags are tested first. There are 19 data inputs for this test: 8'h11 (0), 8'h22 (1), 8'h33 (2), 8'h44 (3), 8'h55 (4), 8'h66 (5), 8'h77 (6), 8'h88 (7), 8'h99 (8), 8'haa (9), 8'hbb (10), 8'hcc (11), 8'hdd (12), 8'hee (13), 8'hff (14), 8'h01 (15), 8'h03 (16), 8'h05 (17), 8'h06 (18). As such, the last bit of data that should enter is 8'h01 (the 16th piece of data, as there are 16 memory locations in the memory). It is seen from figure 8 that this is the case. Data is read in to the correct memory address. When all 15 memory locations have been written to, and no data has been read, the FULL flag goes high as required. No more data is written to the memory.

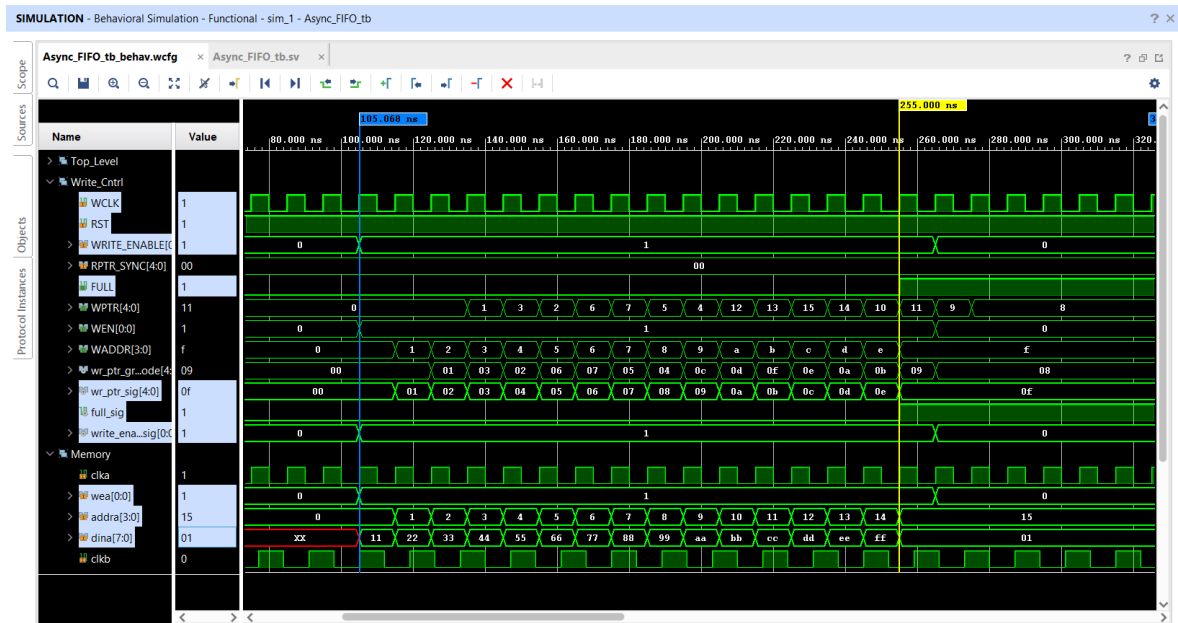


Figure 8: Simulation showing data is read in correctly, FULL flag operates correctly, and data is not read in once FULL flag is set.

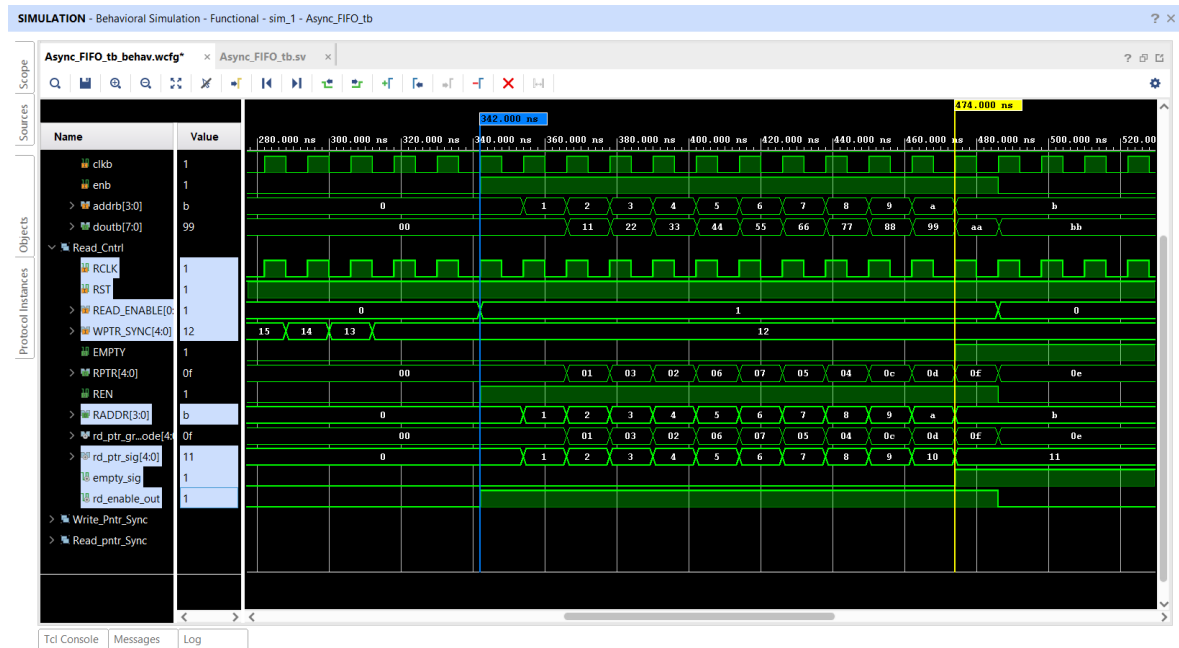
Update FIFO\_read\_control.vhd: The design for the write enable output(i.e. WEN) has been corrected. Before, the WEN signal was delayed by 1 clock cycle, due to it being dependent on an internal signal write\_enable\_sig. This caused data to be read a clock cycle too late, such that the first data input was read to address 1 and not 0. To correct for this, the write pointer was delayed by 1 clock cycle by instantiating another flip flop. Therefore, data was written to the correct address. However, a better design is now used whereby write enable out (WEN) is directly dependent on the enable in port (WRITE\_ENABLE(0)) and the full\_sig i.e. we do not use any internal signals. Therefore the extra flip-flop is not needed. These changes can be seen in figure 9.





Figure 9: Screenshot from Git Desktop showing changes made to the design of the FIFO\_write\_control entity, particularly with respect to the WEN output signal.

The read side however does not work as expected (figure 10). Firstly, although the read\_pointer works as required (and the RD\_ADDR is thus correct), the output from the dual-port memory is delayed by 2 clock cycles. It is seen that 0x11 is read out at address 0x2 instead of 0x0. The author is unable to explain the cause of this. Furthermore, the synchronised write pointer WPTR\_SYNC also has an incorrect value. This means the design inside write\_pointer\_sync.vhd is incorrect. Its simulation output is shown in figure 11. As WPTR\_SYNC is incorrect, the empty flag is thus incorrect. The RDPTR\_SYNC signal also increments incorrectly. The author did not have enough time to fully debug either of these issues. This will be done, but not in time for this report. It is not immediately clear as to the reasons for any of these issues, thus likely causes are held in reserve.



has been selected. This is so the Async FIFO can be used for large dual port memories. This has the affect of delaying the output of the block memory (doutb) by 1 additional clock cycle (see figure 17 in appendix). The corrected testbench waveforms are shown in figure 12.

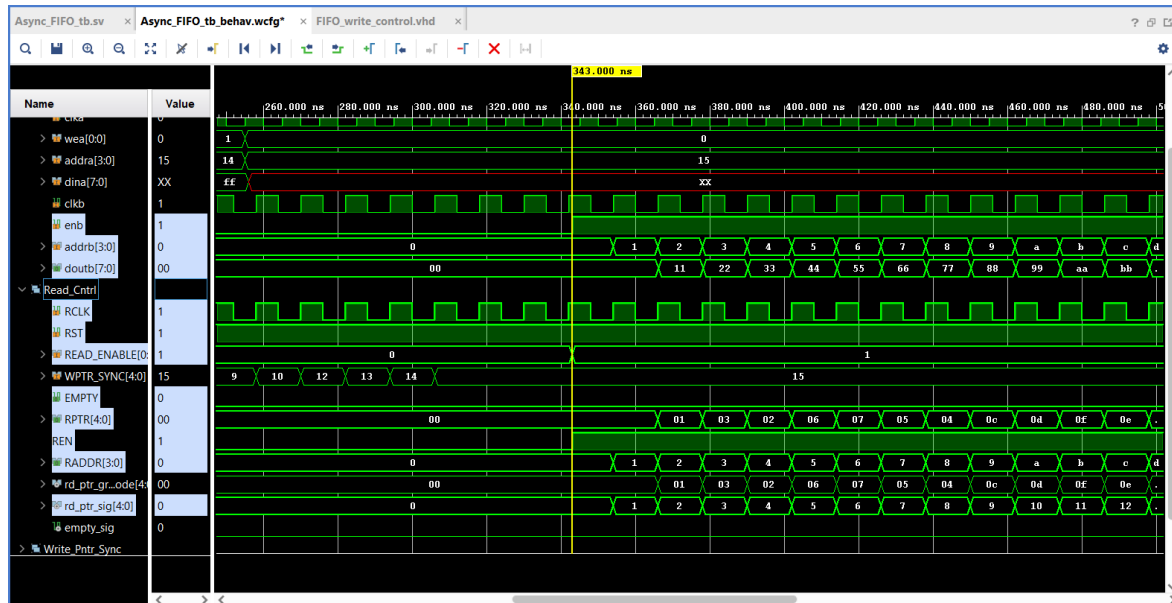


Figure 12: Waveform showing correct operation of the read control portion of the Async FIFO.

```

35 - rd_enable_out <= '0';
36 33 - rd_ptr_sig <= (others => '0'); --this gives multiple load error
37 - rd_ptr_sig_delay <= (others => '0');
38 - rd_ptr_sig2 <= (others => '0');
39 +
39 35
40 36
41 37
42 38
43 39
44 40
45 41
46 42
47 43
48 44
49 45
50 46
51 47
52 48
53 49
54 50
55 51
56 52
57 53
58 54
59 55
60 56
61 57
62 58
63 59
64 60
65 61
66 62
67 63
68 64
69 65
70 66
71 67
72 68
73 69
74 70
75 71
76 72
77 73
78 74
79 75
80 76
81 77
82 78
83 79
84 80
85 81
86 82
87 83
88 84
89 85
90 86
91 87
92 88
93 89
94 90
95 91
96 92
97 93
98 94
99 95
100 96
101 97
102 98
103 99
104 100
105 101
106 102
107 103
108 104
109 105
110 106
111 107
112 108
113 109
114 110
115 111
116 112
117 113
118 114
119 115
120 116
121 117
122 118
123 119
124 120
125 121
126 122
127 123
128 124
129 125
130 126
131 127
132 128
133 129
134 130
135 131
136 132
137 133
138 134
139 135
140 136
141 137
142 138
143 139
144 140
145 141
146 142
147 143
148 144
149 145
150 146
151 147
152 148
153 149
154 150
155 151
156 152
157 153
158 154
159 155
160 156
161 157
162 158
163 159
164 160
165 161
166 162
167 163
168 164
169 165
170 166
171 167
172 168
173 169
174 170
175 171
176 172
177 173
178 174
179 175
180 176
181 177
182 178
183 179
184 180
185 181
186 182
187 183
188 184
189 185
190 186
191 187
192 188
193 189
194 190
195 191
196 192
197 193
198 194
199 195
200 196
201 197
202 198
203 199
204 200
205 201
206 202
207 203
208 204
209 205
210 206
211 207
212 208
213 209
214 210
215 211
216 212
217 213
218 214
219 215
220 216
221 217
222 218
223 219
224 220
225 221
226 222
227 223
228 224
229 225
230 226
231 227
232 228
233 229
234 230
235 231
236 232
237 233
238 234
239 235
240 236
241 237
242 238
243 239
244 240
245 241
246 242
247 243
248 244
249 245
250 246
251 247
252 248
253 249
254 250
255 251
256 252
257 253
258 254
259 255
260 256
261 257
262 258
263 259
264 260
265 261
266 262
267 263
268 264
269 265
270 266
271 267
272 268
273 269
274 270
275 271
276 272
277 273
278 274
279 275
280 276
281 277
282 278
283 279
284 280
285 281
286 282
287 283
288 284
289 285
290 286
291 287
292 288
293 289
294 290
295 291
296 292
297 293
298 294
299 295
300 296
301 297
302 298
303 299
304 300
305 301
306 302
307 303
308 304
309 305
310 306
311 307
312 308
313 309
314 310
315 311
316 312
317 313
318 314
319 315
320 316
321 317
322 318
323 319
324 320
325 321
326 322
327 323
328 324
329 325
330 326
331 327
332 328
333 329
334 330
335 331
336 332
337 333
338 334
339 335
340 336
341 337
342 338
343 339
344 340
345 341
346 342
347 343
348 344
349 345
350 346
351 347
352 348
353 349
354 350
355 351
356 352
357 353
358 354
359 355
360 356
361 357
362 358
363 359
364 360
365 361
366 362
367 363
368 364
369 365
370 366
371 367
372 368
373 369
374 370
375 371
376 372
377 373
378 374
379 375
380 376
381 377
382 378
383 379
384 380
385 381
386 382
387 383
388 384
389 385
390 386
391 387
392 388
393 389
394 390
395 391
396 392
397 393
398 394
399 395
400 396
401 397
402 398
403 399
404 400
405 401
406 402
407 403
408 404
409 405
410 406
411 407
412 408
413 409
414 410
415 411
416 412
417 413
418 414
419 415
420 416
421 417
422 418
423 419
424 420
425 421
426 422
427 423
428 424
429 425
430 426
431 427
432 428
433 429
434 430
435 431
436 432
437 433
438 434
439 435
440 436
441 437
442 438
443 439
444 440
445 441
446 442
447 443
448 444
449 445
450 446
451 447
452 448
453 449
454 450
455 451
456 452
457 453
458 454
459 455
460 456
461 457
462 458
463 459
464 460
465 461
466 462
467 463
468 464
469 465
470 466
471 467
472 468
473 469
474 470
475 471
476 472
477 473
478 474
479 475
480 476
481 477
482 478
483 479
484 480
485 481
486 482
487 483
488 484
489 485
490 486
491 487
492 488
493 489
494 490
495 491
496 492
497 493
498 494
499 495
500 496
501 497
502 498
503 499
504 500
505 501
506 502
507 503
508 504
509 505
510 506
511 507
512 508
513 509
514 510
515 511
516 512
517 513
518 514
519 515
520 516
521 517
522 518
523 519
524 520
525 521
526 522
527 523
528 524
529 525
530 526
531 527
532 528
533 529
534 530
535 531
536 532
537 533
538 534
539 535
540 536
541 537
542 538
543 539
544 540
545 541
546 542
547 543
548 544
549 545
550 546
551 547
552 548
553 549
554 550
555 551
556 552
557 553
558 554
559 555
560 556
561 557
562 558
563 559
564 560
565 561
566 562
567 563
568 564
569 565
570 566
571 567
572 568
573 569
574 570
575 571
576 572
577 573
578 574
579 575
580 576
581 577
582 578
583 579
584 580
585 581
586 582
587 583
588 584
589 585
590 586
591 587
592 588
593 589
594 590
595 591
596 592
597 593
598 594
599 595
600 596
601 597
602 598
603 599
604 600
605 601
606 602
607 603
608 604
609 605
610 606
611 607
612 608
613 609
614 610
615 611
616 612
617 613
618 614
619 615
620 616
621 617
622 618
623 619
624 620
625 621
626 622
627 623
628 624
629 625
630 626
631 627
632 628
633 629
634 630
635 631
636 632
637 633
638 634
639 635
640 636
641 637
642 638
643 639
644 640
645 641
646 642
647 643
648 644
649 645
650 646
651 647
652 648
653 649
654 650
655 651
656 652
657 653
658 654
659 655
660 656
661 657
662 658
663 659
664 660
665 661
666 662
667 663
668 664
669 665
670 666
671 667
672 668
673 669
674 670
675 671
676 672
677 673
678 674
679 675
680 676
681 677
682 678
683 679
684 680
685 681
686 682
687 683
688 684
689 685
690 686
691 687
692 688
693 689
694 690
695 691
696 692
697 693
698 694
699 695
700 696
701 697
702 698
703 699
704 700
705 701
706 702
707 703
708 704
709 705
710 706
711 707
712 708
713 709
714 710
715 711
716 712
717 713
718 714
719 715
720 716
721 717
722 718
723 719
724 720
725 721
726 722
727 723
728 724
729 725
730 726
731 727
732 728
733 729
734 730
735 731
736 732
737 733
738 734
739 735
740 736
741 737
742 738
743 739
744 740
745 741
746 742
747 743
748 744
749 745
750 746
751 747
752 748
753 749
754 750
755 751
756 752
757 753
758 754
759 755
760 756
761 757
762 758
763 759
764 760
765 761
766 762
767 763
768 764
769 765
770 766
771 767
772 768
773 769
774 770
775 771
776 772
777 773
778 774
779 775
780 776
781 777
782 778
783 779
784 780
785 781
786 782
787 783
788 784
789 785
790 786
791 787
792 788
793 789
794 790
795 791
796 792
797 793
798 794
799 795
800 796
801 797
802 798
803 799
804 800
805 801
806 802
807 803
808 804
809 805
810 806
811 807
812 808
813 809
814 810
815 811
816 812
817 813
818 814
819 815
820 816
821 817
822 818
823 819
824 820
825 821
826 822
827 823
828 824
829 825
830 826
831 827
832 828
833 829
834 830
835 831
836 832
837 833
838 834
839 835
840 836
841 837
842 838
843 839
844 840
845 841
846 842
847 843
848 844
849 845
850 846
851 847
852 848
853 849
854 850
855 851
856 852
857 853
858 854
859 855
860 856
861 857
862 858
863 859
864 860
865 861
866 862
867 863
868 864
869 865
870 866
871 867
872 868
873 869
874 870
875 871
876 872
877 873
878 874
879 875
880 876
881 877
882 878
883 879
884 880
885 881
886 882
887 883
888 884
889 885
890 886
891 887
892 888
893 889
894 890
895 891
896 892
897 893
898 894
899 895
900 896
901 897
902 898
903 899
904 900
905 901
906 902
907 903
908 904
909 905
910 906
911 907
912 908
913 909
914 910
915 911
916 912
917 913
918 914
919 915
920 916
921 917
922 918
923 919
924 920
925 921
926 922
927 923
928 924
929 925
930 926
931 927
932 928
933 929
934 930
935 931
936 932
937 933
938 934
939 935
940 936
941 937
942 938
943 939
944 940
945 941
946 942
947 943
948 944
949 945
950 946
951 947
952 948
953 949
954 950
955 951
956 952
957 953
958 954
959 955
960 956
961 957
962 958
963 959
964 960
965 961
966 962
967 963
968 964
969 965
970 966
971 967
972 968
973 969
974 970
975 971
976 972
977 973
978 974
979 975
980 976
981 977
982 978
983 979
984 980
985 981
986 982
987 983
988 984
989 985
990 986
991 987
992 988
993 989
994 990
995 991
996 992
997 993
998 994
999 995
1000 996
1001 997
1002 998
1003 999
1004 1000
1005 1001
1006 1002
1007 1003
1008 1004
1009 1005
1010 1006
1011 1007
1012 1008
1013 1009
1014 1010
1015 1011
1016 1012
1017 1013
1018 1014
1019 1015
1020 1016
1021 1017
1022 1018
1023 1019
1024 1020
1025 1021
1026 1022
1027 1023
1028 1024
1029 1025
1030 1026
1031 1027
1032 1028
1033 1029
1034 1030
1035 1031
1036 1032
1037 1033
1038 1034
1039 1035
1040 1036
1041 1037
1042 1038
1043 1039
1044 1040
1045 1041
1046 1042
1047 1043
1048 1044
1049 1045
1050 1046
1051 1047
1052 1048
1053 1049
1054 1050
1055 1051
1056 1052
1057 1053
1058 1054
1059 1055
1060 1056
1061 1057
1062 1058
1063 1059
1064 1060
1065 1061
1066 1062
1067 1063
1068 1064
1069 1065
1070 1066
1071 1067
1072 1068
1073 1069
1074 1070
1075 1071
1076 1072
1077 1073
1078 1074
1079 1075
1080 1076
1081 1077
1082 1078
1083 1079
1084 1080
1085 1081
1086 1082
1087 1083
1088 1084
1089 1085
1090 1086
1091 1087
1092 1088
1093 1089
1094 1090
1095 1091
1096 1092
1097 1093
1098 1094
1099 1095
1100 1096
1101 1097
1102 1098
1103 1099
1104 1100
1105 1101
1106 1102
1107 1103
1108 1104
1109 1105
1110 1106
1111 1107
1112 1108
1113 1109
1114 1110
1115 1111
1116 1112
1117 1113
1118 1114
1119 1115
1120 1116
1121 1117
1122 1118
1123 1119
1124 1120
1125 1121
1126 1122
1127 1123
1128 1124
1129 1125
1130 1126
1131 1127
1132 1128
1133 1129
1134 1130
1135 1131
1136 1132
1137 1133
1138 1134
1139 1135
1140 1136
1141 1137
1142 1138
1143 1139
1144 1140
1145 1141
1146 1142
1147 1143
1148 1144
1149 1145
1150 1146
1151 1147
1152 1148
1153 1149
1154 1150
1155 1151
1156 1152
1157 1153
1158 1154
1159 1155
1160 1156
1161 1157
1162 1158
1163 1159
1164 1160
1165 1161
1166 1162
1167 1163
1168 1164
1169 1165
1170 1166
1171 1167
1172 1168
1173 1169
1174 1170
1175 1171
1176 1172
1177 1173
1178 1174
1179 1175
1180 1176
1181 1177
1182 1178
1183 1179
1184 1180
1185 1181
1186 1182
1187 1183
1188 1184
1189 1185
1190 1186
1191 1187
1192 1188
1193 1189
1194 1190
1195 1191
1196 1192
1197 1193
1198 1194
1199 1195
1200 1196
1201 1197
1202 1198
1203 1199
1204 1200
1205 1201
1206 1202
1207 1203
1208 1204
1209 1205
1210 1206
1211 1207
1212 1208
1213 1209
1214 1210
1215 1211
1216 1212
1217 1213
1218 1214
1219 1215
1220 1216
1221 1217
1222 1218
1223 1219
1224 1220
1225 1221
1226 1222
1227 1223
1228 1224
1229 1225
1230 1226
1231 1227
1232 1228
1233 1229
1234 1230
1235 1231
1236 1232
1237 1233
1238 1234
1239 1235
1240 1236
1241 1237
1242 1238
1243 1239
1244 1240
1245 1241
1246 1242
1247 1243
1248 1244
1249 1245
1250 1246
1251 1247
1252 1248
1253 1249
1254 1250
1255 1251
1256 1252
1257 1253
1258 1254
1259 1255
1260 1256
1261 1257
1262 1258
1263 1259
1264 1260
1265 1261
1266 1262
1267 1263
1268 1264
1269 1265
1270 1266
1271 1267
1272 1268
1273 1269
1274 1270
1275 1271
1276 1272
1277 1273
1278 1274
1279 1275
1280 1276
1281 1277
1282 1278
1283 1279
1284 1280
1285 1281
1286 1282
1287 1283
1288 1284
1289 1285
1290 1286
1291 1287
1292 1288
1293 1289
1294 1290
1295 1291
1296 1292
1297 1293
1298 1294
1299 1295
1300 1296
1301 1297
1302 1298
1303 1299
1304 1300
1305 1301
1306 1302
1307 1303
1308 1304
1309 1305
1310 1306
1311 1307
1312 1308
1313 1309
1314 1310
1315 1311
1316 1312
1317 1313
1318 1314
1319 1315
1320 1316
1321 1317
1322 1318
1323 1319
1324 1320
1325 1321
1326 1322
1327 1323
1328 1324
1329 1325
1330 1326
1331 1327
1332 1328
1333 1329
1334 1330
1335 1331
1336 1332
1337 1333
1338 1334
1339 1335
1340 1336
1341 1337
1342 1338
1343 1339
1344 1340
1345 1341
1346 1342
1347 1343
1348 1344
1349 1345
1350 1346
1351 1347
1352 1348
1353 1349
1354 1350
1355 1351
1356 1352
1357 1353
1358 1354
1359 1355
1360 1356
1361 1357
1362 1358
1363 1359
1364 1360
1365 1361
1366 1362
1367 1363
1368 1364
1369 1365
1370 1366
1371 1367
1372 1368
1373 1369
1374 1370
1375 1371
1376 1372
1377 1373
1378 1374
1379 1375
1380 1376
1381 1377
1382 1378
1383 1379
1384 1380
1385 1381
1386 1382
1387 1383
1388 1384
1389 1385
1390 1386
1391 1387
1392 1388
1393 1389
1394 1390
1395 1391
1396 1392
1397 1393
1398 1394
1399 1395
1400 1396
1401 1397
1402 1398
1403 1399
1404 1400
1405 1401
1406 1402
1407 1403
1408 1404
1409 1405
1410 1406
1411 1407
1412 1408
1413 1409
1414 1410
1415 1411
1416 1412
1417 1413
1418 1414
1419 1415
1420 1416
1421 1417
1422 1418
1423 1419
1424 1420
1425 1421
1426 1422
1427 1423
1428 1424
1429 1425
1430 1426
1431 1427
1432 1428
1433 1429
1434 1430
1435 1431
1436 
```

as previous values of `wptr_ff_2` (the grey code after the 2nd flip-flop) were being used to do the binary code to grey code conversion as opposed to the current values. Therefore the output did not increment as required. Instead, `grey2binary` was redesigned as a variable to allow for it's immediate update and correct incrementation of `WRITE_POINTER_SYNC` and `WRITE_POINTER_SYNC`.

Course_work\Ex_2_Async_Fifo\VHDL Code\sources\Read_pointer_sync\read_pointer_sync.vhd			
	23	23	begin
	24	24	if RST = '0' then
	25	25	READ_POINTER_SYNC <= (others => '0');
	26	26	rptr_ff_1 <= (others => '0');
	27	27	rptr_ff_2 <= (others => '0');
✓	✓	28	-
✓	✓	29	-
✓	✓	28	+      elsif rising_edge(WCLK) then
✓	✓	29	+      READ_POINTER_SYNC <= (others => '0');
✓	✓	30	+      elsif rising_edge(WCLK) then
	30	31	
	31	32	rptr_ff_1 <= RPTR;
	32	33	rptr_ff_2 <= rptr_ff_1;
✓	✓	33	-
✓	✓	34	-
✓	✓	35	-      -- MSB of binary is the same as MSB of gray code
✓	✓	36	-      grey2binary(4) <= rptr_ff_2(4);
✓	✓	37	-      -- Other bits of binary are the XOR of corresponding gray code and previous binary bit
✓	✓	38	-      grey2binary(3) <= rptr_ff_2(3) xor grey2binary(4);
✓	✓	39	-      grey2binary(2) <= rptr_ff_2(2) xor grey2binary(3);
✓	✓	40	-      grey2binary(1) <= rptr_ff_2(1) xor grey2binary(2);
✓	✓	40	-      grey2binary(0) <= rptr_ff_2(0) xor grey2binary(1);
✓	✓	34	+      grey2binary(4) := rptr_ff_2(4);
✓	✓	35	+      grey2binary(3) := rptr_ff_2(3) xor grey2binary(4);
✓	✓	36	+      grey2binary(2) := rptr_ff_2(2) xor grey2binary(3);
✓	✓	37	+      grey2binary(1) := rptr_ff_2(1) xor grey2binary(2);
✓	✓	38	+      grey2binary(0) := rptr_ff_2(0) xor grey2binary(1);
✓	✓	40	+      READ_POINTER_SYNC <= grey2binary; -- Needs to be inside process so it happens on rising_edge of clock!
	41	41	end if;
✓	✓	42	-      READ_POINTER_SYNC <= grey2binary; -- Needs to be inside process so it happens on rising_edge of clock!
✓	✓	42	+      end process;
	43	43	
✓	✓	44	-      --READ_POINTER_SYNC <= grey2binary;
	45	44	end architecture rtl;

Figure 14: Updates to the `read_point_sync` entity. Similar changes are made in the `write_point_sync` entity.

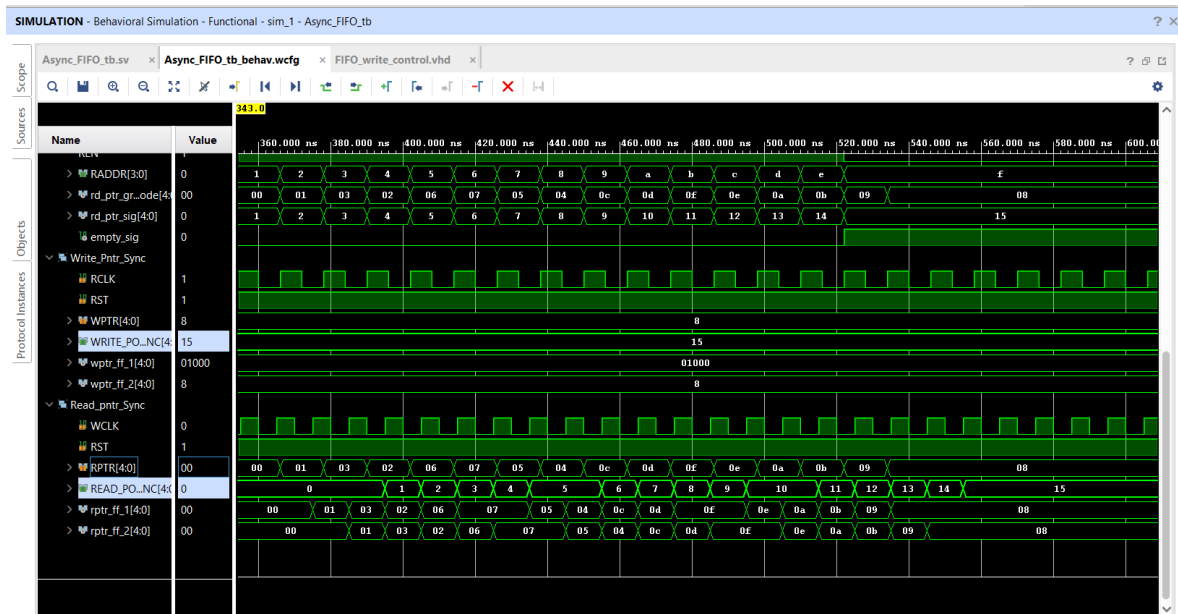


Figure 15: Waveform showing correct operation of the synchronised read and write pointers..

Update to testbench: Lastly, some slight changes were made to the testbench. The original testbench caused delta cycles when inputting data to the DUT. Thus, a slight delay has been added before the if statement to write data, and before the if statement for reading data. Further, when data is not being written, X's i.e. unknowns are read out. This allows for easier viewing of incorrect operation if it exists.

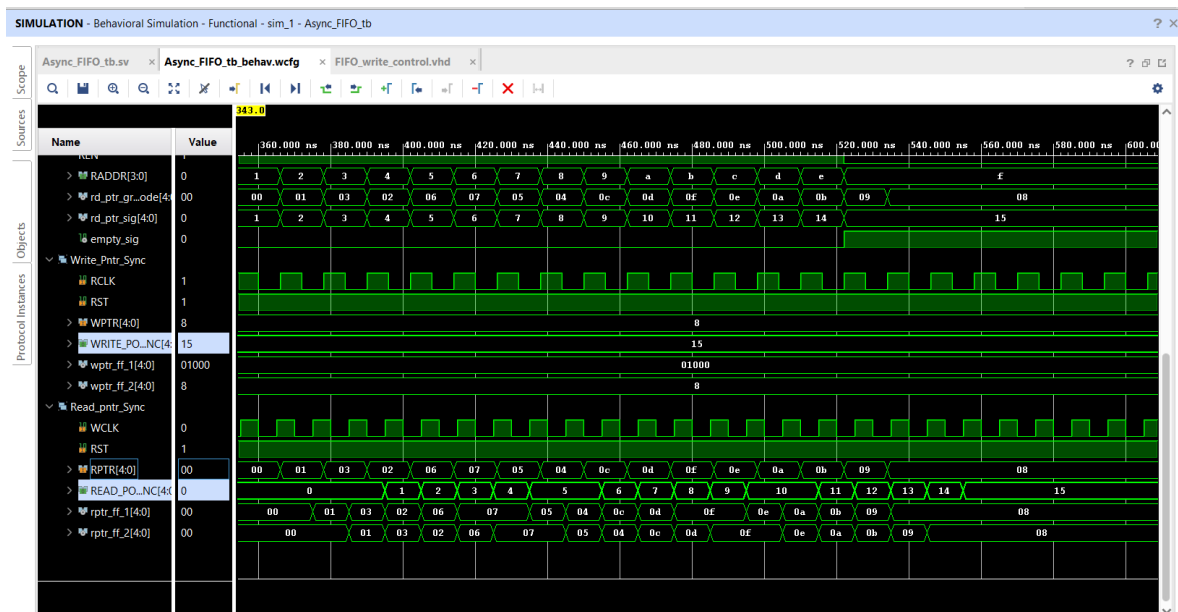


Figure 16: Changes made to the test bench.

## 5 Appendix

### 5.1 Async\_FIFO.vhd

```
1  -- vhdl-linter-disable type-resolved component. component
2  -- Author Daniel Duggan
3  library IEEE;
4  use ieee.std_logic_1164.all;
5  use ieee.numeric_std.all;
6  -- LIBRARY blk_mem_gen_v8_4_7;
7  -- USE blk_mem_gen_v8_4_7.blk_mem_gen_v8_4_7;
8
9  entity Async_FIFO is
10     port (
11         RST_top : in std_logic;
12         WCLK_top : in std_logic;
13         RCLK_top : in std_logic;
14         WRITE_ENABLE_TOP : in std_logic_vector(0 downto 0);
15         READ_ENABLE_TOP : in std_logic_vector(0 downto 0);
16         FULL_top : out std_logic; --output if memory is full
17         EMPTY_top : out std_logic; -- output if memory is empty
18         WRITE_DATA_IN_top : in std_logic_vector(7 downto 0);
19         WRITE_DATA_OUT_top : out std_logic_vector(7 downto 0)
20     );
21 end entity;
22
23 architecture rtl of Async_FIFO is
24
25     -- Declare the component
26     COMPONENT blk_mem_gen_0
27         PORT (
28             clka    : IN std_logic;
29             wea      : IN std_logic_vector(0 DOWNTO 0);
30             addra    : IN std_logic_vector(3 DOWNTO 0);
31             dina     : IN std_logic_vector(7 DOWNTO 0);
32             clkb     : IN std_logic;
33             enb      : IN std_logic;
34             addrb    : IN std_logic_vector(3 DOWNTO 0);
35             doutb    : OUT std_logic_vector(7 DOWNTO 0)
36         );
37     END COMPONENT;
38
39     -- FIFO_WRITE_CONTROL SIGNALS.
40     signal wen_sig : std_logic_vector(0 downto 0); -- write enable from
41         FIFO_WRITE_Control
```

```

41     --write address from FIFO_WRITE_Control
42     signal wr_pointer_sig : unsigned(4 downto 0); --write address from
        FIFO_WRITE_Control are bits (3 downto 0)
43     signal wr_addr_a : unsigned(3 downto 0);
44     signal wr_pointer_sync : unsigned(4 downto 0);
45
46     --FIFO_READ_CONTROL SIGNALS.
47     signal ren_sig : std_logic; -- write enable from FIFO_READ_CONTROL
48     signal rd_pointer_sig : unsigned(4 downto 0);
49     signal rd_addr_b : unsigned(3 downto 0);
50     signal rd_pointer_sync : unsigned(4 downto 0);
51
52     begin
53
54         Fifo_write_control_inst : entity work.FIFO_write_control
55         port map(
56             WCLK => WCLK_top,
57             RST => RST_top,
58             WRITE_ENABLE => WRITE_ENABLE_TOP,
59             RPTR_SYNC => rd_pointer_sync, --Input from Read_pointer_sync.
                Synchronised read pointer.
60             FULL => FULL_top,
61             WPTR => wr_pointer_sig, --Write pointer goes to the sync
62             WEN => wen_sig, -- goes to block_mem via sig wen_sig in async_FIFO
63             WADDR => wr_addr_a -- Output to addra of Memory (Write Address).
64
65         );
66
67         Dual_port_memory_inst : blk_mem_gen_0 -- vhdl-linter-disable-line not-
            declared
68
69         port map(
70             clka => WCLK_top,
71             wea => wen_sig, -- write enable from FIFO_WRITE_Control
72             addra => std_logic_vector(wr_addr_a), -- write address from
                FIFO_WRITE_Control
73             dina => WRITE_DATA_IN_top,
74             clk_b => RCLK_top,
75             enb => ren_sig, -- read enable from FIFO_READ_CONTROL
76             addr_b => std_logic_vector(rd_addr_b), --read address from
                FIFO_READ_CONTROL
77             dout_b => WRITE_DATA_OUT_top
78         );
79

```

```

80     Fifo_read_control_inst : entity work.FIFO_read_control -- vhdl-linter-
        disable-line not-declared
81     port map(
82         RCLK => RCLK_top,
83         RST => RST_top,
84         READ_ENABLE => READ_ENABLE_TOP, -- read enable from TOP
85         REN => ren_sig, -- write enable to BLOCK_MEM
86         WPTR_SYNC => wr_pointer_sync, -- Input sig from WRITE_POINTER_SYNC.
            To determine occupancy and empty/full.
87         EMPTY => EMPTY_top, -- Output sig. Goes to TOP
88         RPTR => rd_pointer_sig, -- output sig to Read_pointer_sync entity.
            Signal to be synchronised with wr_pointer_sig
89         RADDR => rd_addr_b -- Output sig to Dual port memory. Read address.
90
91     );
92
93     Write_pointer_sync_inst : entity work.write_pointer_sync
94     port map(
95         RCLK => RCLK_top,
96         --WCLK => WCLK_top,
97         RST => RST_top,
98         WPTR => wr_pointer_sig,
99         WRITE_POINTER_SYNC => wr_pointer_sync
100    );
101
102     Read_pointer_sync_inst : entity work.read_pointer_sync
103     port map(
104         --RCLK => RCLK_top,
105         WCLK => WCLK_top,
106         RST => RST_top,
107         RPTR => rd_pointer_sig, -- input from FIFO_read_control.
108         READ_POINTER_SYNC => rd_pointer_sync -- Output to
            FIFO_write_control. Synchronised read_pointer.
109    );
110
111 end architecture rtl;

```

## 5.2 FIFO\_write\_control.vhd

```

1  -- vhdl-linter-disable type-resolved
2  -- Author Daniel Duggan
3  library ieee;
4  use ieee.std_logic_1164.all;
5  use ieee.numeric_std.all;

```



```

6
7  entity FIFO_WRITE_CONTROL is
8      port (
9          WCLK : in std_logic;
10         RST : in std_logic;
11         WRITE_ENABLE : in std_logic_vector(0 downto 0);
12         RPTR_SYNC : in unsigned(4 downto 0); --rd pointer that comes from
            the sync
13         FULL : out std_logic;
14         WPTR : out unsigned(4 downto 0); --Write pointer goes to the sync i.
            e. grey-coded
15         WEN : out std_logic_vector(0 downto 0); -- goes to block_mem via sig
            wen_sig in async_FIFO
16         WADDR : out unsigned(3 downto 0) -- write address. Goes to block_mem
            via signal waddr_sig in async_FIFO
17     );
18 end entity;
19
20 architecture rtl of FIFO_WRITE_CONTROL is
21
22     signal wr_ptr_grey_code : unsigned(4 downto 0);
23     signal wr_ptr_sig : unsigned(4 downto 0);
24     signal full_sig : std_logic;
25
26
27 begin
28
29     write_control_process : process (WCLK,RST) --asyn reset
30     begin
31         if RST = '0' then --reset active low
32             wr_ptr_grey_code <= (others => '0');
33             wr_ptr_sig <= (others => '0');
34
35
36
37
38             elsif rising_edge(WCLK) then
39
40                 wr_ptr_grey_code(4) <= wr_ptr_sig(4); -- Binary code to grey
                    code
41                 wr_ptr_grey_code(3) <= wr_ptr_sig(4) xor (wr_ptr_sig(3));
42                 wr_ptr_grey_code(2) <= wr_ptr_sig(3) xor (wr_ptr_sig(2));
43                 wr_ptr_grey_code(1) <= wr_ptr_sig(2) xor (wr_ptr_sig(1));
44                 wr_ptr_grey_code(0) <= wr_ptr_sig(1) xor (wr_ptr_sig(0));
45

```

```

46         if WRITE_ENABLE(0) = '1' and full_sig = '0' then --don't
            write to full memory
47             wr_ptr_sig <= (wr_ptr_sig + 1);
48         end if;
49         WPTR <= wr_ptr_grey_code; -- WPTR is now in grey code. Sent
            to write_pointer_sync for sync
50     end if;
51 end process;
52
53 full_sig <= '1' when (wr_ptr_sig - RPTR_SYNC = 15) else '0';
54
55
56 --WPTR <= wr_ptr_grey_code; -- WPTR is now in grey code. Sent to
    write_pointer_sync for sync
57 FULL <= full_sig;
58 WADDR <= (wr_ptr_sig(3 downto 0)); -- sent to the Dual-port memory
59 WEN <= (others => '1') when (WRITE_ENABLE(0)='1' AND full_sig = '0')
    else (others => '0'); --sent to the Dual-port memory '1' when (
    rd_ptr_sig_delay = WPTR_SYNC) else '0';
60 end architecture rtl;

```

### 5.3 FIFO\_read\_control.vhd

```

1  -- vhdl-linter-disable type-resolved
2  -- Author Daniel Duggan
3  library ieee;
4  use ieee.std_logic_1164.all;
5  use ieee.numeric_std.all;
6
7  entity FIFO_READ_CONTROL is
8      port (
9          RCLK : in std_logic;
10         RST : in std_logic;
11         READ_ENABLE : in std_logic_vector(0 downto 0);
12         WPTR_SYNC : in unsigned(4 downto 0); --wp pointer that comes from
            the sync
13         EMPTY : out std_logic;
14         RPTR : out unsigned(4 downto 0); --Write pointer goes to the sync i.
            e. grey-coded
15         REN : inout std_logic; -- goes to block_mem via sig wen_sig in
            async_FIFO
16         RADDR : out unsigned(3 downto 0) -- write address. Goes to block_mem
            via signal waddr_sig in async_FIFO
17     );

```

```

18 end entity;
19
20 architecture rtl of FIFO_READ_CONTROL is
21
22     signal rd_ptr_grey_code : unsigned(4 downto 0);
23     signal rd_ptr_sig : unsigned(4 downto 0);
24     signal empty_sig : std_logic;
25
26
27 begin
28
29     FIFO_read_control_process : process (RCLK, RST) --asyn reset
30     begin
31         if RST = '0' then --reset active low
32             --RPTR <= (others => '0');
33             rd_ptr_grey_code <= (others => '0');-- this gives multiple load
               error
34             rd_ptr_sig <= (others => '0'); --this gives multiple load error
35
36
37
38             elsif rising_edge(RCLK) then
39
40                 rd_ptr_grey_code(4) <= rd_ptr_sig(4); -- Binary code to grey
                   code
41                 rd_ptr_grey_code(3) <= rd_ptr_sig(4) xor (rd_ptr_sig(3));
42                 rd_ptr_grey_code(2) <= rd_ptr_sig(3) xor (rd_ptr_sig(2));
43                 rd_ptr_grey_code(1) <= rd_ptr_sig(2) xor (rd_ptr_sig(1));
44                 rd_ptr_grey_code(0) <= rd_ptr_sig(1) xor (rd_ptr_sig(0));
45
46                 if READ_ENABLE(0) = '1' and empty_sig = '0' then --don't read
                   from empty memory
47                     rd_ptr_sig <= (rd_ptr_sig + 1); --unsigned so naturally
                       wraps to 0
48                 end if;
49             end if;
50
51         end process;
52
53         empty_sig <= '1' when (rd_ptr_sig = WPTR_SYNC) else '0'; --Check if
           empty or not. Uses synced wr_ptr which has 5 cc delay
54         RADDR <= (rd_ptr_sig(3 downto 0));
55         RPTR <= rd_ptr_grey_code; -- RPTR is now in grey code.
56         EMPTY <= empty_sig;

```

```

57     REN <= '1' when (READ_ENABLE(0)='1' AND empty_sig = '0') else '0'; --use
        rd_eable
58 end architecture rtl; --all 5 bits and equal, empty, all 5 bits and
        difference is 16 is full.

```

## 5.4 write\_pointer\_sync.vhd

```

1  -- vhdl-linter-disable type-resolved
2  -- Author Daniel Duggan
3  library IEEE;
4  use ieee.std_logic_1164.all;
5  use ieee.numeric_std.all;
6
7  entity write_pointer_sync is
8      port (
9          RCLK : in std_logic;
10         --WCLK : in std_logic;
11         RST : in std_logic;
12         WPTR : in unsigned(4 downto 0);
13         WRITE_POINTER_SYNC : out unsigned(4 downto 0)
14
15     );
16 end entity;
17
18 architecture rtl of write_pointer_sync is
19     signal wptr_ff_1 : unsigned(4 downto 0);
20     signal wptr_ff_2 : unsigned(4 downto 0);
21 begin
22
23     second_FF_process : process (RCLK, RST)
24         variable grey2binary : unsigned(4 downto 0);
25     begin
26         if rst = '0' then
27             WRITE_POINTER_SYNC <= (others => '0');
28             wptr_ff_1 <= (others => '0');
29             wptr_ff_2 <= (others => '0');
30         elsif rising_edge(RCLK) then
31             -- wptr_ff_0 <= WPTR;
32             wptr_ff_1 <= WPTR;
33             wptr_ff_2 <= wptr_ff_1;
34
35             -- Grey code to binary code
36             grey2binary(4) := wptr_ff_2(4);
37             grey2binary(3) := wptr_ff_2(3) xor grey2binary(4);

```

```

38         grey2binary(2) := wptr_ff_2(2) xor grey2binary(3);
39         grey2binary(1) := wptr_ff_2(1) xor grey2binary(2);
40         grey2binary(0) := wptr_ff_2(0) xor grey2binary(1);
41         WRITE_POINTER_SYNC <= grey2binary; -- Needs to be inside process
                                         so it happens on rising_edge of clock!
42     end if;
43 end process;
44 end architecture rtl;

```

## 5.5 read\_pointer\_sync.vhd

```

1  -- vhdl-linter-disable type-resolved
2  -- Author Daniel Duggan
3  library IEEE;
4  use ieee.std_logic_1164.all;
5  use ieee.numeric_std.all;
6
7  entity read_pointer_sync is
8      port (
9          --RCLK : in std_logic;
10         WCLK : in std_logic;
11         RST : in std_logic;
12         RPTR : in unsigned(4 downto 0);
13         READ_POINTER_SYNC : out unsigned(4 downto 0)
14     );
15 end entity;
16
17 architecture rtl of read_pointer_sync is
18     signal rptr_ff_1 : unsigned(4 downto 0);
19     signal rptr_ff_2 : unsigned(4 downto 0);
20
21 begin
22     second_FF_process : process (WCLK, RST)
23         variable grey2binary : unsigned(4 downto 0);
24     begin
25         if RST = '0' then
26             READ_POINTER_SYNC <= (others => '0');
27             rptr_ff_1 <= (others => '0');
28             rptr_ff_2 <= (others => '0');
29             READ_POINTER_SYNC <= (others => '0');
30
31         elsif rising_edge(WCLK) then
32
33             rptr_ff_1 <= RPTR;

```

```

34         rptr_ff_2 <= rptr_ff_1;
35
36         grey2binary(4) := rptr_ff_2(4);
37         grey2binary(3) := rptr_ff_2(3) xor grey2binary(4);
38         grey2binary(2) := rptr_ff_2(2) xor grey2binary(3);
39         grey2binary(1) := rptr_ff_2(1) xor grey2binary(2);
40         grey2binary(0) := rptr_ff_2(0) xor grey2binary(1);
41         READ_POINTER_SYNC <= grey2binary; -- Needs to be inside process
           so it happens on rising_edge of clock!
42     end if;
43
44     end process;
45 end architecture rtl;

```

## 5.6 Async\_FIFO\_tb.sv

```

1  // Testbench for Async_FIFO
2  // Author Daniel Duggan
3  module Async_FIFO_tb;
4
5  // Signals
6  logic RST_top;
7  logic WCLK_top;
8  logic RCLK_top;
9  logic [0:0] WRITE_ENABLE_TOP;
10 logic [0:0] READ_ENABLE_TOP;
11 logic FULL_top;
12 logic EMPTY_top;
13 logic [7:0] WRITE_DATA_IN_top;
14 logic [7:0] WRITE_DATA_OUT_top;
15
16 // Clock periods
17 parameter CLK_PERIOD_WR = 10; // Write clock period (e.g., 100MHz)
18 parameter CLK_PERIOD_RD = 12; // Read clock period (e.g., 66.6MHz)
19
20 // FIFO instance
21 Async_FIFO uut (
22     .RST_top(RST_top),
23     .WCLK_top(WCLK_top),
24     .RCLK_top(RCLK_top),
25     .WRITE_ENABLE_TOP(WRITE_ENABLE_TOP),
26     .READ_ENABLE_TOP(READ_ENABLE_TOP),
27     .FULL_top(FULL_top),
28     .EMPTY_top(EMPTY_top),

```

```

29     .WRITE_DATA_IN_top(WRITE_DATA_IN_top),
30     .WRITE_DATA_OUT_top(WRITE_DATA_OUT_top)
31 );
32
33 // Clock Generation
34 always #(CLK_PERIOD_WR/2) WCLK_top = ~WCLK_top;
35 always #(CLK_PERIOD_RD/2) RCLK_top = ~RCLK_top;
36
37 // Predefined data array
38 logic [7:0] predefined_data [0:18] = '{
39     8'h11, 8'h22, 8'h33, 8'h44, 8'h55, 8'h66, 8'h77, 8'h88,
40     8'h99, 8'haa, 8'hbb, 8'hcc, 8'hdd, 8'hee, 8'hff, 8'h01,
41     8'h03, 8'h05, 8'h06
42 };
43
44 // Queue for verification
45 logic [7:0] fifo_queue [$];
46
47 initial begin
48     // Initialize
49     WCLK_top = 0;
50     RCLK_top = 0;
51     RST_top = 0; //reset active low
52     WRITE_ENABLE_TOP = 0;
53     READ_ENABLE_TOP = 0;
54     //WRITE_DATA_IN_top = 8'h00;
55
56     // Apply Reset
57     #(5*CLK_PERIOD_WR);
58     RST_top = 1;
59     #(5*CLK_PERIOD_WR);
60
61     // Write to FIFO
62     for (int i = 0; i < 19; i++) begin
63         @(posedge WCLK_top);
64         #1;
65         if (!FULL_top) begin
66             WRITE_ENABLE_TOP = 1;
67             WRITE_DATA_IN_top = predefined_data[i];
68             fifo_queue.push_back(predefined_data[i]); // Store for
               verification
69             $display("Written:_%h", predefined_data[i]);
70         end else begin
71             WRITE_ENABLE_TOP = 0;
72             WRITE_DATA_IN_top = 8'hx;

```

```

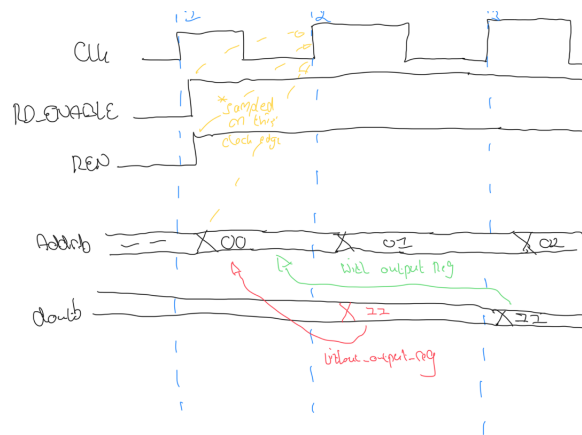
73         end
74     end
75     WRITE_ENABLE_TOP = 0;
76     WRITE_DATA_IN_top = 8'hx;
77
78     // Small delay before reading
79     #(5*CLK_PERIOD_WR);
80
81     // Read from FIFO
82     for (int i = 0; i < 20; i++) begin
83         @(posedge RCLK_top);
84         #1;
85         if (!EMPTY_top) begin
86             READ_ENABLE_TOP = 1;
87             // #7;
88             // @(posedge RCLK_top); // Wait for data to be valid
89             if (fifo_queue.size() > 0) begin
90                 automatic logic [7:0] expected_value = fifo_queue.pop_front
91                     ();
92                 $display("Read_□Data:□%h,□Expected:□%h,□Match:□%s",
93                     WRITE_DATA_OUT_top, expected_value,
94                     (WRITE_DATA_OUT_top == expected_value) ? "YES" : "
95                     NO");
96             end
97         end else begin
98             READ_ENABLE_TOP = 0;
99         end
100     end
101     READ_ENABLE_TOP = 0;
102
103     // End Simulation
104     #(20*CLK_PERIOD_WR);
105     $stop;
106 end
endmodule

```





### 5.6.1 Output Delayed by Extra Clock due to Output Registers



- First clock edge, read enables go high
  - Address & read-penal on at 0
- 2nd clock edge, address is incremented to 0x1
  - Data from address 0x00 is read out [11]. This is when there are no output registers
  - When there are output registers, the data [11] is put into registers
- 3rd clock edge: address is incremented to 0x2
  - Data from address 0x00 [11] is read out (system. low. output registers)
  - Data from address 0x01 [02] is put into the registers (system. low. output registers)
- \* On this clock edge: Both read-enables are seen to be high. This means data can be read. The address is seen to be 0x00. Thus data at 0x00 [0x02] is output [if no output registers]. If output registers data is output on next clock edge.

Figure 17: Hand drawn waveforms displaying where doutb would first output data if no output registers were enabled v were it does when output registers are enabled