

EE533: Network Processor Design & Programming

Lab #8: Convertible First-In-First-Out (SRAM) Memory

Instructor: Prof. Young Cho, PhD

Team Number: **#3**

Project Partners:

Member #1: **Sarthak Jain**

Member #2: **Archit Sethi**

Member #3: **Justin Santos**

Designed, Created, and Submitted by Team #3

University of Southern California

Los Angeles, CA 90007

Team #4 GitHub Repository Link: [Team#3 Link](#)

1. Design and Implementation

a. FIFO Structure: The FIFO Structure is built using dual-port SRAM which enables the read and write operations simultaneously. The architecture consists of:

i. Memory block (SRAM-based)

1. Stores the incoming network packets
2. It is organized as a Circular Buffer

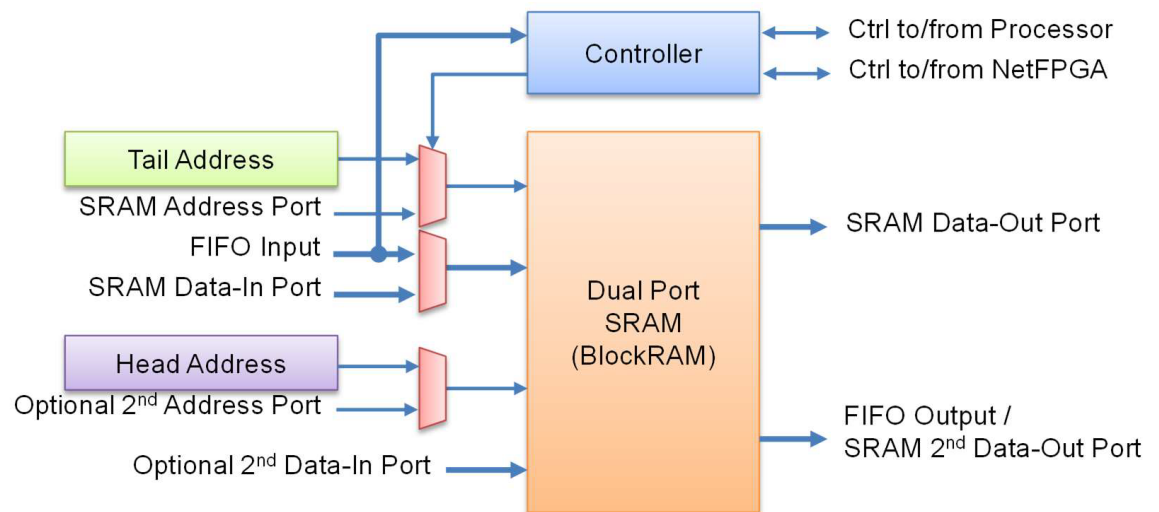
ii. Head and Tail Pointers

1. The HEAD pointer displays the read position
2. The TAIL pointer marks the write position
3. These 2 pointers are updated when the data is being read or written

iii. Control signals

1. FIFO FULL: This control signal prevents overwriting when the buffer is FULL.
2. FIFO EMPTY: It indicates no data is available.
3. Data Ready: It signals when a complete packet is stored in FIFO.

b. Block Diagram



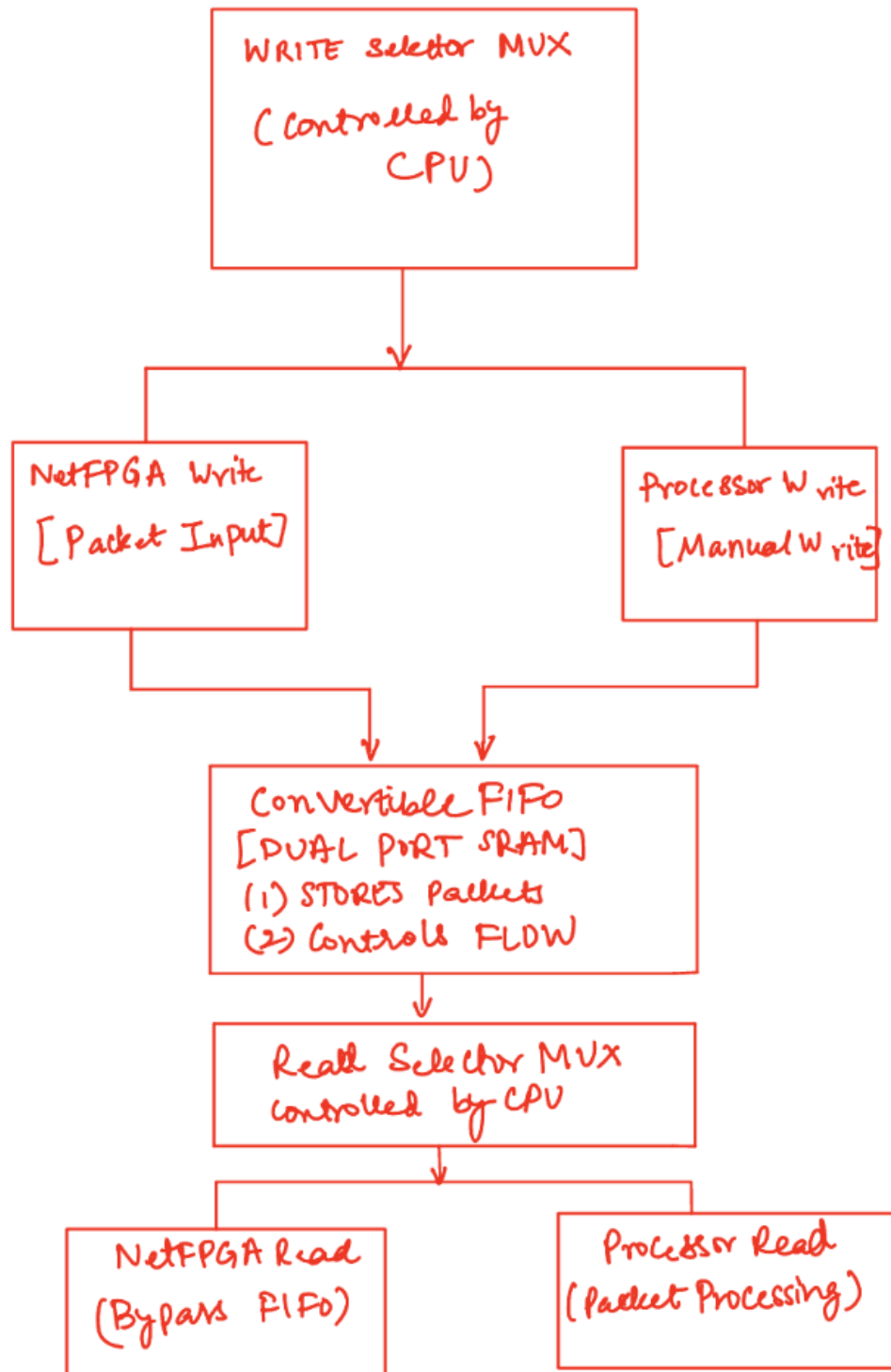
Let us explain the complete Block diagram data flow step-by-step:

Dual Port SRAM: This is the actual memory storage for our FIFO Design. It has 2 independent address and data ports which allows to write data and another port to read data.

Address and Data Ports:

<u>Port Name</u>	<u>Purpose</u>	<u>Controlled By</u>
<i>SRAM Address Port</i>	Points to the memory location where data will be read/written	Tail Address (write), Head Address (read)
<i>SRAM Data-In Port</i>	Data input into the FIFO (from NetFPGA)	FIFO Input
<i>FIFO Output / SRAM 2nd Data-Out Port</i>	Data output (to the processor)	Head Address (read pointer)
<i>Optional 2nd Address/Data Port</i>	Allows a second processor or interface to read/write data	Not always used, but allows more flexibility

Multiplexer (MUX) Integration: In order to allow the NetFPGA and the processor to access FIFO, we used a MUX to select between (a) NetFPGA's Datapath-for writing incoming packets & (b) Processor's MEM stage for reading and processing packets. The BRAM is acting as both FIFO and Processor's MEM stage.



2. Simulation *Convertible FIFO.v*:

```

1  module CONVERTIBLE_FIFO
2
3  --reg CPU_mode;
4
5  --assign bram_re<= (state == PROCESS_PACKET && cpu_read_enable);
6  --assign bram_we<= (state == RECEIVE_PACKET && netfpga_write_enable) || (state == CPU_WRITE && cpu_write_enable);
7  --assign cpu_mem_mode<= cpu_read_enable | cpu_write_enable;
8
9  --always @(posedge clock)
10
11  --begin
12  --if(reset)
13  --begin
14  --state<= IDLE;
15  --head<= 1'd0;
16  --tail<= 1'd0;
17  --status_register<= 1'd0;
18  --fifo_full<= 1'd0;
19  --fifo_empty<= 1'd1;
20  --bram_we<= 1'd0;
21  --bram_re<= 1'd0;
22  --CPU_mode<= 1'd1;
23  --end
24  --else
25  --begin
26  --case(state)
27  --//IDLE: Waiting for NetFPGA to start writing a packet or CPU control signal to decide between Dmem mode or FIFO.
28  --IDLE:
29  --begin
30  --bram_we<= 0;
31  --bram_re<= 0;
32
33  --if(cpu_mem_mode)
34  --begin
35  --CPU_mode<= 1'd1;
36  --state<= MEMORY_MODE;
37  --end
38  --else-if(netfpga_write_enable)
39  --begin
40  --state<= RECEIVE_PACKET;
41  --CPU_mode<= 1'd0;
42  --end
43  --end
44
45  --//MEMORY_MODE: Processor Reads or Writes to BRAM
46  --MEMORY_MODE:
47  --begin
48  --bram_we<= cpu_write_enable;
49  --bram_re<= cpu_read_enable; //Enable BRAM read when CPU requests
50
51  --if(!cpu_write_enable && !cpu_read_enable)
52  --begin
53  --state<= IDLE;
54  --CPU_mode<= 1'd0;
55  --end
56  --end
57
58  --//RECEIVE_PACKET: Enable BRAM write for incoming packet data
59  --RECEIVE_PACKET:
60  --begin
61  --bram_we<= 1; //Enable BRAM Write
62  --bram_re<= 0;
63  --if(!fifo_full)
64  --begin
65  --tail<= (tail==1023)? 0: tail + 1; //Move tail
66  --fifo_empty<= 0;
67  --end
68
69  --if(packet_end)
70  --begin //When NetFPGA finishes sending packet
71  --state<= PACKET_BUFFERED;
72  --bram_we<= 0; //Stop BRAM writes
73  --status_register<= 1; //Notify processor
74
75  --end
76
77  --end
78
79  --end
80
81  --end
82
83  --end
84
85  --end
86
87  --end
88
89  --end
90
91  --end
92
93  --end
94
95  --end
96
97  --end
98
99  --end
100
101  --end
102
103  --end
104
105  --end
106
107  --end
108
109  --end
110
111  --end
112
113  --end
114
115  --end
116
117  --end
118
119  --end
120
121  --end
122
123  --end
124
125  --end
126
127  --end
128
129  --end
130
131  --end
132
133  --end
134
135  --end
136
137  --end
138
139  --end
140
141  --end
142
143  --end
144
145  --end
146
147  --end
148
149  --end
150
151  --end
152
153  --end
154
155  --end
156
157  --end
158
159  --end
160
161  --end
162
163  --end
164
165  --end
166
167  --end
168
169  --end
170
171  --end
172
173  --end
174
175  --end
176
177  --end
178
179  --end
180
181  --end
182
183  --end
184
185  --end
186
187  --end
188
189  --end
190
191  --end
192
193  --end
194
195  --end
196
197  --end
198
199  --end
200
201  --end
202
203  --end
204
205  --end
206
207  --end
208
209  --end
210
211  --end
212
213  --end
214
215  --end
216
217  --end
218
219  --end
220
221  --end
222
223  --end
224
225  --end
226
227  --end
228
229  --end
230
231  --end
232
233  --end
234
235  --end
236
237  --end
238
239  --end
240
241  --end
242
243  --end
244
245  --end
246
247  --end
248
249  --end
250
251  --end
252
253  --end
254
255  --end
256
257  --end
258
259  --end
260
261  --end
262
263  --end
264
265  --end
266
267  --end
268
269  --end
270
271  --end
272
273  --end
274
275  --end
276
277  --end
278
279  --end
280
281  --end
282
283  --end
284
285  --end
286
287  --end
288
289  --end
290
291  --end
292
293  --end
294
295  --end
296
297  --end
298
299  --end
300
301  --end
302
303  --end
304
305  --end
306
307  --end
308
309  --end
310
311  --end
312
313  --end
314
315  --end
316
317  --end
318
319  --end
320
321  --end
322
323  --end
324
325  --end
326
327  --end
328
329  --end
330
331  --end
332
333  --end
334
335  --end
336
337  --end
338
339  --end
340
341  --end
342
343  --end
344
345  --end
346
347  --end
348
349  --end
350
351  --end
352
353  --end
354
355  --end
356
357  --end
358
359  --end
360
361  --end
362
363  --end
364
365  --end
366
367  --end
368
369  --end
370
371  --end
372
373  --end
374
375  --end
376
377  --end
378
379  --end
380
381  --end
382
383  --end
384
385  --end
386
387  --end
388
389  --end
390
391  --end
392
393  --end
394
395  --end
396
397  --end
398
399  --end
400
401  --end
402
403  --end
404
405  --end
406
407  --end
408
409  --end
410
411  --end
412
413  --end
414
415  --end
416
417  --end
418
419  --end
420
421  --end
422
423  --end
424
425  --end
426
427  --end
428
429  --end
430
431  --end
432
433  --end
434
435  --end
436
437  --end
438
439  --end
440
441  --end
442
443  --end
444
445  --end
446
447  --end
448
449  --end
450
451  --end
452
453  --end
454
455  --end
456
457  --end
458
459  --end
460
461  --end
462
463  --end
464
465  --end
466
467  --end
468
469  --end
470
471  --end
472
473  --end
474
475  --end
476
477  --end
478
479  --end
480
481  --end
482
483  --end
484
485  --end
486
487  --end
488
489  --end
490
491  --end
492
493  --end
494
495  --end
496
497  --end
498
499  --end
500
501  --end
502
503  --end
504
505  --end
506
507  --end
508
509  --end
510
511  --end
512
513  --end
514
515  --end
516
517  --end
518
519  --end
520
521  --end
522
523  --end
524
525  --end
526
527  --end
528
529  --end
530
531  --end
532
533  --end
534
535  --end
536
537  --end
538
539  --end
540
541  --end
542
543  --end
544
545  --end
546
547  --end
548
549  --end
550
551  --end
552
553  --end
554
555  --end
556
557  --end
558
559  --end
560
561  --end
562
563  --end
564
565  --end
566
567  --end
568
569  --end
570
571  --end
572
573  --end
574
575  --end
576
577  --end
578
579  --end
580
581  --end
582
583  --end
584
585  --end
586
587  --end
588
589  --end
590
591  --end
592
593  --end
594
595  --end
596
597  --end
598
599  --end
600
601  --end
602
603  --end
604
605  --end
606
607  --end
608
609  --end
610
611  --end
612
613  --end
614
615  --end
616
617  --end
618
619  --end
620
621  --end
622
623  --end
624
625  --end
626
627  --end
628
629  --end
630
631  --end
632
633  --end
634
635  --end
636
637  --end
638
639  --end
640
641  --end
642
643  --end
644
645  --end
646
647  --end
648
649  --end
650
651  --end
652
653  --end
654
655  --end
656
657  --end
658
659  --end
660
661  --end
662
663  --end
664
665  --end
666
667  --end
668
669  --end
670
671  --end
672
673  --end
674
675  --end
676
677  --end
678
679  --end
680
681  --end
682
683  --end
684
685  --end
686
687  --end
688
689  --end
690
691  --end
692
693  --end
694
695  --end
696
697  --end
698
699  --end
700
701  --end
702
703  --end
704
705  --end
706
707  --end
708
709  --end
710
711  --end
712
713  --end
714
715  --end
716
717  --end
718
719  --end
720
721  --end
722
723  --end
724
725  --end
726
727  --end
728
729  --end
730
731  --end
732
733  --end
734
735  --end
736
737  --end
738
739  --end
740
741  --end
742
743  --end
744
745  --end
746
747  --end
748
749  --end
750
751  --end
752
753  --end
754
755  --end
756
757  --end
758
759  --end
760
761  --end
762
763  --end
764
765  --end
766
767  --end
768
769  --end
770
771  --end
772
773  --end
774
775  --end
776
777  --end
778
779  --end
780
781  --end
782
783  --end
784
785  --end
786
787  --end
788
789  --end
790
791  --end
792
793  --end
794
795  --end
796
797  --end
798
799  --end
800
801  --end
802
803  --end
804
805  --end
806
807  --end
808
809  --end
810
811  --end
812
813  --end
814
815  --end
816
817  --end
818
819  --end
820
821  --end
822
823  --end
824
825  --end
826
827  --end
828
829  --end
830
831  --end
832
833  --end
834
835  --end
836
837  --end
838
839  --end
840
841  --end
842
843  --end
844
845  --end
846
847  --end
848
849  --end
850
851  --end
852
853  --end
854
855  --end
856
857  --end
858
859  --end
860
861  --end
862
863  --end
864
865  --end
866
867  --end
868
869  --end
870
871  --end
872
873  --end
874
875  --end
876
877  --end
878
879  --end
880
881  --end
882
883  --end
884
885  --end
886
887  --end
888
889  --end
890
891  --end
892
893  --end
894
895  --end
896
897  --end
898
899  --end
900
901  --end
902
903  --end
904
905  --end
906
907  --end
908
909  --end
910
911  --end
912
913  --end
914
915  --end
916
917  --end
918
919  --end
920
921  --end
922
923  --end
924
925  --end
926
927  --end
928
929  --end
930
931  --end
932
933  --end
934
935  --end
936
937  --end
938
939  --end
940
941  --end
942
943  --end
944
945  --end
946
947  --end
948
949  --end
950
951  --end
952
953  --end
954
955  --end
956
957  --end
958
959  --end
960
961  --end
962
963  --end
964
965  --end
966
967  --end
968
969  --end
970
971  --end
972
973  --end
974
975  --end
976
977  --end
978
979  --end
980
981  --end
982
983  --end
984
985  --end
986
987  --end
988
989  --end
990
991  --end
992
993  --end
994
995  --end
996
997  --end
998
999  --end
1000

```

```

82 //PACKET_BUFFERED: FIFO has full packet, wait for processor
83 PACKET_BUFFERED:
84 begin
85     bram_we <= 0;
86     fifo_full <= (tail + 1 == head); //FIFO Full Check
87     if (cpu_read_enable)
88     begin
89         state <= MEMORY_MODE;
90     end
91     else if (cpu_write_enable)
92     begin
93         state <= CPU_WRITE;
94     end
95 end
96
97 //PROCESS_PACKET: Enable BRAM Read for 5 stage Processor
98 PROCESS_PACKET:
99 begin
100     bram_re <= 1; //Read from BRAM
101     if (cpu_read_enable && !fifo_empty)
102     begin
103         head <= (head == 1023) ? 0 : head + 1; //Move head
104     end
105     if (head == tail)
106     begin
107         state <= FIFO_EMPTY;
108         status_register <= 0;
109         fifo_full <= 0;
110     end
111     else
112     begin
113         state <= MEMORY_MODE;
114     end
115 end
116
117 //CPU_WRITE: Enable BRAM Write for Processor
118 CPU_WRITE:
119 begin
120     bram_we <= 1; //Write to BRAM
121     if (cpu_write_enable && !fifo_full)
122     begin
123         tail <= (tail == 1023) ? 0 : tail + 1; //Move tail
124     end
125
126     if (cpu_write_complete)
127     begin
128         status_register <= 1; //Indicate new packet available
129         state <= PACKET_BUFFERED;
130     end
131     else
132     begin
133         state <= MEMORY_MODE;
134     end
135 end
136
137 //FIFO_EMPTY: FIFO is empty, waiting for new packets
138 FIFO_EMPTY:
139 begin
140     bram_we <= 0;
141     bram_re <= 0;
142     fifo_empty <= 1;
143     if (netfpga_write_enable)
144     begin
145         state <= RECEIVE_PACKET;
146     end
147 end
148
149 default: state <= IDLE;
150 endcase
endcase

```