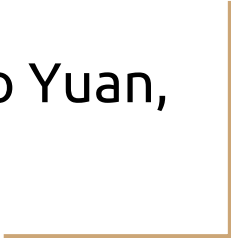




Coder Internal Interleaver

Vinith Sharma, Yao Yuan,
Cheng Lyu

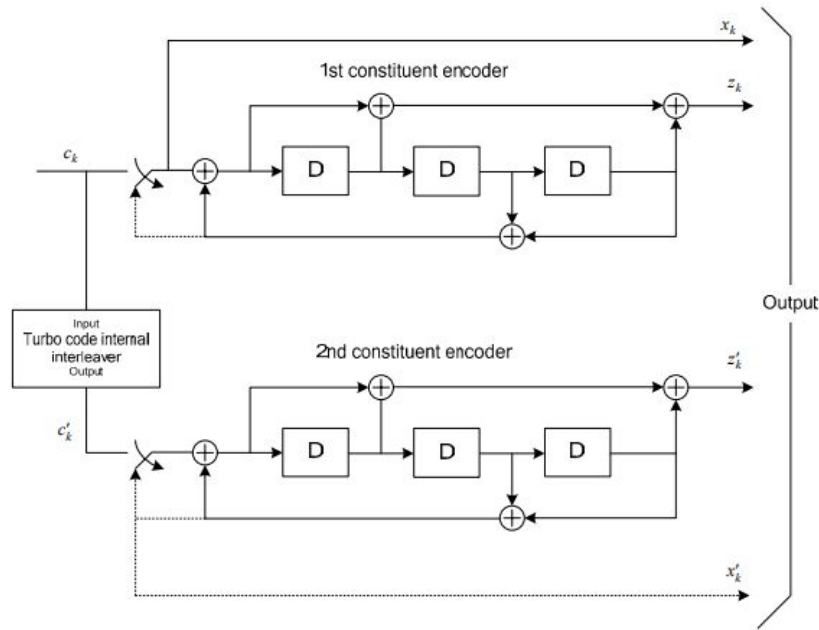


Introduction



1. LTE Advanced
2. Coding, multiplexing and mapping to/from physical channels
3. Data control with the MAC layer
4. Coding schemes
 - a. Error detection
 - b. Error correction
 - c. Rate matching
 - d. Interleaving
 - e. Transport

Internal Interleaver



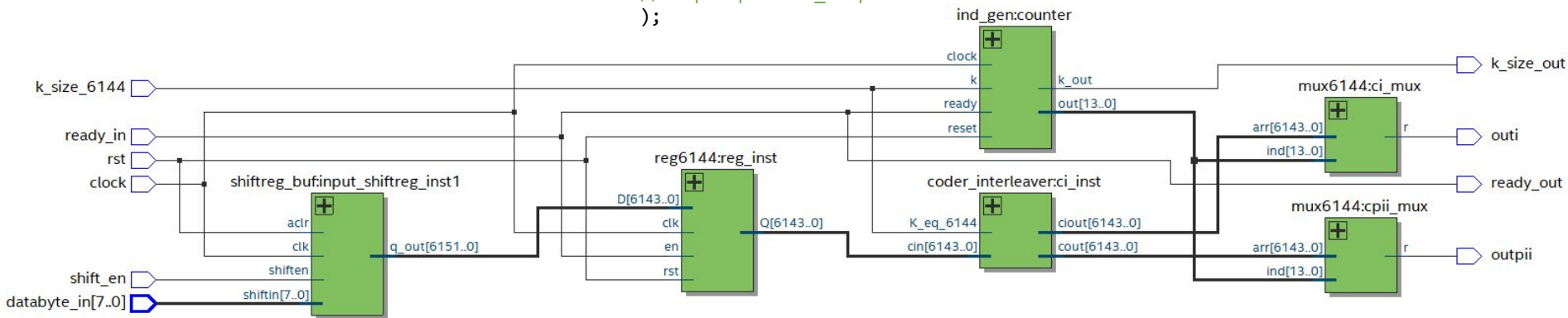
$$c'_i = c_{\Pi(i)}, i=0, 1, \dots, (K-1) \quad \Pi(i) = (f_1 \cdot i + f_2 \cdot i^2) \bmod K$$

Table 5.1.3-3: Turbo code internal interleaver parameters.

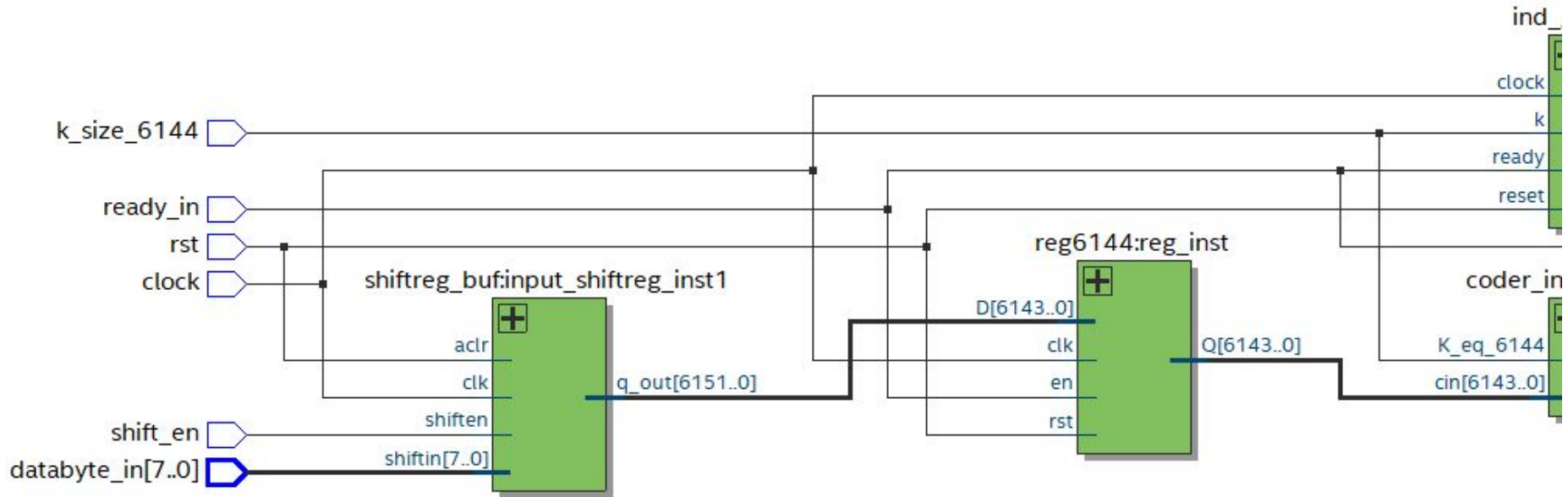
i	K	f_1	f_2	i	K	f_1	f_2	i	K	f_1	f_2	i	K	f_1	f_2
1	40	3	10	48	416	25	52	95	1120	67	140	142	3200	111	240
2	48	7	12	49	424	51	106	96	1152	35	72	143	3264	443	204
3	56	19	42	50	432	47	72	97	1184	19	74	144	3328	51	104
4	64	7	16	51	440	91	110	98	1216	39	76	145	3392	51	212
5	72	7	18	52	448	29	168	99	1248	19	78	146	3456	451	192
6	80	11	20	53	456	29	114	100	1280	199	240	147	3520	257	220
7	88	5	22	54	464	247	58	101	1312	21	82	148	3584	57	336
8	96	11	24	55	472	29	118	102	1344	211	252	149	3648	313	228
9	104	7	26	56	480	89	180	103	1376	21	86	150	3712	271	232
10	112	41	84	57	488	91	122	104	1408	43	88	151	3776	179	236
11	120	103	90	58	496	157	62	105	1440	149	60	152	3840	331	120
12	128	15	32	59	504	55	84	106	1472	45	92	153	3904	363	244
13	136	9	34	60	512	31	64	107	1504	49	846	154	3968	375	248
14	144	17	108	61	528	17	66	108	1536	71	48	155	4032	127	168
15	152	9	38	62	544	35	68	109	1568	13	28	156	4096	31	64
16	160	21	120	63	560	227	420	110	1600	17	80	157	4160	33	130
17	168	101	84	64	576	65	96	111	1632	25	102	158	4224	43	264
18	176	21	44	65	592	19	74	112	1664	183	104	159	4288	33	134
19	184	57	46	66	608	37	76	113	1696	55	954	160	4352	477	408
20	192	23	48	67	624	41	234	114	1728	127	96	161	4416	35	138
21	200	13	50	68	640	39	80	115	1760	27	110	162	4480	233	280
22	208	27	52	69	656	185	82	116	1792	29	112	163	4544	357	142
23	216	11	36	70	672	43	252	117	1824	29	114	164	4608	337	480
24	224	27	56	71	688	21	86	118	1856	57	116	165	4672	37	146
25	232	85	58	72	704	155	44	119	1888	45	354	166	4736	71	444
26	240	29	60	73	720	79	120	120	1920	31	120	167	4800	71	120
27	248	33	62	74	736	139	92	121	1952	59	610	168	4864	37	152
28	256	15	32	75	752	23	94	122	1984	185	124	169	4928	39	462
29	264	17	198	76	768	217	48	123	2016	113	420	170	4992	127	234
30	272	33	68	77	784	25	98	124	2048	31	64	171	5056	39	158
31	280	103	210	78	800	17	80	125	2112	17	66	172	5120	39	80
32	288	19	36	79	816	127	102	126	2176	171	136	173	5184	31	96
33	296	19	74	80	832	25	52	127	2240	209	420	174	5248	113	902
34	304	37	76	81	848	239	106	128	2304	253	216	175	5312	41	166
35	312	19	78	82	864	17	48	129	2368	367	444	176	5376	251	336
36	320	21	120	83	880	137	110	130	2432	265	456	177	5440	43	170
37	328	21	82	84	896	215	112	131	2496	181	468	178	5504	21	86
38	336	115	84	85	912	29	114	132	2560	39	80	179	5568	43	174
39	344	193	86	86	928	15	58	133	2624	27	164	180	5632	45	176
40	352	21	44	87	944	147	118	134	2688	127	504	181	5696	45	178
41	360	133	90	88	960	29	60	135	2752	143	172	182	5760	161	120
42	368	81	46	89	976	59	122	136	2816	43	88	183	5824	89	182
43	376	45	94	90	992	65	124	137	2880	29	300	184	5888	323	184
44	384	23	48	91	1008	55	84	138	2944	45	92	185	5952	47	186
45	392	243	98	92	1024	31	64	139	3008	157	188	186	6016	23	94
46	400	151	40	93	1040	17	66	140	3072	47	96	187	6080	47	190
47	408	155	102	94	1056	171	204	141	3136	13	28	188	6144	263	480

Design & Interfaces

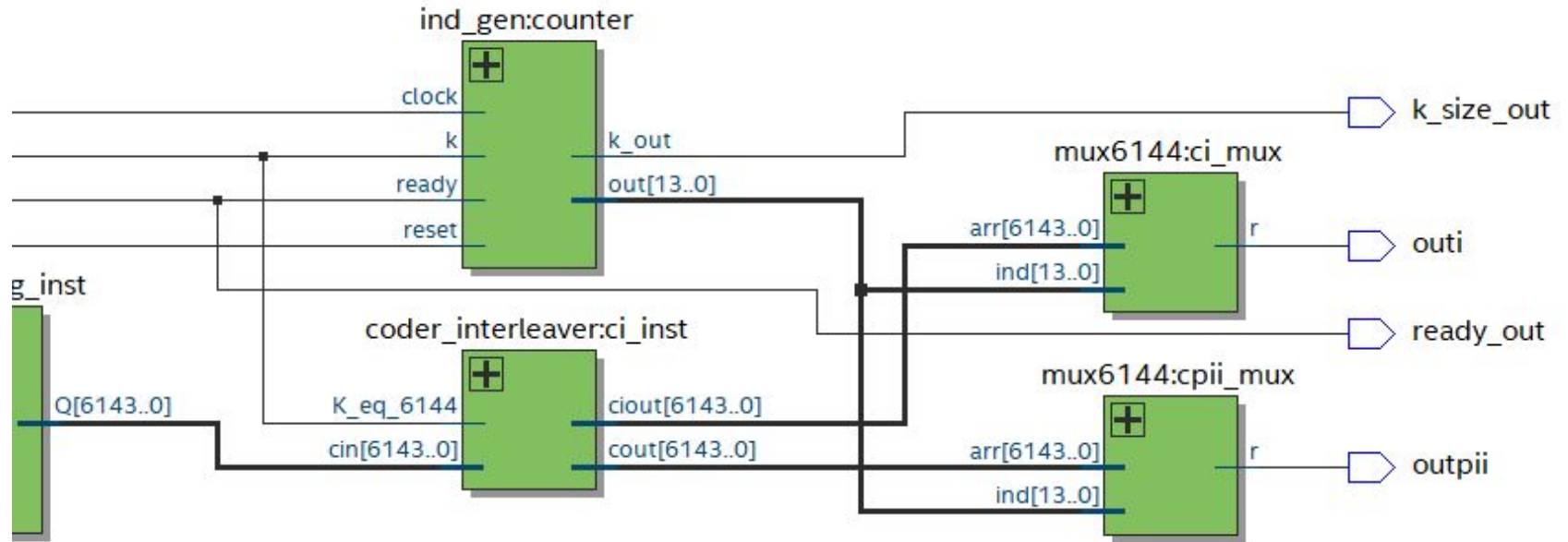
```
module interleaver_top_level_early(
    input k_size_6144,           // 0 if 1056, 1 if 6144 block size
    input [7:0] databyte_in,    // byte-wise serial input
    input clock,                // clock
    input rst,                  // reset
    input shift_en,              // data needs to be shifted in
    input ready_in,              // input bit from CSU to indicate block is done
    output ready_out,            // asserted to tell turbo coder to start
    output k_size_out,           // 0 if K = 1056, 1 if K = 6144; K is block size
    output outi,                 // bit-serial output of input, i.e. ci
    output outpii                // bit-serial output after interleaving, i.e. cpii
    //output process_complete
);
```



Design - Left Half



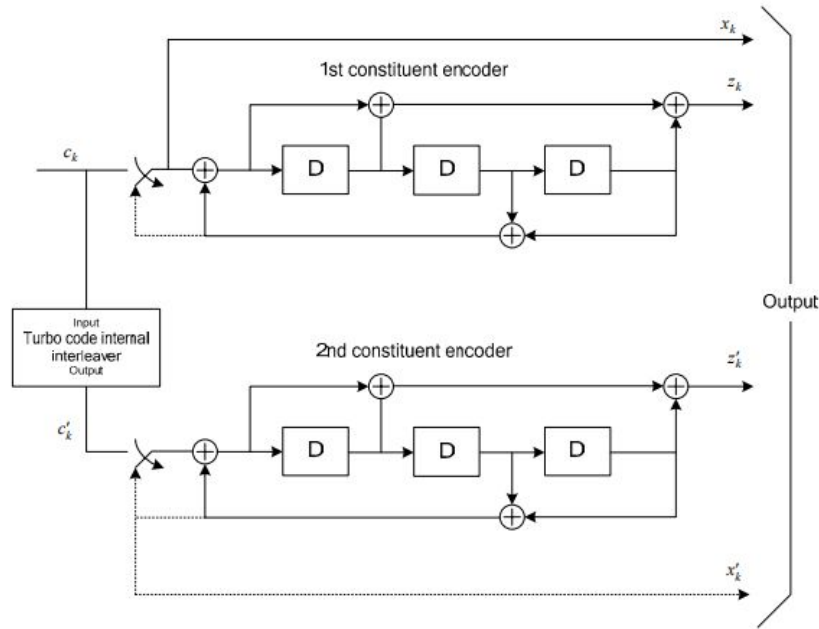
Design - Right Half



Interface

```
module interleaver_top_level_early(  
    input k_size_6144,           // 0 if 1056, 1 if 6144 block size  
    input [7:0] databyte_in,    // byte-wise serial input  
    input clock,                 // clock  
    input rst,                   // reset  
    input shift_en,              // data needs to be shifted in  
    input ready_in,              // input bit from CSU to indicate block is done  
    output ready_out,            // asserted to tell turbo coder to start  
    output k_size_out,           // 0 if K = 1056, 1 if K = 6144; K is block size  
    output outi,                 // bit-serial output of input, i.e. ci  
    output outpii                // bit-serial output after interleaving, i.e. cp  
    // output_complete  
);
```

Internal Interleaver

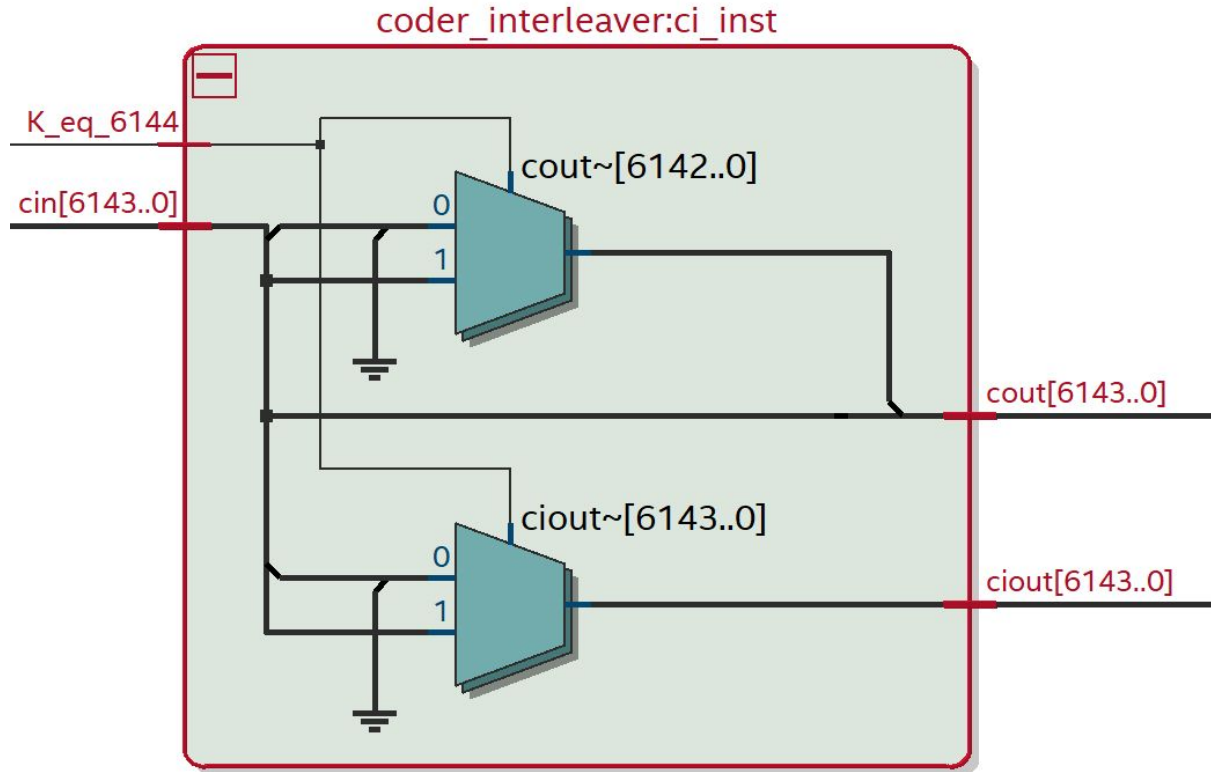


$$c'_i = c_{\Pi(i)}, i=0, 1, \dots, (K-1) \quad \Pi(i) = (f_1 \cdot i + f_2 \cdot i^2) \bmod K$$

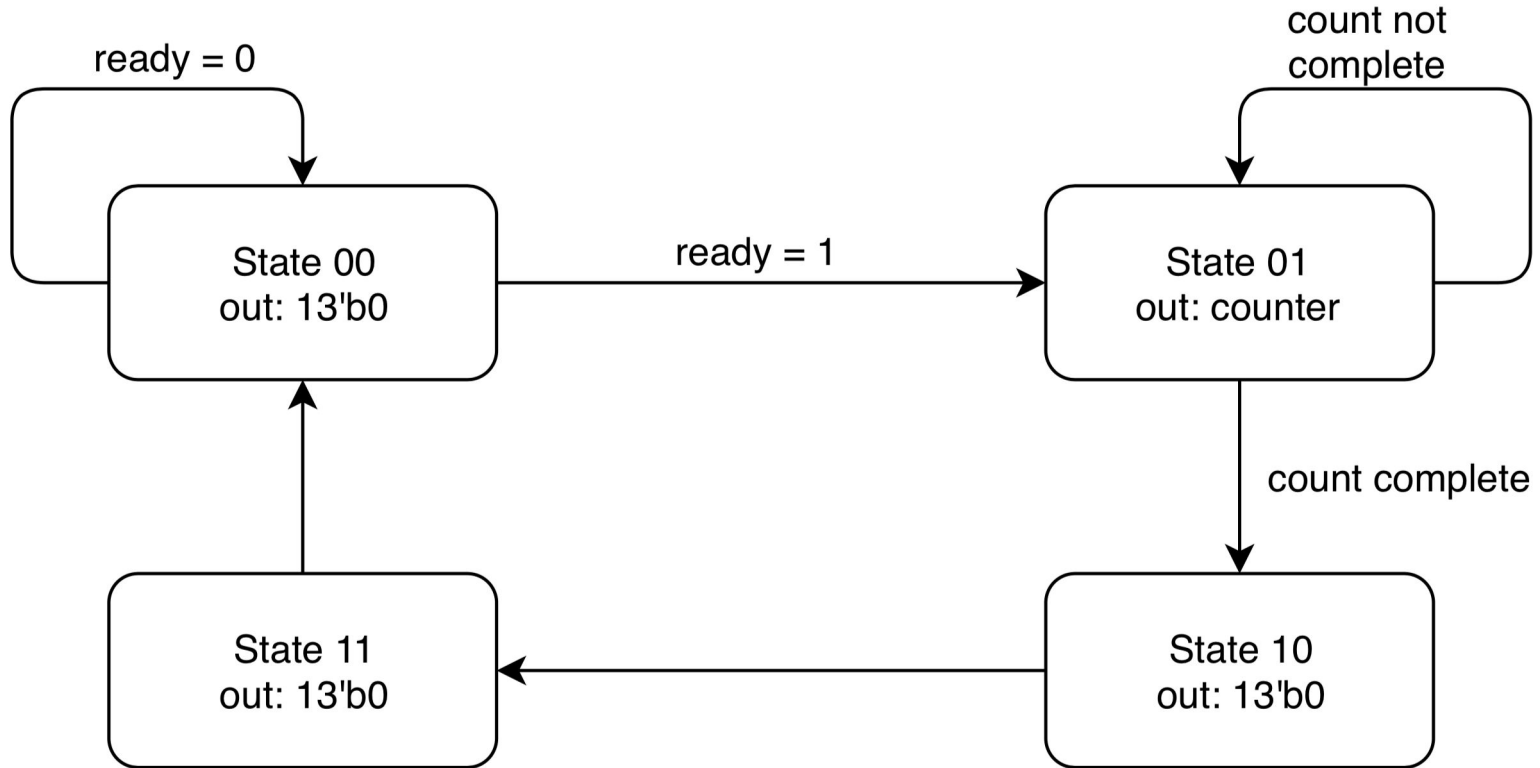
Table 5.1.3-3: Turbo code internal interleaver parameters.

i	K	f_1	f_2	i	K	f_1	f_2	i	K	f_1	f_2	i	K	f_1	f_2
1	40	3	10	48	416	25	52	95	1120	67	140	142	3200	111	240
2	48	7	12	49	424	51	106	96	1152	35	72	143	3264	443	204
3	56	19	42	50	432	47	72	97	1184	19	74	144	3328	51	104
4	64	7	16	51	440	91	110	98	1216	39	76	145	3392	51	212
5	72	7	18	52	448	29	168	99	1248	19	78	146	3456	451	192
6	80	11	20	53	456	29	114	100	1280	199	240	147	3520	257	220
7	88	5	22	54	464	247	58	101	1312	21	82	148	3584	57	336
8	96	11	24	55	472	29	118	102	1344	211	252	149	3648	313	228
9	104	7	26	56	480	89	180	103	1376	21	86	150	3712	271	232
10	112	41	84	57	488	91	122	104	1408	43	88	151	3776	179	236
11	120	103	90	58	496	157	62	105	1440	149	60	152	3840	331	120
12	128	15	32	59	504	55	84	106	1472	45	92	153	3904	363	244
13	136	9	34	60	512	31	64	107	1504	49	846	154	3968	375	248
14	144	17	108	61	528	17	66	108	1536	71	48	155	4032	127	168
15	152	9	38	62	544	35	68	109	1568	13	28	156	4096	31	64
16	160	21	120	63	560	227	420	110	1600	17	80	157	4160	33	130
17	168	101	84	64	576	65	96	111	1632	25	102	158	4224	43	264
18	176	21	44	65	592	19	74	112	1664	183	104	159	4288	33	134
19	184	57	46	66	608	37	76	113	1696	55	954	160	4352	477	408
20	192	23	48	67	624	41	234	114	1728	127	96	161	4416	35	138
21	200	13	50	68	640	39	80	115	1760	27	110	162	4480	233	280
22	208	27	52	69	656	185	82	116	1792	29	112	163	4544	357	142
23	216	11	36	70	672	43	252	117	1824	29	114	164	4608	337	480
24	224	27	56	71	688	21	86	118	1856	57	116	165	4672	37	146
25	232	85	58	72	704	155	44	119	1888	45	354	166	4736	71	444
26	240	29	60	73	720	79	120	120	1920	31	120	167	4800	71	120
27	248	33	62	74	736	139	92	121	1952	59	610	168	4864	37	152
28	256	15	32	75	752	23	94	122	1984	185	124	169	4928	39	462
29	264	17	198	76	768	217	48	123	2016	113	420	170	4992	127	234
30	272	33	68	77	784	25	98	124	2048	31	64	171	5056	39	158
31	280	103	210	78	800	17	80	125	2112	17	66	172	5120	39	80
32	288	19	36	79	816	127	102	126	2176	171	136	173	5184	31	96
33	296	19	74	80	832	25	52	127	2240	209	420	174	5248	113	902
34	304	37	76	81	848	239	106	128	2304	253	216	175	5312	41	166
35	312	19	78	82	864	17	48	129	2368	367	444	176	5376	251	336
36	320	21	120	83	880	137	110	130	2432	265	456	177	5440	43	170
37	328	21	82	84	896	215	112	131	2496	181	468	178	5504	21	86
38	336	115	84	85	912	29	114	132	2560	39	80	179	5568	43	174
39	344	193	86	86	928	15	58	133	2624	27	164	180	5632	45	176
40	352	21	44	87	944	147	118	134	2688	127	504	181	5696	45	178
41	360	133	90	88	960	29	60	135	2752	143	172	182	5760	161	120
42	368	81	46	89	976	59	122	136	2816	43	88	183	5824	89	182
43	376	45	94	90	992	65	124	137	2880	29	300	184	5888	323	184
44	384	23	48	91	1008	55	84	138	2944	45	92	185	5952	47	186
45	392	243	98	92	1024	31	64	139	3008	157	188	186	6016	23	94
46	400	151	40	93	1056	17	66	140	3072	47	96	187	6080	47	190
47	408	155	102	94	1088	171	204	141	3136	13	28	188	6144	263	480

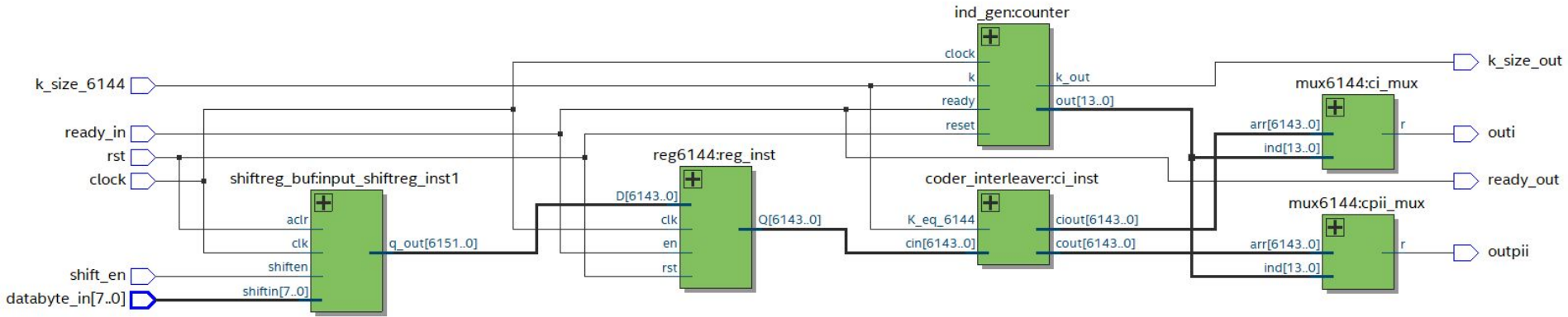
Interleaver remapping



FSM



Design



Design New Design/Final Design



Software Simulation



Output Verification

```
6 def interleaver(input_num, flag_6144):
7     f1 = 263 if flag_6144 == 1 else 17
8     f2 = 480 if flag_6144 == 1 else 66
9     k = 6144 if flag_6144 == 1 else 1056
10    input_inv = input_num[::-1]
11
12    output = [''] * k
13    for i in range(k):
14        pi_i = (f1 * i + f2 * i ** 2) % k
15        output[i] = input_inv[pi_i]
16    return ''.join(output)[::-1]
```

Our group's entire work in one method

```
33 def get_bin(hex_num):
34     hex2bin_map = {
35         "0": "0000",
36         "1": "0001",
37         "2": "0010",
38         "3": "0011",
39         "4": "0100",
40         "5": "0101",
41         "6": "0110",
42         "7": "0111",
43         "8": "1000",
44         "9": "1001",
45         "A": "1010",
46         "B": "1011",
47         "C": "1100",
48         "D": "1101",
49         "E": "1110",
50         "F": "1111",
51     }
52     return "".join(hex2bin_map[i] for i in hex_num)
```

Map of hexadecimal numbers to a binary string

Output Verification

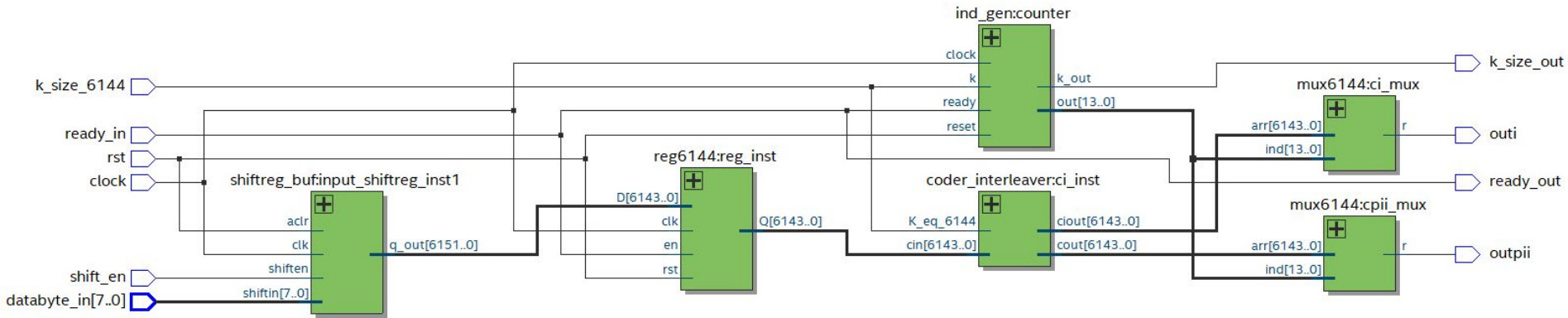
Steps used in testing with the CSU block:

1. Generate hex sequence (99 - 102)
2. Convert sequence to binary (126)
3. Determine k_size_6144 (129)
4. Generate the desired output using the interleaver method previously shown (130)
5. CSU verified using ModelSim

```
99      test_1_hex = ''
100     for _ in range(129):
101         test_1_hex += 'C9'
102     test_1_hex += '2A1438'
```

```
123     test_input_hex = [test_1_hex, test_2_hex, test_3_hex, test_4_hex]
124     test_input_bin = []
125     for hex_num in test_input_hex:
126         test_input_bin.append(get_bin(hex_num))
127     test_out = []
128     for in_bin in test_input_bin:
129         le = 1 if len(in_bin) == 6144 else 0
130         test_out.append(interleaver(in_bin, le))
```

Testbench Simulation



- Fix input byte of data
- `shift_en` for 768 cycles
- Assert `ready_in`

- Monitoring FSM states & index
- Monitoring output and `ci` & `cpii`
- Verified functional correctness
- As well as interface timing

Python Prediction

```
input value of size 6144
1101111111011111110111111101111111011
1101111111011111110111111101111111011
1101111111011111110111111101111111011
1101111111011111110111111101111111011
1101111111011111110111111101111111011
1101111111011111110111111101111111011
1101111111011111110111111101111111011
1101111111011111110111111101111111011
1101111111011111110111111101111111011
1101111111011111110111111101111111011
```

Ci

```
output value
11110111111101111111011111110111111101
11110111111101111111011111110111111101
11110111111101111111011111110111111101
11110111111101111111011111110111111101
11110111111101111111011111110111111101
11110111111101111111011111110111111101
11110111111101111111011111110111111101
11110111111101111111011111110111111101
11110111111101111111011111110111111101
11110111111101111111011111110111111101
```

Cpii

Testbench Simulation

```
ready_in cycle:      6135
Time:                69120, cnt_state:1, mux index: 6138,   Outi 1, OutPii: 1
ready_in cycle:      6136
Time:                69130, cnt_state:1, mux index: 6139,   Outi 1, OutPii: 0
ready_in cycle:      6137
Time:                69140, cnt_state:1, mux index: 6140,   Outi 1, OutPii: 1
ready_in cycle:      6138
Time:                69150, cnt_state:1, mux index: 6141,   Outi 0, OutPii: 1
ready_in cycle:      6139
Time:                69160, cnt_state:1, mux index: 6142,   Outi 1, OutPii: 1
ready_in cycle:      6140
Time:                69170, cnt_state:1, mux index: 6143,   Outi 1, OutPii: 1
ready_in cycle:      6141
Time:                69180, cnt_state:2, mux index: 6143,   Outi 1, OutPii: 1
```

Python Prediction

```
input value of size 6144  
1101111111011111110111111101111111011  
1101111111011111110111111101111111011  
1101111111011111110111111101111111011  
1101111111011111110111111101111111011  
1101111111011111110111111101111111011  
1101111111011111110111111101111111011  
1101111111011111110111111101111111011  
1101111111011111110111111101111111011  
1101111111011111110111111101111111011  
1101111111011111110111111101111111011
```

Ci

```
output value  
11110111111101111111011111110111111101  
11110111111101111111011111110111111101  
11110111111101111111011111110111111101  
11110111111101111111011111110111111101  
11110111111101111111011111110111111101  
11110111111101111111011111110111111101  
11110111111101111111011111110111111101  
11110111111101111111011111110111111101  
11110111111101111111011111110111111101  
11110111111101111111011111110111111101
```

Cpii

Static Timing Analysis

```
wire [6143:0] remap_in,  
assign remap_in = reg_buf_out;  
coder_interleaver ci_inst(  
    .cin(remap_in),  
    .K_eq_6144(k_size_6144),  
    .cout(remap_out),  
    .ciout(ci_array)  
);
```

Worst Slack: 8.079
Max Frequency: 83.89 MHz

Wire Reduction

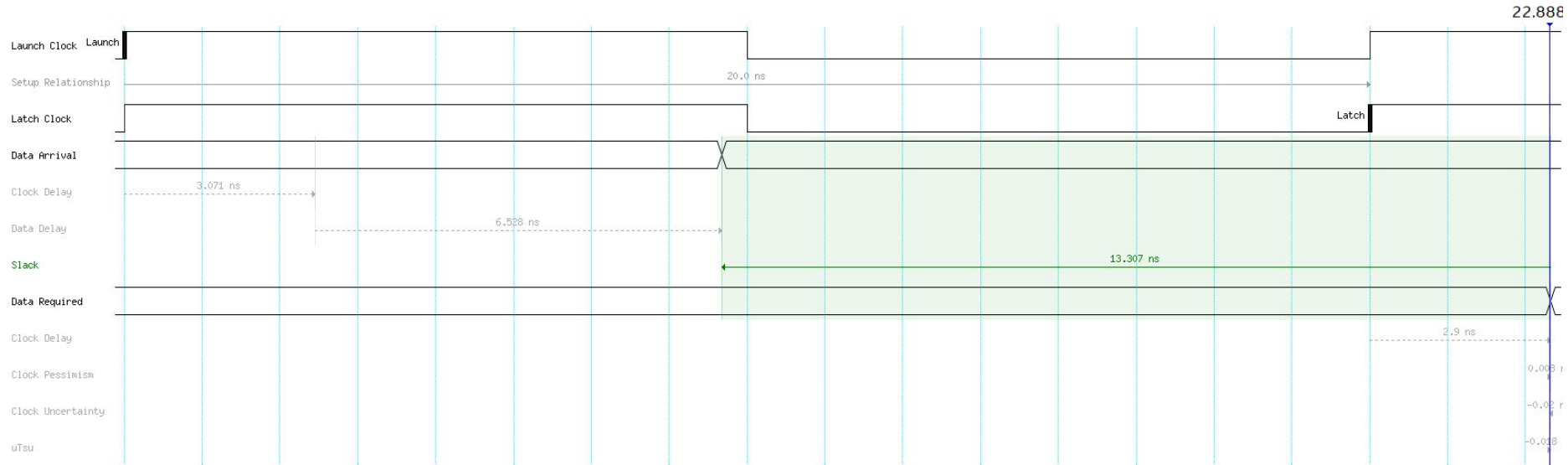
Worst Slack: 13.307
Max Frequency: 149.41 MHz

```
coder_interleaver ci_inst(  
    .cin(reg_buf_out),  
    .K_eq_6144(k_size_6144),  
    .cout(remap_out),  
    .ciout(ci_array)  
);
```

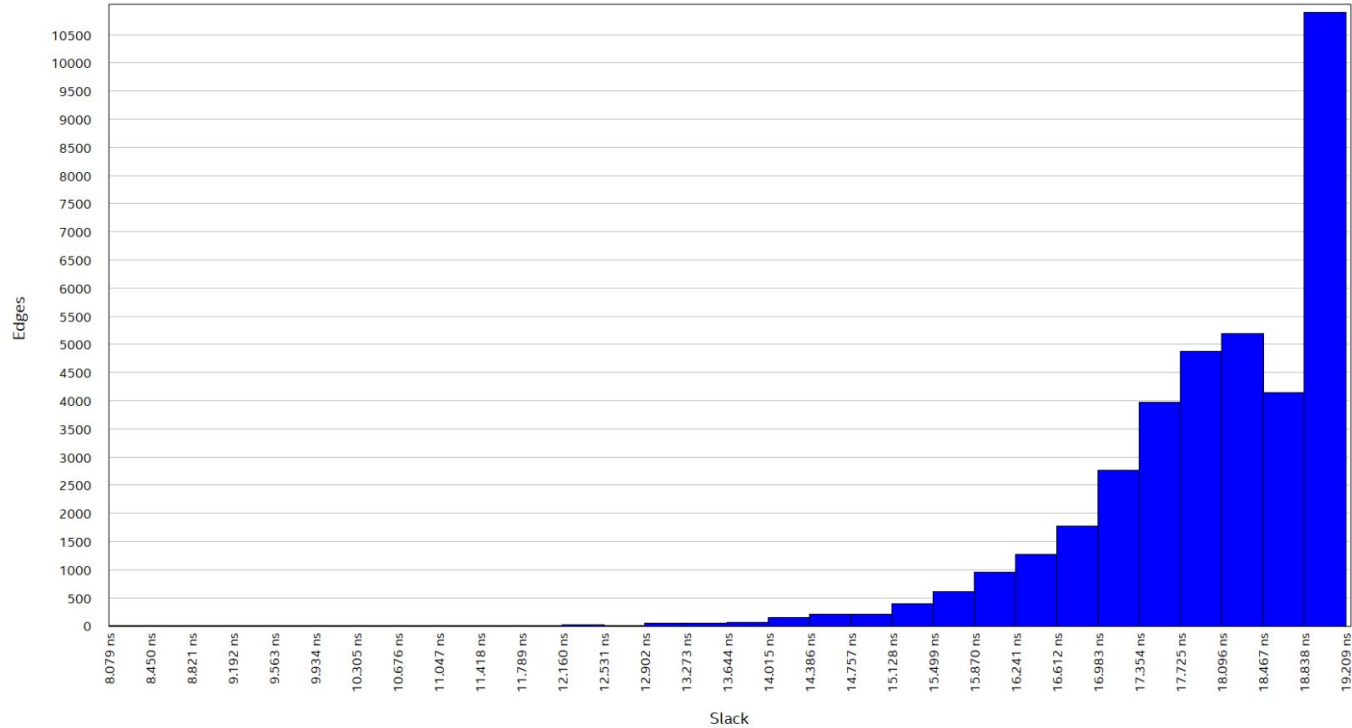
Static Timing Analysis

	Slack	From Node	To Node	Launch Clock	Latch Clock	Relationship	Clock Skew	Data Delay
1	13.307	shiftreg_buf:input_shiftreg_inst1 mem[1464]	reg6144:reg_inst Q[1464]	clock	clock	20.000	-0.163	6.528
2	13.632	ind_gen:counter cnt[7]	ind_gen:counter cnt[0]	clock	clock	20.000	-0.081	6.285
3	13.632	ind_gen:counter cnt[7]	ind_gen:counter cnt[11]	clock	clock	20.000	-0.081	6.285
4	13.632	ind_gen:counter cnt[7]	ind_gen:counter cnt[6]	clock	clock	20.000	-0.081	6.285
5	13.632	ind_gen:counter cnt[7]	ind_gen:counter cnt[1]	clock	clock	20.000	-0.081	6.285
6	13.632	ind_gen:counter cnt[7]	ind_gen:counter cnt[2]	clock	clock	20.000	-0.081	6.285
7	13.632	ind_gen:counter cnt[7]	ind_gen:counter cnt[3]	clock	clock	20.000	-0.081	6.285
8	13.632	ind_gen:counter cnt[7]	ind_gen:counter cnt[4]	clock	clock	20.000	-0.081	6.285
9	13.632	ind_gen:counter cnt[7]	ind_gen:counter cnt[5]	clock	clock	20.000	-0.081	6.285
10	13.632	ind_gen:counter cnt[7]	ind_gen:counter cnt[7]	clock	clock	20.000	-0.081	6.285
11	13.632	ind_gen:counter cnt[7]	ind_gen:counter cnt[12]	clock	clock	20.000	-0.081	6.285
12	13.632	ind_gen:counter cnt[7]	ind_gen:counter cnt[8]	clock	clock	20.000	-0.081	6.285
13	13.632	ind_gen:counter cnt[7]	ind_gen:counter cnt[9]	clock	clock	20.000	-0.081	6.285
14	13.632	ind_gen:counter cnt[7]	ind_gen:counter cnt[10]	clock	clock	20.000	-0.081	6.285
15	13.632	ind_gen:counter cnt[7]	ind_gen:counter cnt[13]	clock	clock	20.000	-0.081	6.285
16	13.770	ind_gen:counter cnt[10]	ind_gen:counter cnt[0]	clock	clock	20.000	-0.081	6.147
17	13.770	ind_gen:counter cnt[10]	ind_gen:counter cnt[11]	clock	clock	20.000	-0.081	6.147
18	13.770	ind_gen:counter cnt[10]	ind_gen:counter cnt[6]	clock	clock	20.000	-0.081	6.147
19	13.770	ind_gen:counter cnt[10]	ind_gen:counter cnt[1]	clock	clock	20.000	-0.081	6.147
20	13.770	ind_gen:counter cnt[10]	ind_gen:counter cnt[2]	clock	clock	20.000	-0.081	6.147
21	13.770	ind_gen:counter cnt[10]	ind_gen:counter cnt[3]	clock	clock	20.000	-0.081	6.147
22	13.770	ind_gen:counter cnt[10]	ind_gen:counter cnt[4]	clock	clock	20.000	-0.081	6.147
23	13.770	ind_gen:counter cnt[10]	ind_gen:counter cnt[5]	clock	clock	20.000	-0.081	6.147
24	13.770	ind_gen:counter cnt[10]	ind_gen:counter cnt[7]	clock	clock	20.000	-0.081	6.147
25	13.770	ind_gen:counter cnt[10]	ind_gen:counter cnt[12]	clock	clock	20.000	-0.081	6.147
26	13.770	ind_gen:counter cnt[10]	ind_gen:counter cnt[8]	clock	clock	20.000	-0.081	6.147
27	13.770	ind_gen:counter cnt[10]	ind_gen:counter cnt[9]	clock	clock	20.000	-0.081	6.147
28	13.770	ind_gen:counter cnt[10]	ind_gen:counter cnt[10]	clock	clock	20.000	-0.081	6.147
29	13.770	ind_gen:counter cnt[10]	ind_gen:counter cnt[13]	clock	clock	20.000	-0.081	6.147
30	13.809	shiftreg_buf:input_shiftreg_inst1 mem[2024]	reg6144:reg_inst Q[2024]	clock	clock	20.000	-0.118	6.071

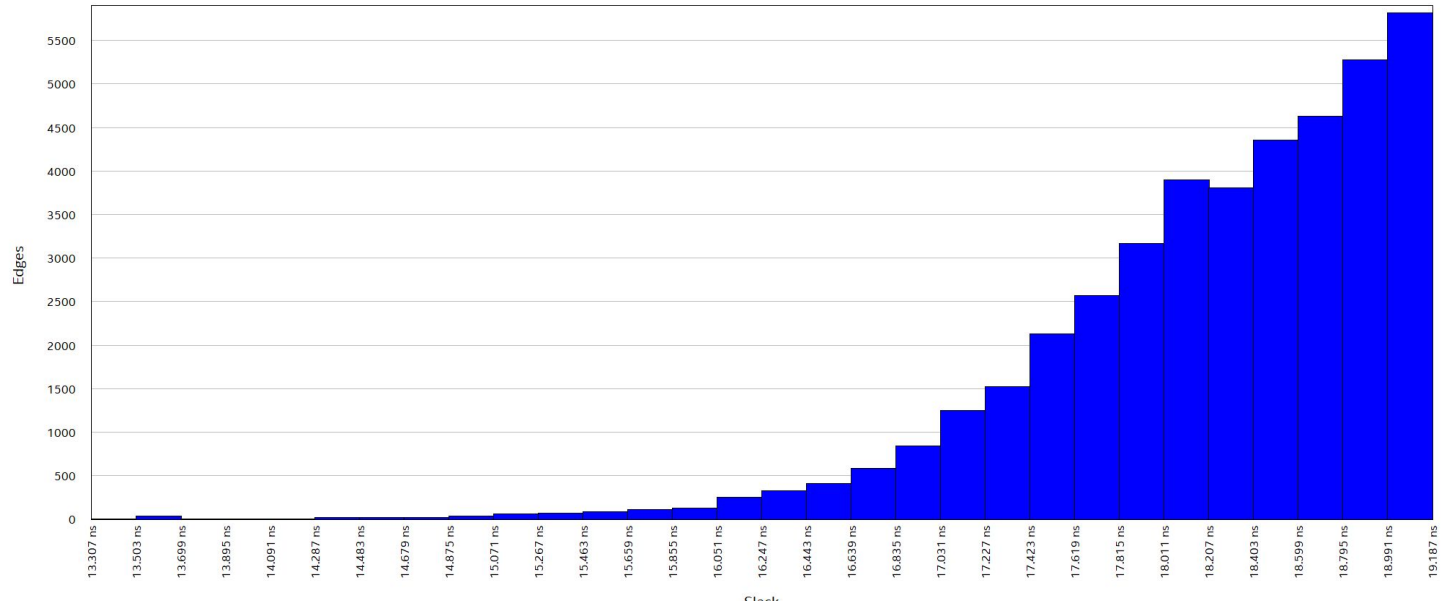
Static Timing Analysis



Static Timing Analysis



Static Timing Analysis



Hardware Results

Input patterns repeat themselves in the output sequence.

```
19 def create_val(last_number):
20     out_str = ''
21     for x in range(6143):
22         out_str += str(random.randint(0, 1))
23     out_str += last_number
24     return out_str
```

Uniform random bit string generator with a specified most significant bit.

```
27 def print_mif(input):
28     # print_in = list(input)
29     for x in range(768):
30         print '%s : %s;' % (x, input[x * 8:x * 8 + 8])
```

MIF file generator.

Creates the output shown below (starting at line 24).

```
23 CONTENT BEGIN
24     0 : 11110111;
25     1 : 00111100;
26     2 : 11000110;
27     3 : 11000100;
28     4 : 11100101;
29     5 : 00001100;
30     6 : 01010100;
31     7 : 00100111;
32     8 : 10111001;
```

Hardware Results

1110 0011 1011 0001 1011 1111

E 3 B 1 B F

24 LSB of the output sequence

```
27 def print_mif(input):
28     # print_in = list(input)
29     for x in range(768):
30         print '%s : %s;' % (x, input[x * 8:x * 8 + 8])
```

MIF file generator.
Creates the output shown below.

CONTENT BEGIN

0 : 11110111; F7

1 : 00111100;

763 : 11101101;

764 : 00011000;

765 : 11111000;

766 : 01111110;

767 : 00111011; 3B

Thanks you!