Convolution Codes II

6.0 Trace Back and Survivor memory

6.1 Trace-back

Up to now we have considered doing a complete trace back for each output bit. This is not the most efficient method. It is better to wait until one knows that several bits (say D bits) at the end of the trace back are consider correct. Then one can extract multiple D bits from only one trace back, and the decoder throughput can greatly improve.

Consider a decoder with 4 add-compare-select (ACS) blocks. Let the survivor memory which stores the outputs of the ACS blocks, consist of (D+S+W) 4-bit words. Define what will be called a *trace-back cycle*. On each cycle every address in memory is accessed, W of them for writing, S of them for tracing back to establish the best path, and D of them for reading back and extracting the output data. The total memory address space is always W+S+D, but the areas assigned to writing, trace-back and data output rotate from one *trace-back cycle* to the next.

To reiterate:
W is the address area where the ACS modules write their bits.

S is the surviving path memory address area used for tracing back through the trellis to find the path that does not die out. We have been using S=5*k=15, that is one must trace back 15 cycles before one is adequately sure one has the best data.

D is the address area of the data which will be sent out by the decoder at the end of the trace-back.

In the simple trace-back W=1=D. One traces back over the depth of S+D each time a new input symbol is processed. The final bit in D at the trace-back is used as output.

In this method the trace-back is made through S+D. S is made large enough to establish the survivor path. The trace-back through D is then used to obtain several bits of data. Thus one trace-back can give more than one data bit, and the decoder can run faster.

.D, S and W are three partitions of the address space. These partitions have a constant length but move dynamically by rotating around the memory.

There are two address pointers:
- A read address pointer which decrements after each read.
- Write address pointer which increments after each write.
One does not read and write at the same time.

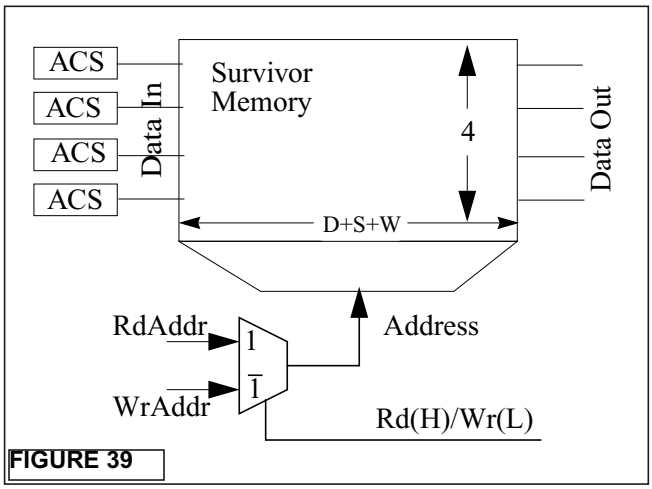


FIGURE 39

Simple Example with D=W=1

The simplest case is D=1, W=1. In this case, one writes the ACS outputs into the W area of memory. Then one traces-back through 15 previously written ACS outputs stored in the S area of memory. Then one traces back one more bit and sends it to the output as valid data. We say this last bit is in the D area of memory, not in the S area.

This is the 1st trace back cycle and takes 17 memory access cycles (1 write+16 reads).

In the 2nd trace-back cycle, the D, S and W address areas will all rotate one word. The previously written word becomes part of the new S area, and the left most word of the S area becomes the D area.

In the 3rd cycle, S will wrap around, and W and D will shift right one word.

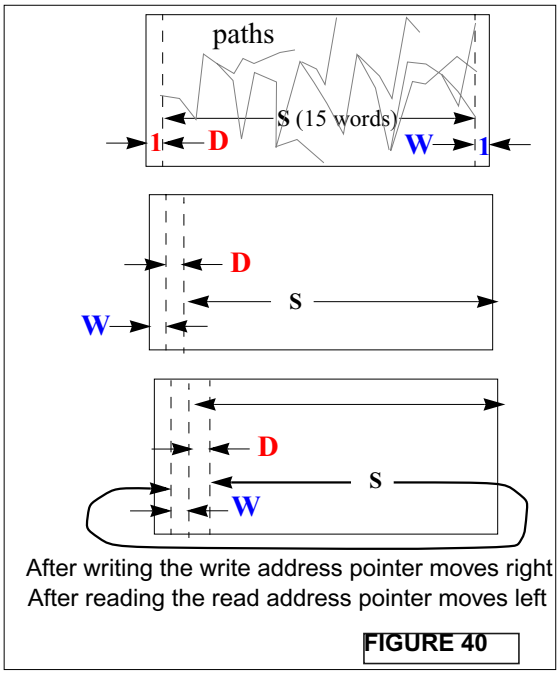


FIGURE 40

There are 17 memory cycles in a *trace-back cycle*, 1 write cycle, 15 read-for-trace back memory cycles, and 1 read-for-data memory cycle.

Example: Using 1 Write Followed by 9 Reads; D=2=W

To allow accessing all of memory in each *trace-back cycle* one must make S= 16 reads in the trace back instead of 15. This is a small price for nearly doubling the throughput.

Consider Figure 41. First write 1 word (a), followed by 9 read-backs (b). This takes one through the 1st read-back section. Then do another write at (c) followed by 7 read-backs (d) plus two reads of the data (e). These last two are sent out as output. This takes one through the 2nd read-back section.

This completes the first *trace-back cycle* which contained 20 memory cycles. At this point, the D, S and W areas are circularly shifted two address right, and the next write will be at (f).

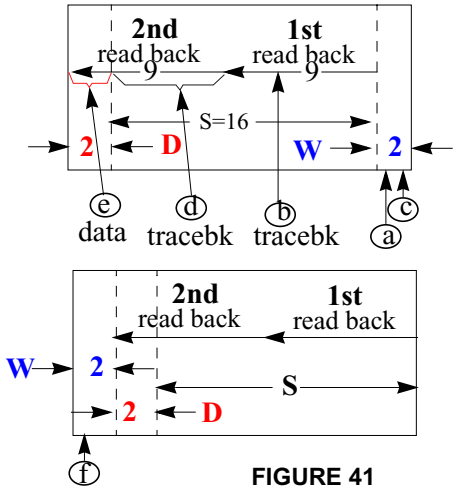


FIGURE 41

Example: Using 1 Write Followed by 2 Reads; D=16=W, S=16

Consider Figure 42. Here the write pointer starts at address 32 and writes one word. The read pointer starts at 31 and reads back two words. This continues until the write pointer has gone from 32 to 40 and the read pointer 31 to 16. Any further read back will be in the D region and should produce valid data.

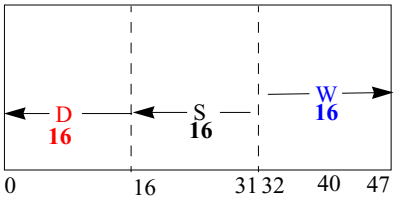


FIGURE 42

Continue writing one word and reading back two data words until the write pointer is at 47 and the read pointer is at 0. Then all 16 words of data have been read, and the meaning of the D, S and W address areas has shifted right 16 addresses. One is ready to start a new trace-back cycle as shown in Figure 43. For a more detailed view see Figure 44.

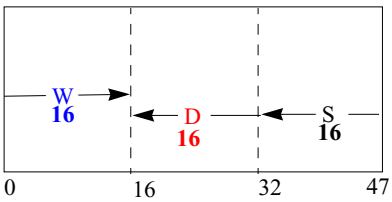
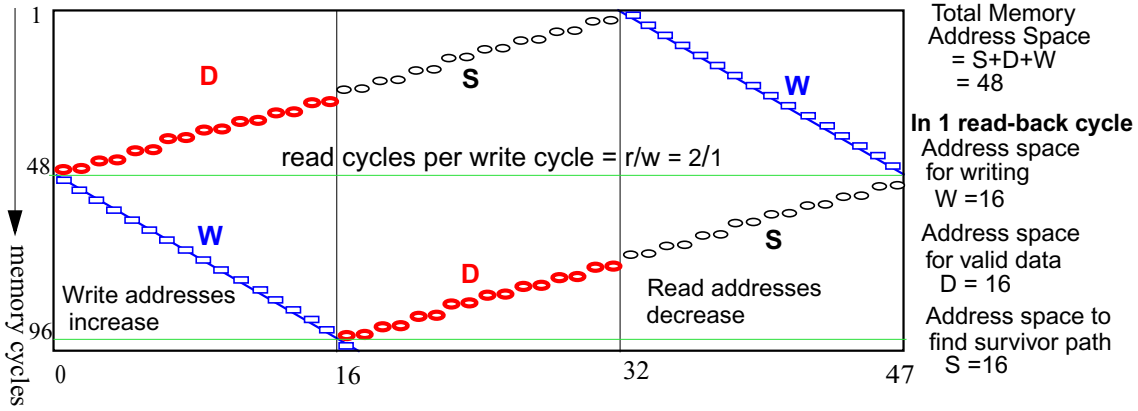


FIGURE 43

The *trace-back cycle* is 48 memory cycles,
16*(1 write followed by 2 reads).

Note this requires triple the memory needed if there is time for 16 reads between writes.

FIGURE 44 Details of read-write interleaving for the first two cycles



6.1.1 The General Case; w Writes Followed by r Reads

D=W always because processing D words for output must free the same number of words for the new writes.

The number of addresses read and written must be in the same proportion as the read speed to the write speed, thus

$$(S+D)/r = W/w$$

Solving gives

$$W = D = S(1/(r/w-1)$$

$$\text{Total memory} = W+D+S = S(w+r)/(r-w)$$

Speed ratio (the rate that ACS unit alone can do)/(rate survivor memory can do)
$= w/(w+r) = 1/(1+r)$ .... for w=1

Note that if w is any number but 1, the ACS module outputs will be rather messy. If you do not see why, think about implementing 2 writes and 3 reads,

Also note r/w must be greater than 1.

For a decoder with trace back starting from an arbitrary state, the rule of thumb is that the surviving path trace-back depth S, should be approximately 5*(constraint length).
For a constraint length of 3 this gives S = 15.

Table 1 on page 26, gives a choice of possible survivor memory schemes.

A timing graph for S=15, 4:1 r:w is shown in Figure 45. Assuming that memory can run at the clock speed, the clock must run at 5 times the symbol rate. A symbol here is 2 bits. 20

read cycles are used for trace-back, the last 5 of which (thick red lines) are sent out as the decoded signal. Then a new trace-back cycle starts using signals 5 cycles younger than the start of the previous trace-back.

FIGURE 45 Interleaving for S=16, r/w=5/1, W=D=4. Nine trace-back cycles shown.

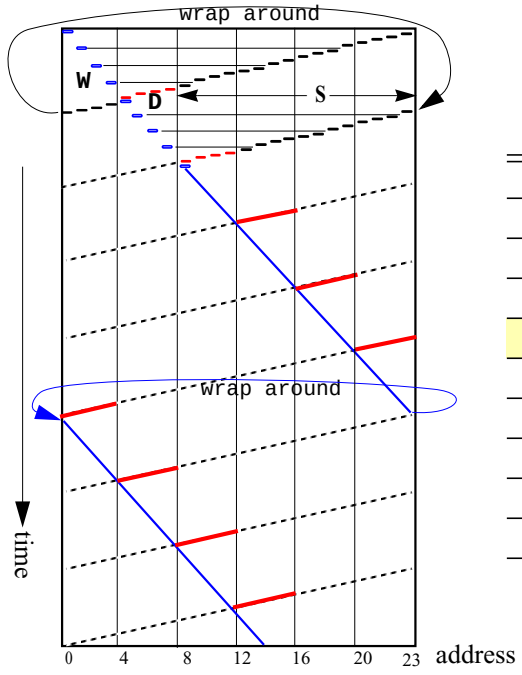


Table 1: Speed and memory size for different trace-back schemes

S	r,w	W=D	S+W+D	Speed
15	2,1	15	45	1/3
16	2,1	16	48	1/3
16	3,1	8	32	1/4
15	4,1	5	25	1/5
16	5,1	4	24	1/6
15	6,1	3	21	1/7
18	7,1	3	24	1/8
14	8,1	2	18	1/9
16	9,1	2	20	1/10
14	15,1	1	16	1/16

Speed is the fraction of the clock rate.

6.1.2 Choice for the Decoder Trace-Back Memory

Table 1 gives most of the useful choices for the trace-back memory parameters.

The choice S is choosen to be very close to 5*(constraint length). The exact value must be matched with r and w so that W and D are integers. The circuit is far simpler if w is made one. The choice of r then defines all the other parameters.

If the decoder must have a high throughput, then a small r is needed. (1/3, 1/4 of the clock rate). However this requires a much larger memory.

If the decoder can run slowly (1/16 of the clock rate), the memory can be reduced to 1/3 of the size needed for a high throughput.

If the D data can all be read consecutively, without a write in the middle, that is r ≥ D (r,w ≥ 5,1), the output shift register is much simpler. See Section 14.0 on page 46 for what happens if you don not do this..

Some possible applications

Compact disc data, is 16 bits x 44100 samples/sec x 2 channels. This would require a clock rate of 1.4112(r+w) Mb/s.

Advanced Television Systems Committee (ATSC) video compression is based on the MPEG-2 video standard. The compressed video and associated audio data streams are packetized along with auxiliary and control data into a 19.4Mb/s data stream. This would require a clock of 19.4(r+w) MHz.

A common application is cell phones. Here the data rate is 7.4 to 13 kbits/s

6.2 Following the Trace-back Path.

In the Viterbi trellis the linkages between states at time t and time t+1 are defined by a butterfly like pattern. This assumes the states are numbered according to the binary number stored in the encoder shift register, and that the data bit 'x' is shifted in on the left in the encoder shift register. See Figure 31 on 21.

To see how the backtrace works, look at the 8-state example in FIGURE 46 . As you already know, to go forward from a state, the transmitter shifts right, and a new bit enters on the left. The result is our new state.

Now, look at the example to go BACKWARDS from a particular state (we call it J). Assume we are somehow in state J=010. From the construction of the trellis, we know we must have come from either state 100 or 101. To determine which, we need to read back the 'winner' of the competition from memory. Recall that this bit, which we'll call "p", was calculated as one of the *came_from[3:0]* bits as we went forward with the ACS operations. As they were stored, the *came_from* bits should have the following associations:

ACS result from State 00 - to - came_from[0]
ACS result from State 10 - to - came_from[2]
ACS result from State 01 - to - came_from[1]
ACS result from State 11 - to - came_from[3]

To retrieve the came-from result for our current traceback state, we can then get it using the simple expression:

p = came_froms_out_of_memory[J]

And from the Figure, we see that our next traceback state is:

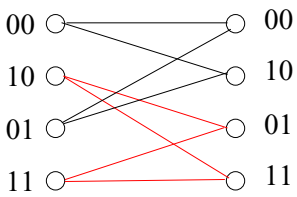
Jnext = {J << 1, p}

Note that you may need to invert the 'p' bit, depending on the convention you used for saving the 'winning' path in the *came_from* word.

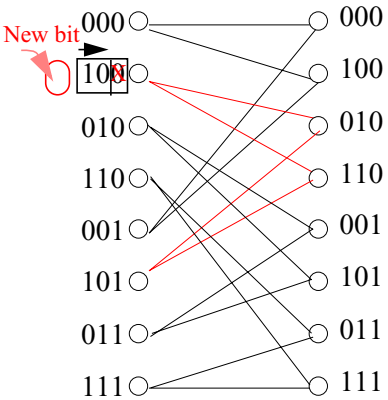
Fancy Extra Info: The 'butterfly' configuration is handy in BIG decoders because you can group butterflies which always communicate as a pair. This can reduce some hardware costs, but don't worry about it.

FIGURE 46 Trellis made of

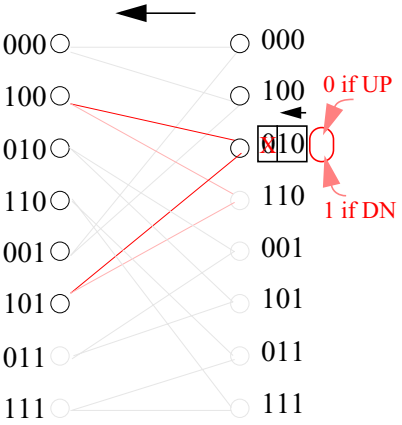
Your 4-state trellis diagram



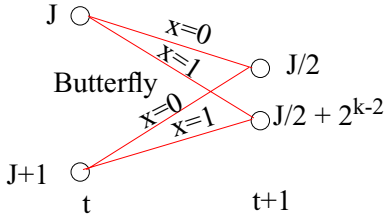
An 8-state trellis K4 Example



To go backwards from a given state



Mathematical Description of Butterfly
(For those who like math...)



6.3 Fourth Exercise: Survivor memory

1. **Problem:** Fill in the survivor memory for one trellis walk through. Figure 47 shows a walk through a trellis with the minimum and maximum path metrics, H, for each branch. The upper metric is for the upper path. In Figure 48, fill in the bits that would be stored in survivor memory to allow a back trace.

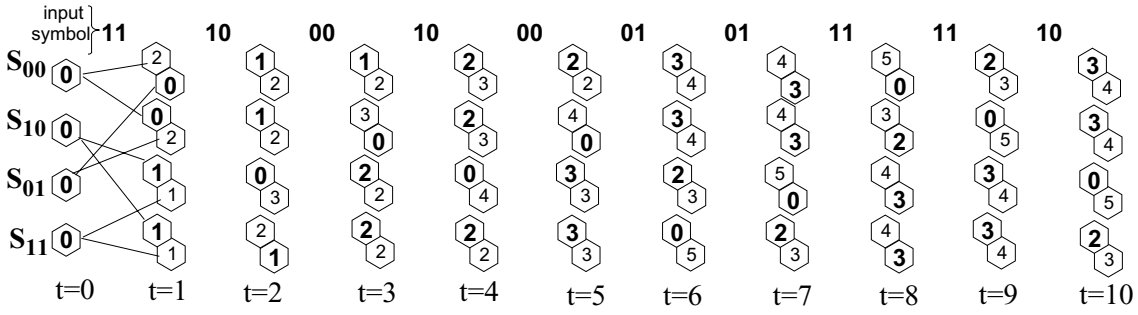
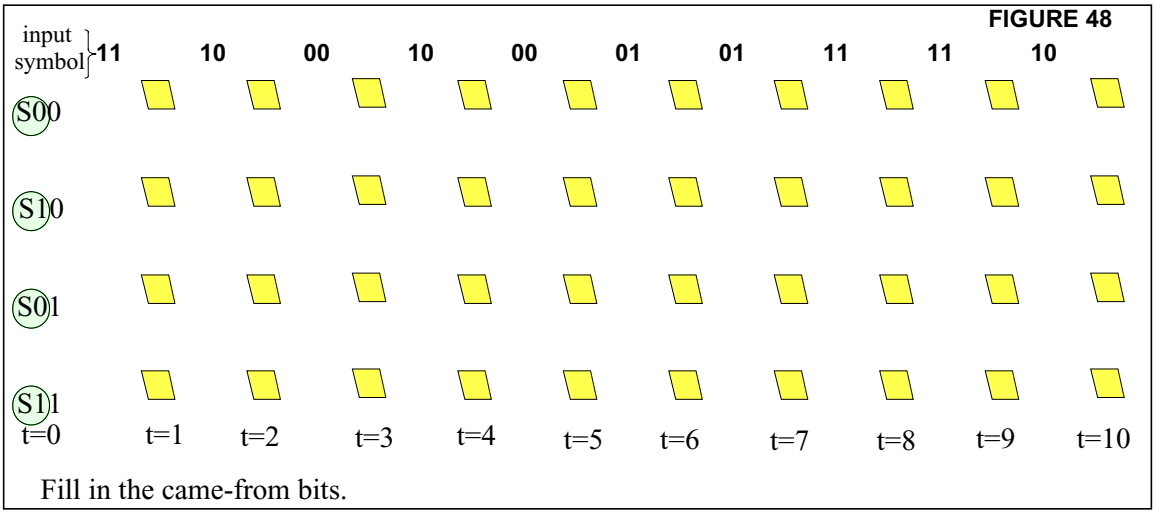


FIGURE 47

FIGURE 48



Fill in the came-from bits.

2. **Problem:** Using the bits stored in problem 1, and no other information, draw in the lines that show a back-track starting from state 0.

3. **Problem:** Assuming that the data is correct, once one has back traced to t=4, calculate the data bit that should be sent out for each of the times t= 4, 3, 2, 1.

4. **Problem:** Decide what sort of an application you might have for your decoder. Then decide whether you want high throughput, smaller memory, or a compromise. Then choose S, W, D, r and w. Write a short explanation for your choice. Fast design time is a reasonable design choice, in which case make $r \geq 5$. Why?

5. **Problem:** Draw a picture like Figure 45, showing the interleaving for your decoder.

6. **Problem:** Draw a picture like Figure 50 for your decoder showing the address, the input symbols, the clock, and seperate write control, read control and output control signals. Show a signals that counts upward for the write address, and one that counts downward for the read address. The write address counter should not increment during reads. Disable read address counting during writes.

7. **Problem:** Combine the write control and the read control into one signal wr(H)_Rd(L), *WrRd_N*. Develop pseudo code for a modulo r counter to generate this signal. Recall r is the number of read cycles per write cycle. Make r a parameter.

8. **Problem:** Develop pseudo code to write the vector of "p"s into survivor memory, read it back on the back trace, and calculate the *came from* state from the word read from the S or D section of memory.

```
always @(WrRd_N or Addr or ACS...)
   if (WrRd_N) // If write cycle
      SurvMem[Addr]<= ACS...;
assign wrd=SurvMem[Addr];
// Trace-Back Bit Select
   assign p = wrd[J];     // up or down ?
   assign Jnext = {J <<1,..};
```

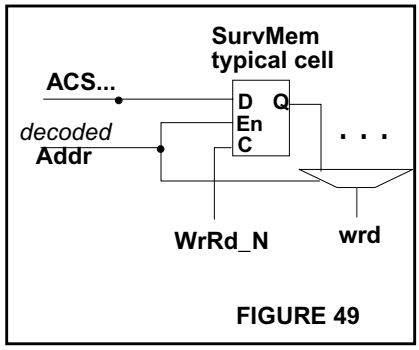


FIGURE 49

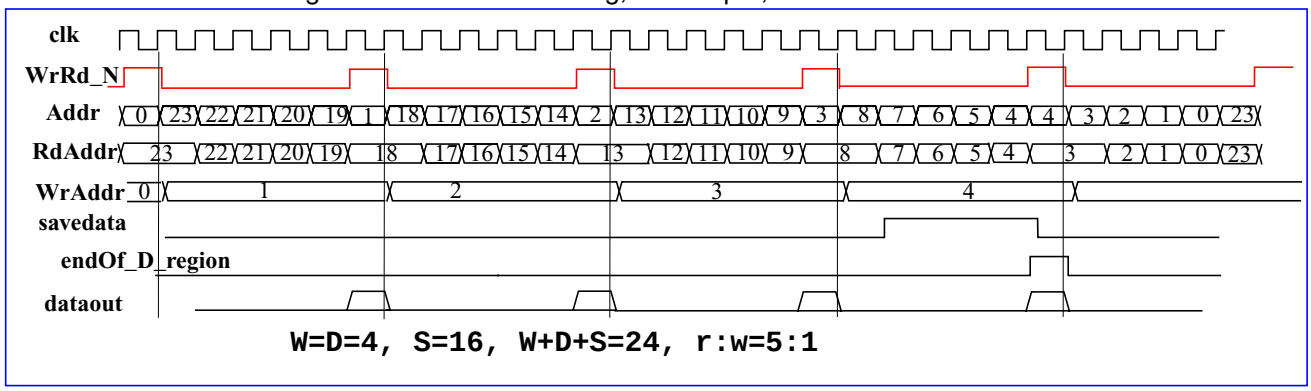
9. **Problem:** Consider the code for a pair of output shift registers called A and B. When the D area of memory is being read, it stores the D bits in the A shift register. It then transfers these bits to the B register which shifts them out on the `dataout` output line in reverse order. If B sends out one bit on every write, the output data will keep up with the incoming data flow.
"`saveddata`" is true during appropriate memory reads, as shown in Figure 50.
"d" is the bit recovered during the trace back.

```
always @(posedge clk)
   A<= Anxt;
always @(savedata or end_of_D_region or WrRd_N or A or d)
   begin: dual_shift_register  // A shifts left/right
    Anxt=A;
    if(savedata)  Anxt<={A,d};
    if(WrRd_N)     Anxt<= A>>1;
  end
assign dataout= A[0] & WrRd_N;
```

Draw a schematic diagram for a circuit that might be synthesized from the code. At a minimum, show the output of the survivor memory, the register or wires holding J, and the shift register drawn as a block showing the direction control lead and the *`dataout`* lead.

6.4 Timing the Survivor Memory

FIGURE 50 Detailed timing of read-write interleaving, and output, for the first few addresses



The timing of the survivor memory, is fairly complicated.There is a temptation to generate several clocks. Do not yield to temptation. The first rule of timing is:

do not generate new clocks without a very good reason.

Extra clocks must be well synchronized with each other. They make synthesis much harder, and they make testing much harder.

6.4.1 Hardware needed for the survivor memory

Four counters are needed to control this part of the decoder. Note that you already have one of these counters and it shouldn't be recreated. **Hint:** Your serial to parallel uses it, but it will need to be modified to match your chosen r+w numbers.

Your read/write control: Mod r+w counter to count the 1 write plus r read cycles.

A write address counter: Mod W+D+S counter to count up for write cycles.

A read address counter: Mod W+D+S counter to count down for the read cycles.

A traceback counter: A Mod D+S counter to tell when to send out the D output data bits. Your traceback consists of two sections. The first set of S reads is 'garbage' just to get you into a converged path, wheras the last D cycles are actually good bits. If, for example, you had S=16 and D=4, you would have a traceback counter from 19 downto 0, where your 'good-bits' come out during cycles 3,2,1,0.

The output data generated by the trace-back is in reverse order. The data is shifted into Reg. A in reverse order. At the end of the D region, the data is shifted into reg. B from which it is shifted out in correct order.

Also needed is an expression to select the proper bit for the path from each `wrd` read back.

FIGURE 51 Block diagram of the survivor memory and the trace-back control circuitry.

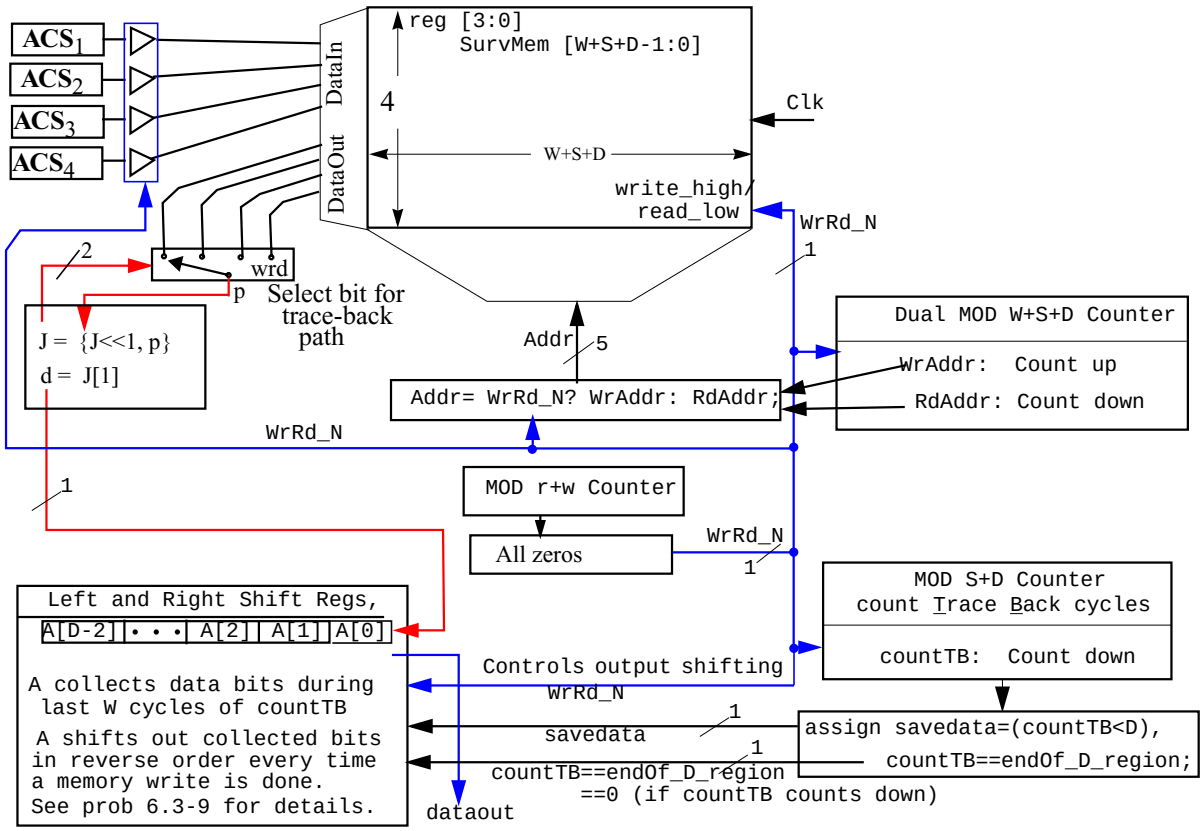


Figure 51 shows the four counters control the trace back. The mod (W+S+D) RdAddr counter should only count during read cycles, wheras the mod (W+S+D) WrAddr counter should only count during write cycles. The mod (S+D) traceback counter (countTB) keeps track of whether you are in a valid or ‘garbage’ traceback section and should advance only on reads. The last D counts of this counter is when the data should be valid and saved in the A shift register ready to be sent out.

Since the data is read back in the reverse order, it is reversed again when it is read out of the B register. There are two choices about how to send out the recovered data. One is to read it in a burst of W bits and then send out a burst. The other way is to send out one data bit every write cycle. Since there are W cycles before new data is ready to send out, this makes the input data rate from the ACS units match the output data rate.

IMPORTANT NOTE: The code above is NOT necessarily right for your system. For example, where the ‘savedata’ is high will depend on how many delays you have between the memory read and the output shift register. The same type of scenario will apply elsewhere. This is why you need to make a ‘map’ of what is occuring when in your system in relation to your master w+r counter.

7.0 Overflow in the Path Metric (H) Registers

7.0.1 Maximum Path Metric H

With errors, the path metric H can grow with the length of the message. The first approximation of the maximum H is the sum of all the maximum branch metrics h.

maximum(H) < 2+2+2+2+ . . .

However if any path has two many two's it will rapidly die out, so a more realistic maximum value is

maximum(H) = 1+1+1+1+1 ... , which sums to the length of the message.

Here, assume your message is no more than 64 bits, so allow 6 bits for H.
A better bound would show you only need 5 bits.

The difference between the minimum and maximum H will be much smaller than the maximum H. This could have been used to reduce register size; see Appendix I.

The gist of Appendix I, is that because the DIFFERENCE between the largest and smallest metrics in the system will be less than 5 bits you can use a mathematical trick to prevent overflow. Compute you metrics as normal, but to compare them use a subtraction rather than the < or > symbol. For example:

top_contender = (1 1000) = 24 or -8 (depending on how we look at the number)

bot_contender = (1 1010) = 26 or -6

---

top-bottom ...111**1** 1110

Because the result's sign bit is -ve, and we subtracted (top-bottom) the bottom number is bigger!

MORAL:, subtract and look at the sign bit to compare and you will not have an overflow problem.

TIP: Make you metric width easy to change so that you can make them very big for debugging - this will help you to look at what's going on without thinking about negative numbers.

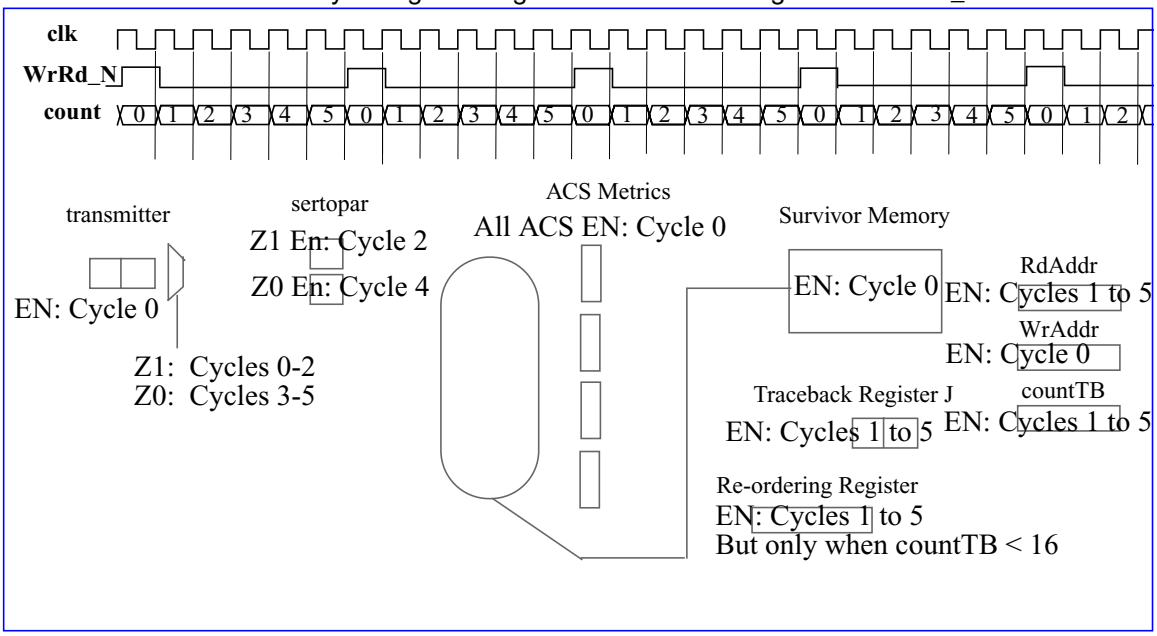
8.0 Laboratory 4: Survivor Memory

1. Laboratory assignment

a) Based on Section 7.0.1 on page 32, revise the size of the H registers in your Add-Compare-Select modules.

b) The original encoder advanced once in every 6 clock cycles. Now it should be apparent why we did this. For each data bit that the decoder processes, it needs w+r cycles. If the encoder sent a new bit every cycle the decoder would have to run off a much faster clock. Figure 52 shows an example timing map of a 'mythical' viterbi system which won't necessarily work. Based on your read:write ratio and register/data flow, draw a detailed map showing on which cycle each flip-flop in your ENTIRE system is enabled. It should look like Figure 52 but may include other registers, etc... It should also work.

FIGURE 52 Survivor memory timing showing the counter used to generate WrRd_N



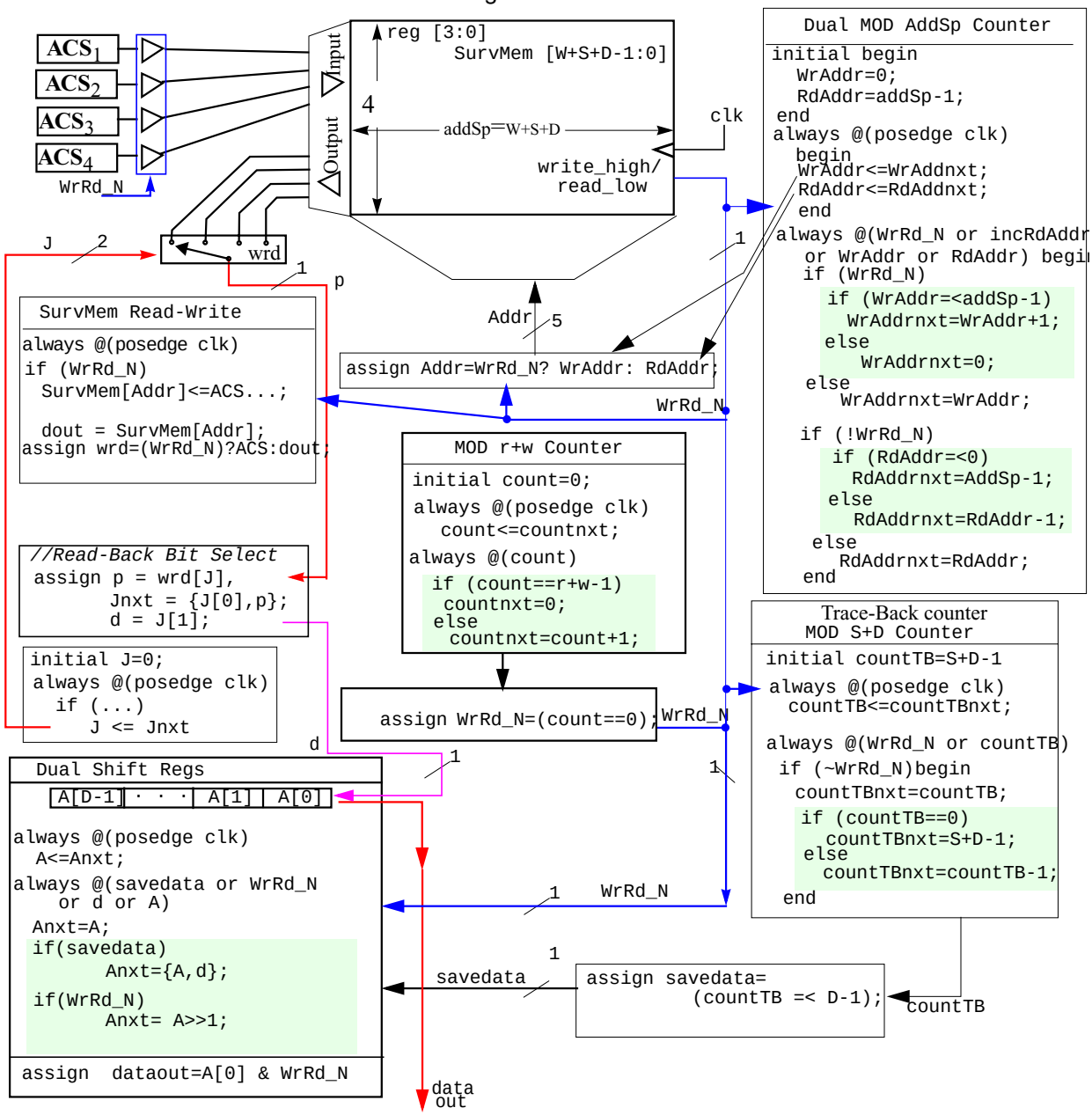
c) Write, or add to, your test bench, so you can simulate the code for parts 2 and 3 below.

2. and 3. Laboratory Assignment: Code the survivor memory circuitry.

Figure 54 shows a more detailed view of possible controlling circuitry for the survivor memory module. Divide the coding into two parts by coloring the picture to define the work for questions 2 and 3.

Remember the boxes contain pseudo-code, not code, `initial` cannot be synthesized, and "begin-ends" are often omitted. There may also be not-so-clever mistakes in the code below just to make your life a lot harder.

FIGURE 53 Details of the read-back circuit timing[1]



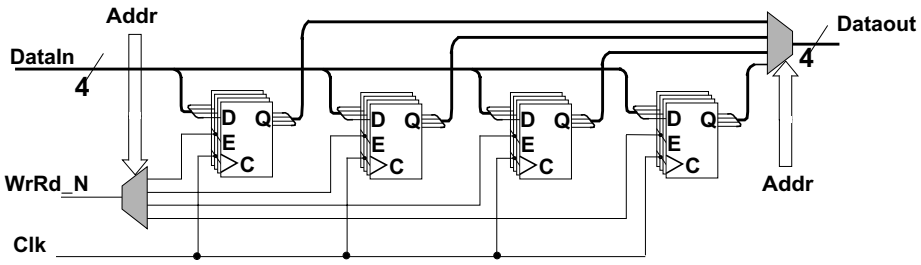
8.0.1 Synchronous Memory

Most of the memory designs for imbedding in logic chips are now synchronous. This means the data is controlled by the clock. In reality, the memories are asynchronous and have a synchronous interface to make it easy for the Synopsys user to interface to them.

We suggest you use the synchronous memory model shown below in Figure 54.

1. A MOD 5 counter (or modulo 5 counter) is one that counts 0,1,2,3,4,0,1,2,3,4,0,1,2, . . .

FIGURE 54 Synchronous memory module.

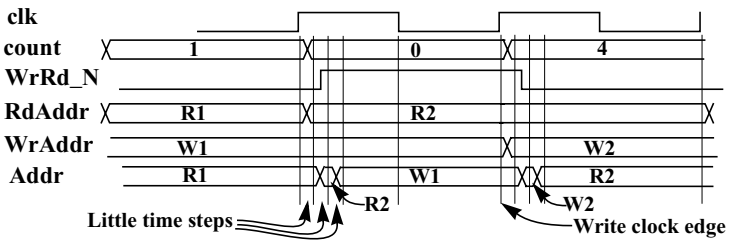


- Real memory uses latches, not flip-flops. However for thinking about how it works, the flip-flops should be a good model.
- One cannot reset the flip-flops. Memory is hard to initialize, fortunately it is not necessary for the circuit. To initialize it in simulation see the last page in:
  http://www.doe.carleton.ca/~jknight/97.478/97.478_03F/PeterVrlR.pdf
- This is a two-port memory with one read and one write port.
- The model shows the same address to both ports. One could use separate addresses.
- The model has one enable *WrRd_N* used only for the write port. One could have separate *RdEn* and *WrEn.*

8.0.2 Addresses In Synchronous Memory

For synchronous memory, the address near the end of the clock cycle determines where the data is stored on the clock edge at the end of the cycle. It is insensitive to races. Asynchronous memory is smaller and faster but it is susceptible to races.

FIGURE 55 Memory timing



In synchronous memory the data would only be captured at the Write clock edge.
The delays in the signals after the clock edge is of little concern.
With latches,controlled by WrRd_N, the data might be latched into the wrong adress>
If you care, the glitch is described in more detail in the 2002 writeup for labs 3-4.

8.1 The output shift register

If you picked a circuit with (r/w < 5/1), see Section 14.0 on page 46; good luck.

The final output, read back from the survivor memory, was read back in reverse order over the D area.Its output order must be reversed again before it is sent on.

One would like the incoming data rate to be the same as the outgoing data rate. To do this it is convenient to send out the data bit in the same cycle that the input data bits are read in. Every WrRd_N cycle is convenient.

For D ≤ r, one can do this with a bidirectional shift register. But if a WrRd_N pulse comes inside the D area, this shift register will put some bits out of place. If you picked such a circuit, see Section 14.0 on page 46.

Note word “a”. This is the final data bit read at the end of the “D” read cycles. It is immediately overwritten with new data in the next clock cycle.

FIGURE 56 Read-write Interleaving as it affects the output shift register.

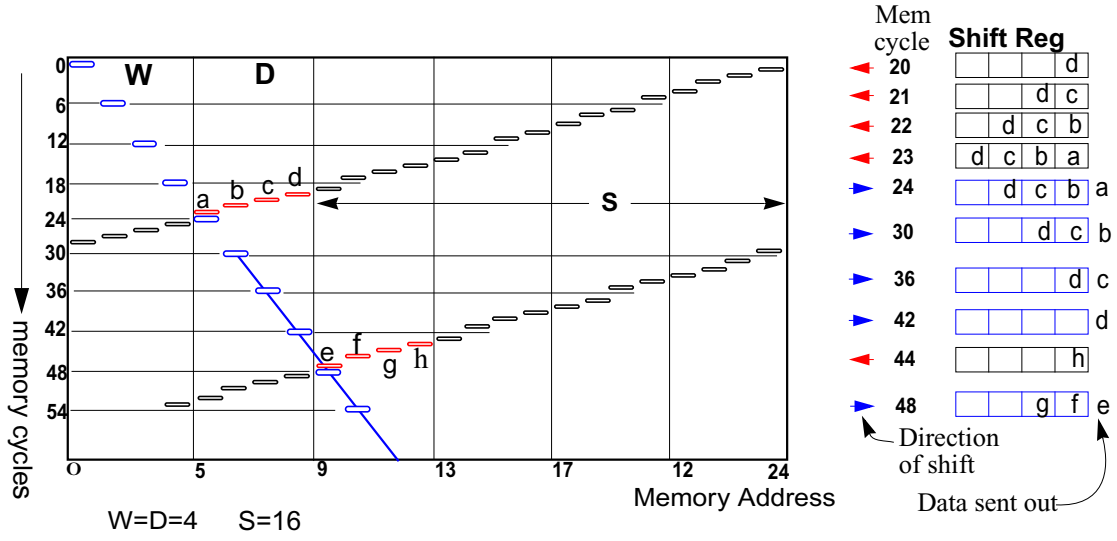
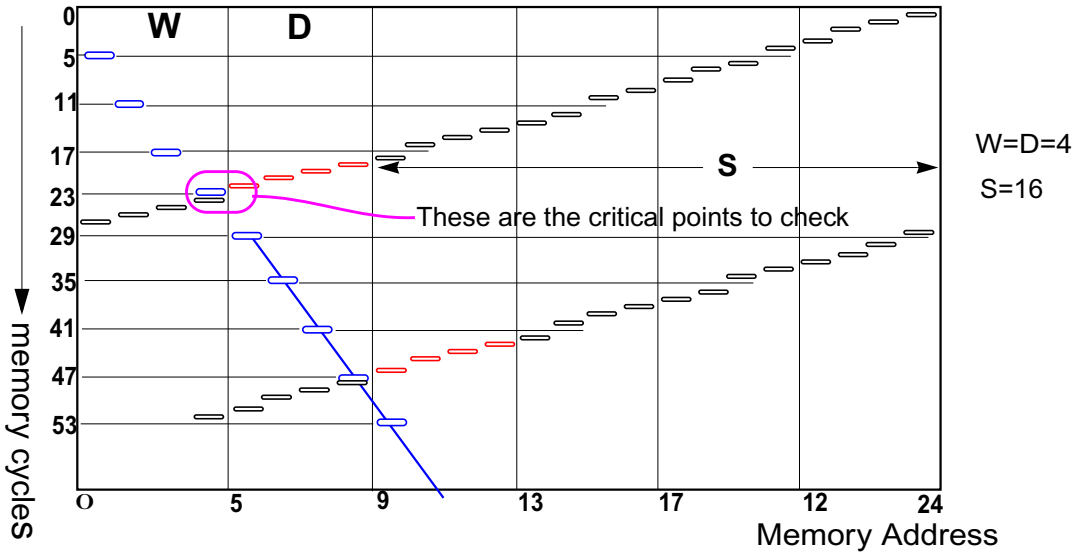


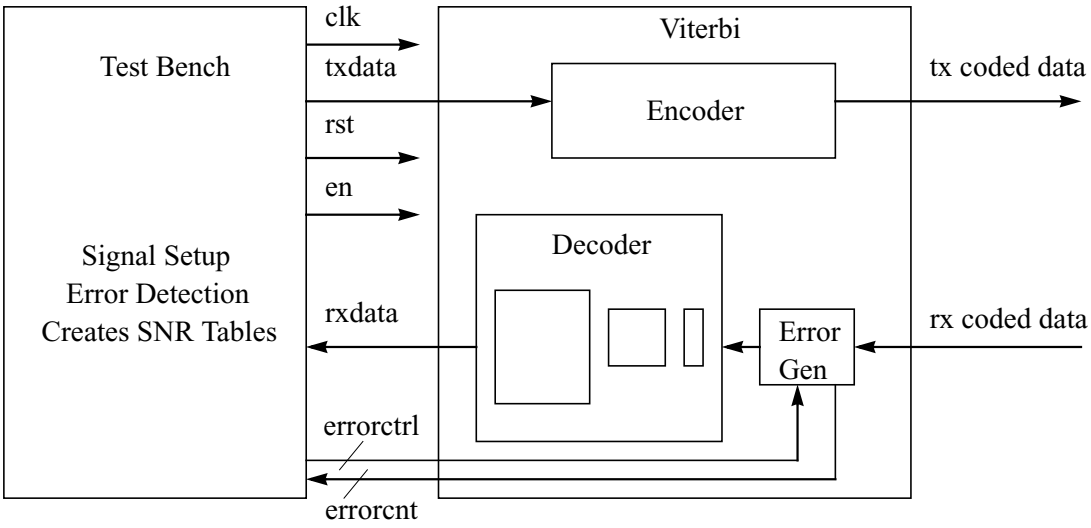
FIGURE 57 Fine detail of read-write Interleaving for an implementation which starts with a read.



9.0 The Complete Encoder-Decoder.

With the exception of the error generator, your system should be complete and an alternative block diagram is shown below.

FIGURE 58 Summary of the Design



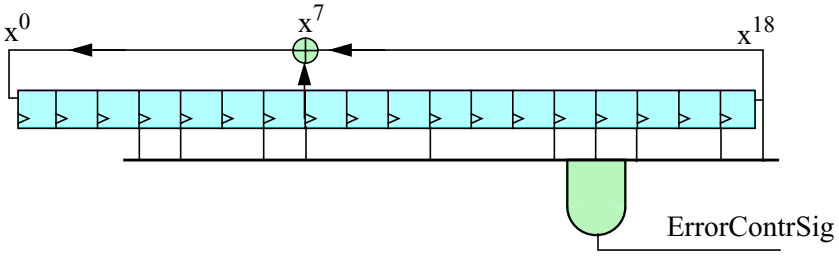
Note that the input txdata and output rxdata only advance once every r+w cycles. Your testbench will need to take this into account when sending data in and comparing the results.

Also, to perform the appropriate tests, you will need to connect the tx coded data to rx coded data for a loopback test.

The errorctrl signal is used to tell the error generator approximately how often to flip bits (eg 1/256), and the errorgenerator should provide the testbench with a count of the number of channel bits it has actually errored.

9.1 The Error Generator

FIGURE 59 A pseudorandom error generator of length 18. Its cycle length is 2^{18}-1.



These circuits made from shift registers with XORs attached at appropriate places, can generate bit streams which pass many, but not all tests for randomness:
First, they give a deterministic sequence which will repeat after 2^n-1 cycles where n is the number of flip flops.

Second, they are a shift register, so if one looks at the pattern within the register, one number is related to the next by a right shift, slightly messed up by the XORs.

Suppose one wants an error every 1024 bits. If one takes 10 random bits, They are all 1s only 1 clock cycle out of 1024. This means that the AND of any 10 bits should generate the desired error control signal.

Since the generator is periodic, if one makes the generator only 10 bits long, one would get error control signals exactly 1023 cycles apart. This is not random, thus one must make the

generator much longer, maybe 20 bits. Any ten bits of this longer register will all be 1s one-out-of-1024 times on average. However this errror pattern will only repeat after 2^{20}-1 clock cycles.

Since the generator is a shift register, one trys to distribute the inputs for the AND gate at random positions along the register as shown. Clumping them at one end may give some unwanted correlations between the errors.

It does little harm to make the shift register shift faster than the encoder puts out new data, provided the period of the shift register, 2^n-1, is not divisable by r+w. Why?

10.0 Laboratory 5:

1. Design and code the error generation unit. Ensure that when it introduces an error into the channel it does NOT always flip both z1 and z0. Create a counter on the error generator so that you can keep track of the number of introduced errors. Note that an introduced error should only 'count' if it got caught in your serial-to-parallel and was used in the ACS calculations.
2. Hook up your entire chip and, using a self-checking testbench, get it to work,without introduced errors, for an input data stream of:
   a) All 0's
   b) All 1's
   c) 00000111111....11111
   d) 00000000 11111111 00000000 11111111 00000000...
   e) 01010101010101010101010101010...
   f) Random input data
3. Synthesise your circuit in an attempt to beat your peers in terms of area and throughput. Ensure you have scripted the process so that it can be easily repeated. Hand in your scripts and the results in terms of area, power and clock rate.
4. If you haven't already done so, fix your ACS units to use the subtraction technique to prevent overflow problems. Repeat question 3 to make sure its still working.

11.0 Lab 5+

5. With the error generator in place and flipping, an average of 1/64 bits, repeat step 2f. If you can't get it working, start with 2a and work your way down. It is much easier to debug simple data streams.
6. Write a testbench which can automatically test the output error rate for long bit sequences and at different channel error rates. Create a plot of output error rate vs 'counted' input error rate for sequences of at least 100k bits long. This plot is the ultimate test of whether your circuit works.
7. Simulate the gate level code for a point on the curve from question 4.

12.0 The Project

The laboratories from now on will be to:

- Finish the simulation

- Synthesize each module.
- Synthesise the complete circuit
- Simulate the synthesised circuit
  Write the final report.

For Lab 5+, and after there will be no more intermediate reports. Just the final report.

12.1 Extra Details on Bit Error Rate Simulations (not Required)

The final performance test of your design should be a bit error-rate (BER) vs signal-to-noise ratio (SNR) graph.
This graph is discussed in digital communication courses. A typical graph can be found in
http://pw1.netcom.com/~chip.f/viterbi/simrslts.html
This graph will be the same as yours except the x-axis uses a signal with added Gausian white noise (AGWN). This is Eb/No.
In the SNR can be approximated as
(number of error bit received)/(total number of bits transmitted)
Recall that a rate 2 encoder transmits two bits for every data bit.
The bit error rate is
(number of uncorrected errors)/(total number of data bits)

Most of this work will be done in your test bench.

12.1.1 Shannon's Law

Shannon

C is the channel capacity (data rate). If you are transmitting at a data rate R <C, it is possible to have error-free transmission if you can find the right code. If R>C, you will always have errors. Shannon related the maximum rate C with the noise and bandwidth of the channel.

R = User data rate in message bits/sec. If you encode and send several code bits/(data bit), as in Viterbi, R does not change, but the bandwidth needed will.

C = Channel capacity = Maximum error-free data rate. (User data bits)/sec
B = Bandwidth used by the channel, specifically that processed by the receiver.

N_d = Noise spectral density (Watts/Hz)
S = Signal power in Watts

Then Shannon proved that this limiting capacity for error-free transmission was

$$C = B*\log_2(1+ S/N_dB) \text{ baud}$$

Let E_b= Energy in Joules (Watt sec) per data bit.

$$S= E_bR$$

If E_b is per bit in the original users message, then R is the number of user data bits/ sec. However if R is expanded 2 for 1 as in a rate 1/2 Viterbi code, the energy per bit, E_b, will halve and S remains unchanged.

The data rate R must be less than C, that is

$$R< B*\log_2(1+ S/N_dB)$$

or

$$E_b/N_d > (B/R)(2^{1/(B/R)} -1)$$

The inverse ratio $(B/R)^{-1}$ is called the spectral efficiency of the sytem. How much bandwidth does one need to send at rate R. It is a function of the modulation and coding scheme, but typically

$1/2 < R/B < 1$

Nyquist

DataRatemax=2*(bandwidth)*$\log_2$(numberof signal levels) baud

Spectral efficiency R/B is governed by this.

R/B< $\log_2$(numberof signal levels/T) baud/Hz

QPSK =0.96 bits/Hz. Compare 2*$\log_2$(4)= 4

Shannon says bandwidth=(1/2)*(numberof channel sig/sec)*(number of dimensions of sig space)

Example: BPSK: R=n bits/sec; B=(1/2)*n*1 R/B=1/2

Digital modulation http://www.ee.mtu.edu/faculty/ztian/ee5530/lecture_4note.pdf

– The process of mapping digital information into analog waveforms

– The mapping is performed by taking block of $K = \log_2 M$ bits and selecting one of the 2^K waveforms

• Vector representation of waveforms

– Signal space: any M waveforms can be represented in a N-dim signal space spanned by a complex basic function set $\{ .\psi_1(t),\ldots, .\psi_N(t)\}$, $N <= M$

– The orthonormal basic function set is not unique, can be constructed from the Gram-Schmidt procedure

– Signal operations: energy, correlation, mutually orthogonal

Bit Rates and Symbol Rates (3)

§§ If more bits can be sent with each symbol, then the same amount of data can be sent in a narrower spectrum

§ Modulation formats that are more complex and use a higher number of states can send the same information over a narrower band of the RF spectrum

§ **The needed signal bandwidth for the communications channel depends on** the symbol rate, not on the bit rate

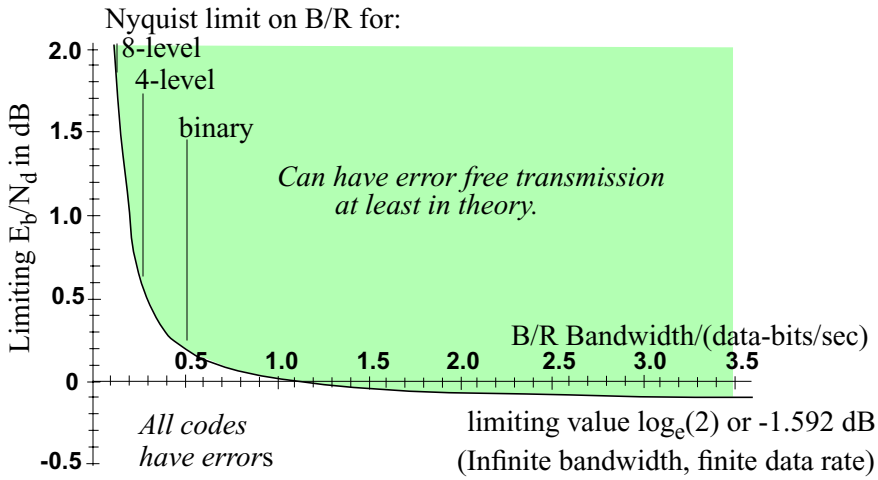
Example: 80kbit/sec data signal

§ If one bit is transmitted per symbol, as with BPSK, then the symbol rate would be the same as the bit rate ⇒ 80kbits per second

§ If two bits are transmitted per symbol, as in QPSK, then the symbol rate would be half of the bit rate ⇒ 40kbits per second

The limiting signal-to-noise ratio Eb/Nd(min) tells how much signal power is needed to get error free communication with various B/R (modulation and coding schemes).

Independent of the number of levels.



Notes:

As you put more levels in your code, the noise is more likely to give a wrong level. You need a higher E_b/N_d.

Coding gain.

Sending n code bits for k data bits increases the bandwidth, moves us right on the graph and gives a better channel.

Increasing the power also helps but only as the log.

When no coding is used $R=1/T_b$, T_b is the period of 1 bit.

Nyquist says $B>R/2 =1/(2T_b)$, call this B_b

When coding is used, k data bits are expanded to n total bits.

R is the rate for actual data. It is unchanged, but the coding reduces the time perbit by k/n.

The bandwidth is increased in proportion.

$B = (n/k) * B_b = (n/k)R/2$

Recall Shannon says for error free transmission,

$R< B*\log_2(1+ S/N_dB)$

For binary signals one gets $B_b = R/2$ if we can do Nyquist sampling

With the increased B, we get

$R< (n/k)(B_b)*\log_2(1+ S/N_dB_b(n/k))$

We increased the channel capacity by n/k, and reduced it by only $\log_2(1+ (k/n)*$Sig-to-noise ratio))

Thus coding clearly helps.

Using Shannon, people have found that as n gets large (k/n can stay the same) there is a code that is as close as one wants to the Shannon limit. Unfortunately the work to decode it is prohibitive.

With infinite bandwidth and finite rate B/R=0

Eb/N

Factors Influencing the Choice of Digital Modulation

Modulator Choice

Eb/Nb = (Signal Energy/bit)/(Noise power/Hz)

Bandwidth Efficiency = ability of the modulation scheme to accomodate data with limited bandwidth

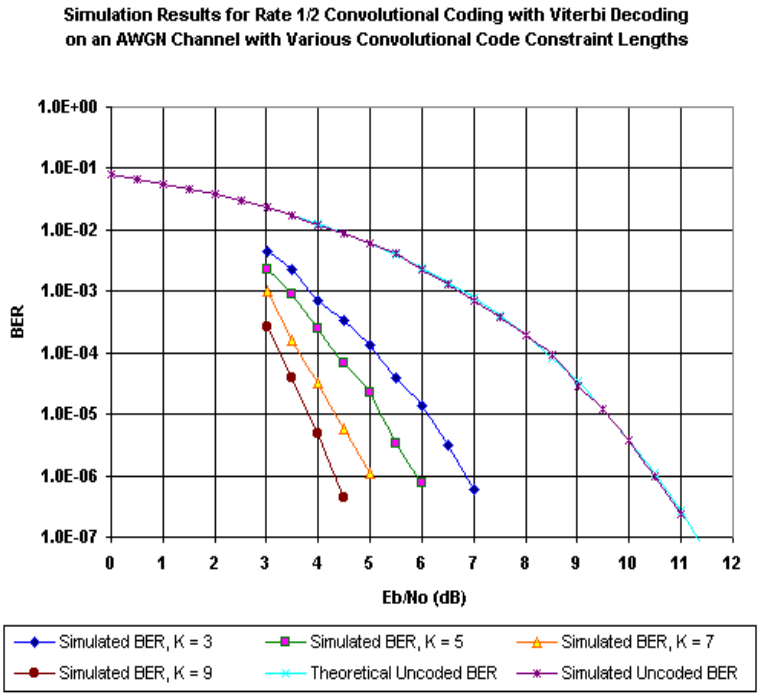
$$\eta_b = R/B \quad \text{bits/sec/Hz}$$

Shannons Theorem tells us that

$\eta_b = C/B$ =(channel capacity in bits/sec)/(bandwidth in Hz)

$= \log_2(1+S/N)$

FIGURE 60. Simulation Results for Convolutional Coding with Measured with a AWGN Channel with Various Convolutional Code Lengths



Key performance criteria of digital modulators :

i. Bandwidth efficiency,

.. A measure of the amount of data that can be accommodated in a limited bandwidth.

.. For all modulators,

$\eta_B = (R/BW)$ bits/sec/Hz

where, R = data rate in b/s

$$\eta_B < \eta_{Bmax} = (C/BW) = (\log_2 (1+S/N)$$

where,

η_{Bmax} ..max. bandwidth efficiency in an AWGN channel as determined by Shannon bound.

S/N .. signal to noise ratio of the channel.

C .. channel capacity in b/s.

ii. Power efficiency,

.. How quickly P(Σ)(prob. of error) improves (decreases) with increasing signal power.

.. The required amount of signal-to-noise ratio required to reach a particular P(Σ) threshold.

.. Typically, modulators trade off between spectral efficiency & power efficiency e.g BPSK vs. QPSK.

Power Spectral Density (PSD)

for transmitted signal,

$s(t)=Re\{g(t)exp(j\pi fct)\}$

$PSD=P_s(t)=(1/4)[P_g(f-f_c)+P_g(-f-f_c)]$

where

$P_g(f)=|G(f)|^2$

Bandwidth, BW

null-to-null Bandwidth, BW

.. width of main spectral lobe in PSD.

.. width from first null in PSD (to right of center freq.) to first null in PSD (to left of center frequency).

Absolute BW

.. range of frequency over which signal has non-zero PSD.

.. often

Half power BW

.. width from first point where PSD has dropped 3dB (to the right of center) to first point where PSD has dropped 3 dB (to left of center).

Book's BW

.. null-to-null BW.

FCC Bandwidth

Where power is 0.5% of peak

13.0 Appendix I
Overflow Correction

13.0.1 Overflow Correction By Subtracting The Smallest H From all the Others

Even though the path metrics, H_i, can grow as long as the message continues. the storage space required for H_i can be limited.

Suppose at each step one took

$H_{min} = \min(H_1, H_2, H_3, H_3)$

If one were to subtract H_{min} from all four H_i , the comparisons would still be correct. Thus one should only need storage for words the size of the maximum difference, that is for

$\max(Hi-Hj)$ i=1 to 4, j=1 to 4

This maximum difference can be calculated theoretically. For this decoder it is 24, or 5 bit words. It also requires finding the minimum and doing the subtractions, which is usually considered to overide any register length saving.

In this decoder, the maximum path metric difference is 24, and one would need a 5 bit word.

13.0.2 Overflow Correction Using Two's Complement

There is a method to avoid finding H_{min} and doing the subtractions if one extends the word length by one bit. That is make the H registers such that

storage word size = $1+\text{ceiling}(\log_2(\max(Hi-Hj)))$ i=1 to 4, j=1 to 4

This method makes use of the detailed properties of overflow when using two's complement subtraction.

Summary of the method

Consider FIGURE 61 . This represents the contents of a 4-bit register holding some H_i. Since one always adds to the H_is, one always travels clockwise around the circle as time increases. If one compares two H values, the most clockwise one is the one we want to consider as larger. This is true even when the counter wraps around, and the wrapped-around H is numerically smaller than one that has not wrapped around. .

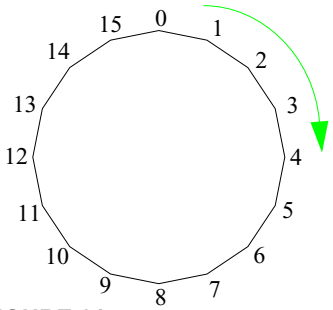


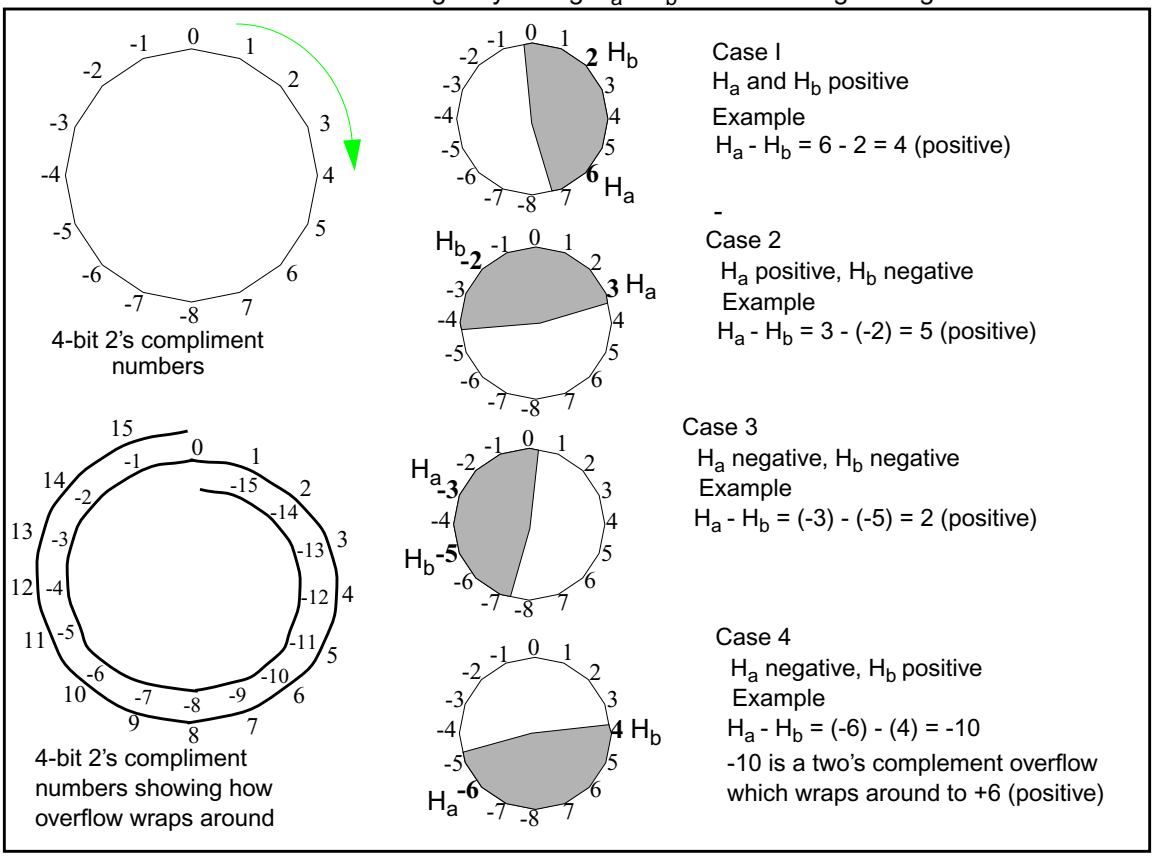
FIGURE 61
Possible values in a 4-bit register holding a metric H

Now one asks, "How can one tell that the wrapped around number is supposedly larger, when a numerical comparison makes it smaller?"

To do this we need to do three things:

- Make sure that the two numbers being compared are always less than a diameter apart.
- Interpret the numbers over 7 as two's compliment numbers as shown in FIGURE 62
- Remember that if one adds two positive numbers with a sum over 7, it overflows into negative number territory. If two negative numbers are added to give a sum less than -8, it overflows into positive number territory. Thus (-6)+(-3) = -9 will appear to be +7.

FIGURE 62 How to tell which H is larger by taking H_a - H_b and checking the sign.



If one subtracts any number on the circle from one clockwise from it, using two's complement subtraction, the result will be a positive number provided the distance between the numbers was less than half the circle circumference.

FIGURE 62 shows that if H_a is clockwise from H_b, but less than 180° away, then H_a - H_b will always be positive. This should be obvious for cases 1, 2 and 3.

For case 4, the subtraction always overflows, and thus the result will still be positive.

Summary

Subtracting H_a - H_b will give a positive number if H_a is clockwise from H_b, as long as it is less than 180 degrees away. The comparison must be done by subtracting and checking the most significant bit. A standard comparison will not work.

Maximum H_a-H_b

Extensive simulations show that for k=3, rate 1/2 encoder, $|\mathbf{H_a - H_b}| \leq 5$.

14.0 Appendix II
Dual Output Shift Registers

For circuits with (r/w < 5/1)

14.1 The output shift register

The final output read back from the survivor memory must have its order reversed, since it is read backwards.

One would like the incoming data rate to be the same as the outgoing data rate. To do this it is convenient to send out the data bit in the same cycle that the input data bits are read in. Every WrRd_N cycle is convenient.

For some combinations of W, D and S, one can do this with a bidirectional shift register. But if a WrRd_N pulse comes inside the D area, a shift register will put some bits out of place.

This is illustrated in FIGURE 56 . The data bits are given letters, “a”, “b”, ... In memory cycle 19 bit “e” is pushed into the shift register, pushing bit “x” left. Bit “x” is still unknown if the decoding started at cycle 0.

Cycle 20 is a WrRd_N cycle and the shift register shifts right and sends out bit ”e”. This is wrong. Assume the “e” bit got put in properly at cycle 21, to help explain the next error.

In cycles 21 through 24 the bits “d”, “c”, “b”, and “a” are stored. In the WrRd_N cycles 25, 30, 35, and 40, the bits are shifted to the output in the correct order. However, before “e” gets shifted out in cycle 45, “k” gets shifted in in cycle “44”.

In cycle 45, “k” gets shifted out instead of “e”. This is wrong.

FIGURE 63 Read-write Interleaving as it affects the output shift register.

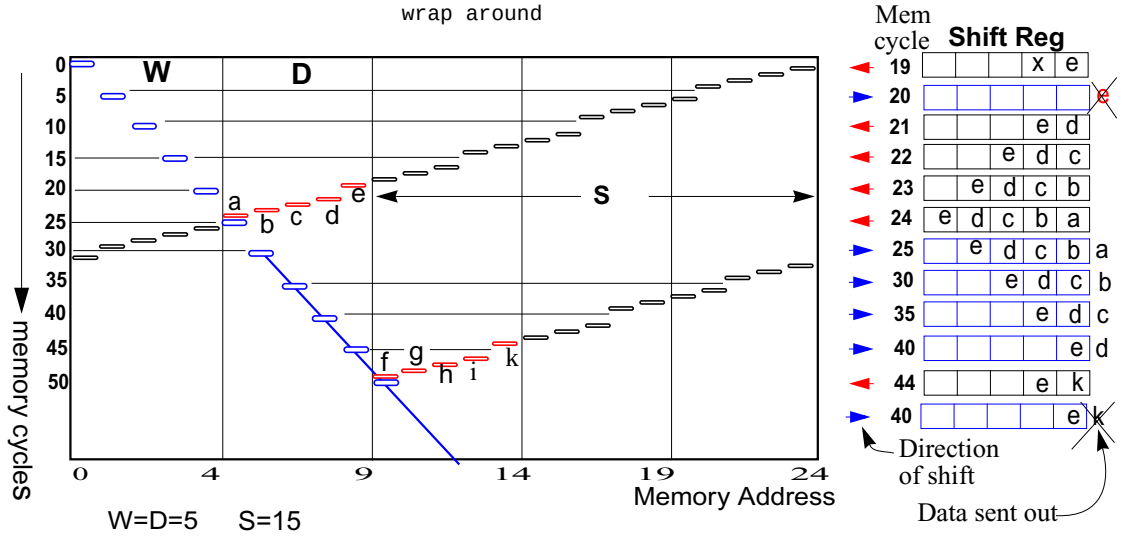
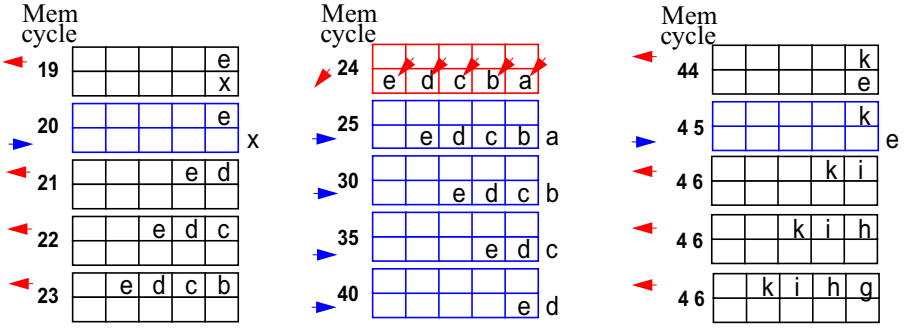


FIGURE 64 Dual shift-register implementation which avoids errors. .



A dual shift-register which avoids the problem is shown in FIGURE 64 . Data bits are always shifted in (left) in the top register and shifted out (right) in the bottom register.

During the last read in the D part of trace back (cycle 24) , the data is shifted left and into the lower register. This system avoids mixing the new and the old data.

14.1.1 Using a single bidirectional register.

If one chooses (r,w ≥ 5,1) in Table 1, page 26, then the D section is short enough so the writes need not come in the middle of the D read, and the single bidirectional register is enough.