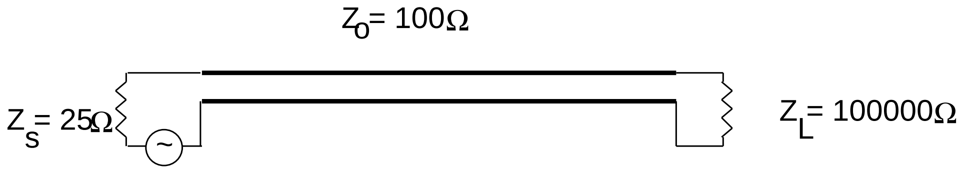
Solution Set 7

gmoy@eecs.berkeley.edu

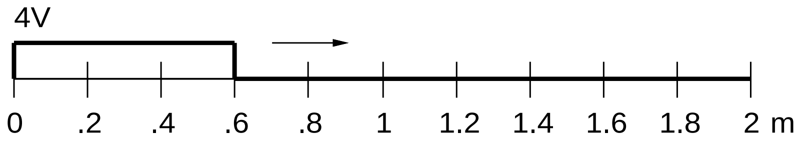
Problem 1

a) Length of wire = 2m, $c = 2 \times 10^8 m/s$. It takes 10ns to propagate from the transmitting side to the receiving side.

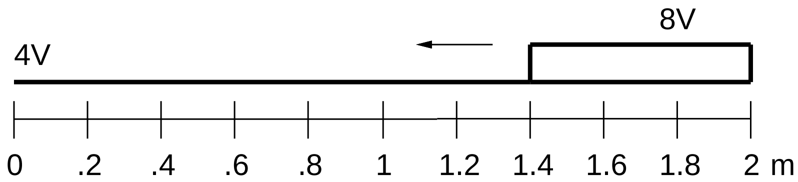


$\Gamma_B = \frac{Z_L - Z_O}{Z_L + Z_O} = \frac{10^5 - 100}{10^5 + 100} \approx 1$
$\Gamma_A = \frac{Z_S - Z_O}{Z_S + Z_O} = \frac{25 - 100}{25 + 100} = -0.6$

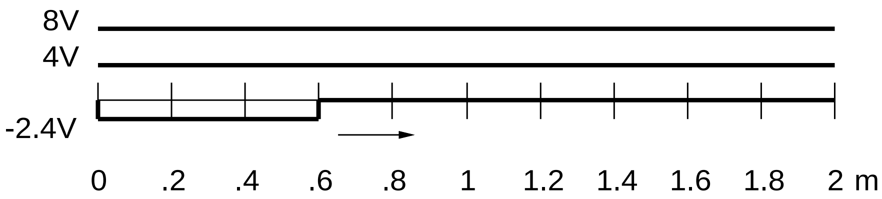
at t = 3ns



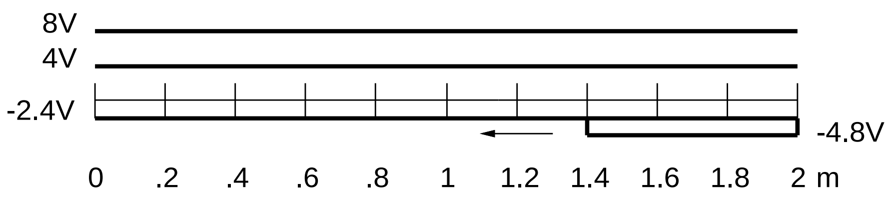
at t = 13ns



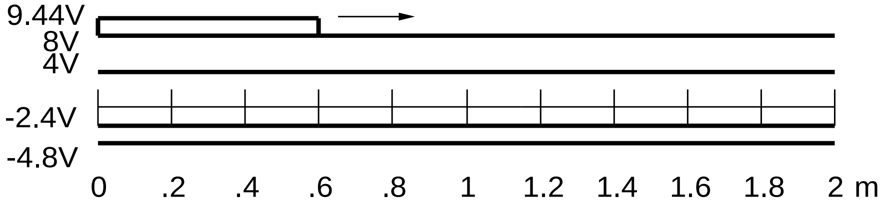
at t = 23ns



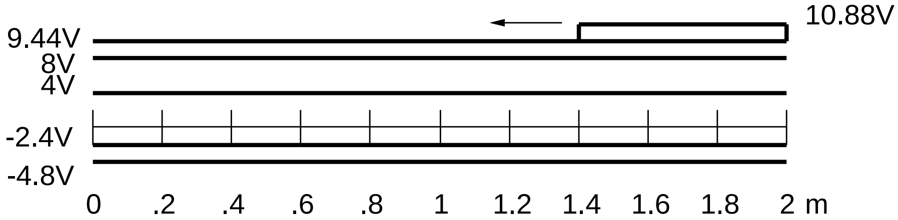
at t = 33ns



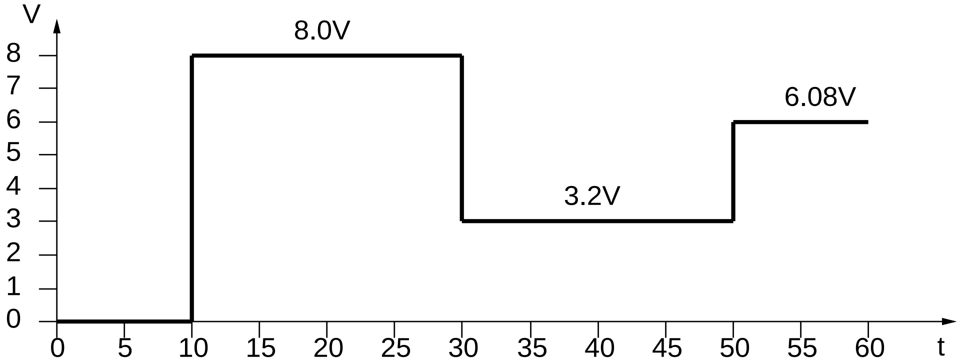
at t = 43ns



at t = 53ns



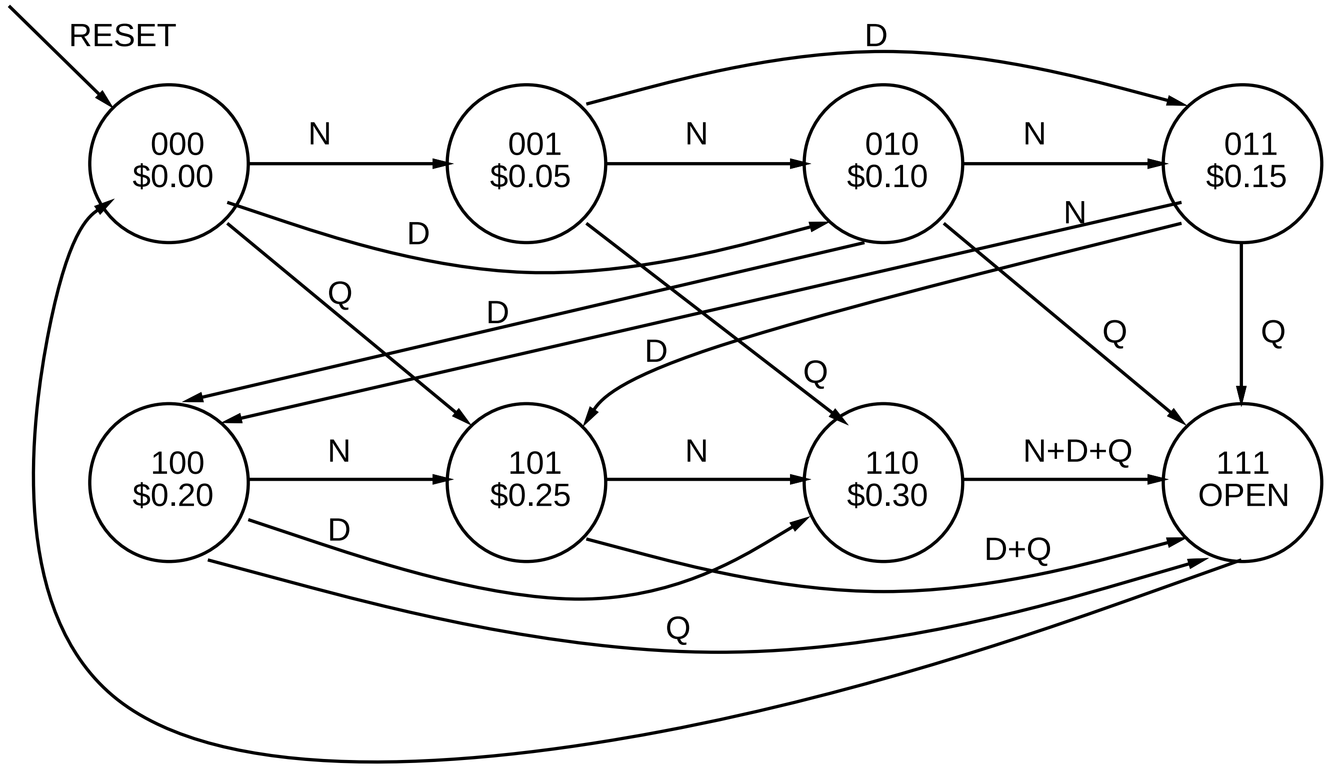
b)



c) No glitches, but there is bad overshoot of 8V which exceeds the absolute maximum ratings of $V_{CC} + 0.5V$. Note that the 1-0 transition would have a glitch. Drawings left to the reader as an exercise.

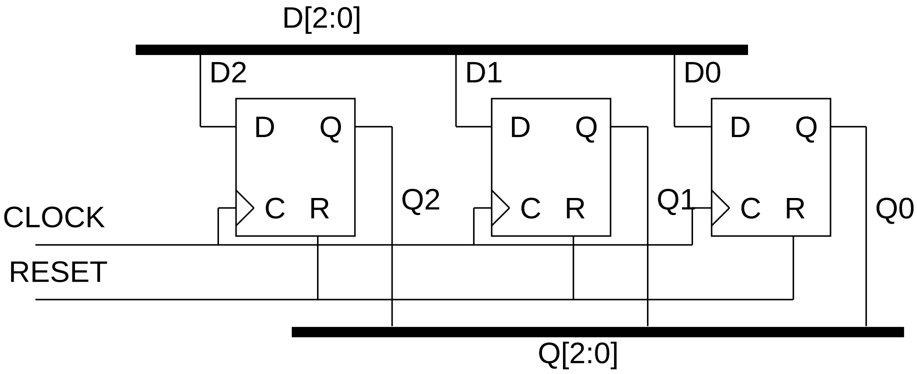
Problem 2

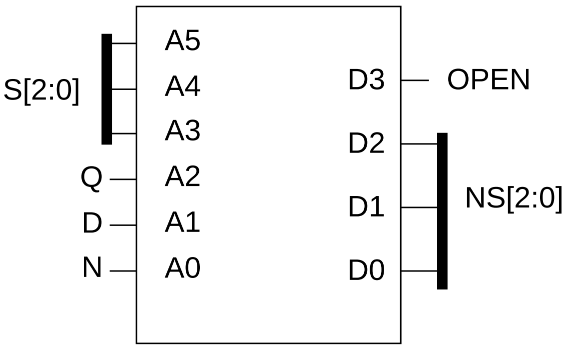
a) Inputs: NDQ; Output: OPEN; There is a global reset. We assume that there will not be any illegal inputs (more than one coin line being asserted high). These states must be accounted in the ROM though.



A_5	A_4	A_3	A_2	A_1	A_0	D_3	D_2	D_1	D_0	
State			Input			OPEN	Next State			ROM
Q_2	Q_1	Q_0	Q	D	N		Q_2	Q_1	Q_0	Contents
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	1	1
0	0	0	0	1	X	0	0	1	0	2
0	0	0	1	X	X	0	1	0	1	5
0	0	1	0	0	0	0	0	0	1	1
0	0	1	0	0	1	0	0	1	0	2
0	0	1	0	1	X	0	0	1	1	3
0	0	1	1	X	X	0	1	1	0	6
0	1	0	0	0	0	0	0	1	0	2
0	1	0	0	0	1	0	0	1	1	3
0	1	0	0	1	X	0	1	0	0	4
0	1	0	1	X	X	0	1	1	1	7
0	1	1	0	0	0	0	0	1	1	3
0	1	1	0	0	1	0	1	0	0	4
0	1	1	0	1	X	0	1	0	1	5
0	1	1	1	X	X	0	1	1	1	7

A_5	A_4	A_3	A_2	A_1	A_0	D_3	D_2	D_1	D_0	
State			Input			OPEN	Next State			ROM
Q_2	Q_1	Q_0	Q	D	N		Q_2	Q_1	Q_0	Contents
1	0	0	0	0	0	0	1	0	0	4
1	0	0	0	0	1	0	1	0	1	5
1	0	0	0	1	X	0	1	1	0	6
1	0	0	1	X	X	0	1	1	1	7
1	0	1	0	0	0	0	1	0	1	5
1	0	1	0	0	1	0	1	1	0	6
1	0	1	0	1	X	0	1	1	1	7
1	0	1	1	X	X	0	1	1	1	7
1	1	0	0	0	0	0	1	1	0	6
1	1	0	0	0	1	0	1	1	1	7
1	1	0	0	1	X	0	1	1	1	7
1	1	0	1	X	X	0	1	1	1	7
1	1	1	0	0	0	1	0	0	0	8
1	1	1	0	0	1	1	0	0	0	8
1	1	1	0	1	X	1	0	0	0	8
1	1	1	1	X	X	1	0	0	0	8



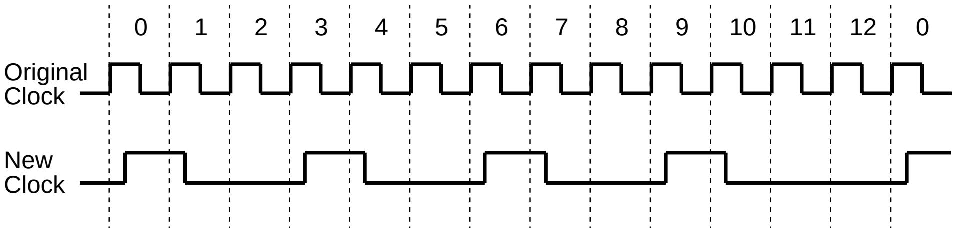


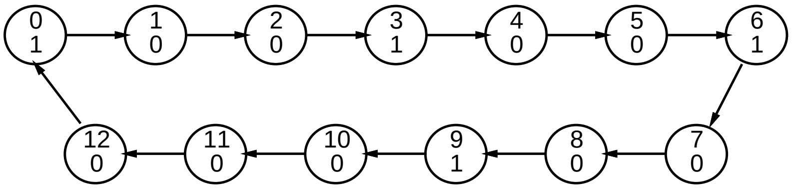
ROM CONTENTS:
{0,1,2,2,5,5,5,5,
1,2,3,3,6,6,6,6,
2,3,4,4,7,7,7,7,
3,4,5,5,7,7,7,7,
4,5,6,6,7,7,7,7,
5,6,7,7,7,7,7,7,
6,7,7,7,7,7,7,7,
8,8,8,8,8,8,8,8}

b) Need $64 \times 4 = 256$ bits. Each 32×1 is 1 CLB. Need 8 CLBs for 64×4 ROM + 2 CLB for FF
Fractional area $= 8/196 = 4\%$.

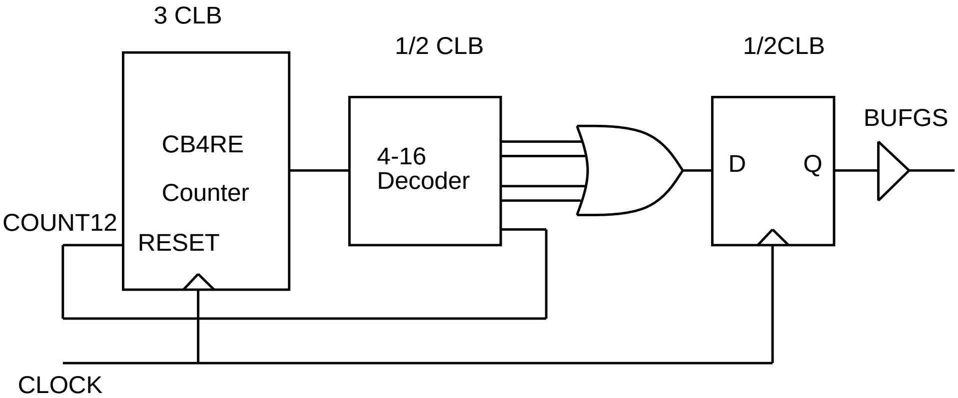
Problem 3

13 states and clock triggered.





Use a 4-bit counter with synchronous reset to count states. Have a separate bit for the output. CLB cost is approx. 4 CLBs.



Problem 4

a) I will calculate the local derivative on each side of the point of interest. With the derivatives for each frame, I can compare them to find a crude estimation of the movement. If there is no change, then the velocity will be zero. One of the problems with this algorithm is that it only considers the points next to the point of interest, so any movement greater than a pixel will not be accurately detected by this algorithm.

Note: This algorithm is VERY crude and bad... It is functional. You are welcome and encouraged to find better and/or simpler algorithms.

This algorithm approximately calculates $\frac{\partial}{\partial t}\left(\frac{\partial I}{\partial x}\right)$ on either side of the point of interest.

```
/* find the local space derivatives */
for i = 1 to NumImages
 for j = 1 to 8
  dx1[i,j] = Image[i,100,j*20] - Image[i,100,j*20-1]
  dx2[i,j] = Image[i,100,j*20+1] - Image[i,100,j*20]
 end
end

/* compare with the next frame's derivatives */
/* (the difference of space derivatives in time) */
for i = 1 to NumImages - 1
 for j = 1 to 8
  vel1[j] = dx1[i+1,j] - dx1[i,j]
  vel2[j] = dx2[i+1,j] - dx2[i,j]
  if ((vel1[j] == 0) & (vel2[j] == 0))
   vel[j] = 0
  else
   vel[j] = -(vel1[j] + vel2[j])
  endif
 end
 print(vel)
end
```

b) Code for `doit.c`

```
#include<stdio.h>

void main(int argc, char *argv[])
{
 FILE *fp;
 int NumImages, i, j, k;
 int width, height, temp;
 int *dx1, *dx2, vel1[8], vel2[8], vel[8];
 unsigned char *image;

 NumImages = argc - 1;
 image = (unsigned char *)calloc(NumImages*256*256,sizeof(unsigned char));
 dx1 = (int *)calloc(NumImages*8,sizeof(int));
 dx2 = (int *)calloc(NumImages*8,sizeof(int));

 /* read in all the images */
 for (i = 0; i < NumImages; i++)
 {
```

```
  fp = fopen(argv[i+1],"r");
  fscanf(fp, "%d", &height);
  fscanf(fp, "%d", &width);
  for (j = 0; j < height; j++)
   for (k = 0; k < width; k++)
   {
    fscanf(fp, "%d", &temp);
    image[i*256*256 + j*256 + k] = (unsigned char)temp;
   }
  fclose(fp);
 }

 /* calculate the local derivatives */
 j = 100;  /* look at the 100th row */
 for (i = 0; i < NumImages; i++)
  for (k = 0; k < 8; k++)
  {
   dx1[i*8 + k] = (int)image[i*256*256 + j*256 + (k+1)*20] -
                  (int)image[i*256*256 + j*256 + (k+1)*20 - 1];
   dx2[i*8 + k] = (int)image[i*256*256 + j*256 + (k+1)*20 + 1] -
                  (int)image[i*256*256 + j*256 + (k+1)*20];
  }

 /* calculate velocities */
 for (i = 0; i < NumImages - 1; i++)
 {
  fprintf(stderr, "Velocity vector between frame %d and %d: ", i+1, i);
  for (k = 0; k < 8; k++)
  {
   vel1[k] = dx1[(i+1)*8+k] - dx1[i*8+k];
   vel2[k] = dx2[(i+1)*8+k] - dx2[i*8+k];
   if ((vel1[k] == 0) && (vel2[k] == 0))
    vel[k] = 0;
   else
    vel[k] = -(vel1[k] + vel2[k]);
   fprintf(stderr, "%d ", vel[k]);
  }
  fprintf(stderr, "\n");
 }
}
```

This is what the program gives:

```
Commandline:
doit away1 away2 away3 away4 away5 away6 away7 away8 away9

Results:
Velocity vector between frame 1 and 0: -1 -1 -2 -14 1 -1 44 -1
Velocity vector between frame 2 and 1: -2 2 -41 0 -1 1 -1 54
Velocity vector between frame 3 and 2: 4 -4 46 -4 4 0 1 -46
Velocity vector between frame 4 and 3: -2 1 -3 3 -1 -1 1 -7
Velocity vector between frame 5 and 4: -2 -38 46 -20 50 0 0 0
Velocity vector between frame 6 and 5: -36 38 -6 -3 -28 -19 6 -16
```

```
Velocity vector between frame 7 and 6: 38 2 -4 16 -20 16 -6 15
Velocity vector between frame 8 and 7: 18 13 -36 -37 -3 2 -1 2

Commandline:
doit right1 right2 right3 right4 right5 right6 right7

Results:
Velocity vector between frame 1 and 0: -3 10 1 -14 9 -1 -2 53
Velocity vector between frame 2 and 1: 1 -11 1 -2 -10 8 2 1
Velocity vector between frame 3 and 2: 1 -38 -2 2 -3 -9 2 -3
Velocity vector between frame 4 and 3: -1 40 1 0 40 1 -3 39
Velocity vector between frame 5 and 4: -13 -1 0 -13 -37 1 -1 -38
Velocity vector between frame 6 and 5: 14 -32 -1 15 0 -1 5 0
```

c) This algorithm is fairly simple to implement. You will need memory to store the image, some more memory to store the velocities, an arithmetic logic unit to do the subtractions and additions, and registers to store the numbers to go into the ALU. All arithmetic should be done in 2's complement since there is a significant amount of subtraction. There is no floating point arithmetic in this algorithm, but if there were, it would be necessary to simplify it. Division and multiplication either takes lots of memory (for a lookup table), or lots of time (for a iterative adder/subtractor).

The memory in bits needed to process 2 images is as follows:

Image1	$256 \times 8bits$
Image2	$256 \times 8bits$
dx1	$16 \times 8bits$
dx2	$16 \times 8bits$
vel1	$8 \times 8bits$
vel2	$8 \times 8bits$
vel	$8 \times 8bits$
total	$568 \times 8bits$

The counters needed to process 2 images are as follows:

Number of images	1 bit
calculate dx1	3 bit

These same counters can be reused to calculate the rest of the information.

All arithmetic is 8 bits wide. You will need 2 registers as the input to the addition block, and another register as the output.