

James Cook University

Electrical and Computer Engineering

EE4305: Electronics and Machine Control

Solutions

Frequency Detector Assignment

C. J. Kikkert October 1998

Task

Design a circuit to have two inputs, A and B and at least two LED outputs. The circuit is to light one LED, or a sequence of LEDs if the frequency of input A is higher than that of input B and it is to light the other LED, or another sequence of LEDs if the frequency of input B is higher than the frequency of input A. The circuit is to operate reliably over a wide range of input frequencies. The design can be done using either synchronous or asynchronous logic techniques. The design should be your own work and publicly available designs, such as those published in hobby magazines will not be accepted. (These designs also do not satisfy all the requirements of this assignment).

This design is to be fitted into a Lattice ispLSI2032. The JCU-ECE EPLD development board will be used to verify your EPLD design. To facilitate testing, the input pins are to be I/O12 and I/O13 and the output pins are to include the LED outputs I/O17 and I/O18 and may include all the 8 LEDs connected to I/O 16-23 if needed.

Asynchronous Sequential Logic

The logic signal flow graph for one of the possible solutions to the required function is shown in Figure 1. It should be noted that there are many possible ways of achieving a similar output function.

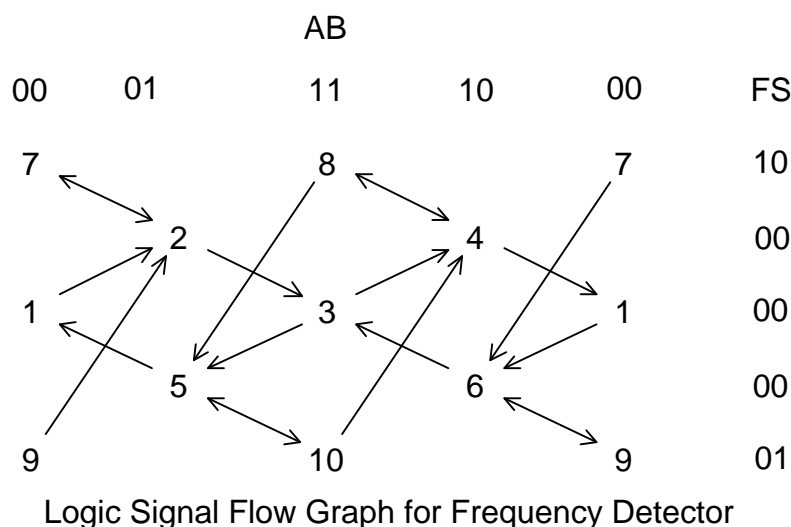


Figure 1

If the inputs A and B are at the same frequency and produce the sequence 00, 01, 11, 10, 00, etc., the circuit will go through states 1, 2, 3, 4, 1 etc., and the outputs F (Fast) and S (Slow) will both be zero. If

the inputs A and B are at the same frequency and produce the sequence 00, 10, 11, 01, 00, etc., the circuit will go through states 1, 6, 3, 5, 1 etc., and the outputs F (Fast) and S (Slow) will again both be zero.

If B is of a higher frequency than A, the circuit will go to states 7 and 2 if A = 0 and to states 8 and 4 if A = 1. Similarly if B is a lower frequency than A, the circuit will end up in states 10 and 5 or 9 and 6. The states 7 and 8 are thus an indication if the frequency of B is high and states 9 and 10 are an indication if the frequency of B is low.

The primitive flow table is shown in Figure 2. In this table the stable states are shown by *bold italic* numbers.

| AB | | | | FS |
|-----------------|-----------------|------------------|-----------------|----|
| 00 | 01 | 11 | 10 | |
| <i>1</i> | 2 | | 6 | 00 |
| 7 | <i>2</i> | 3 | | 00 |
| | 5 | <i>3</i> | 4 | 00 |
| 1 | | 8 | <i>4</i> | 00 |
| 1 | <i>5</i> | 10 | | 00 |
| 9 | | 3 | <i>6</i> | 00 |
| 7 | 2 | | 6 | 10 |
| | 5 | <i>8</i> | 4 | 10 |
| <i>9</i> | 2 | | 6 | 01 |
| | 5 | <i>10</i> | 4 | 01 |

Figure 2. Primitive Flow Table.

This primitive flow table can be merged, to result in a reduced number of rows, as is shown in Figure 2.

| AB | | | | FS |
|-----------------|-----------------|------------------|-----------------|-------|
| 00 | 01 | 11 | 10 | |
| <i>1</i> | 2 | | 6 | 00 |
| <i>7</i> | <i>2</i> | 3 | 6 | 10,00 |
| | 5 | <i>3</i> | 4 | 00 |
| 1 | 5 | <i>8</i> | <i>4</i> | 10,00 |
| 1 | <i>5</i> | <i>10</i> | 4 | 00,01 |
| <i>9</i> | 2 | 3 | <i>6</i> | 01,00 |

Figure 3. Merged Flow Table.

The merged flow table has 6 rows, requiring 3 feedback variables being: F, G and H. No other way of merging is possible.

This merged flow table requires a 3 feedback variable plus 2 input variable Transition Matrix. The rows of this matrix are arranged to ensure that there are no critical races. The resulting Transition Matrix is shown in Figure 4. The matrix has been arranged to keep the length of transitions to a minimum.

Since the output coding requires an output F during state 7 and 8, some saving in hardware could be achieved by swapping rows fgh = 100 and row 111. For a TTL realisation, this hardware saving is significant but for an EPLD realisation with 5 input variables to any gates, there is no real advantage.

| | | AB | | | | | | AB | | | |
|-----|-----|----|----|----|----|-----|-----|----|----|----|----|
| | | 00 | 01 | 11 | 10 | | | 00 | 01 | 11 | 10 |
| fgh | 000 | ⑨ | 2 | 3 | ⑥ | fgh | 100 | 1 | 5 | ⑧ | ④ |
| | 001 | ① | 2 | 3 | 6 | | 101 | 1 | 5 | ③ | 4 |
| | 011 | ⑦ | ② | 3 | 6 | | 111 | 1 | ⑤ | ⑩ | 4 |
| | 010 | | | | | | 110 | | | | |

Figure 4. Transition Matrix

This leads to the Excitation Matrix shown in Figure 5. In this table stable states have been indicated by ***bold italic*** numbers and don't cares have been indicated by X.

| FGH | | AB | | | |
|-----|-----|-------------------|-------------------|-----|-------------------|
| | | 00 | 01 | 11 | 10 |
| fgh | 000 | <i>000</i> | 011 | 001 | <i>000</i> |
| | 001 | <i>001</i> | 011 | 101 | 000 |
| | 011 | <i>011</i> | <i>011</i> | 001 | 000 |
| | 010 | XXX | XXX | XXX | XXX |

| FGH | | AB | | | |
|-----|-----|-----|-------------------|-------------------|-------------------|
| | | 00 | 01 | 11 | 10 |
| fgh | 100 | 000 | 111 | <i>100</i> | <i>100</i> |
| | 101 | 000 | 111 | <i>101</i> | 100 |
| | 111 | 000 | <i>111</i> | <i>111</i> | 100 |
| | 110 | XXX | XXX | XXX | XXX |

Figure 5. Excitation Matrix

It should be noted that multiple transitions are required for state 3 with AB = 11 and f = 0, to prevent the circuit going to stable states 8 and 10. After completion of the Karnaugh maps we need to check that rows fgh = 010 and 110 do not have all stable states in either of them, as the circuit may be locked into that row during start-up. This will result in the Karnaugh maps shown in Figure 5. In these maps, the don't care states are again shown as X. Having a 5 variable Excitation Matrix, results in 3 five variable Karnaugh maps.

The minimisation from these maps is:

$$\begin{aligned}
 F &= A B g' h + (A + B + g + h') \cdot f \\
 G &= A' B + B f g + A' f' g \\
 H &= B f' + A' f + A' f' h + B f h
 \end{aligned}$$

The rows for fgh = 010 and 110 containing don't care states, both result in some non-stable states to be contained on each row, so that the circuit will start properly on power-up.

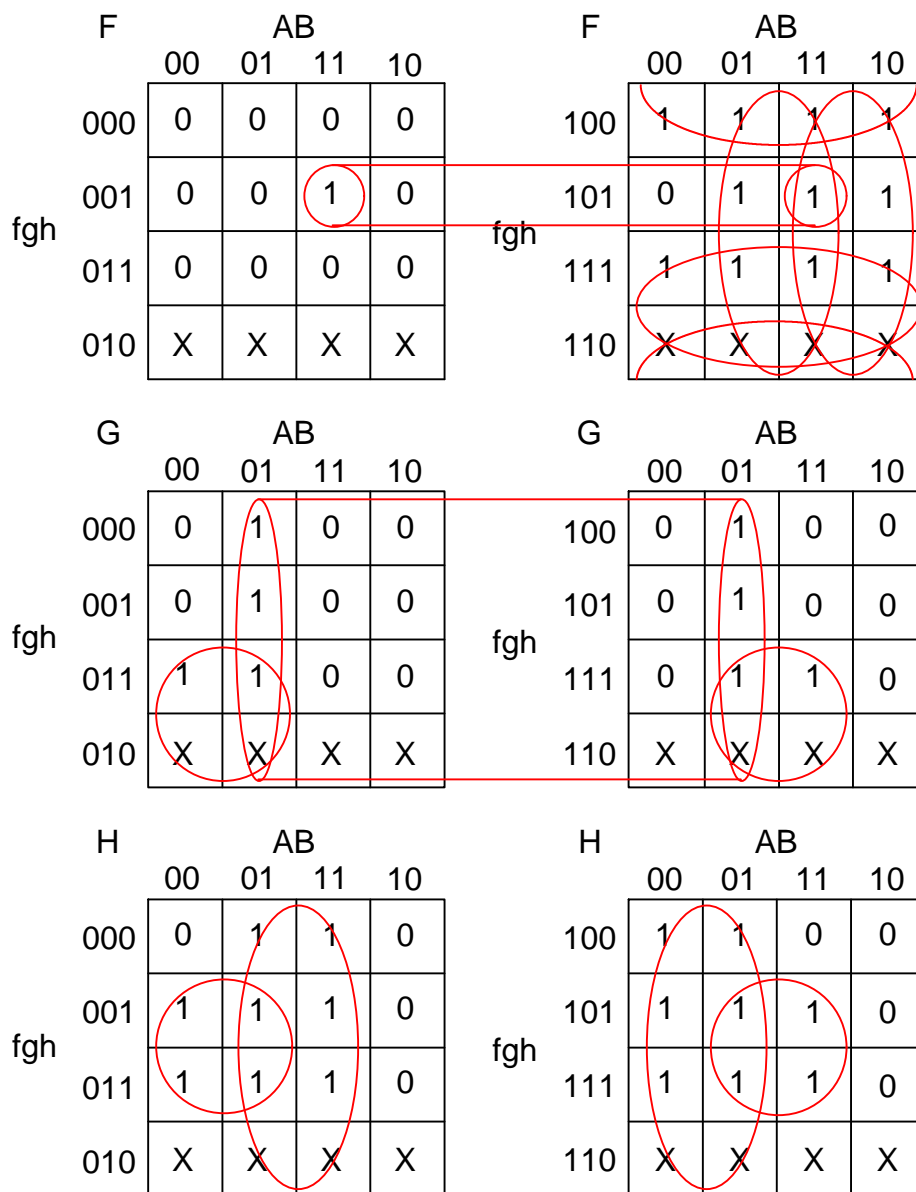


Figure 6. Karnaugh maps for F, G and H

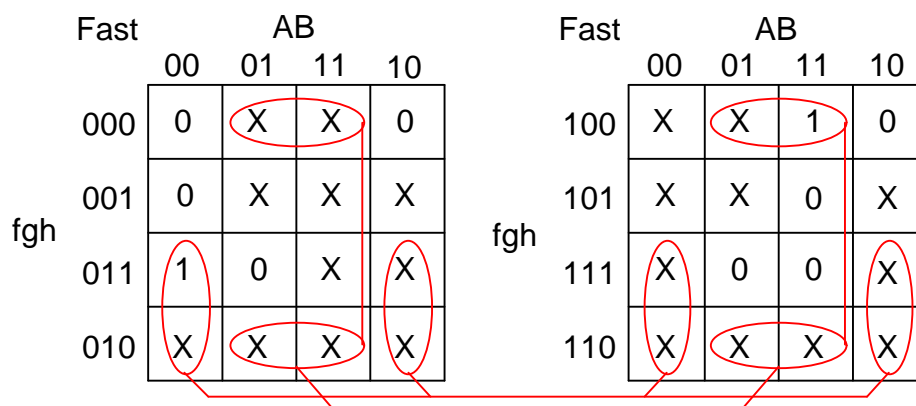


Figure 7 (a) Karnaugh map for Fast output.

| | | AB | | | | | | AB | | | |
|-----|------|-----|-----|-----|-----|-----|-----|-----|-----|----|----|
| | | 00 | 01 | 11 | 10 | | | 00 | 01 | 11 | 10 |
| fgh | Slow | 000 | 001 | 011 | 010 | 100 | 101 | 111 | 110 | | |
| | | 1 | X | X | 0 | X | X | 0 | 0 | | |
| | | 0 | X | X | X | X | X | 0 | X | | |
| | | 0 | 0 | X | X | X | 0 | 1 | X | | |
| | | X | X | X | X | X | X | X | X | | |

Figure 7 (b) Karnaugh map for Slow output.

The output states now need to be coded. The output Fast occurs during states 7 and 8. The output Slow occurs during states 9 and 10. In the Karnaugh maps for the output all other stable states are a one and all transitional states are don't cares.

The resulting output values are:

$$\begin{aligned}\text{Fast} &= B'g + Bh' \\ \text{Slow} &= A'h' + Ag\end{aligned}$$

Realisation

When this function was implemented, the Fast and Slow output pulses are not clearly visible on a LED. To provide a visible output, the Fast and Slow outputs were also used to drive a set-reset flipflop, giving outputs FOF (Fast Output F/F) and SOF (Slow Output F/F). These have been connected to pin 38 and 39, while the Fast and Slow outputs have been connected to pins 37 and 40. These pins are all connected to LEDs on the test board.

There are some glitches on the Fast and Slow outputs. These glitches will prevent a proper operation of the Set-Reset Flip-flop. The glitches can be removed by changing the outputs to:

$$\begin{aligned}\text{Fast} &= B'f'g + Bfh' \\ \text{Slow} &= A'f'h' + Afg\end{aligned}$$

The resulting ABEL File is as follows:

```
MODULE FreqDet
TITLE ' Frequency Detector,
      C. J. Kikkert
      James Cook University
      25 Aug 1998
      ispLSI2032 version '

Declarations
"Inputs
  A, B      pin 29, 30;                "/O 12,13, pin 29,30

"Outputs
  FA, SL    pin 40,37 istype 'com';    "LED outputs 'Fast' and 'Slow', pin 40 and 37
  SOF, FOF  pin 38,39 istype 'com';    "LED outputs 'Fast-RSF/F'and 'Slow-RSF/F',pin 38 and 39
```

```

"Intermediate Outputs
  FF, FG, FH, F, G, H   node   istype 'keep';  "Feedback variables for asynchronous sequential network
                                     "FF, FG, FH outputs (Capitals) F, G, H input (lower case)
  SO, FO node istype 'keep';  "Feedback variables for RS FlipFlop

"constants
  c,x,z = .C.,.X.,.Z.;

equations
  "Frequency Detector Logic
  FF = !G&H&A&B # G&F # !H&F # B&F # A&F ;
  FG = !A&B # F&G&B # !F&G&!A ;
  FH = !F&B # F&!A # H&!A # H&B ;
  FA = !B&F&G # B&F&!H;
  SL = !A&!F&!H # A&F&G;
  F = FF;
  G = FG;
  H = FH;
                                     "Freq B Fast Output
                                     "Freq B Slow Output
                                     "Now put outputs to input

  "Output RS FlipFlop
  SOF = !FA&SO # SL;
  FOF = !SL&FO # FA;
  SO = SOF;
  FO = FOF;
                                     "Now put outputs to input

test_vectors ([A, B ]->[FF,FG,FH, FA,SL])  "state
  [0, 0 ]->[ 0, 0, 0, 0, 1];  "9s
  [0, 1 ]->[ 0, 1, 1, 0, 0];  "2s
  [1, 1 ]->[ 1, 0, 1, 0, 0];  "3s
  [1, 0 ]->[ 1, 0, 0, 0, 0];  "4s
  [0, 0 ]->[ 0, 0, 1, 0, 0];  "1s
  [0, 1 ]->[ 0, 1, 1, 0, 0];  "2s
  [0, 0 ]->[ 0, 1, 1, 1, 0];  "7s
  [0, 1 ]->[ 0, 1, 1, 0, 0];  "2s
  [0, 0 ]->[ 0, 1, 1, 1, 0];  "7s
  [0, 1 ]->[ 0, 1, 1, 0, 0];  "2s
  [1, 1 ]->[ 1, 0, 1, 0, 0];  "3s
  [0, 1 ]->[ 1, 1, 1, 0, 0];  "5s
  [1, 1 ]->[ 1, 1, 1, 0, 1];  "10s
  [1, 0 ]->[ 1, 0, 0, 0, 0];  "4s
  [0, 0 ]->[ 0, 0, 1, 0, 0];  "1s
  [1, 0 ]->[ 0, 0, 0, 0, 0];  "6s
  [0, 0 ]->[ 0, 0, 0, 0, 1];  "9s

  "Note the output variables FOF, SOF not shown in this table are still available
  "in the waveform viewer.

end FreqDet

```

The ABEL file can be used to fit the realisation into an ispLSI2032, using the Synario software and download cable. The resulting waveforms are shown in Figure 8. The realisation fits in one quarter of the ispLSI2032.

The Synario Project file is shown in Figure 9. As the ABEL file above also contains test vectors, it is also shown as a .abv file in the project window.

The circuit works as expected and can detect frequency differences from DC up to tens of MHz.

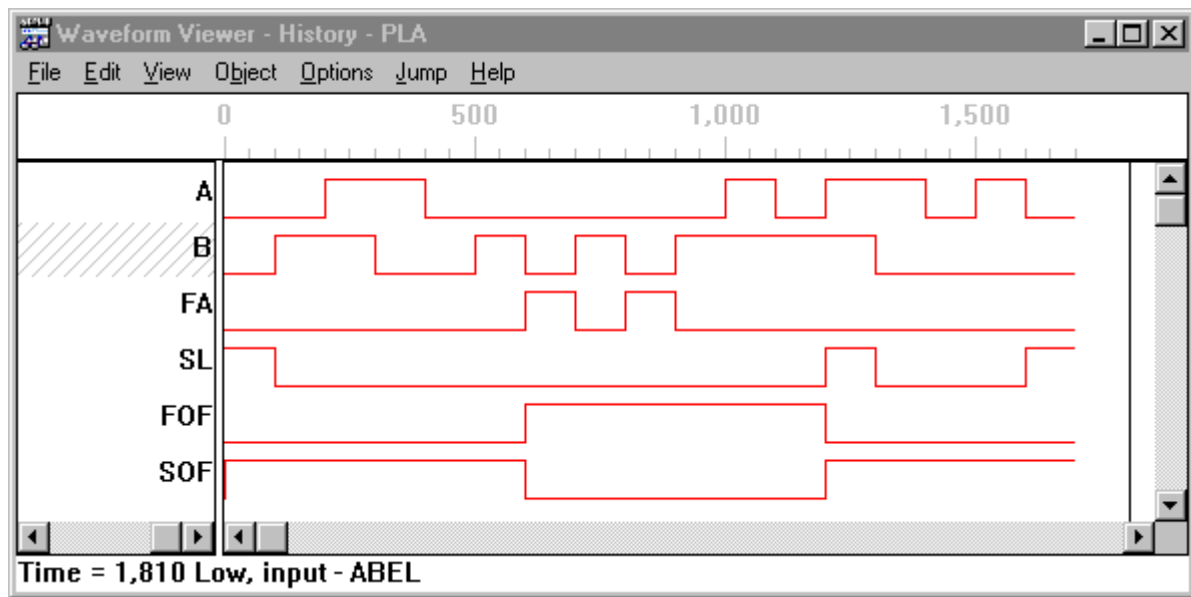


Figure 8. Frequency Detector Waveforms.

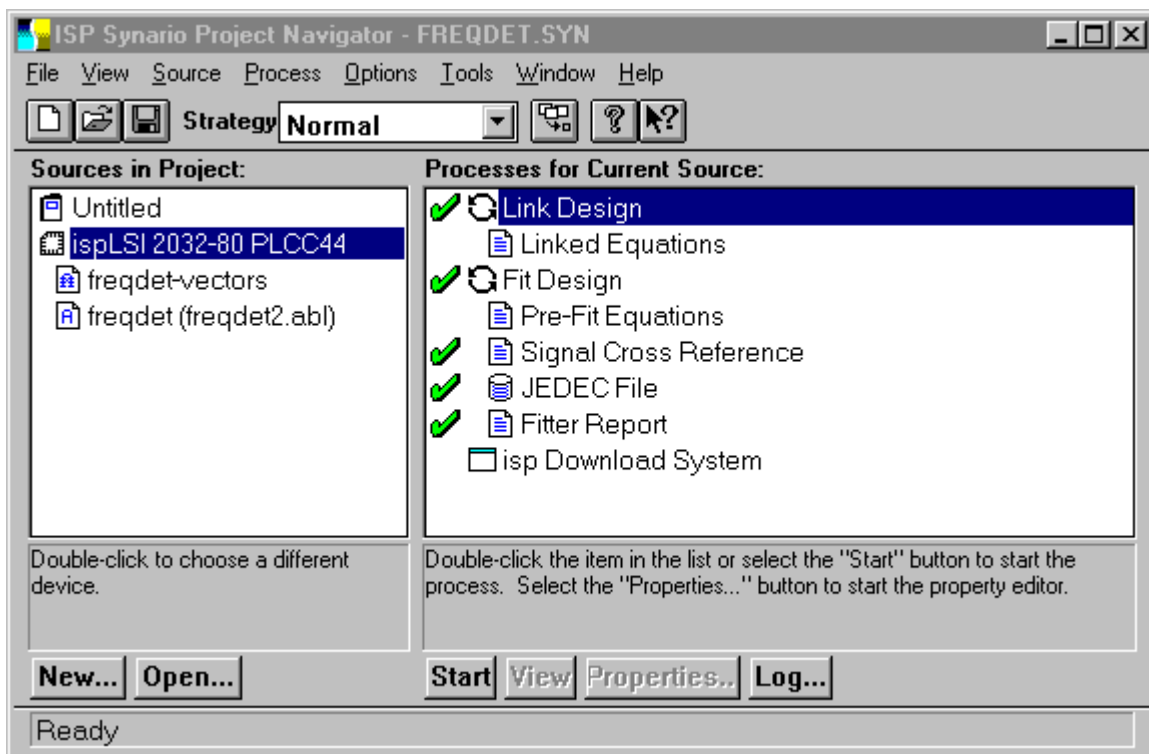


Figure 9. The asynchronous sequential logic realisation.

Synchronous Sequential Logic

The circuit can also be realised using synchronous sequential logic. If each of the inputs A and B is used as a clock for a separate counter. The difference between counter A and counter B is a measure of the phase difference between the waveforms. If the difference is increasing then A is of a higher frequency. If the difference is decreasing then B is of a higher frequency. Since the EPLD test board has 8 LEDs, a 3

bit counter can be used for the A counter and the B counter. The difference can be taken by inverting the output from counter B and using that and the counter A output into a 3 bit full adder. The output of that adder is then the difference again using 3 bits. Using a 3 bit to octal decoder, provides 8 LED outputs corresponding to the difference between the counters. Counter and adder overflows can be ignored as a valid 3-bit difference is always provided.

The schematic for the above circuit is shown in Figure 10.

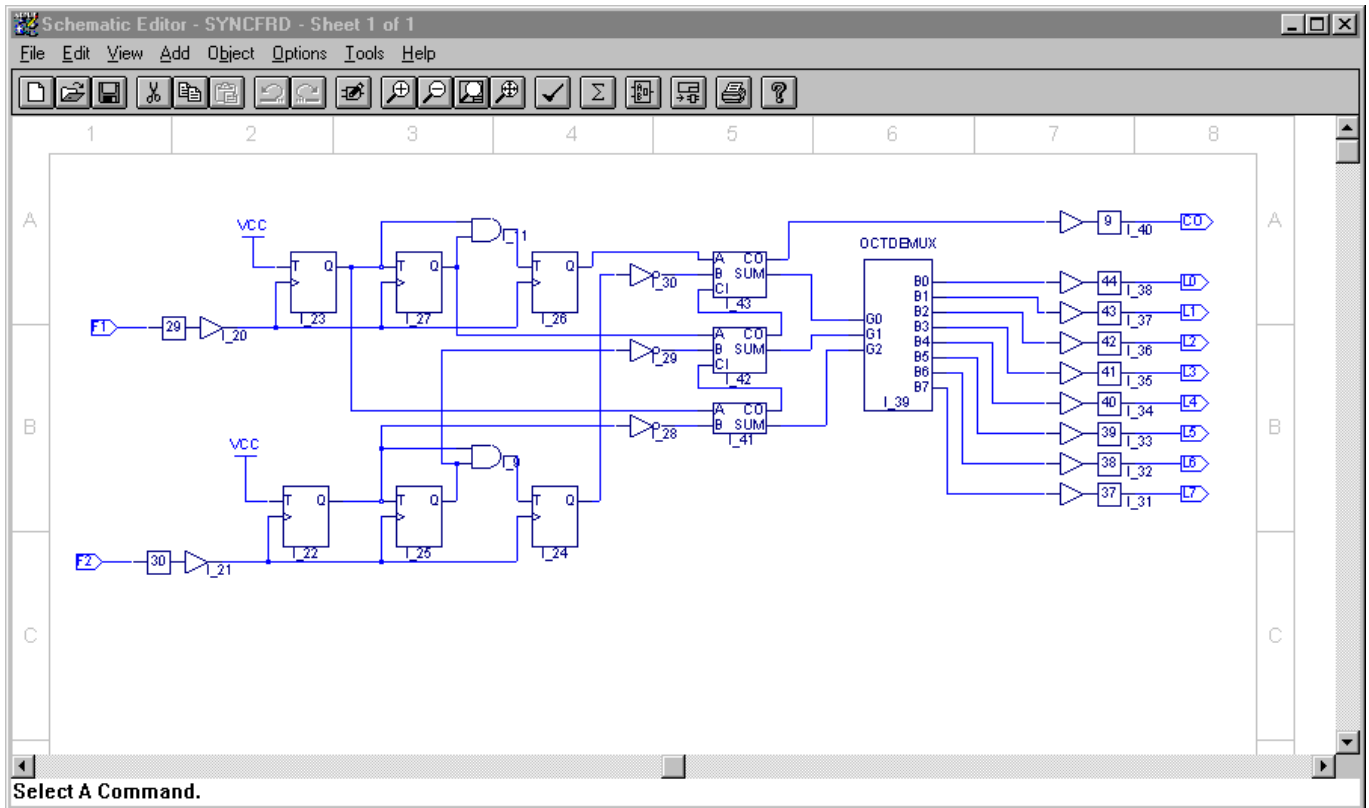


Figure 10. Frequency Detector Schematic.

The counters are on the left, the adder is in the middle and the Octal De-multiplexer Block is at the right. That Octal de-multiplexer is most easily realised using an ABEL file as follows:

```
MODULE octdemux
```

```
TITLE ' 3 to 8 Demultiplexer
```

```
C. J. Kikkert
```

```
James Cook University
```

```
12 Sep 1998'
```

```
"Inputs
```

```
G0, G1, G2 pin;
```

```
"Outputs
```

```
B0, B1, B2, B3, B4, B5, B6, B7 pin istype 'com';
```

```
"Declaration
```

```
InD = [G0,G1,G2];
```

```

truth_table (InD -> [B0,B1,B2,B3,B4,B5,B6,B7])
0 -> [ 1, 0, 0, 0, 0, 0, 0, 0];
1 -> [ 0, 1, 0, 0, 0, 0, 0, 0];
2 -> [ 0, 0, 1, 0, 0, 0, 0, 0];
3 -> [ 0, 0, 0, 1, 0, 0, 0, 0];
4 -> [ 0, 0, 0, 0, 1, 0, 0, 0];
5 -> [ 0, 0, 0, 0, 0, 1, 0, 0];
6 -> [ 0, 0, 0, 0, 0, 0, 1, 0];
7 -> [ 0, 0, 0, 0, 0, 0, 0, 1];

end octdemux

```

It can be seen that the heart of the file is the truth table. The Synario compiler simply processes that to produce the required output. The Synario Project screen is shown in Figure 11.

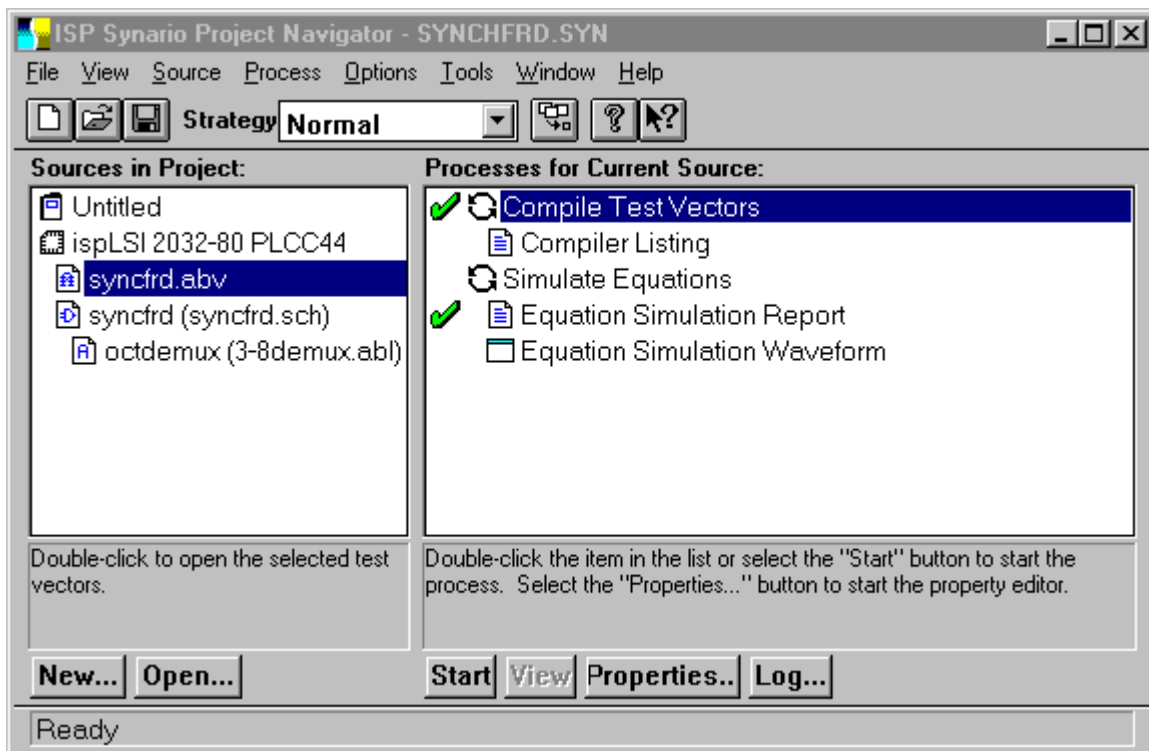


Figure 11. Synchronous Sequential Logic Frequency Detector Project Window.

The resulting waveforms are shown in Figure 12. It can be seen that the count decreases if F2 is of a higher frequency than F1, and vice versa. Some glitches are shown on the waveforms in Figure 12, however these do not cause any problems. As the output is the result of two independent input signals, very small differences in the time of events can occur, resulting in the glitches.

This circuit works very nice and has a good indication of the difference in frequency between the two waveforms. If the frequency difference is very large however, all the 8 LEDs are lit, and as the frequency indication is determined from the increase or decrease of the counter difference, as indicated by the movement of the LED sequence, this circuit does not perform well under those conditions.

The circuit uses 5 of the 8 macrocells for the realisation in a ispLSI2032. As the asynchronous sequential logic circuit only takes 2 macrocells, both realisations can be included into the one ispLSI2032. In such a combined design ten LEDs are required. 8 for the synchronous logic and 2 for the Fast and Slow outputs of the asynchronous logic. Unfortunately the test board only has 8 LEDs.

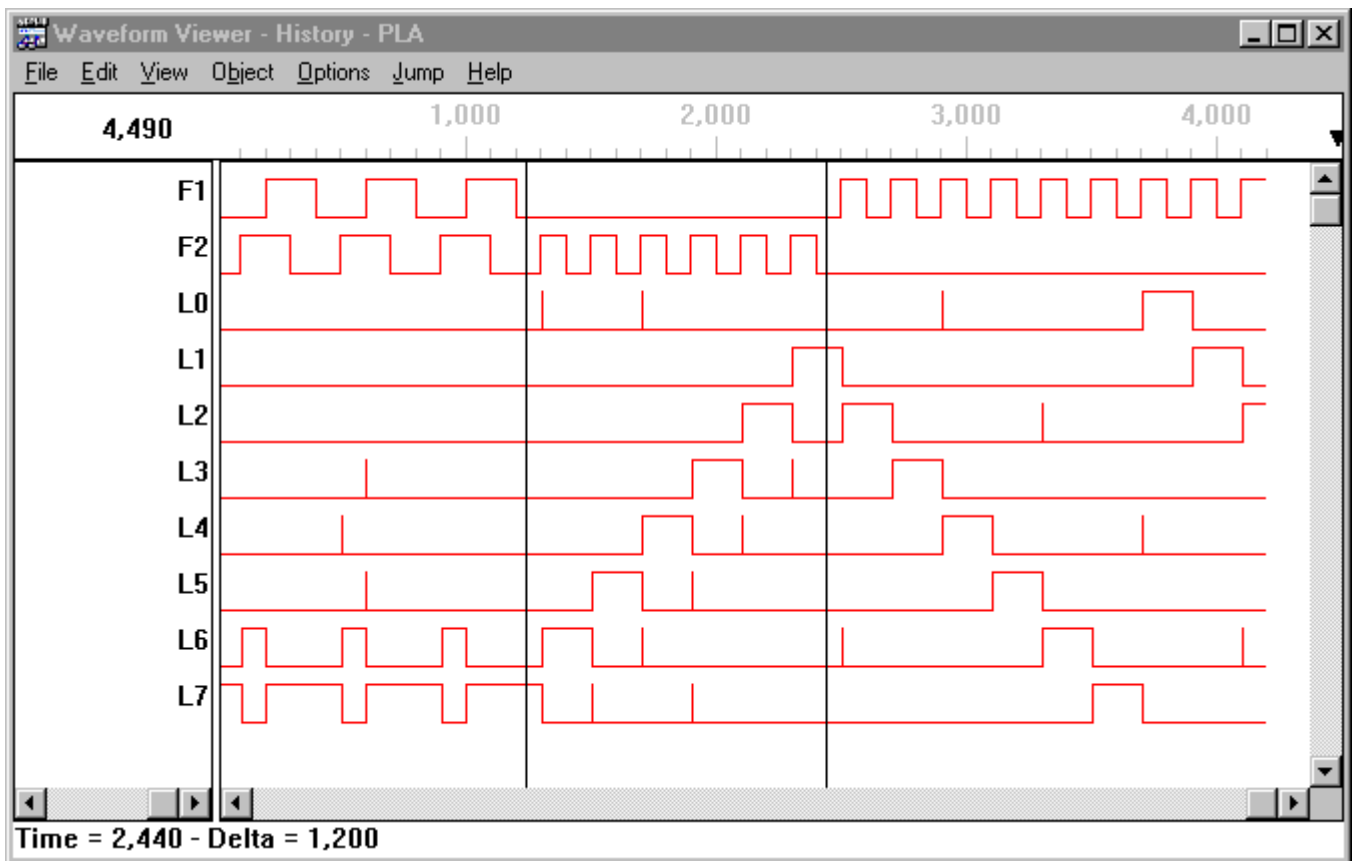


Figure 12 Synchronous Frequency Detector waveforms.

In figure 12 three regions can be identified. The region on the left of the vertical lines is where both F1 and F2 are the same frequency. During this time LEDs L6 and L7 are partially ON. All others are OFF (apart from some glitches on L3, L4 and L5). In the region between the vertical lines, frequency F2 is higher and the LEDs progressively change from L7 to L1. In the region to the right of the vertical lines, frequency F2 is higher and the LEDs progressively change from L1 to L7 and on to L0 and L2.