EE3DSD: Digital System Design (Coursework 3)

The following report details some of the methodology and reasoning behind the work undertaken to implement the bit decoder components for the UK MSF and German DCF77 low frequency radio signals.

Contents

List of Figures

List of Tables

Methodology

As with the previous coursework, development began with creating testbenches for each component, using data recorded from the model answer implementation using minicom. For the bit decoding components, I used data which was recorded from the first coursework, as this meant I had a sample minute of data for both the DCF and MSF signals which I had already hand-decoded (Table 1 and Table 2).

```
1 1 1 0 0 0 0 0 0     8416.53509 ms
0 1 1 0 0 0 0 0 0     8506.42440 ms
1 1 1 0 0 0 0 0 0     9410.87999 ms
0 0 1 0 0 0 0 0 0     9606.97260 ms
1 1 1 0 0 0 0 0 0    10409.72956 ms
0 1 1 0 0 0 0 0 0    10506.94701 ms
```

Bit	Pulse width (ms)	Value
:00	89.889	0
:01	196.093	1
:02	97.217	0

Table 1 An extract from the recorded DCF trace

```
0 1 1 0 0 0 0 0 0     8403.87185 ms
0 0 1 0 0 0 0 0 0     8915.78745 ms
0 1 1 0 0 0 0 0 0     9404.05309 ms
1 0 1 0 0 0 0 0 0     9521.29280 ms
0 1 1 0 0 0 0 0 0    10403.30498 ms
0 0 1 0 0 0 0 0 0    10519.87661 ms
```

Bit	Pulse width (ms)	Value	
:00	511.916	1	1
:01	117.240	0	0
:02	116.572	0	0

Table 2 An extract from the recorded MSF trace

Testbench stimuli

To prepare the testbenches, the recorded data was filtered into a plaintext list of timestamps of alternating rising and falling edge event times. Each value was subtracted from the first value so as to give a list of rising/falling edge times relative to the first rising edge starting at 0 ms, and each falling edge time has an extra column which is equal to the bit out which I hand-decoded, with two bit values for the MSF trace (Figure 1).

```
$ head -n6 dcf.txt
0 ms
89.88931 ms 0
994.3449 ms
1190.43751 ms 1
1993.19447 ms
2090.41192 ms 0
$ head -n6 msf.txt
0 ms
511.9156 ms 1 1
1000.18124 ms
1117.42095 ms 0 0
1999.43313 ms
2116.00476 ms 0 0
```

Figure 1 An extract of the DCF and MSF test stimuli

Testbenches

The testbenches have several concurrent processes, one to set the duration of the test, one to simulate a `clk` signal, one to read the test stimuli from the file and assign `di` and `si`, and another process that reads the hand-decoded bit values and uses the `assert` procedure to verify that the value decoded by the testbench device matches the value that I decoded by hand. Figure 2 shows the additions to the Makefile which were needed to simulate and view the testbenches.

```
TESTBENCHES = dcf_bits_testbench msf_bits_testbench
VIEW_TB := dcf_bits_testbench

all simulation sim:
	@for f in $(SOURCES); do \
		if [ -f $$f ]; then \
			echo "ghdl -a $$f"; \
			ghdl -a $$f; \
		fi; \
	done
	@for t in $(TESTBENCHES); do \
		echo "ghdl -e $$t"; \
		ghdl -e $$t; \
		echo "ghdl -r $$t --vcd=$$t.vcd"; \
		ghdl -r $$t --vcd=$$t.vcd; \
	done

view:
	gtkwave $(VIEW_TB).vcd $(VIEW_TB).sav >/dev/null 2>&1
```

Figure 2 Makefile targets for simulation and testbenches

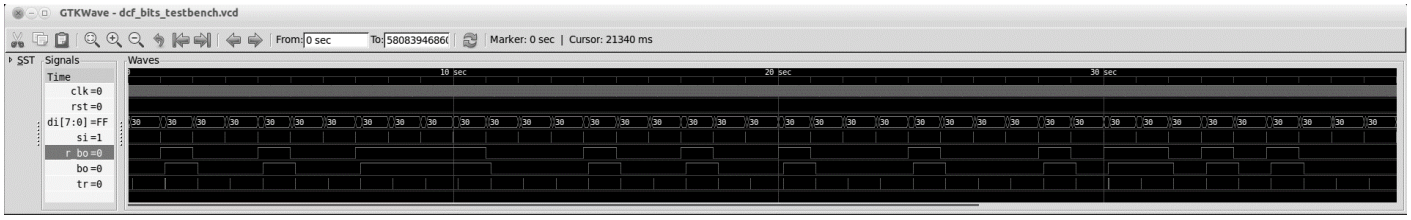

Figure 3 Testbench for the DCF bits component

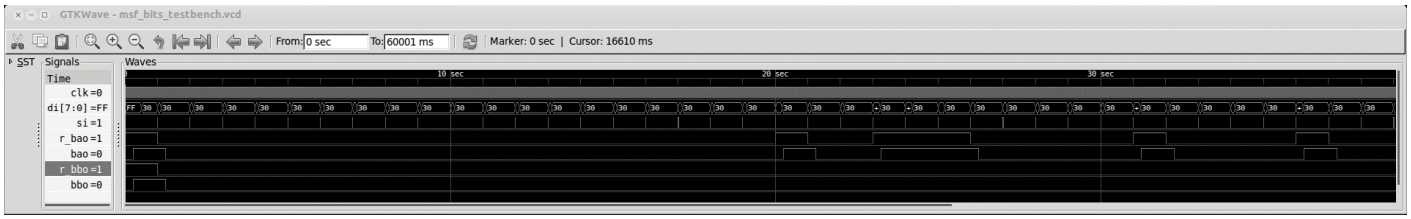

Figure 4 Testbench for the MSF bits component

DCF bit decoder

The DCF bit decoder works by making a sample of `di` 150 ms after receiving an `si` pulse, and using that to determine whether the bit was high or low (Figure 5). The reasoning behind this is that a DCF pulse is either 100 ms or 200 ms in length, so by sampling at the halfway point between those two values, you can most accurately determine the bit value, while leaving the largest room for timing inaccuracy in the received signal.

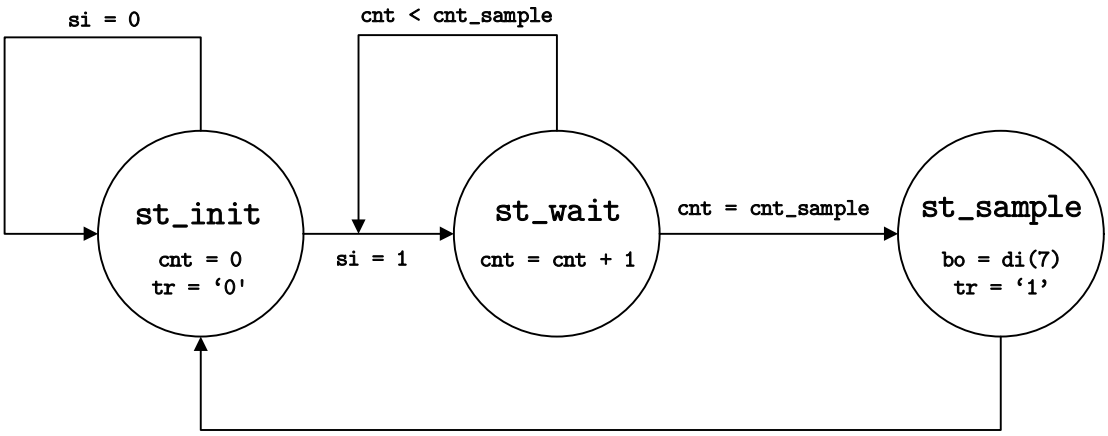

Figure 5 A Moore state diagram for the DCF bits decoder.

Figure 6 shows an enlarged view of the start of a second, which shows how the `si` pulse goes high exactly 150 ms before the `tr` output of the bits component is set high.

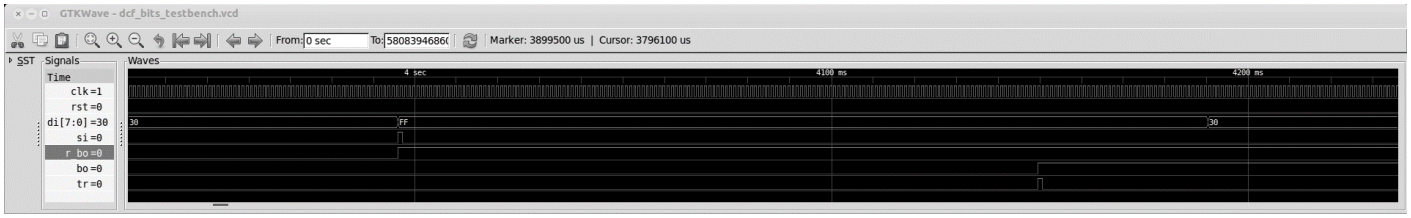

Figure 6 The start of a second for the DCF signal

MSF bit decoder

The MSF bit decoder functions in much the same way as for the DCF component, except for the added complexity of having to sample twice in each second, once at 150 ms and once at 250 ms. This is because the MSF signal encodes two bits in each second.

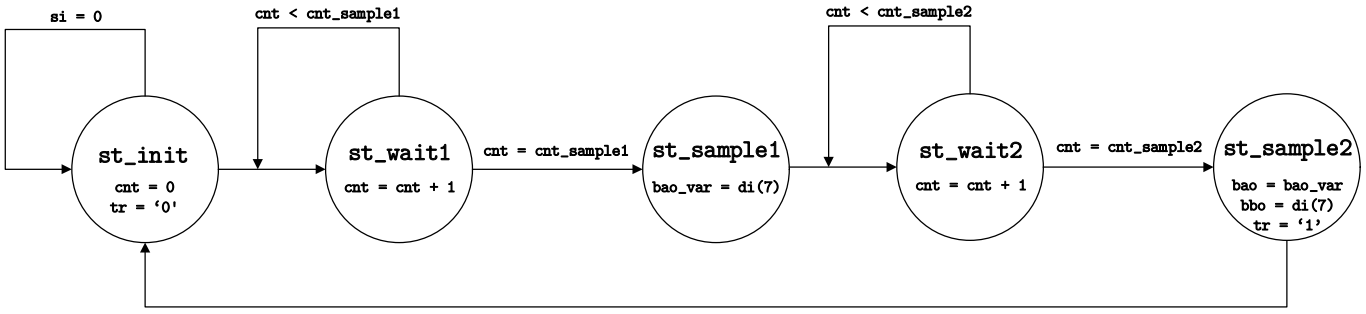

Figure 7 A Moore state diagram for the MSF bits decoder.

Figure 8 shows an enlarged view of the start of a second, which shows how the `si` pulse goes high exactly 250 ms before the `tr` output of the bits component is set high.

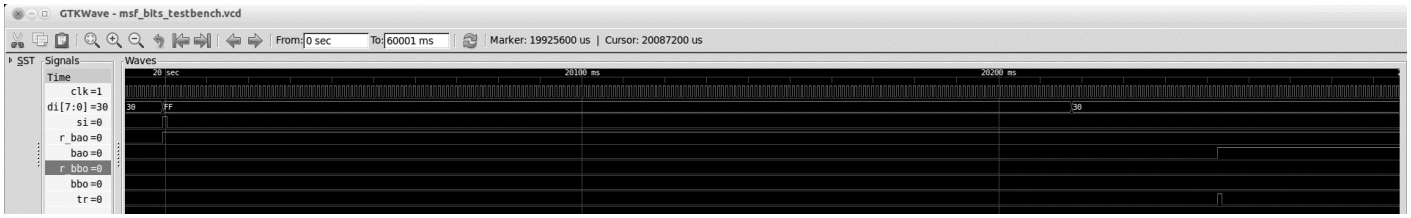

Figure 8 The start of a second for the MSF signal

Seven Segment Display

Figure 9 shows a state diagram of the `ssg` component. In addition to refreshing the values at a rate of 60 Hz (or 16ms, which means 4 ms each between the four displays), the data which is displayed is only updated when a `wr` signal is made. The testbench for the ssg component simply writes a few different values to `di`, and can be used to determine that the correct data is being written to `ka` and `an`.

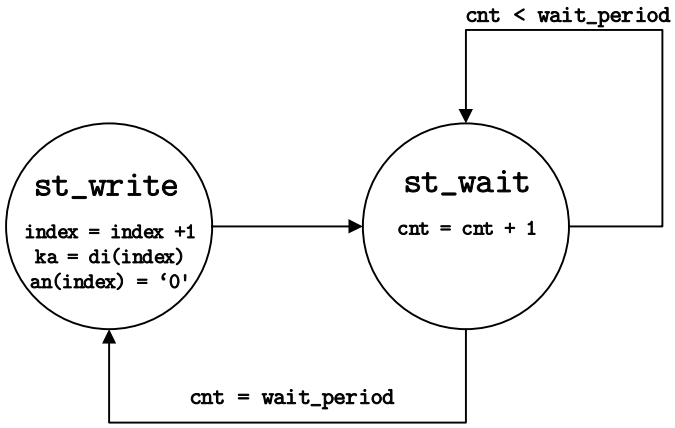

Figure 9 A Moore state diagram of the Seven Segment driver