EE3DSD: Digital System Design (Coursework 4)

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

Contents

Methodology 2
Testbench stimuli 2
Testbenches 2
DCF decoder 4
MSF decoder 4

List of Figures

Figure 1 An extract of the DCF and MSF test stimuli 2
Figure 2 Makefile targets for simulation and testbenches 3
Figure 3 Testbenches for the MSF and DCF decoder components 3
Figure 4 A Moore state diagram for the DCF bits decoder. 4
Figure 5 A Moore state diagram for the MSF bits decoder. 4

List of Tables

Table 1 An extract from the recorded DCF trace 2
Table 2 An extract from the recorded MSF trace 2

Methodology

As with the previous courseworks, development began with creating testbenches for both components, 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). Since the decode components are purely digital logic and do not have any timing considerations, we can discard the timestamps, and use just the list of decoded bit values.

```
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 decoded bit values (Figure 1).

```
$ head -n5 dcf-decode.txt
0
1
0
0
1
$ head -n5 msf-decode.txt
1 1
0 0
0 0
0 0
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 read the test stimuli from files and assign the input bits, one to assert that the component outputs the correct values, and processes to set the `clk`, `si`, and `mi` inputs. Figure 2 shows the additions to the Makefile which were needed to simulate and view the testbenches.

```
TESTBENCHES = dcf_decode_testbench msf_decode_testbench
VIEW_TB := dcf_decode_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 --wave=$$t.ghw"; \
		ghdl -r $$t --wave=$$t.ghf; \
	done

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

Figure 2 Makefile targets for simulation and testbenches

The testbenches simulate a minute of real world data as recorded during coursework one, followed by a further minute which contains inputs which cause the parity checks to fail.

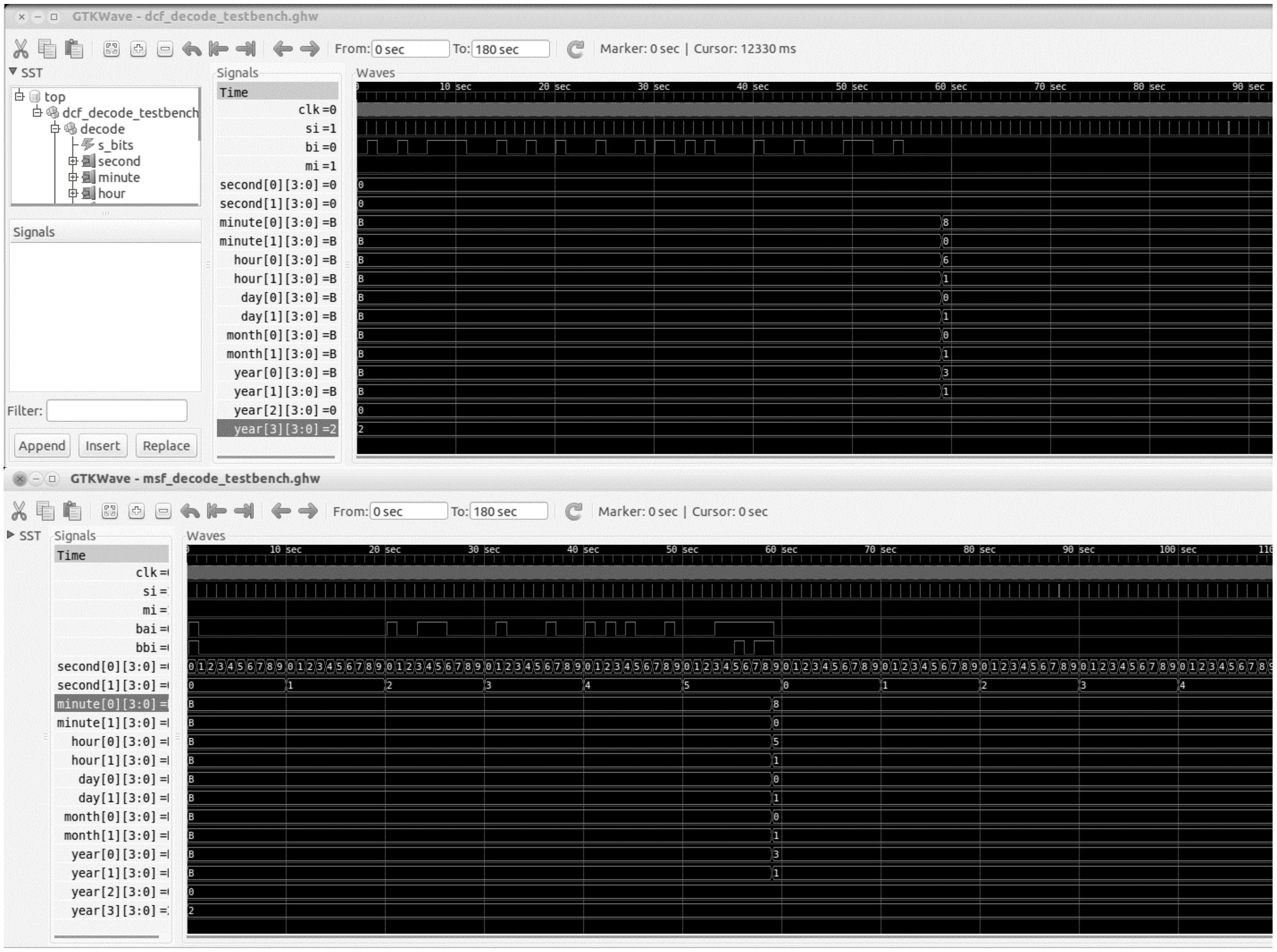

Figure 3 Testbenches for the MSF and DCF decoder components

DCF decoder

The DCF decoder contains a `std_logic_vector` of 59 bits which acts as a buffer to contain a full minute's worth of decoded bit values. A numerical index is increased by one each second, and used to index into this array so that for every `si` pulse, a `bi` value is inserted into the bit array in a unique location. Once the last bit has been received, the buffer is filled with a minute data, and the date and time values are extracted from the buffer and written to the output ports. Parity checks are performed for each of the three parity bits.

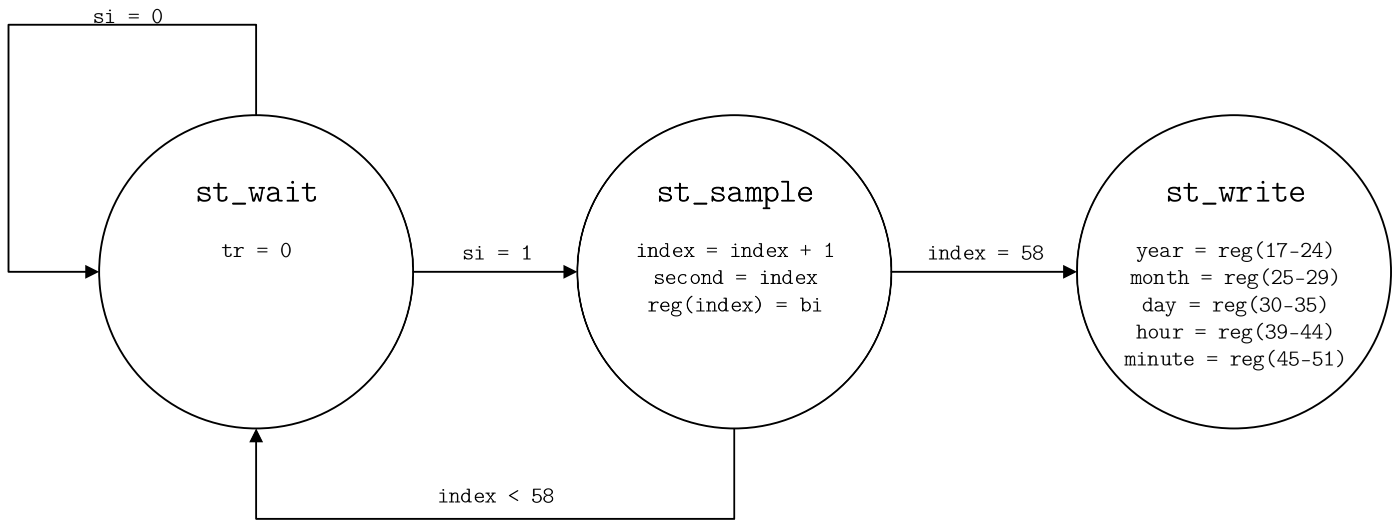

Figure 4 A Moore state diagram for the DCF bits decoder.

MSF decoder

The MSF bit decoder functions in much the same way as for the DCF component, except for the added complexity of having two bit buffers, since there are two bits in each second. In addition to the parity checks, we also perform checks that expected bit values are the correct values.

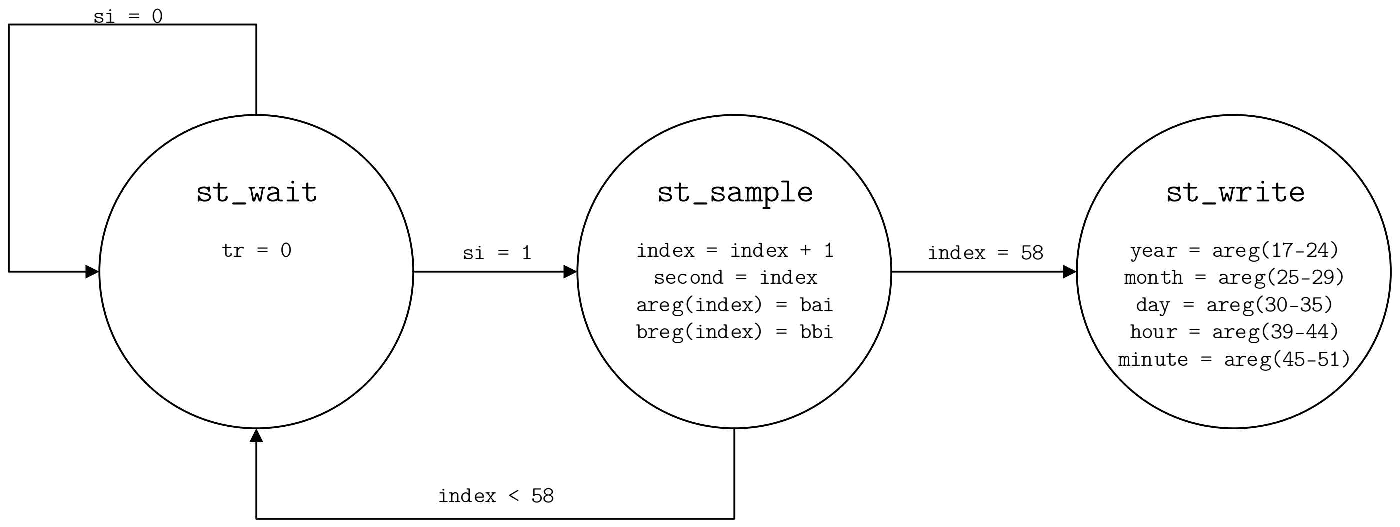

Figure 5 A Moore state diagram for the MSF bits decoder.