Lecture 2: Linear Feedback Shift Registers and Logic Analysis

G. Kemnitz*, TU Clausthal, Institute of Computer Science

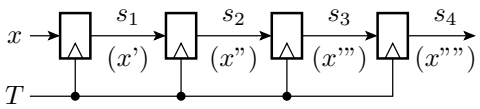
May 25, 2011

Abstract

A linear automaton is a simple sequential circuit to produce a periodic pseudo-random bit sequence and is in this exercise the device under test. The produced bit sequence should first be calculated by simulation, in a second task recorded by an external logic analyzer and in a third task be recorded by the integrated logic analyzer »ChipScope«. Results will be displayed and compared.

1 Linear feedback shift registers

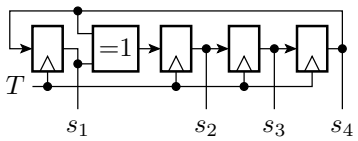
A shift register is a chain of edge triggered flip-flops, in which each successor flip-flop takes the value of its predecessor. With each active clock edge the stored bit vector moves one step along. At the begin one bit is added and at the end one bit gets lost.



x	s_1	s_2	s_3	s_4
1	0	0	1	1
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

Figure 1: Shift register

A linear feedback shift register (LFSR) is a shift register, in which in addition the output value of the last bit is added modulo-2 to selected bit positions. A modulo-2 addition adds two bits without calculating the carry and is realized by an EXOR gate. The special feature of a linear feedback shift register is that it, starting with an initial value unequal zero, traverses a state sequence with a strong similarity to a random sequence (figure 2).



step	s_1	s_2	s_3	s_4	step	s_1	s_2	s_3	s_4
1	1	1	0	1	9	1	0	0	0
2	1	0	1	0	10	0	1	0	0
3	0	1	0	1	11	0	0	1	0
4	1	1	1	0	12	0	0	0	1
5	0	1	1	1	13	1	1	0	0
6	1	1	1	1	14	0	1	1	0
7	1	0	1	1	15	0	0	1	1
8	1	0	0	1	1	1	1	0	1

Figure 2: 4-bit linear feedback shift register

*Tel. 05323/727116

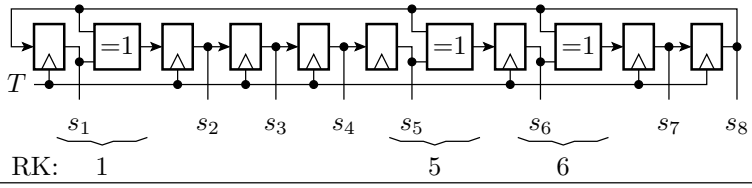
To produce pseudo random sequences primitive feedback shift register are preferred. A primitive feedback shift register is a linear feedback shift register with a maximum state sequence of the length

$$Z = 2^r - 1$$

(r – length of the shift register). This sequence contains all states except the »all zero state«. The zero state is always its own successor. The 4-bit linear feedback shift register in figure 2 has e.g. the cycle length

$$Z = 2^4 - 1 = 15$$

and so a primitive feedback. Figure 3 a shows an 8-bit primitive feedback shift register and figure 3 b possible feedback's for other register lengths.



a)

r	RK	r	RK	r	RK	r	RK	r	RK
8	1, 5, 6	28	3	48	1, 27, 28	68	9	88	1, 71, 72
12	3, 4, 7	32	1, 27, 28	52	3	72	6, 47, 53	92	1, 12, 13
16	2, 3, 5	36	11	56	1, 21, 22	76	1, 35, 36	96	2, 47, 49
20	3	40	2, 19, 21	60	1	80	1, 37, 38	100	37
24	1, 3, 4	44	1, 26, 27	64	1, 3, 4	84	13	104	1, 10, 11

b)

Figure 3: a) 8-bit primitive feedback shift register b) Feedback positions of primitive feedback's for other register lengths

The circuit of a feedback shift register has a clock and an initialization input and the register state, a bit-vector of size r, as output. The bit-vector with the feedback positions (bits with feedback's are »1« and the others »0«) should be a parameter of the entity of the design unit :

```
entity LFSR is
 generic (
  RK: STD_LOGIC_VECTOR:="0010");
 port (T, I: in STD_LOGIC;
  s: out STD_LOGIC_VECTOR(RK'LENGTH-1 downto 0));
end entity;
```

With the default value »0010« the 4-bit register in figure 2 is described. To instantiate the 8-bit register in figure 3 a the default value of the parameter **RK** has to be overwritten by:

```
... generic map(RK => "01100010") ...
```

To sample the initialization signal an internal signal has to be declared. The sampling itself is described in a sampling process, assigning on each rising clock edge the value of the asynchronous external initialization signal to the internal initialization signal:

```
architecture a of LFSR is
 signal I_del: STD_LOGIC;
 signal z: STD_LOGIC_VECTOR(RK'LENGTH-1 downto 0);
begin
 process(T)
 begin
  if RISING_EDGE(T) then I_del <= I; end if;
 end process;
```

The function of the LFSR is described in a process with the internal initialization signal and the clock in the sensitivity list. If the initialization signal is »1«, the initialization state »all ones« and else, if there is a rising clock edge the current state rotated by one position is assigned to the state signal. If the leading bit of the current register state is one in addition the feedback vector **RK** is added bit-wise modulo-2. The assignment of the state signal to the output signal is described by the final concurrent signal assignment:

```
 process(I_del, T)
 begin
  if I_del='1' then
   z <= (others =>'1');
  elsif RISING_EDGE(T) then
   if z(z'HIGH)='0' then
    z <= z(z'HIGH-1 downto 0) & z(z'HIGH);
   else
    z <= (z(z'HIGH-1 downto 0) & z(z'HIGH)) xor RK;
   end if;
  end if;
 end process;
 s <= z;
end architecture;
```

2 Simulation

The simulation requires a testbench, containing the linear feedback shift register as an instance and to produce the input signal T and I. For this, a constant for the clock period and signals for the interface of the DUT are declared. In the following example the default value for the parameter **RK** is kept. It means that the device under Test is the feedback shift register in figure 2:

```
constant tP: DELAY_LENGTH := 20 ns;
signal I, T: STD_LOGIC;
signal y: STD_LOGIC_VECTOR(3 downto 0);
...
DUT: entity WORK.LFSR port map(T=>T, I=>I, s=>y);
```

The following test process produces at the begin of the simulation an initialization pulse with a duration of $2{,}7 \cdot t_\mathrm{P}$ and a clock signal witch toggles always after $t_\mathrm{P}/2$ from »0« to »1« and vice versa. After a total of 0,5 µs simulation time, simulation stops with a wait statement without wake-up condition.

```
test: process
begin
 T<='0'; I<='1', '0' after 2.7*tP;
 loop
  wait for tP/2; T <= not T;
  if NOW > 0.5 us then wait; end if;
 end loop;
end process;
```

The description of the device under test, the testbench, a shell script withe command sequence for the simulation with GHDL and GTKWAVE and a sav-file are given. As in the exercise before, the corresponding archive, here »PrVHDL-A2.zip« has to be unpacked in the working directory of the laboratory course. Also the other steps to carry out the simulation and visualize the simulation results are the same as in the exercise before.

2.1 Synthesis with ISE

The circuit description »LFSR.vhdl« is already fit for synthesis[1]. To improve testability, the LFSR should be embedded in the enclosing circuit in figure 4 a. As primary clock the 50 MHz clock »GCLK0« produced by the external oscillator at the bottom of the test board should be used. The switch »SW0« is used to select between a fast clock with half of the frequency of the primary clock and a slow clock with a frequency of approximately 1 Hz (Frequency of the primary clock divided by 2^{27}). A clock signal must reach all memory cells almost simultaneously. For this, the programmable logic circuit has special clock nets, which are driven by »BUFG« driver. In the VHDL description after the clock divider the BUFG driver has to be inserted manually[2]. The initialization signal is produced by »BTN0«. The output signal of the feedback shift register is connected first to the LEDs and second to the expansion connector A2 (opposite to the switches). The expansion connector is to mount the external logic analyzer to those outputs (figure 4).

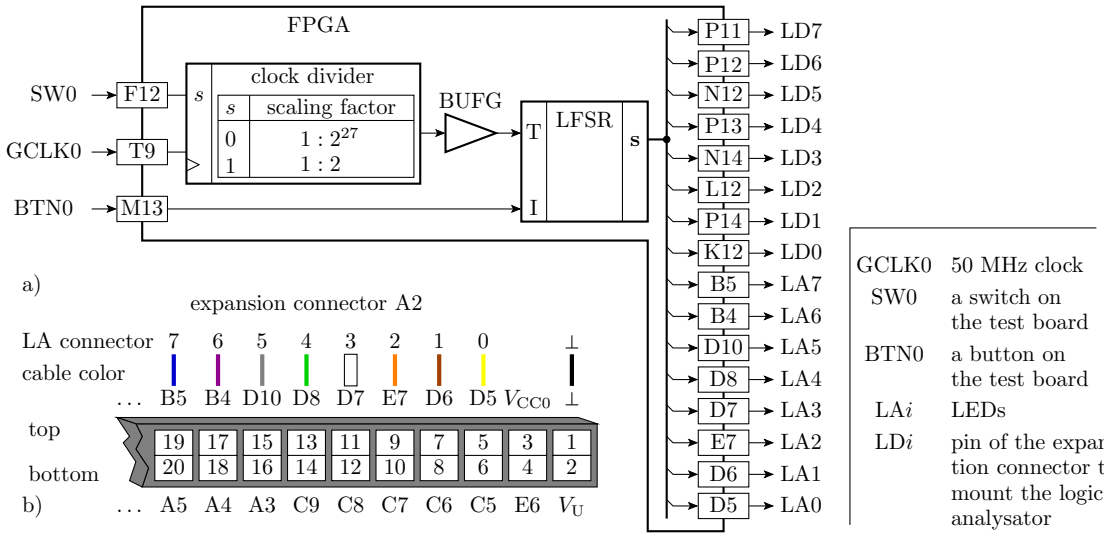


Figure 4: a) Enclosing circuit of the LFSR b) Assignment of the inputs of the logic analyzer extension connector A2

The interface of the enclosing circuit is:

```
entity Gesamtschaltung is
 port(GCLK0, SW0, BTN0: in STD_LOGIC;
      LD, LA: out STD_LOGIC_VECTOR(7 downto 0));
end entity;
```

(Gesammtschaltung – German word for whole circuit). In the architecture description signals has to be declared for the branching output signal and the down scaled clock before and after the »BUFG«. The clock divider is a process that counts up the rising edges of the input clock in a variable. If switch setting is »SW0=0« the generated clock is inverted with every and else with every 25.000.000th rising edge of the input clock.

```
architecture a of Gesamtschaltung is
 signal y: STD_LOGIC_VECTOR(7 downto 0);
 signal T, TBufG: STD_LOGIC:='0';
begin
 ClkDiv: process(GCLK0)
```

[1]It does not contain delay times, output of text messages or other statements not supported by synthesis.

[2]In later designs with internal clock dividers always feed the internally generated clocks via »BUFG« driver into the clock net. Otherwise clock skews may cause difficult to locate malfunctions.

```
 variable Ct: NATURAL range 0 to 25000000;
begin
 Ct := Ct + 1;
 if Ct = Ct'HIGH then
  T <= not T;
 end if;
end process;
```

An important detail in the description of the clock divider is, that the down scaled output clock is declared as a bit signal and signal assignments to it are done in a sampling process. This guarantees that the clock will be taken from the output of a flip-flop which minimizes the clock skew and avoids glitches. This detail also should be adopted in all later designs with a clock divider. The »BUFG« driver – a basic design component – is instantiated as an component. The component declaration is in the package »UNISIM.VComponents«, that has to be imported at the begin of the design file:[3]

```
clock_driver: BUFG port map(I=>T, O=>TBufG);
```

The device under test is here the 4-bit shift register if figure 2. The output signal is correspondingly 4 bit wide. It is connected to the lower output bits and the internal clock to the highest output bit. The rest of the output bits are set to zero.

```
 DUT: entity WORK.LFSR port map(T=>TBufG, I=>BTN0, s=>y(3 downto 0));
 y(7 downto 4) <= T & "000";
 LD <= y;
 LA <= y;
end architecture;
```

Die VHDL file the whole circuit and the project file are given and unpacked from the archive to the directory »Aufg2/ISE«. The constraint file »Aufg2.ucf_« is incomplete and has to be renamed before the start of »ISE« . to »Aufg2.ucf«. To the rest of the circuit connectors the package pin assignments has to be added as shown in figure 4 a. After starting »ISE« change to directory »Aufg2/ISE«, open the project »Aufg2«, synthesize and download the design in the programmable circuit as described in the first lecture.

2.2 Test

To test the circuit via the LEDs »SW0« has to be switched to »0« (1Hz clock). Pushing the button »BTN0« the four low-order LEDs must turn on and »LD7« must blink with the clock. After releasing the reset button the low-order LEDs must display cyclic the generated pseudo random sequence.

2.3 Logic analysis

Switching to the fast clock, the circuit runs so fast that only a steady glowing of the LEDs will be displayed. A logic analyzer is a device that records logical data streams at its inputs with a high speed. Our logic analyzer has to be mounted to the expansion connector as displayed in figure 4 b.

2.3.1 Configuring the logic analyzer

Before testing the logic analyzer has to be configured via an xml-file. The sample rate describes the number of recorded sample values per second. Valid values are the numbers from 1 to 150.000.000 and 600.000.000. For the test with the fast clock 600 million samples per second are a reasonable value:

[3] The template of a component declaration can be found under »Edit« ▷ »Language« ▷ »Templates« ▷ »VHDL« ▷ »Device Primitive Instantiation« ▷ »FPGA« ▷ »Clock Components« ▷ »Clock Buffers«.

```
<la>
 <samplerate>600000000</samplerate>
```

The used logic analyzer always logs 4048 sample values. So the whole recording time is:

$$t_{\text{Aufzeichnung}} = \frac{4048}{600.000.000\,\text{s}^{-1}} \approx 6{,}7\,\mu\text{s}$$

With 25 clocks per microsecond nearly 170 clock period are recorded. To record the desired time interval the trigger and pre-trigger parameters have to be adjusted in an appropriate way. The trigger describes a signal condition, to which the recording window will be aligned. The trigger consists of two auxiliary variables »A« and »B«, each an AND term of bit conditions. Possible bit conditions are the values »0« and »1« or the rising or the falling edge. The whole trigger can be the term »A=1«, »B=1«, »A∨B=1« etc. In the following example the trigger condition is »A=1«, where »A« is the AND term of the conditions »1 at input 0« and »0 at input 1 to 3« (for more Details refer to the short reference of the USB-LOGI-500 at the web site[4]):

```
 <trigger when="A">
  <A>
   <ch when="high">0</ch>
   <ch when="low" >1</ch>
   <ch when="low" >2</ch>
   <ch when="low" >3</ch>
  </A>
 </trigger>
</la>
```

The pre-trigger describes the fraction of the waveform displayed before the input signal matches the trigger condition (figure 5). Valid values are 1 to 7 for 1/8 to 7/8 of the displayed time before the trigger event. In the example it is set to »1« for 1/8:

```
 <pretrigger>1</pretrigger>
```

After starting the logic analyzer waits until it has recorded enough pre-trigger values. Than it continues filling the recording memory circularly until the signal matches the trigger condition. Finaly it records the required post trigger values and returns with the recorded data (see next subsection).

The signal definition defines the names and channel numbers of the signals to be recorded. Signal vectors combine multiple channels, as in the following the signal vector »y« the channels »0« to »3«:

```
 <signals>
  <signal name="Takt">
   <ch>2</ch>
  </signal>
  <signal name="y">
   <ch>0</ch>
   <ch>1</ch>
   <ch>2</ch>
   <ch>3</ch>
  </signal>
 </signals>
```

The channel numbers are printed on the housing of the logic analyzer and on the insulating tubes of the wires.

[4] still to be translated into English

2.3.2 Recording and displaying

The configuration file of the subsection before will be unpacked from the zip-file in the directory »Aufg2/LA« and is named »ConfigLA_Aufg2.xml«. For the experiment

- start a terminal
- change to this directory
- select on the test board by »SW0=1« the fast clock and
- start recording with

```
usb-logi ConfigLA_Aufg2.xml
```

The command creates after finishing recording a lxt- and a sav-file and starts GRKWAVE with both files to display the recorded waveform. Figure 5 shows the result with the described settings. To repeat the recording with the slow clock switch to »SW0=0«, reduce the sample rate to 100 per second and start recording again.

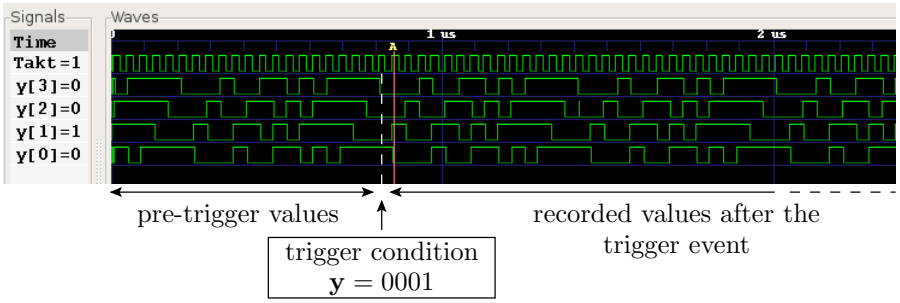


Figure 5: Displayed waveforms with the fast clock and the described settings

3 Chip-Scope

Alternatively to the external logic analyzer the logic analyzer also can be programmed into the FPGA. To generate a logic analyzer circuit the design system »ISE« has a circuit generator asking for a parameter description and producing all necessary design files. In the following example the recording clock of the integrated logic analyzer (ILA) should be the 50 MHz input clock »GCLK«. The logic analyzer should have five date inputs to record the state $\mathbf{z}$ of the feedback shift register and the down-scaled clock T. The sampling point should be the rising edge of »GCLK«. All five signal bits should be used for trigger. The simplest trigger condition – matching with given values – is sufficient (figure 6). The integrated logic analyzer is controlled via the programming cable by the program »Chip-Scope« running on the PC.

To configure the integrated logic analyzer in »ISE« in »Sources for Implementation« a new design object of the type »Chip-Scope Configuration file« has to be created:

- »New Source« ▷ file name: »Chip-Scope«, Source Type: »Chip Scope Definition and Connection File«
- »Next« ▷ »associated to Gesamtschaltung« ▷ »Next« ▷ »Finish«

Open the new source »Chip-Scope.cdc« with a mouse click. In the window that opens in »Trigger Parameters« select for the number of inputs »5«, »Match Type Basic«, one »Match Unit«, no counter and no »Trigger Sequencer«. Left side, in the window »Core Utilization« the required hardware is displayed in terms of look-up tables, flip-flops and block RAMs (figure 7). Each

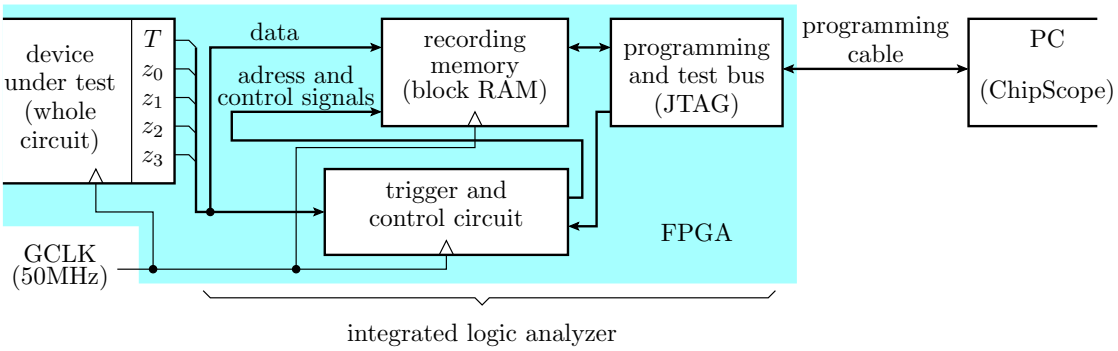


Figure 6: Test of the example circuit with the integrated logic analyzer

expansion of the trigger functionality costs additional hardware, that is not available for the device under test. In our example the circuits is almost empty, so resources for complex trigger condition as multiple match units etc. could be added, but will not be used.

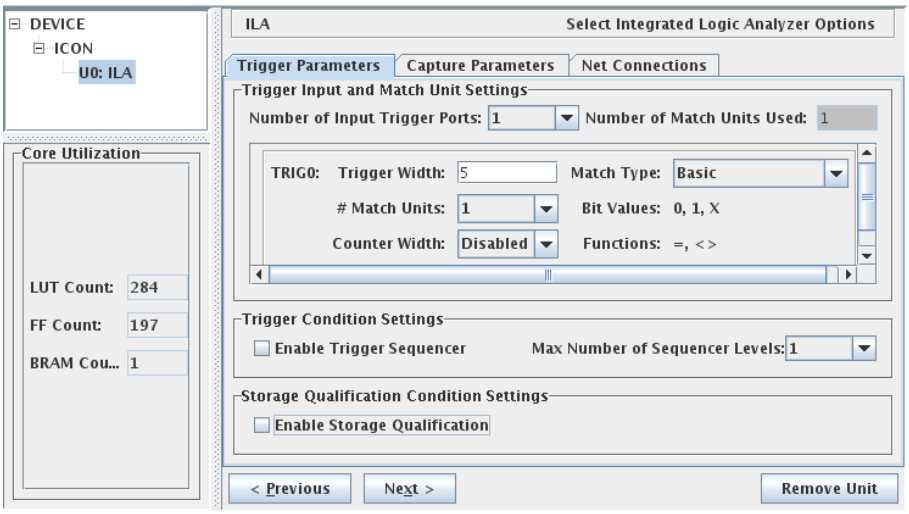


Figure 7: Trigger parameters for the integrated logic analyzer

In the menu »Capture Parameters« select for the depth of the recording memory 1024, for the data channels to be recorded »Data Same As Trigger« and for the record clock edge »Rising«. In the menu »Net Connections« open with »Modify Connections« the menu to assign signals of the device under test to inputs of the logic analyzer. The recording clock should be the 50MHz input clock. However, within the programmable circuit only the output signal of the automatically inserted clock driver »GCLK_BUFGP« is available (figure 8 a). To the data inputs has to be assigned, as shown in figure 6, the bisect clock T and the four state bits of the linear feedback shift register (figure 8 b).

Complete the editing of the chip scope configuration parameters:

- »OK« ▷ »Return to Project Navigator«

After this, select in the window »Sources for Implementation« the »Gesammtschaltung« and start in the tool window »Analyze Design Using Chips-Scope«. This command starts the synthesis of the complete circuit including the integrated logic analyzer, followed by placement, routing etc. up to the launch of the program »Chip-Scope«. This program downloads the configuration file

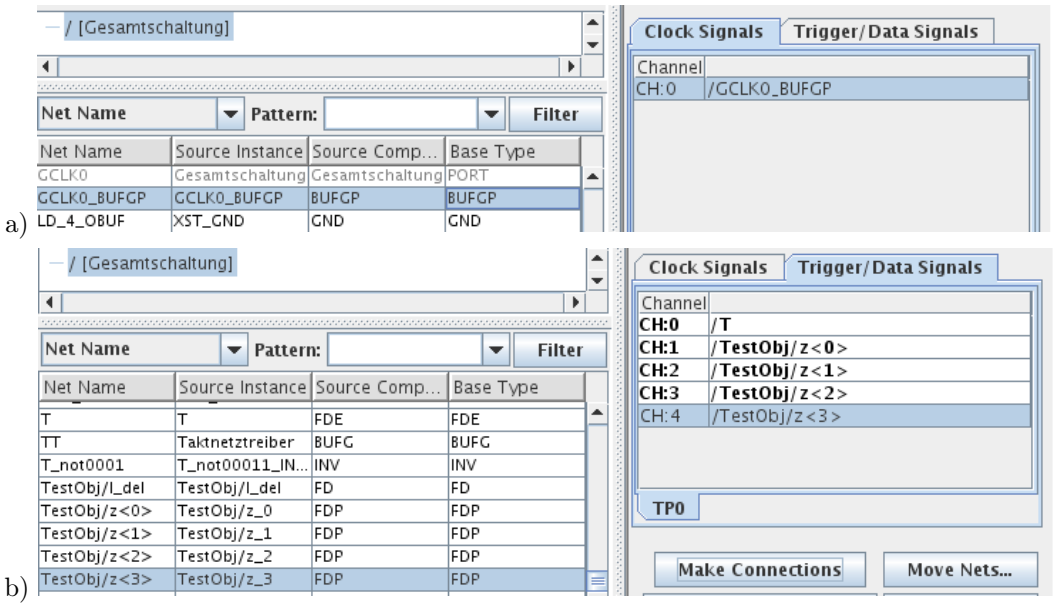


Figure 8: a) Assignment of the recording clock b) Assignment of the data signals to be recorded

into the programmable circuit and allows to select trigger values, start recording, get date back to the PC and display the recorded data.

After opening of the window »Chip-Scope Pro Analyzer«

- connect the test board (if it is still not done) to the power supply and the programming cable
- click on the chain symbol at the top left in the window to connect to the programming cable[5]

After connecting successfully, the detected circuits on the test bus are displayed (figure 9 top left). To the first circuit in the chain »Dev:0« the generated bit-file has to be assigned. Hereupon the circuit will be programmed and in the object space the integrated logic analyzer (ILA) will be displayed.

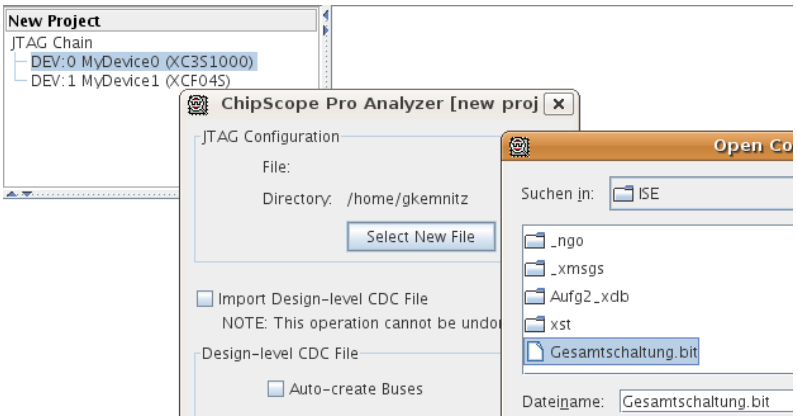


Figure 9: Adjustments in Chip-Scope

[5]The error message »Cable is locked ...« says that another program has locked the cable driver and still not released it. In the current version Chip-Scope has the bug not to delete the lock entry after closing. Until a better workaround is found in this situation only a restart of Linux solves the problem.

Open both sub-windows at the right side in figure 10 »Trigger Setup« and »Waveform« with a right mouse click and »Open ...« to it's name in the object window. Before starting recording, trigger and pre-trigger value (position) has to be selected In figure 10 the trigger event is the first occurrence of $T = 1$ and $\mathbf{z} =$ "0010" after start and recording of the 100 pre-trigger samples. Recording is started with a right click on the triangle in the top menu bar.

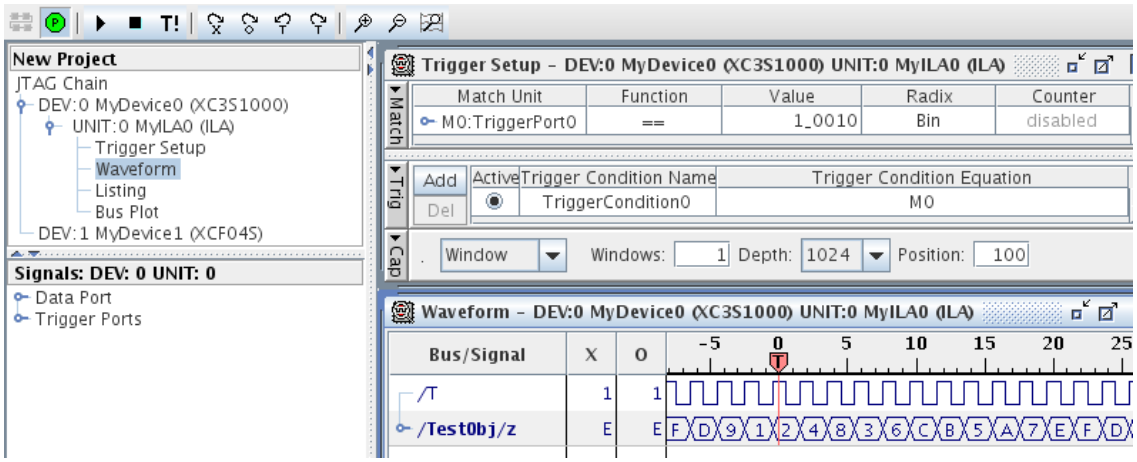


Figure 10: Recording adjustments und simulation result of the test with the high clock speed

4 Exercises

The aim of this lecture first of all is to learn to use the different techniques of testing and troubleshooting. For testing and troubleshooting are the most time consuming tasks in hardware design as also in software design.

1. Run the simulation, the test with the external logic analyzer and the test with the integrated logic analyzer with the original example circuit. Use the example files of the zip-file on the web-site.

2. Modify the example circuit to a 5 or 6 bit feedback shift register with self selected feedback points and draw this LFSR on the handout sheet for exercice 2.

3. Simulate the modified circuit. Determine the cycle length of the state sequence from the initial state until the initial state is reached again. Write the result on the handout sheet and keep the ghw- und sav-file for the final checking by the supervisor.

4. Synthesize the modified circuit and try to get the same results with the logic analyzer as with the simulation. For this also the constraint file for synthesis and the configuration file for the logic analyzer has to be adapted. Keep also this ghw- und sav-file for the final checking.

5. Adapt the integrated logic analyzer to the modified circuit and repeat the test with it. Write the modifications on the handout sheet and keep a screen shoot of the trigger setup and the recorded waveform for the final checking.

Suggestion (not obligatory):

- Do additional experiments with different trigger and pre-trigger values both with the external and the integrated logic analyzer.

- Check as in the first exercise with

– »Synthesis XST« ▷ »View RTL Schematic«

the synthesis result and with

– »Place & Route« ▷ »Analyze Timing / Floorplan Design ...«

the automatic generated placement.

- Run a post-route simulation with the modified feedback shift register and compare the calculated output waveforms with those recorded by the external logic analyzer.

5 Questions for self-monitoring

- Which initial states are allowed for a primitive feedback shift register so that it cycles after initialization through $2^r - 1$ states?
- What frequency has a clock with a period of 20 ns?
- The external logic analyzer records 600 million and the integrated logic analyzer 50 million samples per second. How often each period of the 25 MHz clock is sampled in both cases?