The LSST Sequencer

Introduction

The aim of this document is the description of the functionalities of the sequencer designed to drive LSST CCDs. This system can generate all the signals needed to acquire data from CCDs that are: parallel clocks, serial clocks, ASPIC signals and the ADC conversion start. Some other spare signals are also available. The architecture described below is the first version realized and has to be intended as a starting point to design the final sequencer for LSST.

The Sequencer Architecture

The sequencer is divided in 16 subroutines. Each subroutine generates a synchronous sequence of signals. The user, by means of parameters, defines that sequence. The execution order of different subroutines is written in a memory called stack register.

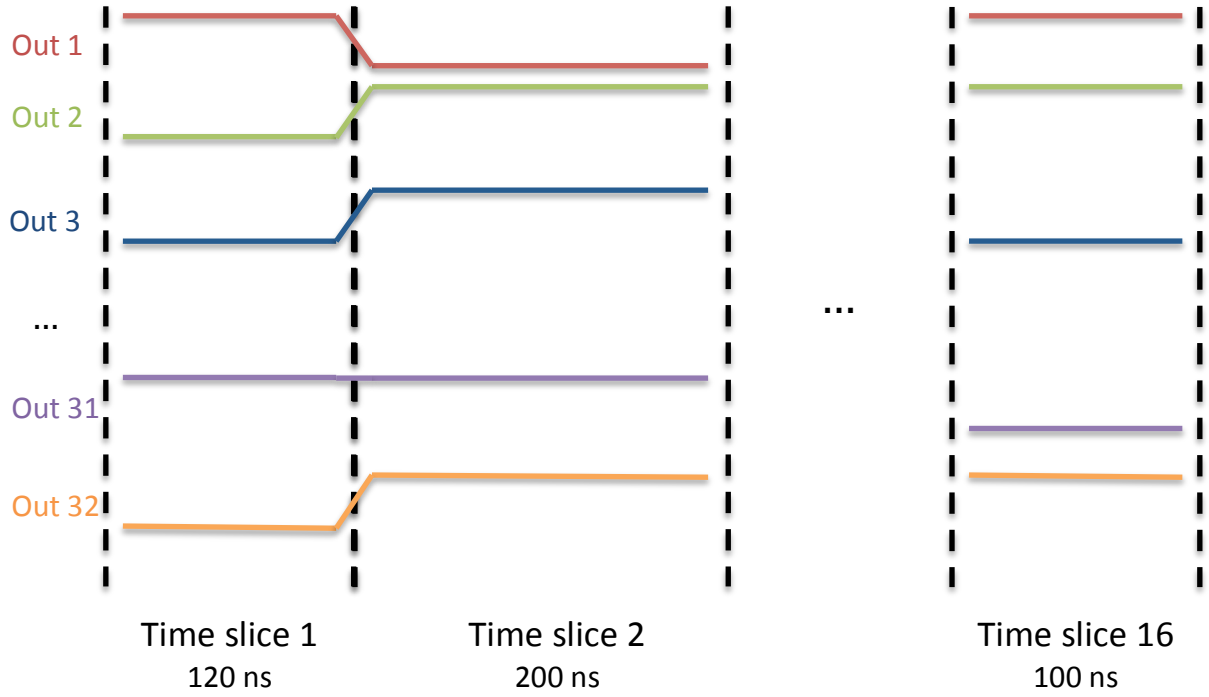
The Subroutine

The subroutine is the basic element of the sequencer and is set by the user to generate a time sequence of 32 output signals. Each subroutine is divided in 16 time slices. Each time slice is defined as a period of time where the outputs have the same value; therefore it is characterized by the state of each output (1 or 0) and by the time interval. An outputs transition is done during the transition between two time slices.

To set a single time slice two configuration words are needed: a 32 bits word for the outputs and a 16 bits word for the time interval.

A subroutine starts executing the first time slice and goes on sequentially until either the 16^{th} time slice is executed or when a time length value set at 0 is found. This means that also a subset of time slices is configurable. The time length is expressed in units of 10ns. This gives a resolution of 10ns for the outputs transitions.

The example below shows how a subroutine works.



In the table below the time slice parameters are described.

	Time Slice 1	Time Slice 2	...	Time Slice 16
Output #1	1	0		1
Output #2	0	1		1
Output #3	0	1		0
...	...	...	...	...
Output #31	1	1		0
Output #32	0	1		1
Time length	12 x 10ns	20 x 10ns		10 x 10ns

The first subroutine is a special subroutine since is composed only by one time slice and is used to set outputs default values. When the system is in a non-operation mode the output values are set as described in this subroutine.

The Stack Register

As mentioned, the subroutines execution order is defined in the stack register that is a memory of 65k 32 bits words. At each address the subroutine to execute and the number of times the same subroutine has to be repeated are written. When the sequence starts the system launch the first subroutine for the required number of times. At the end jumps to the next address and perform the next subroutine.

An “infinite loop” option is also available. Activating a special flag the system is forced to perform the same subroutine until the loop is stopped. To exit from the infinite loop condition two different options are available:

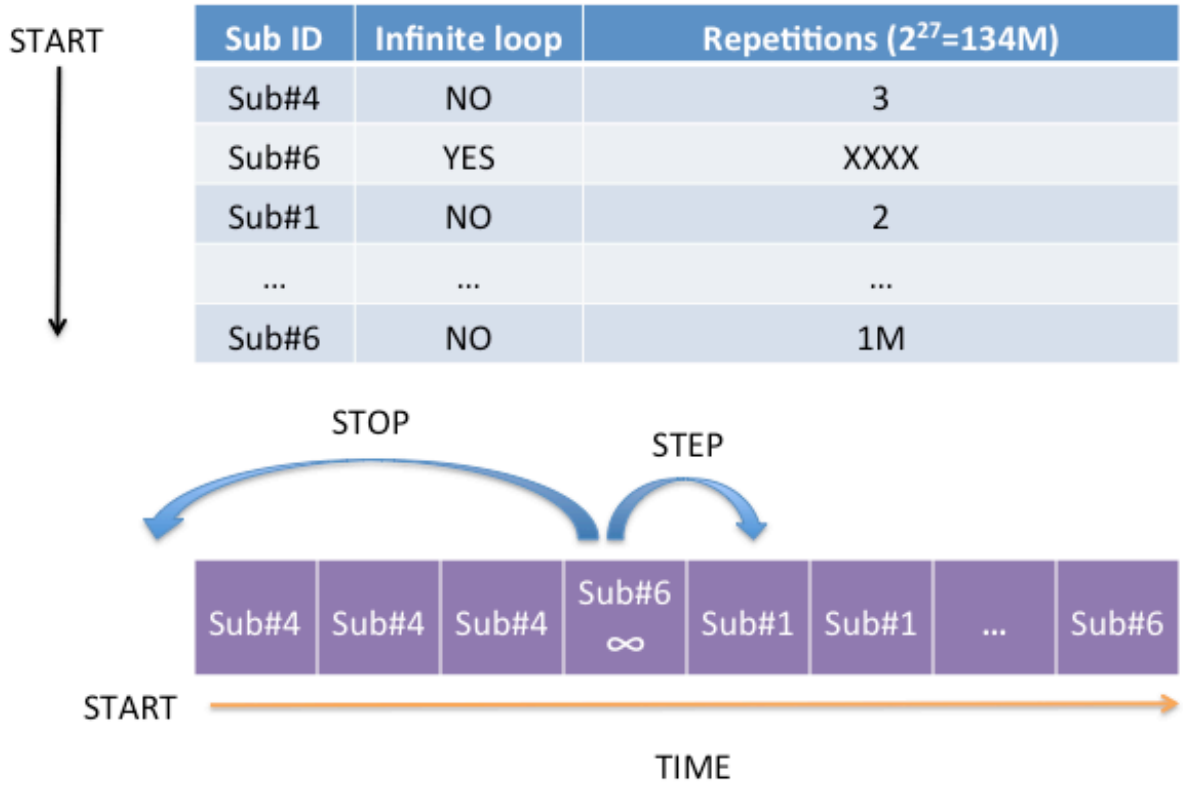
- STEP command;
- STOP command;

With the STEP command the system exit from the infinite loop and jump to the next stack register’s address executing the instruction contained in it. With the STOP command the time sequence ends.

Another parameter to be set is the “last address value”. This represent the last stack register’s address to be executed.

Once the sequencer is configured, the system is ready to start. When the START command sent, the sequencer read the parameters contained at the stack register’s address 0 and execute the corresponding subroutine for as many times as indicated in the stack. At the end jumps to the address number 2 and launch the corresponding subroutine. The process goes on sequentially trough all addresses until the “last address value” is reached.

The figure below shows an example of the timing sequence.



The written configuration is valid until a reset command is sent. This means that, after the configuration phase, the system performs the same sequence each time a START command is sent.

The sequencer's main characteristics are summarized below:

- Number of Subroutines: 3 + default state;
- Maximum number of time slices per subroutine: 16;
- Time resolution: 10ns;
- Maximum time slice length: 2^{16} x 10ns = 655µs;
- Stack register length: 256 words;
- Maximum number of repetitions per subroutine: 2^{26} = 67M;
- "Infinite loop" execution;
- STOP & STEP commands.

Programming the sequencer using the LSST's DAQ

When the LSST's DAQ is used all the configuration parameters are sent using that system. From the user point of view the interface consist of an address and a data filed; for this reason each kind of sequencer parameter is identified by an address. The associated data is the value that will be written. In other words, to write a value for a specific parameter, the user will write that value into a specific address.

In the following table all the address and data needed to configure the sequencer are reported:

Address value	Data value	action
0x100000 "ST"	0x"oooooooo"	Set the outputs values "oooooooo" for the time slice "T" of the subroutine "S"
0x100000 "ST"	0x0000"tttt"	Set the time value " tttt" for the time slice "T" of the subroutine "S". The time is in unit of 10 ns
0x300000"aa"	0x"rrrrrrrr"	Write in the Stack memory at the address "aa" the word "rrrrrrrr" (see below for details)
0x40000000	0x"000000"mm"	Write "mm" as the last stack max address used
0x50000000	0x00000000	Send the command "step" (infinite loop mode)

0x60000000	0x00000000	Send the command “stop” (infinite loop mode)
0x00000009	0x00000004	Sequencer start

Data in the stack register are mapped in this way:

Bit 31 – 30: not used.

Bits 29-28: subroutine to execute.

Bit 27: infinite loop.

Bits 26 to 0: sub repetitions .

On the current version only 12 sequencer outputs are used. The current outputs mapping is this:

```
ADC_trigger          <= sequencer_outputs(0);
ASPIC_r_up           <= sequencer_outputs(1);
ASPIC_r_down         <= sequencer_outputs(2);
ASPIC_reset          <= sequencer_outputs(3);
ASPIC_clamp          <= not(sequencer_outputs(4)); the signal is inverted on BEB
CCD_ser_clk(0)       <= sequencer_outputs(5);
CCD_ser_clk(1)       <= '0';
CCD_ser_clk(2)       <= '0';
CCD_reset_gate       <= sequencer_outputs(6);
CCD_par_clk          <= sequencer_outputs(13 downto 10);
```