A General Purpose FPGA-based Programmable Digital Patter Generator

stefano russo
srusso@slac.stanford.edu

Summary

A pattern generator on a Field Programmable Gate Array (FPGA) is presented.
The proposed design feature **32 outputs** a **10 ns time resolution** and a **fully programmable structure** that allows the generation of patterns from 10 ns to minutes of length. The architecture looks like a simple processor hence the pattern is defined writing a program that resides inside the FPGA's memory.
The sequence **execution time is tightly controlled** to ensure a resolution of 10 ns over the entire sequence without glitches and latencies. The design is written in vhdl and **can be ported on all kinds of FPGA**.

Context

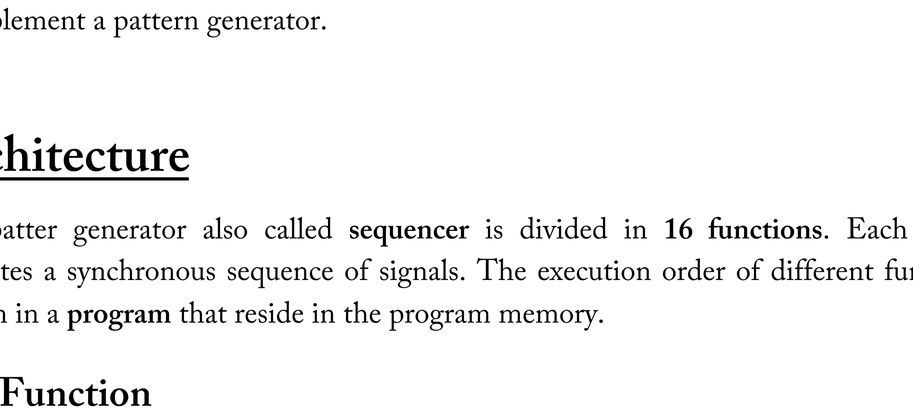
In many modern physics experiments a fully synchronous programmable pattern generator is often required. A typical case is represented by experiments that use Charge Coupled Device (CCD) as detectors where many signals need to be generated in a timing deterministic manner to read an image from this device.
Another field for a digital pattern generator is the design verification of electronics devices where it is used to stimulate the Device Under Test (DUT).
A commercial pattern generator usually address the needs described above, but in some specific cases, it is not practical or impossible to be used. An example is a telescope camera built in a tight environment where a commercial CCD controlled would not fit. Another case is a camera composed by hundreds of CCDs where the use of commercial controllers would be very expensive. A way to address these difficulties is to implement a pattern generator on an FPGA. This device is very flexible, not expensive, low power and digital designs implemented on it can achieve high clock frequencies. Hence it is a perfect platform to implement a pattern generator.

Architecture

The patter generator also called **sequencer** is divided in **16 functions**. Each function generates a synchronous sequence of signals. The execution order of different functions is written in a **program** that reside in the program memory.

The Function

The function is the sequencer basic element and is set by the user to generate a time sequence of 32 output signals. Each function is divided in a maximum of 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 binary state of each output (1 or 0) and by its 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. The smallest configurable function has to have 2 time slices.
A function starts executing the first time slice and goes on sequentially until either the 16th time slice is executed or a time length value set at 0 is found. The time length is expressed in units of 10ns. This gives a resolution of 10ns for the outputs transitions.



	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 Program

The functions' execution order is defined in program. The first 4 bits of each word in the program contains an Operation Code (**OP Code**) that defines the meaning of the rest of the word. 10 OP Codes are available:

- 0x1 – function execution;
- 0x2 – function execution with function pointer;
- 0x3 – function execution with repetitions pointer;
- 0x4 – function execution with function and repetitions pointers;
- 0x5 – jump to address;
- 0x6 – jump to address with address pointer;
- 0x7 – jump to address with repetitions pointer;
- 0x8 – jump to address with address and repetitions pointer;
- 0xE – subroutine trailer;
- 0xF – end of program;

Function Execution

When OP code 0x1 is called a function execution is requested and the program word is defined as follows:

31-28	27-24	23	22-0
OP code (0x1)	Function ID	Infinite Loop	Repetitions

- The **Function Identification Number** (ID) is the number used to identify the function to be called.
- An **Infinite Loop** option is available. The activation of this special flag force the system to execute the same function until the loop is stopped. To exit form the loop two different options are available: **STEP** command and **STOP** command.
- **Repetitions** number is the field where the number of times a function executions is specified. Writing a 0 makes the sequencer skipping the call.

Jump to Address

The OP code 0x5 is used to jump to a specific program memory address to executing a so-called **Subroutine**. A subroutine is a set instructions delimited by OP code 0xE. When the execution of the subroutine is finished the program restarts from the jump address. In this case the program word is defined as follows:

31-28	27-26	25-16	22-0
OP code (0x5)	Reserved MBZ	Address	Repetitions

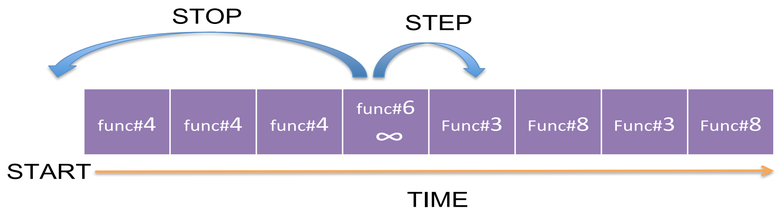
- The **Address** is filed specify the position where the subroutine starts.
- **Repetitions** number is the field where the number of times a subroutine executions is specified. Writing a 0 makes the sequencer skipping the call.

Pointers

In the program memory the concept of pointer is implemented. By means of pointers some parameters can be written in different memories and can be "recalled" from the program memory. In the program memory, instead of the parameter, a memory address is written and, during the program execution, this address is used to extract the corresponding value.

START

Add	OP Code	Func ID/Add	Infinite Loop	Rep
0	1	Func # 4	NO	3
1	1	Func # 6	YES	---
2	5	100	---	2
3	F		---	
..				
100	1	Func # 3	NO	1
101	1	Func # 8	NO	1
102	E		---	



Resources	Count
Slice LUT	2000
LUT used as Memory	1400
Slice Flip Flop	220

Implemented on a Xilinx Kitex 7 160T