Documentation for digital pattern generator-based imaging controller

Ryan Thomas

March 17, 2020

1 Introduction

This document describes the design and operation of the digital pattern generator (DPG)-based imaging controller used in the Kjærgaard lab. While both this controller and its predecessor are implemented using a field-programmable gate array (FPGA), namely the Xlinx Spartan 3AN development board, the DPG imaging controller has a much simpler FPGA architecture and is more easily customizable than the previous, bespoke version.

Whereas the previous imaging controller had many different modules, each of which controlled a different aspect of the timing sequence, the DPG solution has only two modules: the DPG itself and a module to generate triggers for the FlexDDS. The reason that these are separate is that we need millions of triggers for the FlexDDS at a fixed rate and duty cycle, and using a DPG to generate these would require an enormous amount of memory. It is easier to use a dedicated module for this purpose.

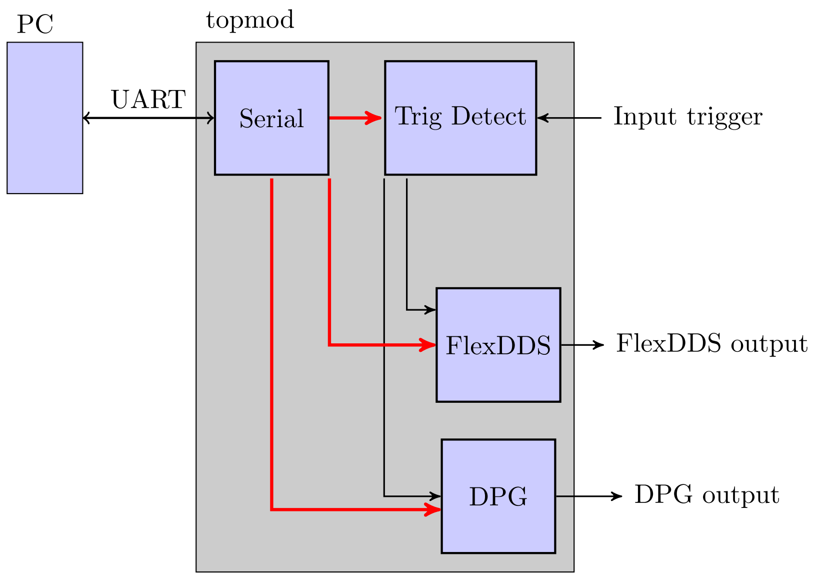

Figure 1: Diagram of the DPG-based imaging controller. A PC transmits parameters to the FlexDDS trigger generator and a digital pattern to the DPG. The sequence starts either when an input trigger is received or when a software trigger is issued by the host PC over serial. Thick red lines indicate the transmission of information to and from the serial controller.

2 Digital Pattern Generator (DPG)

A DPG is simple to understand – it is a device that stores a list of digital output patterns and the times at which they should be enacted. In the case of the current DPG, these are stored as a series of 40-bit (5 byte) instructions, where the most-significant byte (MSB) is the type of instruction and bytes 0 to 3 (for a total of 32 bits) are the *data*. There are currently three types of instructions:

1. MSB = 0x00 This instruction tells the DPG to wait for a time equal to *data* sample clock cycles.

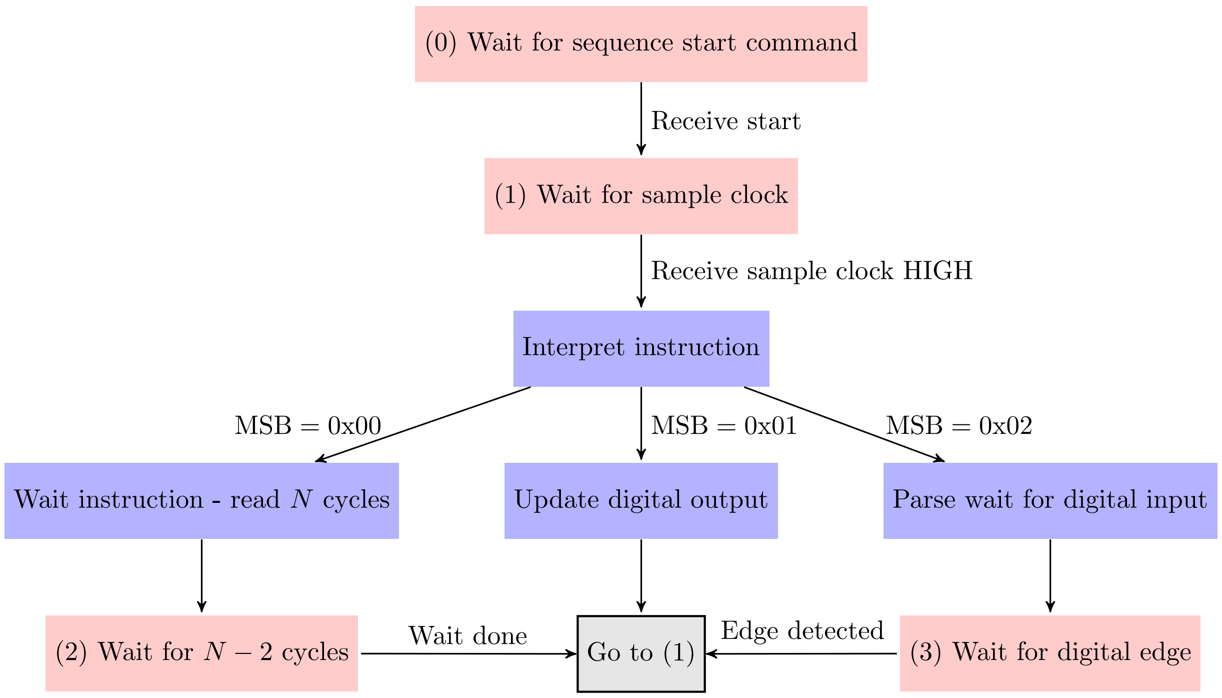
(0) Wait for sequence start command
Receive start
(1) Wait for sample clock
Receive sample clock HIGH
Interpret instruction
MSB = 0x00
MSB = 0x01
MSB = 0x02
Wait instruction - read N cycles
Update digital output
Parse wait for digital input
(2) Wait for $N-2$ cycles
Wait done
Go to (1)
Edge detected
(3) Wait for digital edge

Figure 2: Diagram of the finite-state machine (FSM) at the heart of the DPG. Red boxes indicate states of the state machine; blue boxes indicate conditional statements. New instructions are requested when a "start" command is received and also whenever a sample clock HIGH is detected. When all instructions are read, the FSM returns to state 1. If a sequence "stop" command is received, the controller returns to state 0.

2. MSB = 0x01 This instruction tells the DPG to output the pattern equal to *data*.
3. MSB = 0x02 This instruction tells the DPG to wait for a digital input event before continuing to the next instruction. Bits 0 to 3 of *data*, interpreted as an unsigned integer, indicate the digital input channel to look at, and bits 8 to 9 indicate the type of edge to look for.

At the heart of the DPG is a finite-state machine (FSM) that executes the reads, parses, and executes new instructions: a diagram is shown in Fig. 2. The idle state of the DPG is the "wait-for-start" state, where the FSM waits for the sequence start command from the PC over the UART (serial) connection. When it receives this trigger, it raises the `seqEnabled` signal and request a new instruction from the block memory that holds the instruction list. Raising `seqEnabled` enables the detection of input triggers, and when the input trigger is received the sample clock starts. The sample clock is set to 1/4 of the FPGA master clock (but with a duty cycle of 1/4), so with the current master clock frequency of 100 MHz the sample clock is 25 MHz. This means that new instructions are processed every 40 ns. The factor of 1/4 was chosen to ensure that there is sufficient time to read new instructions from the memory between successive sample clock edges.

When the FSM detects that the sample clock is high it issues a request for a new instruction at the same time that it parses the current one. When the current instruction number, given by the memory address, is less than the number of stored instructions the FSM also raises the `seqRunning` signal which tells the sample clock generator to continue running. If the current instruction number is equal to the maximum address, the FSM raises the `seqDone` signal which tells the controller on the next sample clock edge to read the first instruction again and return to the "wait-for-sample-clock" state (state 1 in Fig. 2). If at any time a "sequence stop" command is received, the FSM immediately returns to the idle state (state 0).

If the current instruction is a digital output instruction, the controller routes the *data* part of the instruction to the signal `dOutSig`, which is routed to the physical output when `seqRunning` is high. If `seqRunning` is low, then the digital output is routed from the signal `dOutManual` which is the manually set value. This signal can either be a default output state, or the user can set it as needed for testing.

If the current instruction is a wait instruction, the FSM reads the wait time from *data* as a delay of N sample clock cycles, where N is a 32-bit unsigned integer. The FSM proceeds to the "wait" state (state

2) and waits for $N-2$ cycles - the -2 is necessary to account for the sample period during which the instruction is parsed and the periods during which the counting is done. By choosing to wait for $N-2$, the delays between updates of the digital output are N sample clock cycles. Note that the controller cannot wait for 0 cycles, and asking it to wait for 1 cycle will result in it waiting for 2 cycles. In order to have a delay of 1 cycle between successive updates to the digital output, the user should have two successive digital output instructions rather than separating them by a wait time.

Finally, if the current instruction is a wait for input instruction, the FSM reads the input bit from bits 0 to 3 of *data* corresponding to an unsigned integer from 0 to 7. The edge type to look for is encoded in bits 8 to 9 with a "00" being a falling edge, "01" being either a falling or rising edge, and "10" being a rising edge. When the correct edge is detected, the FSM moves to state 1. An edge type of "11" is not used and will immediately move to state 1. Digital inputs are detected synchronously with the master clock, so any input signal must be high for at least one master clock period (10 ns).

2.1 Programming the DPG

The DPG is programmed using a relatively simple set of commands which are sent to the FPGA using a serial (UART) connection. Commands are interpreted least-significant bit (and byte) first. Some commands set the controller up to read the next command(s) as a numerical parameter, or, in the case of memory uploading, the next series of commands as writes to memory. A table of commands is given in Table 1

Command	Action	Subsequent command(s)
0xID_00_00_00	Sequence start	None
0xID_00_00_01	Sequence stop	None
0xID_00_00_02	Sends status information back to PC	None
0xID_00_00_03	Sends current value of `dOutManual` back to PC	None
0xID_01_00_00	Set value of `dOutManual`	Next 32-bit command is interpreted as `dOutManual`
0xID_02_*xx*_*xx*	Upload instructions to memory - see text	Next 0x00_00_*xx*_*xx* commands must be 40-bits - see text

Table 1: Command options for DPG. ID is the DPG's serial ID, currently set to 00.