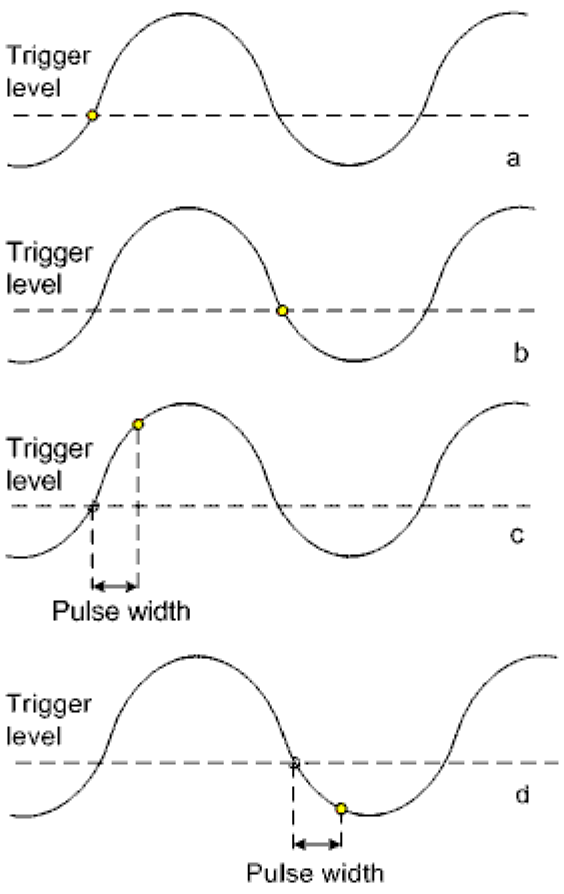
How to implement a digital oscilloscope in Structured ASIC fabric

By Mircea Moldovan, Dan Nicula, and Traian Tulbure, eASIC Corporation

Page 2 of 3
Programmable Logic DesignLine
(07/12/2006 7:07 PM EDT)

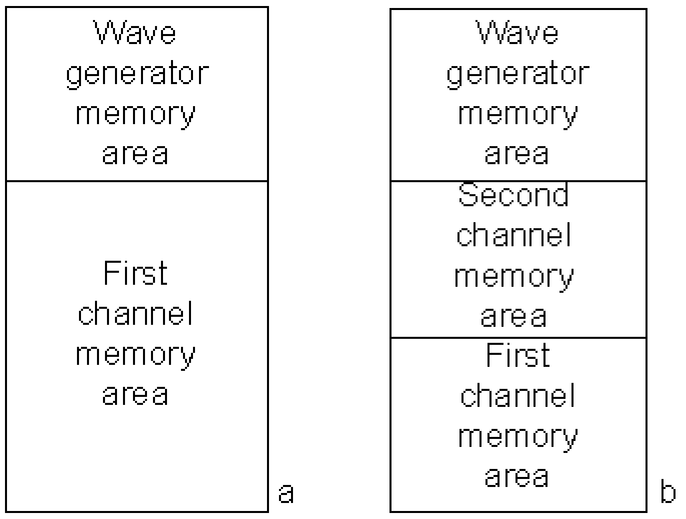
The eScope's acquisition part can work with four different trigger types as specified through 2-bit hfTriggerType_i input from the hfif module. These four types are as follows (and as illustrated in Fig 5:

- *Trigger on positive edge.*
- *Trigger on negative edge.*
- *Trigger on positive pulse.*
- *Trigger on negative pulse.*



5. Trigger methods: (a) positive edge, (b) negative edge, (c) positive pulse, and (d) negative pulse.

Valid samples are packed into 96-bits words and are written into memory using the OCP protocol. Each channel has a separate address counter. When a write request occurs, the proper counter is selected based on the acquisition mode. The entire memory is managed as illustrated in Fig 6.


Wave generator memory area
First channel memory area
a
Wave generator memory area
Second channel memory area
First channel memory area
b

6. Memory management: (a) single and interleaved sampling mode, and (b) double channel acquisition mode.

The wave generator path

The main on-chip memory is shared between acquisition and wave generator functions. User defined samples can be uploaded into the wave generator memory. Once this is done, the wave generator outputs this signal at a frequency selected by the user.

*The wave Generator output path is formed by two modules: **waveGen** and **dacOutput**. The role of the **waveGen** module is to generate OCP read commands to memory and transfer the received 96-bits data words to the **dacOutput** module. The transfers are done using OCP protocol synchronized to the scClk4x_i clock.*

*The **dacOutput** module receives data from the **waveGen** module and sends it to the DAC outputs. The **dacOutput** module synchronizes the data from the scClk4x_i clock to the wgClk_i using asynchronous FIFO memories. These FIFOs are 96-bit wide by 32-locations deep. The FIFO memory is created using the dual-port RAM memory that is available throughout the eASIC programmable ASIC logic fabric.*

Each 96-bit data word is sent out to the DAC on a 12-bit interface over (8 x hfSampleWidth_i) consecutive wgClk_i clock periods. Each sample is held on DAC input a for hfSampleWidth_i periods of . This mechanism implements a simple frequency divider for the waveform generator output. The frequency divider parameter (hfSampleWidth_i) is written in the eScope control registers.

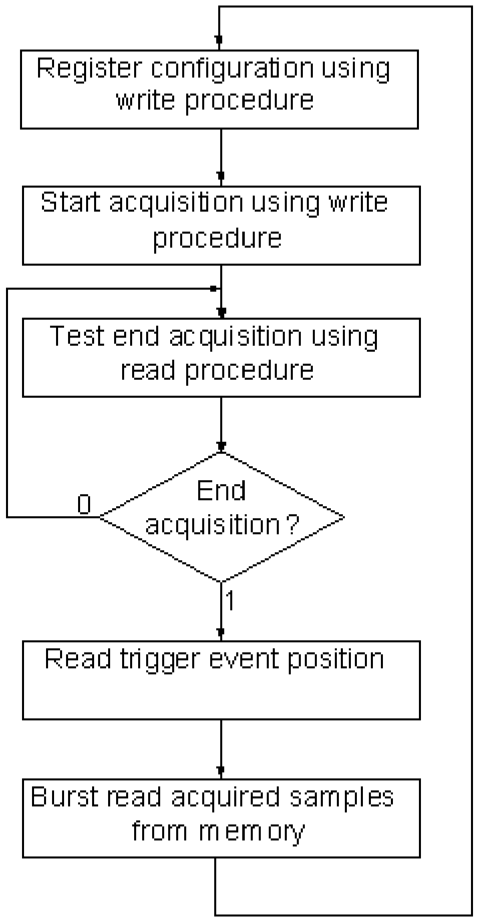
*Memory requests from the **waveGen** module are made by an embedded FSM sequencer that issues memory requests when space is available in the FIFO memories.*

The functional scenario

All accesses to eScope are made using a GUI in conjunction with the EZ-USB driver procedures, which are summarized as follows:

- *The **write** procedure configures hfif registers by specifying register address on address bus and also by specifying control signals.*
- *The **read** procedure gets data from hfif registers by specifying register address on address bus and also by specifying control signals.*
- *The **burst write** procedure stores samples in wave generator memory area by specifying samples on data bus and also by specifying control signals. hfif module manages the address.*
- *The **burst read** procedure gets samples from eScope acquisition memory area by specifying control signals. hfif module manages the address.*

Using these four procedures, the application environment is programmed to support acquisitions and wave generator operations. The acquisition path functional scenario is as shown in Fig 7.


Register configuration using write procedure
Start acquisition using write procedure
Test end acquisition using read procedure
End acquisition ?
0
1
Read trigger event position
Burst read acquired samples from memory

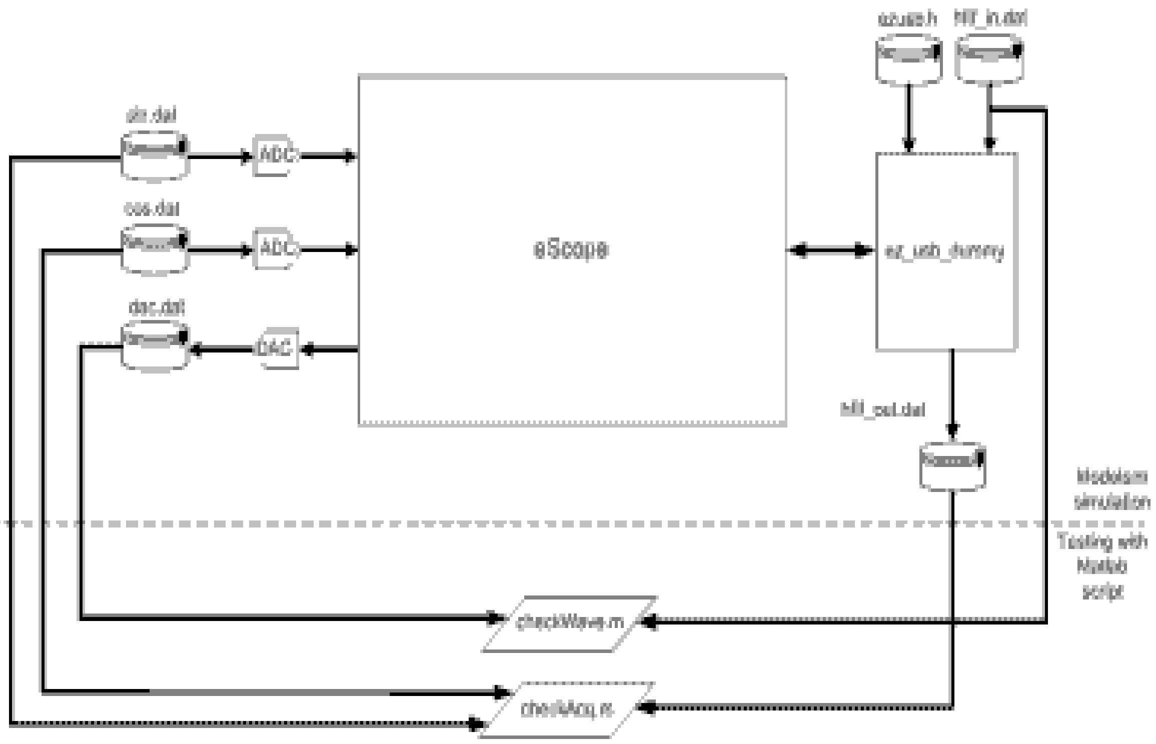
7. The acquisition path functional scenario.

By comparison, the wave generator path functional scenario is as follows:

- *Setup register configuration using write procedure.*
- *Burst write wave generator samples into memory.*
- *Start wave generator using write procedure.*
- *Run.*

The simulation environment

The simulation environment is as illustrated in Fig 8. The simulation bench contains an eScope instance, which includes logic (instantiated in eScopeLogic), a clock signals generator (in eScopeClk), pad instances (in eScopePads); ADC and DAC instances, an EZ-USB interface called ez_usb_dummy and two additional modules that generate reference clock (clk_gen) and reference reset (reset_gen) signals. An ezusb.h file is included which describes the functionality of EZ-USB interface. This file programs the design with different functional scenarios.



8. The eScope simulation environment.
(Click this image to view a larger, more detailed version)

The ADC modules used for simulation contains a memory for each channel from which samples are read. These memories are loaded from initialization files. Each sample is multiplied by a gain parameter to amplify the signal sent to the eScope inputs. The DAC module receives data from wave generator into a file for analysis.

*The ez_usb_dummy module implements four tasks ("procedures") for accessing eScope: **write**, **read**, **burst write** and **burst read** as previously described.*

Because eScope operation is highly configurable, it isn"t possible to simulate all operating conditions. Operating conditions were created by initializing each register with one of four possible values: low limit value, high limit value, and two intermediate values. A 15-bit number is created using a seeded random function generator. Groups of bits from this number are associated with registers their initialization value for a particular simulation

Sine, cosine, and complex waveforms were used as test signals during simulation, and the simulations were partitioned into 3 types:

1. *Tests which determine if a trigger event is correct using Verilog simulator assertions.*
2. *Tests which check if the acquired data matches the original "generated" data from the ADCs. The sampled signal is transferred over the USB interface to a virtual host and checked against the original data signal using a MATLAB script.*
3. *Tests which check that the wave generator outputs match the original data uploaded in wave generator memory, also using a MATLAB script.*