Sequential Memory Sequencer with Simple Loops and FIFO

Frank J. LaRosa, LaRosa Engineering, Inc.

Introduction

The FPGA used for this Sequencer was a Xilinx Spartan XC2S100-6PQ208, and the design was created in VHDL using the Xilinx ISE Web-Pack software. This Sequencer was designed as part of a program called Event-Driven CCD (EDCCD), thus the Xilinx project name is EDCCD, and the top-level design file is edccd_top.vhd.

The following design files and other referenced documents are available for download here:

EDCCD Xilinx Project: http://www.larosa-eng.com/ref/edccd2003.zip
Sequencer Specification: http://www.noqsi.com/images/SeqSpec.rtfd.pdf
Cypress CY7C1329 RAM Data Sheet:
http://www.larosa-eng.com/ref/CY7C1329.pdf

Design Organization

The Sequencer FPGA design is mostly contained in the file edccd_top.vhd, the only sub-module being the Command FIFO, which is an entity called fifo_16_255.vhd, which in turn instantiates a Xilinx Block RAM vcomponent called RAMB4_S16_S16. The bulk of the design is segregated into separate processes all in the top module.

Reference

The Sequence Spec should be reviewed before continuing to the design walk-through below. It is a short and sweet description of the functionality of this FPGA.

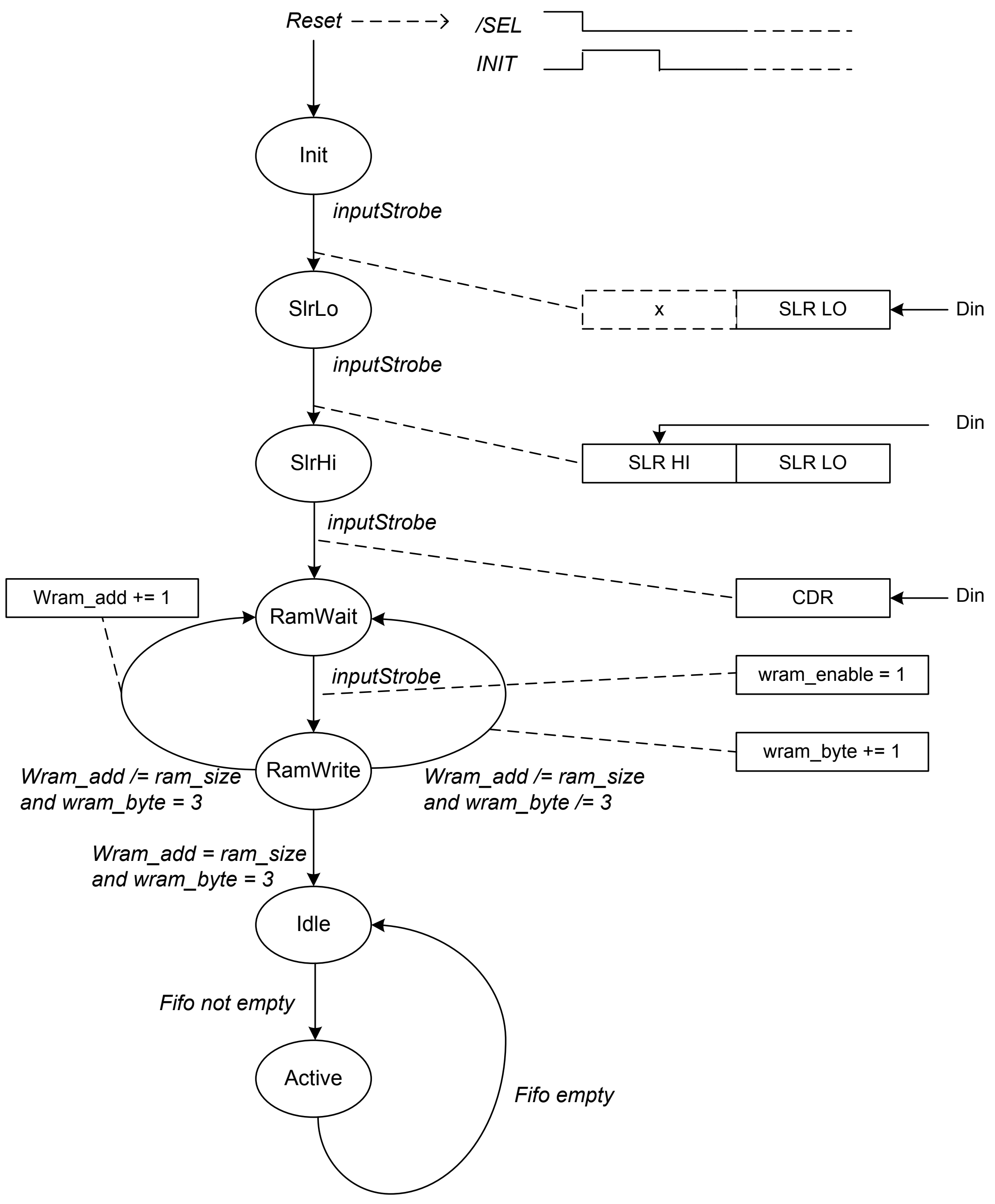
Design Walk-Through

The FPGA is implemented as a synchronous design, using a 30MHz clock, aptly named ***clock*** internally. Starting with edccd_top.vhd line 213, the process PORST generates an internal Power-On-Reset signal (***poReset_sync***) which is active when an external low-active reset signal is active. The de-assertion of ***poReset_sync*** is synchronized to ***clock***.

poReset_sync is only used by the process MASTERRST (line 228), which provides the final internal reset signal ***Reset***, which is used by all remaining processes in the design. After Power-On, ***Reset*** is also asserted by a specific control sequence from the Printer Port, namely, ***nSelect*** and ***nInit*** both asserted (low) simultaneously.

The process PPORT (line 241) - and the two concurrent lines after it - generate leading-edge and trail-edge pulses from the Printer Port signal ***nStrobe***. ***nStrobe*** can be more than 50uS long, as compared to the 33.33nS internal ***clock*** period, and all internal control signals must be one clock period in length, and synchronized to ***clock***.

Before describing the rest of the logic, and overview of the main state machine, implemented in the process (MAIN_SM (line 341), is necessary. Below is the state diagram:



On Power-On Reset or a reset sequence from the Printer Port, the state machine enters the "Init" state. From this point on, the state machine expects a specific order of printer port WRITEs, as follows:

Sequence Length Register (SLR) LOW byte
SLR HIGH byte
Clock Divider Register (single byte)
256K (262,144) Memory bytes

Once the entire memory has been filled (the state machine counts the byte writes), the state machine enters the “Idle” state. In this state, any Printer Port WRITE is interpreted as a Command WRITE. A command is received as a pair of bytes, Low followed by High. The low byte is buffered in a register, and then the entire word is written to the Command FIFO when the High byte is received (described later in a separate process). Referring back to the state diagram, when the Command FIFO is not empty, the MAIN_SM enters the “Active” state. As the Sequencer pulls commands from the FIFO and generates CCD output timing signals, if the FIFO empties out the MAIN_SM will revert to the “Idle” state and wait for new commands to be received.

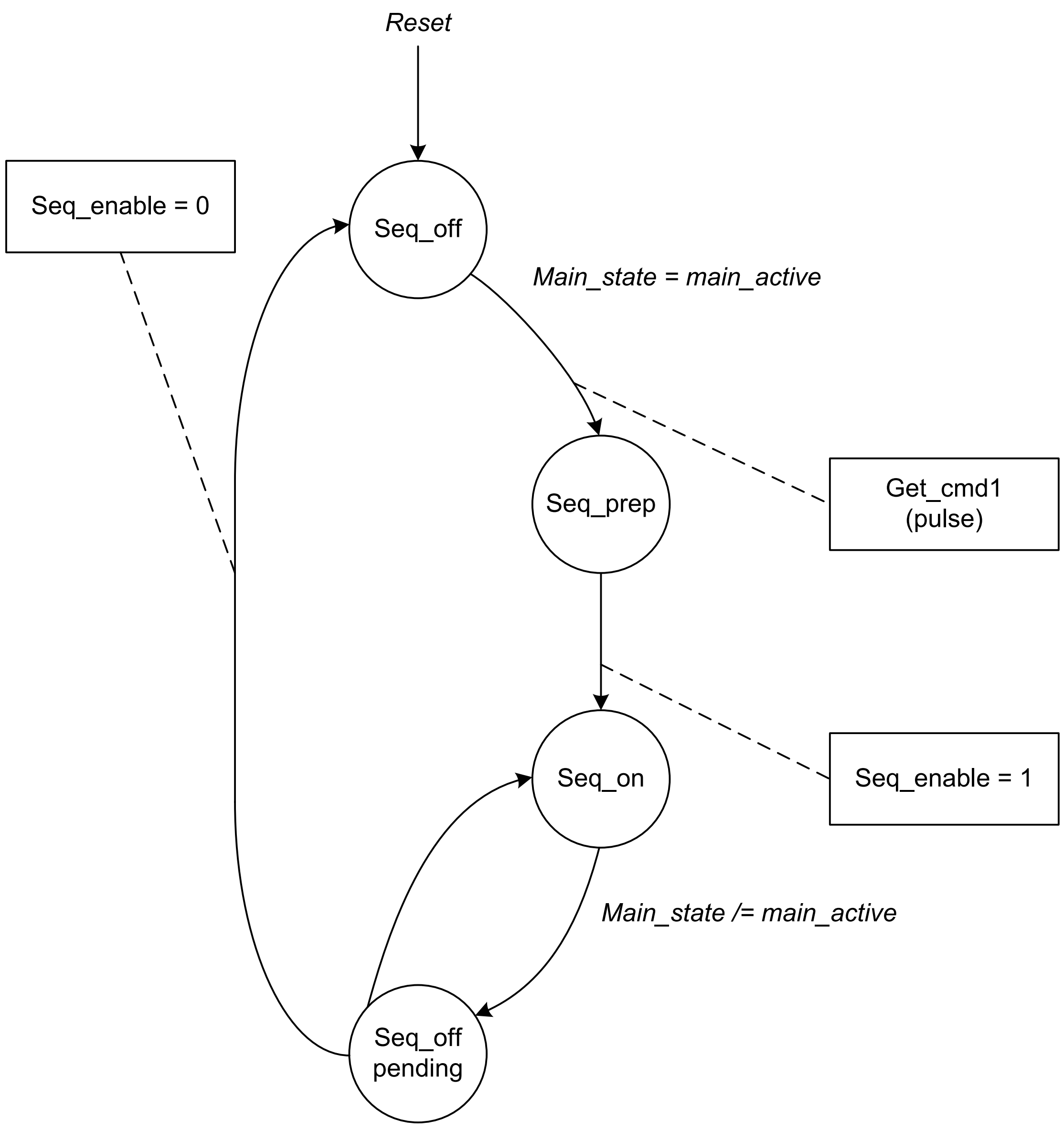
There are three outputs connected to the Printer Port: ***Busy***, ***nAck*** and ***nError***. ***nError*** is currently unused, and is set permanently to ‘1’ (de-asserted). ***nAck*** is generated by the process ACKSTATE (line 258), which, as the name implies, is another state machine. ACKSTATE (starting in the state “aIdle”) is triggered by the nStrobe trailing-edge pulse ***inputStrobe2***. The first thing it does is wait 533nS (the constant ACK1 - line 91 - is set to the value 15, which results in a count of 16 ***clock*** periods). This is a simple guarantee of polite handshaking between host and logic. Next, the state machine issues in internal signal called ***fifo_busy***, which is de-asserted after another 533nS unless the FIFO turns out to be FULL after the current transaction (note that this state machine does not actually know the state of MAIN_SM, so it doesn’t know whether Memory or FIFO is being written to, except when the ***fifo_full*** flag turns out to be asserted (see line 296).

If the FIFO is not FULL (***fifo_full*** = ‘0’), then ACKSTATE returns to the “aIdle” state. If the FIFO is FULL, the state machine will park in “CheckFifo” until the FIFO is no longer full (the Sequencer logic has to pop at least one command from the FIFO for this to happen).

The remaining Printer Port handshake signal is ***Busy***, which is managed in BUSYPROC (line 320). ***Busy*** is generated under two conditions: 1) held true for 16 clock periods after a reset, and 2) any time the FIFO is FULL, more specifically, the signal from ACKSTATE ***fifo_busy***.

The process CMD_RCV (line 457) handles the Low Command byte buffering, and the 16-bit write to the Command FIFO.

The Sequencer logic comprises the rest of the design file, starting with the process SEQEN (line 493). For this state machine to enable the Sequencer, MAIN_SM must be in the “Active” state, which can only occur after a full initialization and at least one Command written to the Command FIFO. When the “Active” state is detected, SEQEN exits its default “seq_off” state, issues an internal pulse (***get_cmd1***) to read the currently available command from the FIFO, and then enters the “seq_on” state. Below is the state diagram for SEQEN:



When the "Active" state de-asserts (MAIN_SM will be in "Idle") then SEQ_EN goes into a "seq_off_pending" state to wait for the completion of the current Sequencer pattern, before returning to the "seq_off" state.

The process CLKDIV (line 541) uses the CDR register to control the finest timing increment in the Sequencer.

The process OUTSTROBE (line 595) in conjunction with OSTROBE (line 634) create the output latching timing, the Active signal and the out_strobe signal.

SEQ (line 654) is the Sequencer state machine, and manages the internal counters with the help of the process TC (line 691).

DOUTLATCH (line 738) latches the 32-bit output.

Test Bench Simulation and Waveforms

The test bench for this design is edccd_TB.vhd. Since simulating 262,144 RAM writes consumes unnecessary simulation time, a way of limiting the RAM depth has been provided in the top-level file edccd_top.vhd. Lines 88 through 98 contain constants declarations that define printer port timing and RAM depth. This set of lines should be uncommented to implement the device (synthesis, place&route, etc.):

```
-- UNCOMMENT THESE BEFORE IMPLEMENTING DEVICE:
constant RAMSIZE : std_logic_vector(15 downto 0) := X"ffff";
-- Times for the ACK State Machine:
constant ACK1 : std_logic_vector := "01111";    -- .533 uS
constant ACK2 : std_logic_vector := "11111";    -- 1.07 uS
```

This set of lines should be uncommented (and the lines above commented) in order to run the simulation:

```
-- Test Values. Comment these out before implementing device:
constant RAMSIZE : std_logic_vector(15 downto 0) := X"007f";
-- Times for the ACK State Machine:
constant ACK1 : std_logic_vector := "00001";    --  66 nS
constant ACK2 : std_logic_vector := "00011";    -- 133 nS
```

Note that the second set of constants declarations sets the RAM depth to 32. referring to the test bench, the 32 RAM values are defined as bytes in a table starting at line 102. The test bench performs the initialization sequence and fills the shortened RAM with 32 words, then fills the Command FIFO. As soon as the first command goers in, the Sequencer starts. The test bench is scripted to have more than the 255 commands the FIFO can hold. This serves to test the fifo_full and busy flags.

The simulation was done using ModelSim Xilinx Edition, supplied with the ISE WebPack tools. Once the simulation is started, the file edccd_behave_wave.do may be loaded into the waveform window.

Three wave form plots have been provided to illustrate the timing:

http://www.larosa-eng.com/docs/edccd_init_timing.jpg
showing the INIT sequence and the start of the RAM loading.

http://www.larosa-eng.com/docs/edccd_cmd_timing.jpg
showing the first two Command FIFO loads, and the Sequencer logic starting up as soon as the first Command has been loaded.

http://www.larosa-eng.com/docs/edccd_output_timing.jpg
showing the walking one's pattern in the 8 least significant bits, along with the output CLK and STRB signals.

Frank J. LaRosa, LaRosa Engineering, Inc.
frank@larosa-eng.com