Design: Command Decoder for LumiMulti ADC

Introduction:

The aim of this work is to implement a slow interface for LumiMulti ADC chip using a serial SPI like interface. The LumiMulti ADC is a slave, that works in SPI mode '0'(active posedge). The serial command protocol defined for the LumiMulti ADC is shown in the figure below.

1	0	1	0	1	1	C1	C0	D7	D6	D5	D4	D3	D2	D1	D0

Figure 1. Serial command/data format.

The hierarchy of the serial command protocol is as follows:

(1) Header: 101011
(2) Command set: C1-C0
(3) Data set: D7-D0

The MSB is transmitted first. Field header is 6 bit long and one should always transmit 101011. Command set is 2 bit long and hence there are four possible commands: **config**, **active-adc**, **dac0** and **dac1** with codes given in table 1. Data set are always 9-bit long and for commands with shorter data the trailing bits are set to zero.

Specification:

The data interpretation for the different commands are summarized in table1. Also some description of the command set is described in this section.

Name	***Code***	***Data interpretation***
config	'00'	Mode(2b), Test-ADC(3b), Low Power(2b)
active-adc	'01'	Select-ADC (8b)
dac0	'10'	Dac0 value (9b)
dac1	'11'	Dac1 value (9b)

Table 1. Command set.

The **config** command is used to operate in LumiMulti ADC in three different modes: Parallel, Serial and Test. Modes of operation are chosen by setting appropriate value in the field mode of config command (refer table 2).

Mode	***Field***
Parallel	'00'
Test	'01'
Serial	'10'

Table 2. config command set.

Parallel Mode: This is the base mode of operation, after every hard reset mode is set to parallel. In this mode each ADC is connected to on LVDS output and sends data serially. Internal ADC clock is 10 times slower than the input clock signal. During transmission MSB are sent out first.

Serial Mode: In this mode all ADC sends data through one serial LVDS output. Internal ADC clock is 80 times slower than input clock signal. The transmission pattern is, MSB of ADC7, ADC6....ADC0. They are all bit number 9 of those ADC's samples. Next in sequence are bit 8 and so on from all ADCs.

Test Mode: In test mode only one ADC reads out data. Consecutive samples are presented on LVDS output 9 to 0 after each clock cycle so the internal ADC clock is exactly the same as input clock signal.

Low Power: Selects power mode of LVDS ports and internal buffer controls. The low power field has only two bits: first bit controls the LVDS power mode and the second bit controls both internal buffers.

Active-adc: The active-adc command permits one choose which ADC is to be turned ON and which to be turned OFF. When an ADC is OFF, the power in analog part is OFF and the clock in the correction logic is stopped. Active-ADC command has 8 bits of data. First bit if set to 1/0 turns ADC7 ON/OFF, similarly second bit controls ADC6, and so on till ADC0.

Both **dac0/dac1** commands set the DACs in the analog part. DAC number 0 controls the biasing current in the main part of ADC while DAC1 is responsible for current in the sample and hold circuit.

Implementation:

Command decoder module was implemented in Verilog HDL. Only the top level design, with submodules and primary inputs-outputs are shown here for clarity.

Tools used: Cadence NC-Verilog.

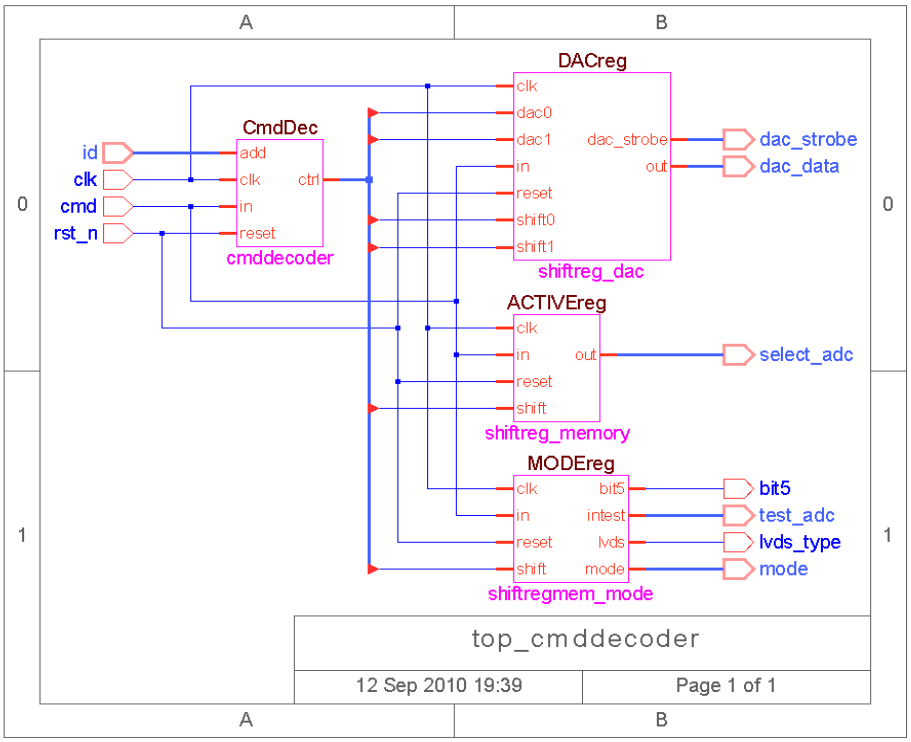

Figure 2. Top-level Module.

Simulation and Synthesis:

The design provided with necessary input test vectors has been simulated successfully, the simulation test results for one such scheme is presented below.

Tools used: Cadence NC-Sim, RTL compiler and SOC Encounter.

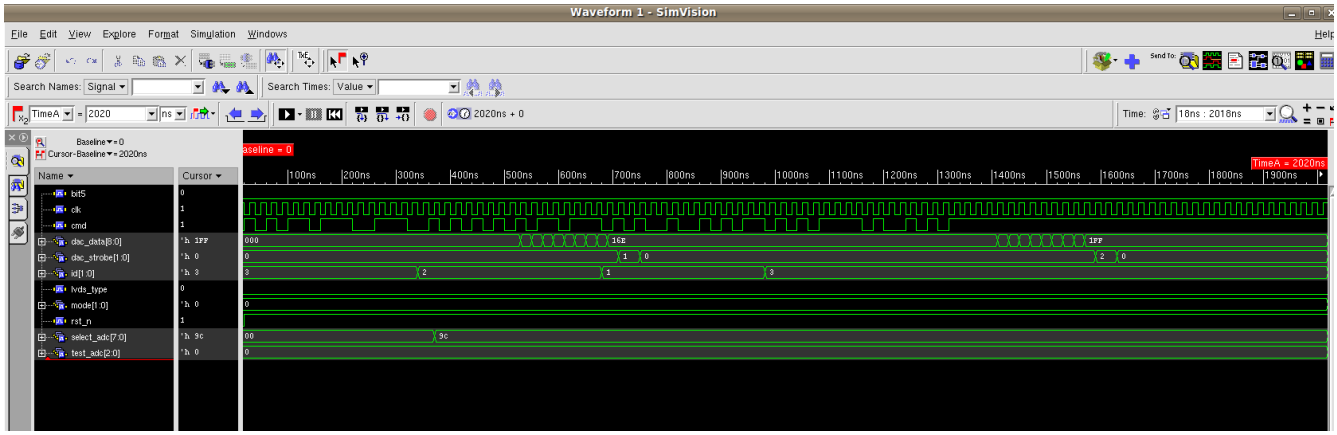

Figure 2. Simulation waveforms.

Command decoder module has been synthesized and gate level netlist generated in AMS 0.35u technology. Synthesis reports summary has been added here.

```
============================================================
  Generated by:           Encounter(R) RTL Compiler v07.20-s009_1
  Generated on:           Sep 13 2010  09:20:00 AM
  Module:                 top_cmddecoder
  Technology libraries:   c35_CORELIB 2.1
                          physical_cells
  Operating conditions:   _nominal_
  Interconnect mode:      ple2
  Area mode:              physical library
============================================================

  Gate  Instances    Area       Library
-----------------------------------------------
ADD21          1    145.600    c35_CORELIB
AOI210         8    582.400    c35_CORELIB
AOI2110        4    364.000    c35_CORELIB
AOI220         6    546.000    c35_CORELIB
AOI310         3    273.000    c35_CORELIB
CLKIN2        11    400.400    c35_CORELIB
CLKIN3         1     36.400    c35_CORELIB
DFC3           7   2165.800    c35_CORELIB
DFEC1         39  13486.200    c35_CORELIB
DFSC1          1    382.200    c35_CORELIB
DLQ3           1    182.000    c35_CORELIB
IMUX20         4    364.000    c35_CORELIB
INV1           2     72.800    c35_CORELIB
INV2           1     36.400    c35_CORELIB
INV3          12    436.800    c35_CORELIB
NAND20        24   1310.400    c35_CORELIB
NAND30         5    364.000    c35_CORELIB
NOR20         25   1365.000    c35_CORELIB
NOR30          4    291.200    c35_CORELIB
NOR31          2    145.600    c35_CORELIB
NOR40          3    218.400    c35_CORELIB
```

OAI210 11 800.800 c35_CORELIB
OAI2110 3 273.000 c35_CORELIB
OAI220 2 182.000 c35_CORELIB
OAI310 5 455.000 c35_CORELIB
---
total 185 24879.400

Type	Instances	Area	Area %
sequential	48	16216.200	65.2
inverter	27	982.800	4.0
logic	110	7680.400	30.9
total	185	24879.400	100.0

==========
Generated by: Encounter(R) RTL Compiler v07.20-s009_1
Generated on: Sep 13 2010 09:20:00 AM
Module: top_cmddecoder
Technology libraries: c35_CORELIB 2.1
physical_cells
Operating conditions: _nominal_
Interconnect mode: ple2
Area mode: physical library
==========

Instance	Leakage Cells	Power(nW)	Dynamic Power(nW)	Total Power(nW)
top_cmddecoder	185	22.908	518256.637	518279.545
CmdDec	126	7.396	319387.691	319395.086
ACTIVEreg	17	5.840	58753.496	58759.336
MODEreg	15	5.114	64519.164	64524.279
DACreg	27	4.558	34138.056	34142.614

Verification with Self-checking test-benches:

Here we perform a predictive analysis to ensure that the design when manufactured will perform the given I/O functions with certain accuracy. Detailed self-checking test-benches are written to perform the verification, along with code coverage tools. The commonly used code coverage analysis offers Code/Data coverage, FSM coverage and Functional coverage. If any coverage is found to be low, analysis of the unused code helps to determine additional tests needed.

Tools used: Cadence ICCR.

A simple top level FSM coverage result is shown below:

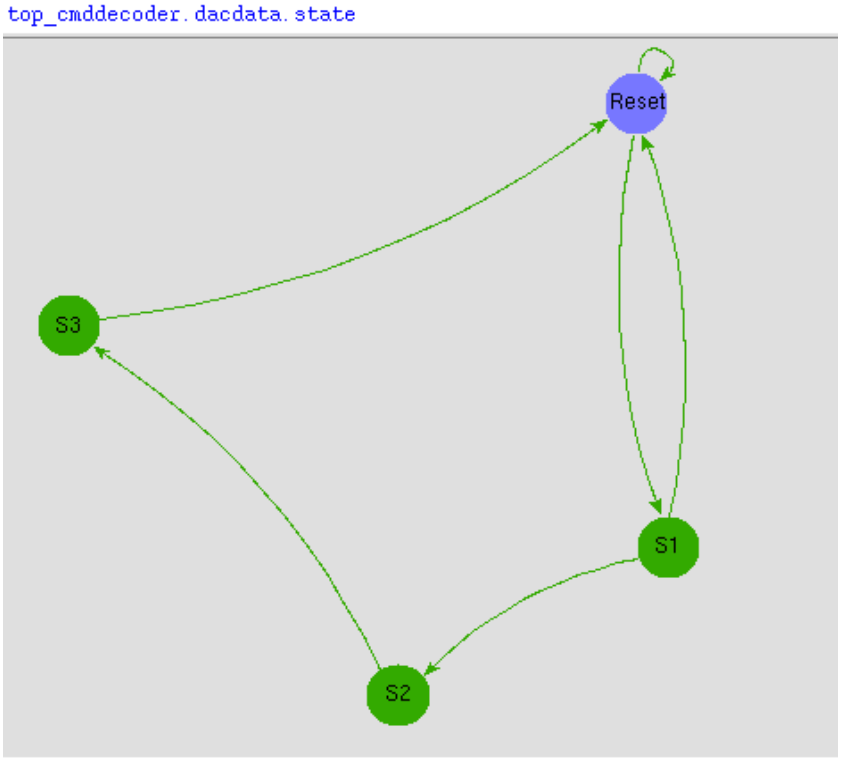

Figure 3. FSM coverage chart.

The simple state machine shown above has 100% coverage, as all the states has been visited during the exercise, and there are no unused states. Results of the code coverage analysis has been summarized below for the entire design in table 2.

Type	***Coverage***	***Passing Ratio***
Module/Unit	97.00%	398/410
Instance	97.00%	398/410
State	100.00%	49/49
Arc	97.00%	58/60

Table 2: Result-Code coverage analysis.