94x/92x Video Timing Detection Mechanism Analysis

Texas Instruments

6/12/19

Overview

- 94x Deserializers include a PATGEN BIST feature which may be utilized in the final production system to verify RGB data from upstream SER
- 94x and 92x deserializers include registers which can read back video timings information
- These two features may be used to detect and correct display issues during system init to mitigate potential system level problems
  - Ex. Excessive jitter/skew on startup
- Based on preliminary analysis, this feature may be useful in detecting a data reception issue at the remote serializer even where BIST and internal PATGEN do not show issues
  - Ex. Can be used to mitigate clock to data skew issues at the 947 OLDI input which cause link up but jittering screen or black screen symptoms
- Reading the detected resolution on the DES does not require serializer to take any action, and running the feature does not trigger any interruption to the display so it can be used in real time

Feature Usage and Limitations

- The PATGEN BIST feature can be used for RGB bitwise error detection with 94x DES
  - Can only be used when both SER and DES are operated in external timing modes
  - SER can activate PATGEN with external timing and select a static color pattern (no scrolling)
  - DES PATGEN must be programmed for external timing and the same color pattern as the SER
  - SER may also use no PATGEN but the source must provide the expected color pattern of the remote DES
  - PATGEN BIST Error register counts number of bit errors up to 0xFF once activated
  - RGB information can be used to validate end to end bitwise errors as long as timing information is correct
- Timing information can be used to detect sync issues
  - Active frame dimensions can be read back on 94x and 92x devices based on the received data

Registers (on DES)

0x68	**PGDBG**					
		7:4	PATGEN_DBG_SEL	RW	0000	Test Mux Select: This field selects the signals to be brought out on the test output bus as well as read in the PGTSTDAT register. See the Debug Monitor section of the Pattern Generator DDS for details.
	(94x Only)	3	PATGEN_BIST_EN	RW	0	Pattern Generator BIST Enable: Enables Pattern Generator in BIST mode. Pattern Generator will compare received video data with local generated pattern. Upstream device must be programmed to the same pattern.
		2	PATGEN_RAND	RW	0	Random Pattern Generation Select: 1: Output a pseudo-random pattern, overriding all other pattern selection. 0: Output a pattern as configured in Fixed or Auto-Scrolling Pattern Modes.
		1	PATGEN_DBG_FREERUN	RW	0	Test Mux Freerun: Enables continuous output of test mux data. If set to 0, data will be sampled and held following setting of the PATGEN_DBG_SAMPLE register bit. Freerun operation is most useful for viewing on external pins. Sample/Hold is most useful for reading through the PGTSTDAT register.
		0	PATGEN_DBG_SAMPLE	RW	0	Test Mux Sample/Hold: Enables sampling of the test mux data within its source clock domain. Guarantees valid data readback through the PGTSTDAT register. This bit is self-clearing

Registers (on DES)

0x69	**PGTSTDAT**					
	(94x Only)	7	PATGEN_BIST_ERR	R	0	Pattern Generator BIST Error Flag During Pattern Generator BIST mode, this bit indicates if the BIST engine has detected errors. If the BIST Error Count (available in the Pattern Generator indirect registers) is non-zero, this flag will be set.
		6	reserved	R	00	Reserved: Reads return 0, writes are ignored.
		5:0	PATGEN_TST_DATA	R	000000	Test Data: This field contains the Debug Monitor output. See the Debug Monitor section of the Pattern Generator DDS for details.

Registers (on 94x DES)

- In the PATGEN indirect registers at address 0x19:

Bit	Access	Field	Default (bin)	Description
7:0	R	PATGEN_BIST_ERRS	00000000	PATGEN BIST error count - Clear on read

Registers (on DES)

(For bits [7:4] in register 0x68)

Table 7. PATGEN_DBG_SEL Assignments

PATGEN_DBG_SEL	Source	Bit Assignments
0000	Timing Control Block (patgen_control_mux.v)	[5:0] = pat_ahw[5:0]
0001	Timing Control Block (patgen_control_mux.v)	[5:0] = pat_ahw[11:6]
0010	Timing Control Block (patgen_control_mux.v)	[5:0] = pat_avw[5:0]
0011	Timing Control Block (patgen_control_mux.v)	[5:0] = pat_avw[11:6]
0100	Timing Control Block (patgen_control_mux.v)	[5] = next_active_line [4] = pat_de [3] = pat_hs [2] = pat_vs [1] = start_frame [0] = start_line
0101	Timing Control Block (hdcp_vsync_detect.v)	[5] = no_vsync_det [4] = vsync_det [3] = vsync_error [2:0] = vsync_state
0110	Timing Control Block (patgen_control_det.v)	[5] = Reserved [4] = long_blank_det [3] = vs_active [2] = vs_asserted [1] = vs_det_complete [0] = vs_inact_state

Timing Extraction Method – 92x or 94x

- Activate link between SER and DES
- Activate video input to the SER
- Set 0x68[7:4] bits for desired monitoring test point
  - Video dimensions are 12 bits each, split between two readings
- Set 0x68[0] = 1
  - Delay ~100ms (still debugging why this is needed and only the first time a test point is selected)
  - Set 0x68[0] = 1 to trigger an update of the data again (still debugging why this is needed)
- Read 0x69[5:0]
- Repeat for all 4 values
- Combine read back values into 12 bit numbers and convert to decimal to get active horizontal and vertical dimensions detected at the deserializer

Timing Extraction Method – Tool

- Place values for pat_ahw[11:6], pat_ahw[5:0], pat_avw[11:6], and pat_avw[5:0] into this Excel tool to calculate detected resolution

Microsoft Excel Worksheet

Timing Extraction – Basic Example Script

```
import time
UB947 = 0x18
UB948 = 0x58

board.WriteI2C(UB947,0x01,0x02) # Reset 947 (optional)
time.sleep(0.1)
board.WriteI2C(UB947,0x4F,0x00) # Set dual OLDI (optional)
board.WriteI2C(UB947,0x03,0xFA) # Set I2C passthrough all and auto ack
board.WriteI2C(UB948,0x01,0x02) # Reset 948 (optional)
time.sleep(0.1)
board.WriteI2C(UB947,0x03,0xDA) # Set I2C passthrough all
board.WriteI2C(UB948,0x68,0x19) # H active High monitor
time.sleep(0.1) # Critical delay! Needed for proper operation!
board.WriteI2C(UB948,0x68,0x19) # Need to write this register again after the
delay
Hhigh = board.ReadI2C(UB948, 0x69, 1)
board.WriteI2C(UB948,0x68,0x09) # H active Low monitor
Hlow = board.ReadI2C(UB948, 0x69, 1)
board.WriteI2C(UB948,0x68,0x39) # V active High monitor
Vhigh = board.ReadI2C(UB948, 0x69, 1)
board.WriteI2C(UB948,0x68,0x29) # V active Low monitor
Vlow = board.ReadI2C(UB948, 0x69, 1)
mask = int('00111111',2)
hlowmask = Hlow & mask
hhighmask = Hhigh & mask
vlowmask = Vlow & mask
vhighmask = Vhigh & mask
hhighmask = hhighmask << 6
vhighmask = vhighmask << 6
Hactive = hhighmask | hlowmask
Vactive = vhighmask | vlowmask
print "Detected Resolution = ",Hactive,"x",Vactive
```

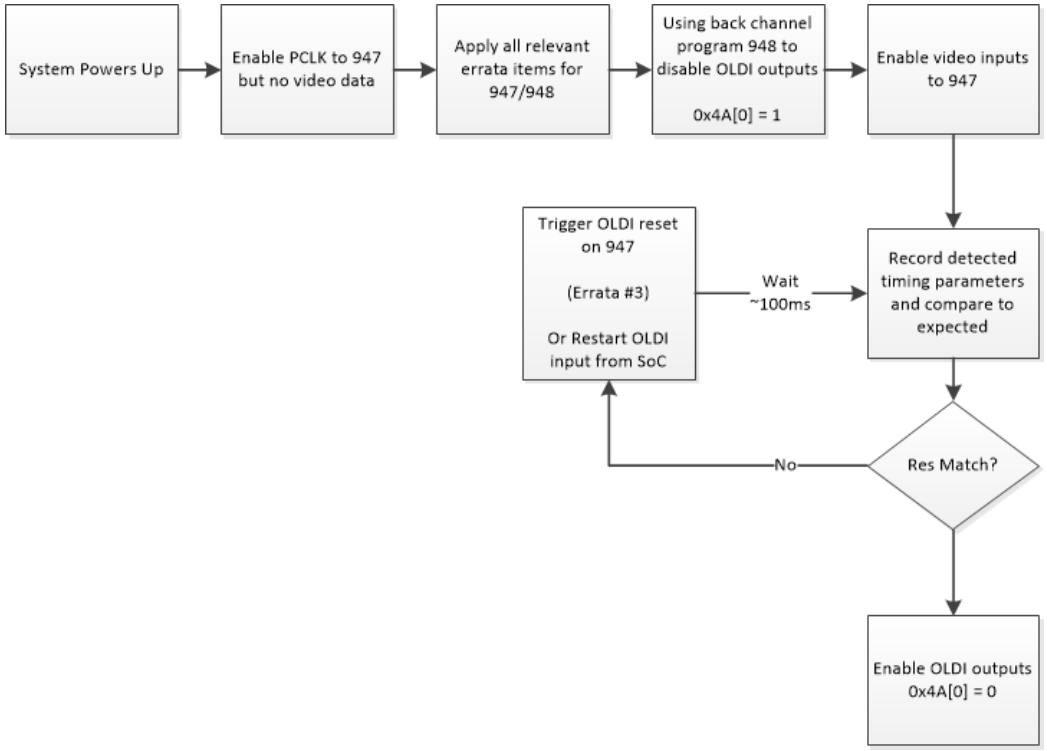
- Tested with 948 EVM internal PATGEN -> 947 EVM -> second 948 EVM
- Script runs from 947 EVM
- Detected resolution can be recorded during normal system operation across multiple units to build confidence in consistent reading
- If there is a timing issue related to the serializer side, it can most likely be detected as an error in the active frame dimensions when compared against the normal case
- In example testing, intentional skew provided to 947 input to simulate system marginality and resolution shows mismatch

Verification and Testing

- Timing extraction has been tested with 5 different resolutions on the following setup at TI:
  - 948 EVM (internal PATGEN) -> 947 EVM -> 948 EVM (All dual OLDI)
- Error condition forced on 947 input by increasing 0x4C indirect OLDI register to force a clock/data misalignment
  - In the forced error condition timing information read back clearly shows mismatch to expected active dimensions
  - In all normal conditions with different resolutions and frame rates the detected resolution is correct
- Testing ongoing with customers in real system
  - Read back of working system shows correct resolutions in multiple tests
  - Error condition (jittery screen) forced by adjusting 0x4C register and error can be detected through registers
  - Ongoing tests to try reading timings in non-forced error condition with jittery screen

Timing Error Detection Example (947->948)

Goal: During system initialization, utilize timing extraction to detect and correct issue at 947 OLDI inputs before issue propagates to display



Example Issue Detection and Prevention

```
import time

UB947 = 0x18
UB948 = 0x58

## Script inputs (system dependant)
Hres = 1280
Vres = 720
##
##
##

BISTerr = 1
attempts = 0

board.WriteI2C(UB947,0x01,0x02) # Reset 947
time.sleep(0.1)

board.WriteI2C(UB947,0x4F,0x00) # Set dual OLDI

board.WriteI2C(UB947,0x03,0xFA) # Set I2C passthrough all and auto ack
board.WriteI2C(UB948,0x01,0x02) # Reset 948
time.sleep(0.1)
board.WriteI2C(UB947,0x03,0xDA) # Set I2C passthrough all

board.WriteI2C(UB948,0x4A,0x01) # Disable OLDI outputs on 948

while BISTerr > 0 and attempts < 5:

    board.WriteI2C(UB947,0x40,0x10) # OLDI block reset (Errata item 3)
    board.WriteI2C(UB947,0x41,0x49)
    board.WriteI2C(UB947,0x42,0x16)
    board.WriteI2C(UB947,0x41,0x47)
    board.WriteI2C(UB947,0x42,0x20)
    board.WriteI2C(UB947,0x42,0xA0)
    board.WriteI2C(UB947,0x42,0x20)
    board.WriteI2C(UB947,0x42,0x00)
    board.WriteI2C(UB947,0x41,0x49)
    board.WriteI2C(UB947,0x42,0x00)
```

Example Issue Detection and Prevention

```
    time.sleep(0.1)

    board.WriteI2C(UB947,0x64,0x05) # Enable PATGEN with color bars and external timing on 947
    board.WriteI2C(UB948,0x64,0x04) # Set PATGEN on 948 to color bars

    board.WriteI2C(UB948,0x68,0x08) # Enable PATGEN BIST

    time.sleep(0.5)

    board.WriteI2C(UB948,0x68,0x19) # H active High monitor
    Hhigh = board.ReadI2C(UB948, 0x69, 1)
    board.WriteI2C(UB948,0x68,0x09) # H active Low monitor
    Hlow = board.ReadI2C(UB948, 0x69, 1)
    board.WriteI2C(UB948,0x68,0x39) # V active High monitor
    Vhigh = board.ReadI2C(UB948, 0x69, 1)
    board.WriteI2C(UB948,0x68,0x29) # V active Low monitor
    Vlow = board.ReadI2C(UB948, 0x69, 1)

    board.WriteI2C(UB948,0x66,0x19)
    BISTerr = board.ReadI2C(UB948, 0x67, 1) # Read twice to get correct value
    BISTerr = board.ReadI2C(UB948, 0x67, 1)

    board.WriteI2C(UB948,0x68,0x00) # Disable PATGEN BIST
    board.WriteI2C(UB947,0x64,0x00) # Disable PATGEN with color bars and external timing on 947

    # print "0x18 Hhigh = ",hex(Hhigh)
    # print "0x08 Hlow = ",hex(Hlow)
    # print "0x38 Hhigh = ",hex(Vhigh)
    # print "0x28 Hlow = ",hex(Vlow)

    mask = int('00111111',2)

    hlowmask = Hlow & mask
    hhighmask = Hhigh & mask
    vlowmask = Vlow & mask
    vhighmask = Vhigh & mask


    hhighmask = hhighmask << 6
    vhighmask = vhighmask << 6
```

Example Issue Detection and Prevention

```
80
81      Hactive = hhighmask | hlowmask
82      Vactive = vhighmask | vlowmask
83
84      attempts = attempts + 1
85
86      if Hactive != Hres:
87          BISTerr = BISTerr + 1
88      if Vactive != Vres:
89          BISTerr = BISTerr + 1
90
91      print "System Init Attempts =",attempts
92      print "Detected Resolution = ",Hactive,"x",Vactive
93      print "BIST Errors = ",BISTerr
94
95  board.WriteI2C(UB948,0x4A,0x00) # Enable OLDI outputs on 948
```

RGB Bitwise Error Detection Method – Internal Pattern

- Activate link between SER and DES (94x DES only)
- Activate video input to the SER and start PATGEN with external timing using a internal fixed color pattern (no scrolling)
- Set the DES to the same color pattern and set the PATGEN configuration to use external timing. Do not enable PATGEN on the DES
- On the DES, set 0x68[3] = 1 to enable PATGEN BIST
- On the DES, register 0x69[7] shows an error flag
- On the DES, in the PATGEN indirect registers at address 0x19, the number of errors is counted up to 0xFF
  - This register is clear on read and when the value in this register is 0x00, the 0x69[7] flag will be de-asserted

* If there are timing errors causing sync issues then this feature may not report correct error checks

Important notice and disclaimer

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements.

These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.