HAMAMATSU

PHOTON IS OUR BUSINESS

InGaAs Linear Sensor Reference Circuit Design

Technical Note

Authors
John Gilmore, Hamamatsu Corporation
Lu Cheng, Hamamatsu Corporation
Scott Hunt, Analog Devices, Inc.

Web info
Find this tech note online at:

https://hub.hamamatsu.com/us/en/technical-notes/image-sensors/ingaas-linear-sensor-reference-circuit-design-section-1.html

Hamamatsu Corporation: www.hamamatsu.com

Analog Devices, Inc.: www.analog.com

HAMAMATSU
PHOTON IS OUR BUSINESS

Table of Contents

Introduction

Section 1

The objective of this tech note is to provide the reference circuit design for our most demanded InGaAs linear image sensor (LIS) series. This reference design can be used to drive the Hamamatsu one-stage TE-cooled InGaAs LIS G9201-256SB, G9202-512SB, G9203-256SA, G9204-512SA, G9211-256SB, G9212-512SB, G9213-256SA, G9214-512SA, and the two-stage TE-cooled InGaAs LIS with extended cutoff wavelength G9205-256/512WB, G9206-256/512WB, G9207-256WB, G9208-256/512WB. Please note: this reference circuit cannot drive the InGaAs LIS series including G14237-512WA, G11508-256/512SA, G11475-256/512WB, G11476-256WB, G11477-256/512WB, G11478-256/512WB, G11620-256/512SA, and G12230-512WB. For further information about the G11508/G1147x series please refer to **https://www.hamamatsu.com/resources/pdf/ssd/g11508_etc_kmir1032e.pdf**.

These InGaAs LIS series are designed for near infrared multi-channel spectrophotometry, non-destructive inspection, and DWDM wavelength monitor applications. These sensors consist of an InGaAs photodiode array, charge amplifiers, offset compensation circuit, and timing generator formed on a CMOS chip. The spectral response characteristics of these series that the reference circuit can support is as follows:

Spectral response

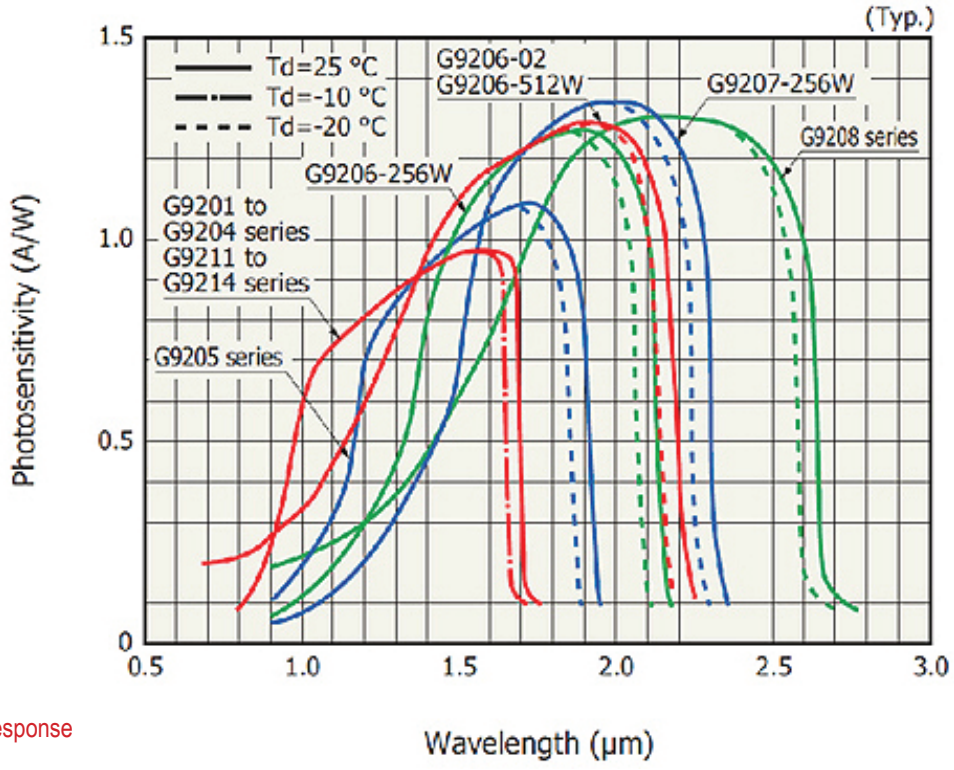


Figure 1-1: Spectral response

In this tech note a system design including the analog front-end sensor board, the FPGA modules, the USB processor, and the data acquisition software is described. And the test results of the I/O signals, noise, and linearity are presented for various pixel formats and timing control modes.

System Description

Section 2

The system includes four boards: the analog front-end (AFE) sensor board, the FPGA board, the USB board, and the interconnect board. The AFE sensor board was developed by Analog Devices, Inc. in collaboration with Hamamatsu; this board includes a Hamamatsu InGaAs linear array, followed by a 1MSPS SAR ADC with integrated ADC driver. The board also includes a TEC controller and all of the required voltages conditioning to drive the sensor. The FPGA board issues the control signals to the sensor via the Analog Devices AFE sensor board. The interconnect board is a passive device joining the FPGA development board, USB development board and Analog Devices AFE sensor board together. The overall system interconnection is shown in Figure 2-1.

The data returned by the sensor in response to the FPGA control signals is processed through the A/D converter on the Analog Devices AFE sensor board and received by the FPGA board. The FPGA sends the processed data to the USB processor via the EZ-USB FX3® Slave FIFO Interface for subsequent transfer to a PC.

The data stream received from the Analog Devices sensor board consists of 16-bit words, each representing a single pixel value. The system supports two pixel formats: 256-pixel and 512-pixel mode. The pixel data rate is adjustable from 10KHz to 500KHz, resulting in data throughput of 160Kbps~8Mbps.

The FPGA board is controlled by the USB processor (EZ-USB FX3®) via an I2C interface with the processor acting as an I2C master and the FPGA as an I2C slave. The control is accomplished via a set of the R/W control/status registers in the FPGA memory space. Refer to subsequent sections for the details of the control/status register space and the I2C access protocol.

The logic/flow diagram is shown in Figure 2-2.

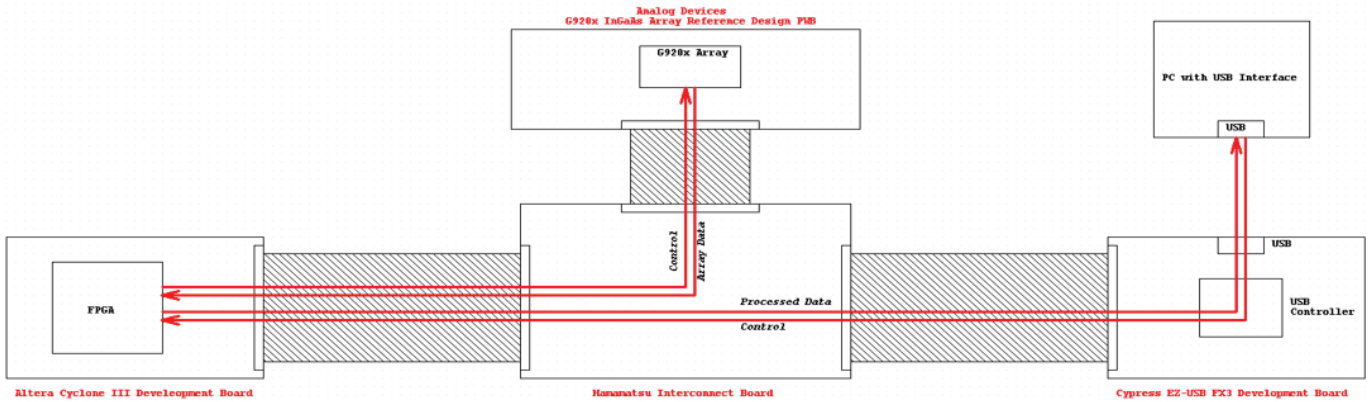


Figure 2-1: System interconnection

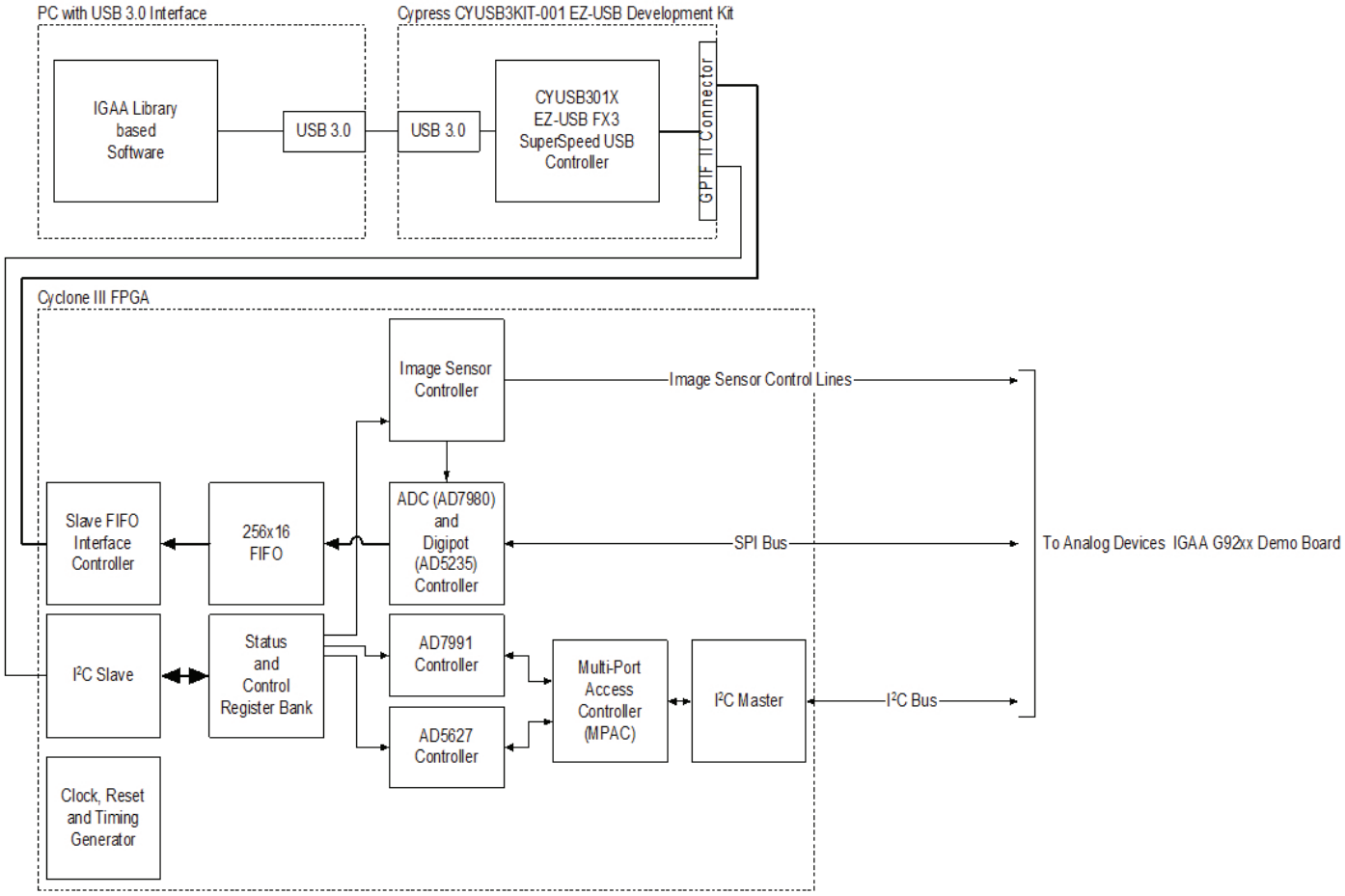


Figure 2-2: Logic/Flow diagram

3A. Theory of Operation and I2C and SPI Device Descriptions

Developed by Analog Devices, Inc. in collaboration with Hamamatsu, this board includes a Hamamatsu G920x InGaAs array, followed by two buffer amplifiers, a multiplexer to select a voltage from one of these two amplifiers, and a 1MSPS SAR ADC with integrated ADC driver. The board also includes a TEC controller and all of the required power conditioning to power the board from an AC adapter. The product selections were chosen to exceed the performance targets of the solution and provide high integration to enable a small footprint and simplicity of design. For more information about the devices chosen for use in this design, please refer to
https://www.analog.com/en/technical-articles/integration-collab-at-heart-of-hi-perf-image-sensor-ref-design.html

Although the board does not conform to ANSI/VITA FMC standards due to the form factor and AC adapter power input, the connector area is designed to be able to interface with most FMC-compliant FPGA development boards in order to provide increased flexibility of target platform.

3B. Providing Power to the Board

Provide 9VDC, 1.5A or greater through barrel connector J2 (recommended: CUI, Inc. SMI24-9-V-P5 or similar). Although there is reverse power protection on the board, ensure that the polarity of the AC adapter is center positive for proper operation. For alternative InGaAs array variants, consider the power required by the TEC and ensure enough power is provided. This reference design board can support up to 3A maximum TEC current at up to 5V. A 9V, 2A AC adapter is enough to support a full 3A TEC current due to the high-efficiency 9V to 5V step-down buck regulator (the LTM8053) that is on the board. The power input features reverse protection, a 6A slow-blow fuse, and a 15V bidirectional TVS to protect the board from power supply transients. When using one of the G920x variants which has a 5V, 1.8A TEC, total power draw from the 9V AC wall adapter can be roughly 1.2A at maximum TEC drive.

3C. Array and Buffers

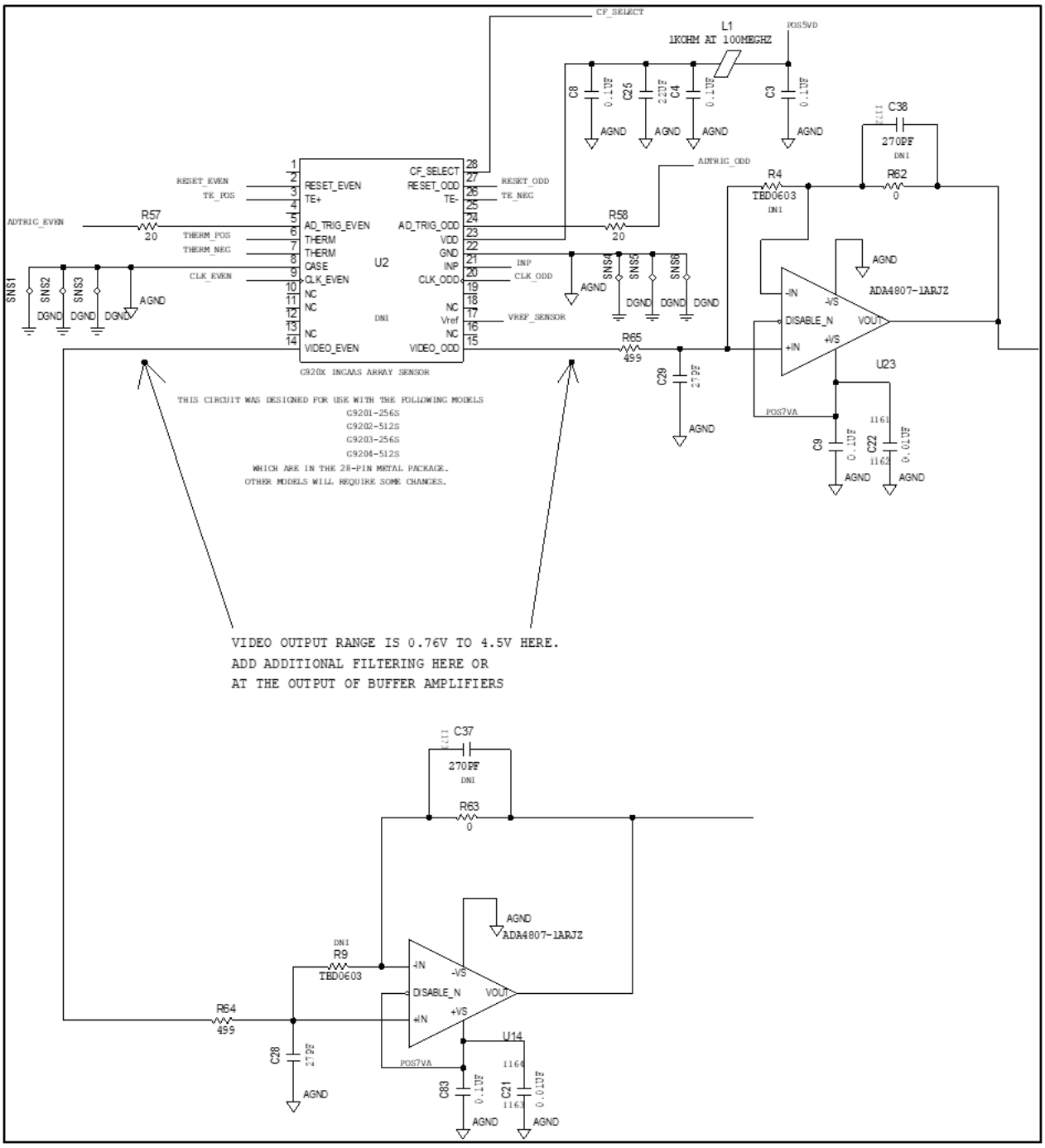


Figure 3-1: Sensor and buffers

Depending on the variant, the array has up to two analog outputs, called ‘Video-even’ and ‘Video-odd’, which are suggested to be buffered in the array datasheet. For the 512-pixel versions of the array, the even pixels come out on one line while the odd pixels are shifted out on the other line. The amplifier and filters are chosen to settle faster than the array, which specifies a 600ns output settling time. Provisions are included to match input resistances for the amplifier by placing 499Ω at R62 and R63 to reduce DC errors due to input bias current. If the 499Ω R62 and R63 are installed, it may be necessary to install C37 and C38 to neutralize the resulting input pole and avoid instability. The analog output of the array can go from 0.76V to 4.5V.

3D. Even-Odd Switch

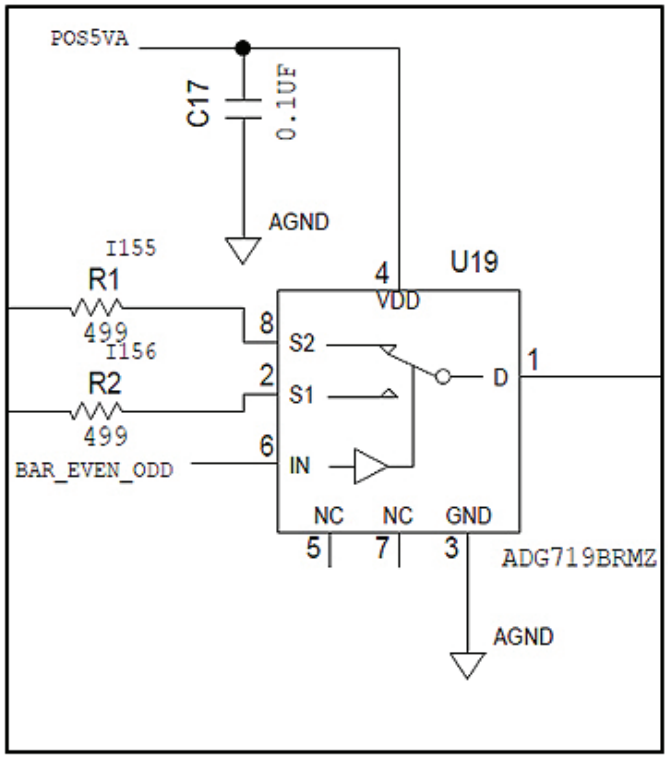


Figure 3-2: Even-odd multiplexer

A single-pole dual throw (SPDT) solid-state switch follows the even and odd pixel buffer amplifiers and connects one of the two to the ADC. A default pull-up resistor sets the switch to odd when floating, which is the correct side for the 256-pixel arrays. A 499Ω resistor ensures stability of the buffers driving the switch capacitance and reduces current spikes during switching.

3E. ADC Driver and ADC

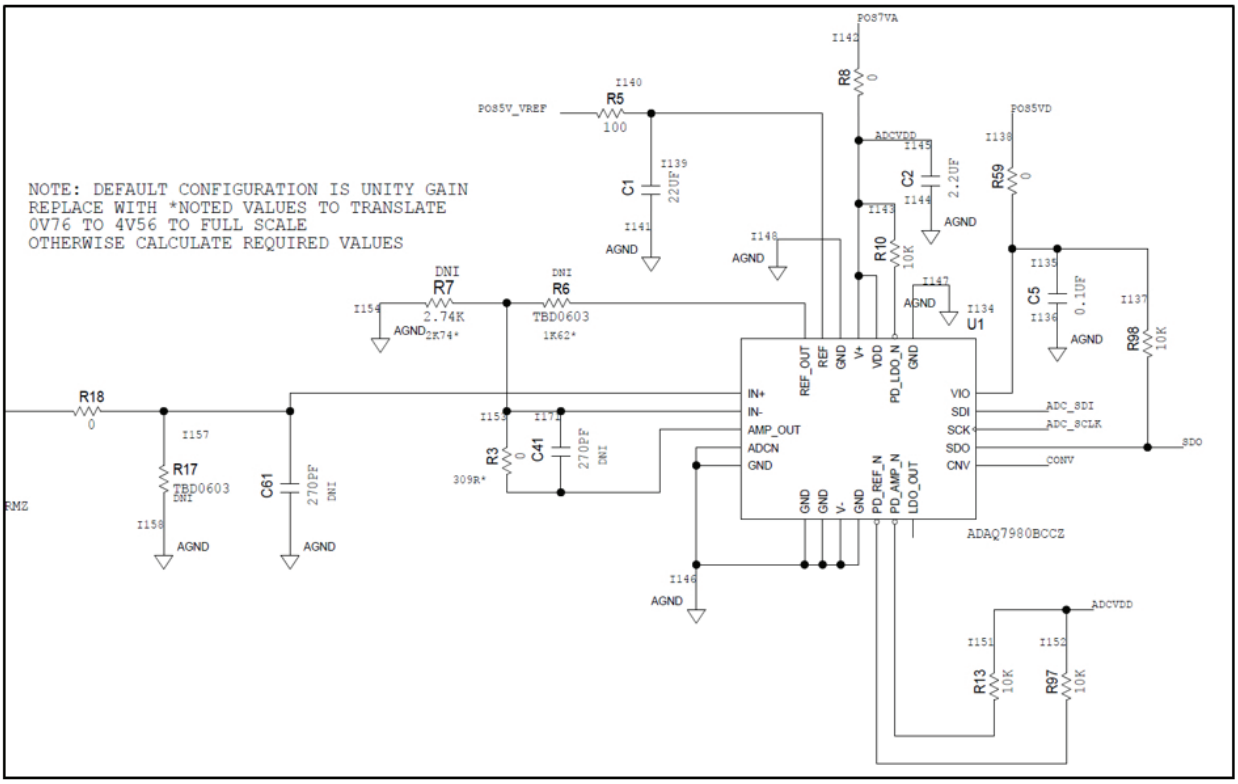


Figure 3-3: ADC

The ADC driver is integrated into the ADAQ7980. By default, this stage is in unity-gain, which provides the optimal settling and does not negatively affect the noise. Provisions and recommendations are included to match the input to the full-scale 0-5V input range of the ADC if desired. The ADAQ7980 ADC is a 16-bit, 1MSPS SAR ADC with an integrated ADC driver, reference buffer, LDO, and necessary passives that is connected to the SPI bus. For more information about this integrated signal chain, refer to **https://www.analog.com/en/technical-articles/integration-collab-at-heart-of-hi-perf-image-sensor-ref-design.html**. The reference voltage is 5V, so one LSB represents 76.3µV.

3F. Serial Devices

There are a total of 4 serial devices on the reference board:

SPI Bus

- One AD5235 dual 25kΩ, 1024-position digital potentiometer
- One ADAQ7980 16-bit ADC

I2C Bus

- One AD7991 12-bit ADC
- One AD5627 12-bit DAC

3G. AD5235 Dual Potentiometer and Bias Divider

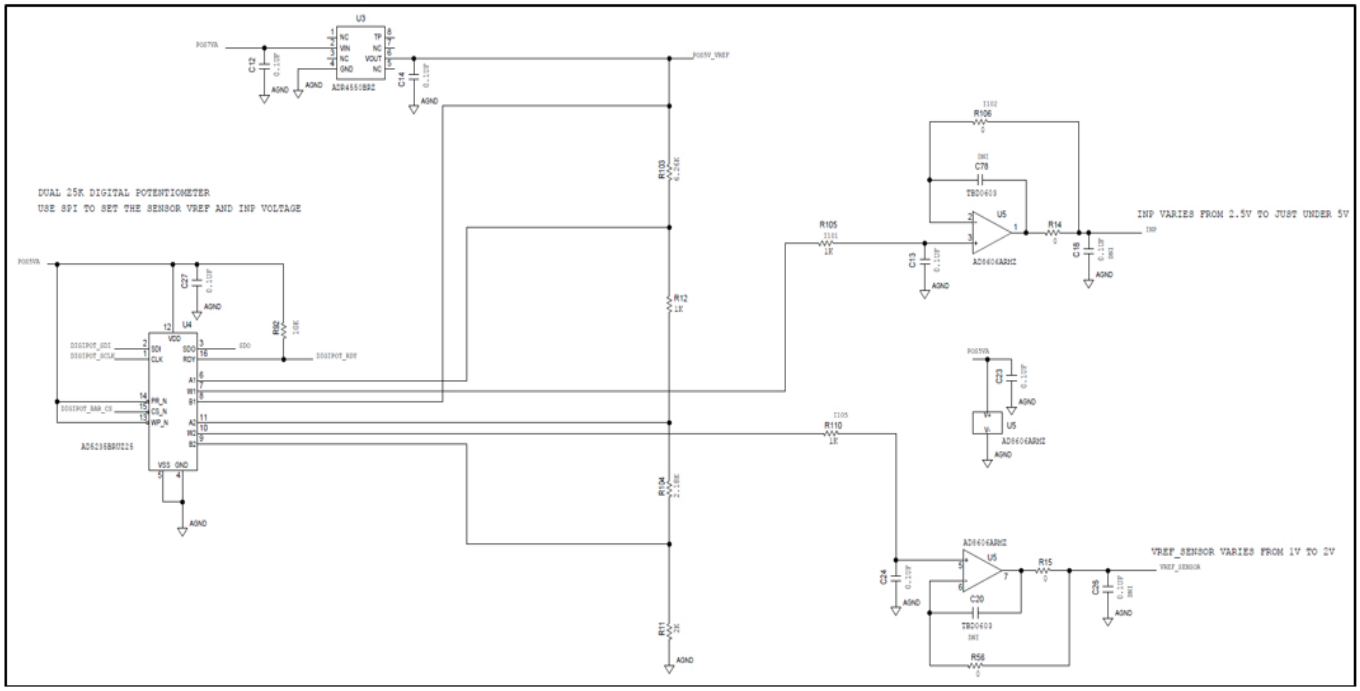


Figure 3-4: Dual POT for Vref and INP bias voltages

The programmable bias voltages for the sensor are derived from a voltage divider off of the precision ADR4550 5V voltage reference used for the ADC. Potentiometer 1 controls the INP voltage from approximately 2.5V at maximum code to approximately 5V at minimum code. Potentiometer 2 controls the VREF_SENSOR voltage from approximately 1V at minimum code to approximately 2V at maximum code. When potentiometer 1 is set to its lowest value of 0, the INP voltage is 5V. When INP is set to the maximum value of 1023, the INP voltage is 2.5V. Intermediate values can be calculated according to the following equation:

Equation 1

$$V_{INP} = 5V - 2.5V\left(\frac{D}{1024}\right)$$

where D can range from 0 to 1023. Note that although the AD8606 is a ‘rail to rail’ output amplifier, it will only get to about 4.96V, so writing values lower than ‘16’ to the rheostat may not result in the desired output. This should not be a problem because the maximum bias voltage specified in the G920x datasheet is 4.6V. The default value written to this rheostat should be 205d (0xCD) to get 4.5V.

Potentiometer 2 controls the Vref voltage going to the array. When the potentiometer is set to its lowest value of 0, the output on the Vref pin is 1V. When the potentiometer is set to its highest value of 1024, the output on the Vref pin is 2V. Intermediate values can be calculated according to the following equation:

Equation 2

$$V_{Vref} = 1V + 1V\left(\frac{D}{1024}\right)$$

where D can range from 0 to 1023. Vref should be set to a default value of 1.26V by writing 266d to the potentiometer (0x10A).

3H. AD7991 ADC

Note: I2C address: 010 1001

The AD7991 is a 12-bit ADC with a 2.5V reference provided by the ADN8835, so each LSB represents 610µV. Channel 0 is used to measure the amplified and linearized thermistor output from the ADN8835.

The thermistor uses a simple linearization circuit that results in an output of approximately 26.53mV/°C from -10°C to 40°C, with an offset of 0.566V.

To convert from volts to approximate temperature, use the following equation:

Equation 3

$$Temperature = \left(\frac{ADC\ Output}{4096} \cdot 2500mV - 566mV\right) \cdot \frac{1^{\circ}C}{26.53mV}$$

This approximation will result in errors of slightly over 1°C at -10°C and 40°C, and nearly zero error at 15°C. Note that it is possible to write voltages lower than 0.239V, which will result in a lower setpoint temperature. However, the above approximation will result in large errors. For example, the equation above predicts that 0V corresponds to -21.3°C, while the true temperature would be close to -29°C.

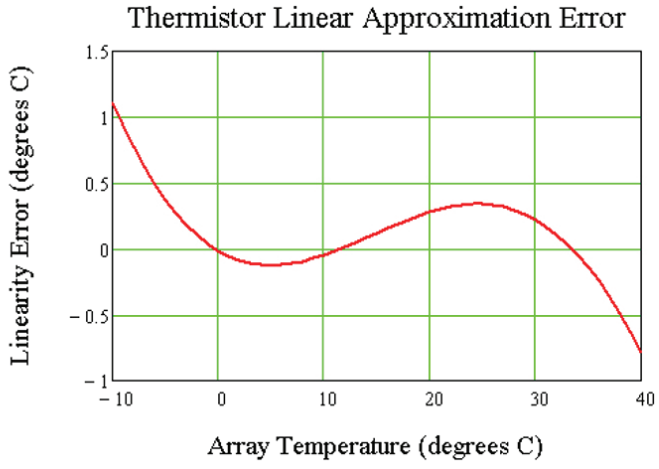


Figure 3-5: Thermistor linear approximation error

If higher accuracy conversion is required, use the following equation:

Equation 4

$$Temperature\ (degrees\ K) = \left(\frac{1}{T_r} + \frac{ln\left(-\frac{R_x - \frac{R_{fb}}{\frac{R_{fb}}{R_{top}} + \frac{V_{out}}{1.25V} - 1}}{R_r} \right)}{B} \right)^{-1}$$

where:
Temperature = The thermistor temperature in degrees Kelvin
T_r = 298.15K (Temperature at which nominal thermistor resistance is specified)
R_r = 5kΩ (Nominal thermistor resistance at T_r)
B = 3200K (Thermistor constant given in the datasheet)
R_x = 5.11kΩ (R43 in the schematic)
R_{top} = 10kΩ (R32 in the schematic)
R_{fb} = 12kΩ (R31 in the schematic)

Note: Using this equation results in linearity error dominated by the number format precision used, and should be negligible if implemented in floating point on a PC.

Channel 1 is used to measure the linearized thermistor output from the 'hot' side of the heatsink. The same equation as above can be used to convert volts to temperature, assuming the same thermistor is used. Note that once again, the approximate equation will lose accuracy above 40°C, and the full equation should be used if higher temperatures are expected. If a different thermistor is used, then the equation will vary depending on the thermistor's characteristics. For proper operation, the REF_SEL bit in the AD7991's Configuration Register must be set to '1' (this uses the Vin3/Vref pin as the converter reference).

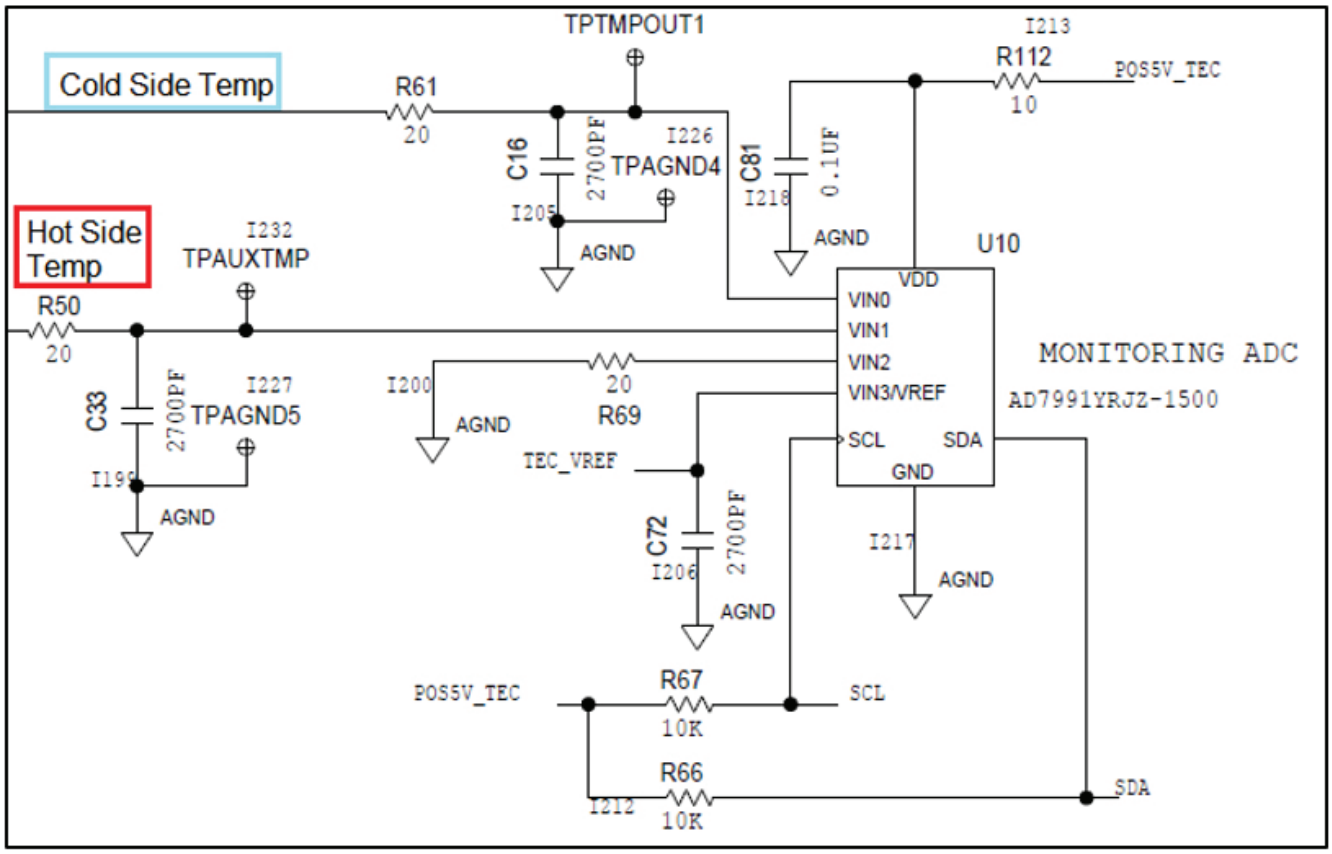


Figure 3-6: Temperature monitoring ADC

3I. DAC

Note: I2C address: 000 1110

The AD5627 DAC controls the temperature setpoint for the TEC controller. To calculate the temperature setpoint based on the output data written, use the following equation:

Equation 5

$$Temperature\ Setpoint = \left(\frac{DAC\ Output}{4096} \cdot 5000mV - 564mV\right) \cdot \frac{1^{\circ}C}{26.53mV}$$

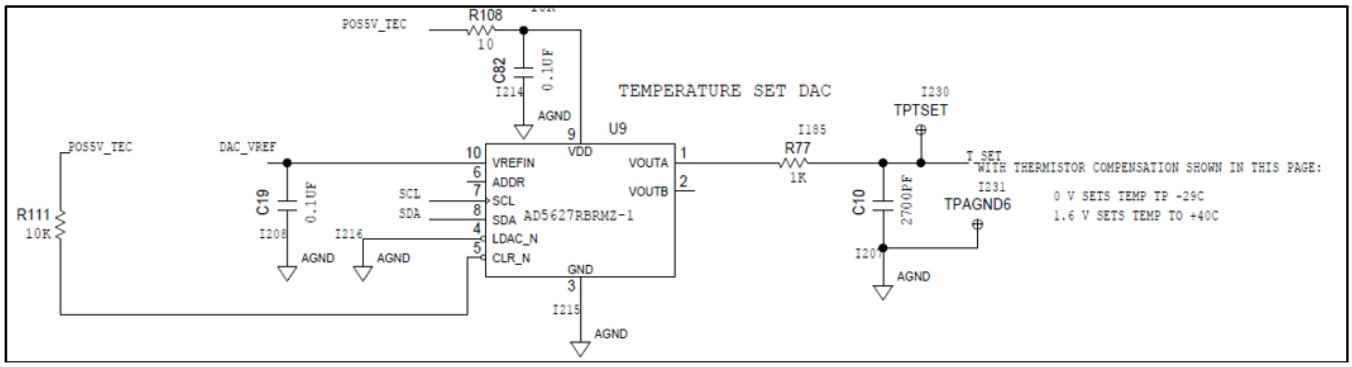


Figure 3-7: Temperature setting DAC

3J. TEC Controller

The ADN8835, a TEC controller with integrated 3A power FETs, is used to very accurately control the temperature of the image sensor array. The AD5627 sets the temperature setpoint and the AD7991 reads the hot side and cold side thermistor temperatures. The TEC Voltage Limit and Current Limit are set by resistor dividers tailored to the values needed for the G920x family. To change these limits to accommodate a different family of image sensors with different TEC requirements, see Analog Devices UG-951 for suggestions on the resistor value. For more information on thermoelectric cooler control, refer to ADN8835 datasheet. It may also be necessary to adjust the analog PID components and thermistor components in order to accommodate other sensors. The EN/SY pin is pulled up in order to be enabled by default, but the TEC controller can be shut down if this signal is pulled low by the FPGA. This pin can also be used for synchronization, as described below.

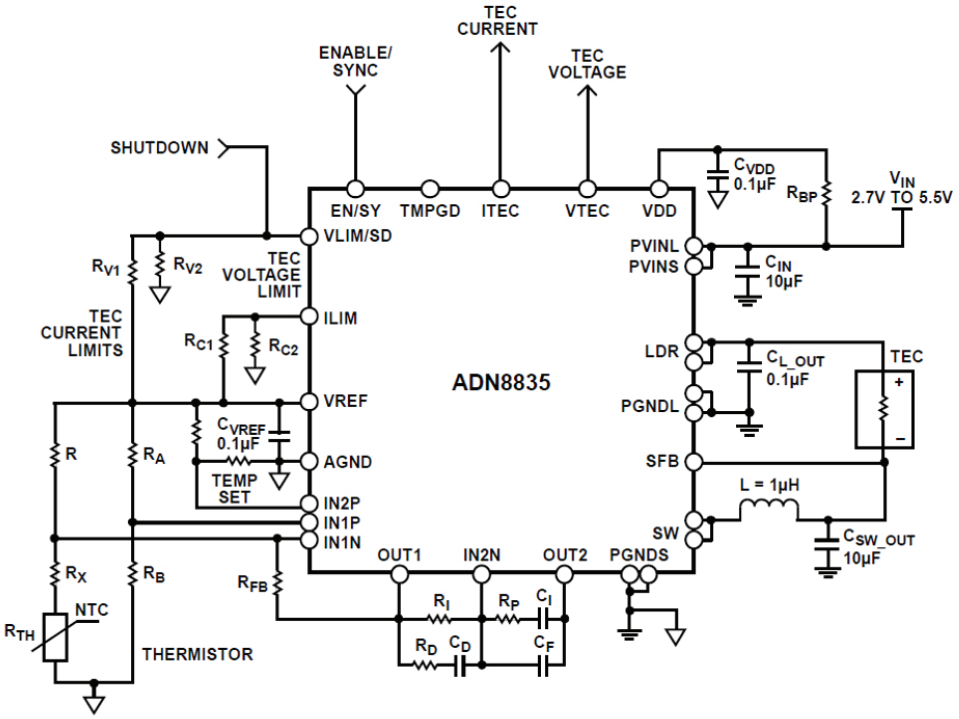


Figure 3-8: TEC control

Indicators and Test Points

Ultralow power LEDs are provided to indicate temp good from the ADN8835 (DS4) and power good for all power rails: DS3 indicates POS5V_TEC, DS2 indicates POS7VA, and DS1 indicates POS5VD/POS5VA power good. Surface mount test points are provided for many signals to be probed, especially in the dense TEC control portion of the circuit, along with corresponding ground test points.

Synchronization Feature and LTM8053 Modes

To remove any switching clock intermodulation from the circuit, this design can be fully synchronized. ADN8835 can be synchronized by driving the shutdown signal, BAR_TEC_SD, with a clock between 1.85MHz and 3.25MHz. LTM8053 can also be synchronized by installing a provisional zero ohm resistor in position R35 to connect its SYNC pin to the BAR_TEC_SD signal. This reduces the maximum synchronization clock frequency to 3MHz. The clocks of both of these devices may be synchronized to a clock output from the FPGA that is synchronous with the sensor pixel clock signals, and the ADAQ7980 samples may be taken at integer multiples of the sync clock frequency, eliminating the effect of switching noise on the measurement. For more information, see the notes on the schematic and the ADN8835 and LTM8053 datasheets. The full complement of SYNC/MODE options for LTM8053 are described in the schematic. By default, the ADN8835 and the LTM8053 are both running on their internal clocks and the LTM8053 is in pulse-skip mode.

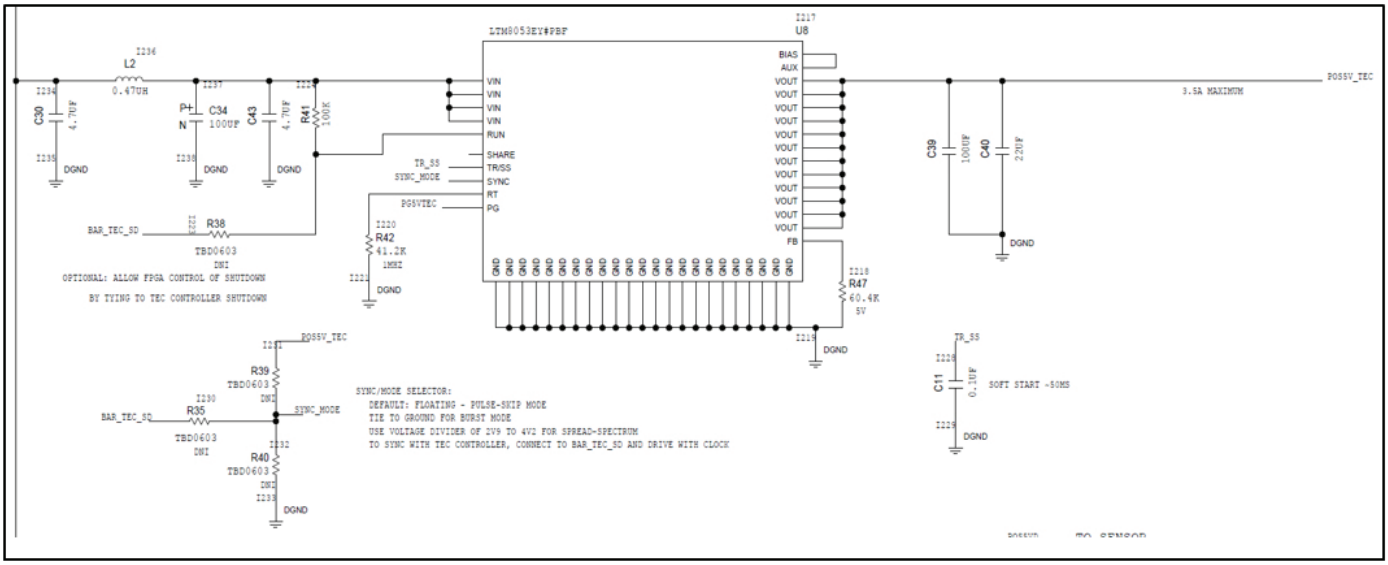


Figure 3-9: SYNC / mode selection

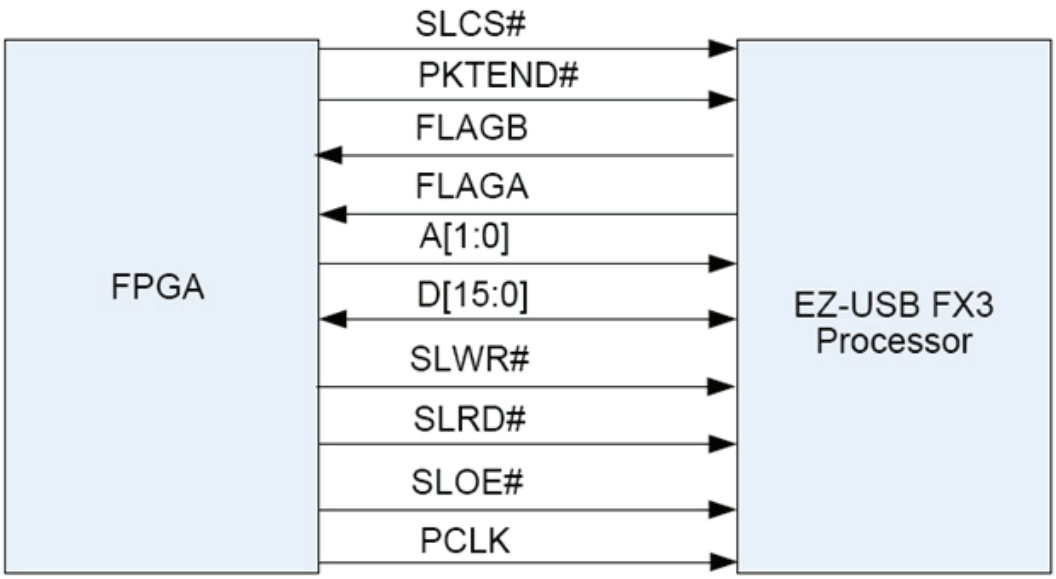


Figure 4-1: FPGA to EZ-USB FX3 interface

The pixel data is transferred between the FPGA and the USB processor using a 16-bit Synchronous Slave FIFO interface operating at 80MHz.

The data stream consists of 16-bit pixel word comprising 256 or 512 pixels per line.

The line-to-line separator consists of 2x 16-bit marker words:
Marker Word 1: 0xAAAA
Marker Word 2: 0x5555

The marker words appear in sequence: Marker Word 1, followed by Marker Word 2. The marker words are intended to serve the function equivalent to V-sync in video frames. *Note: There is no equivalent H-syncs as the frame is a single line.*

4A. Interface Configuration

Table 4-1 Data Path I/O Configuration

CYUSB3014-BZX Pin		Synchronous Slave FIFO Interface with 16-bit Data Bus	Interconnect Board Net Name	FPGA Pin	EZ-USB PWB I/O Voltage	Description
Name	Location					
Data Path						
GPIO[17]	K8	SLCS#	CTL0	L1	VIO1	This is the chip select signal for the Slave FIFO interface. It must be asserted to access Slave FIFO.
GPIO[18]	K7	SLWR#	CTL1	L4	VIO1	This is the write strobe for the Slave FIFO interface. It must be asserted for performing write transfers to Slave FIFO.
GPIO[19]	J7	SLOE#	CTL2	M2	VIO1	This is the output enable signal. It causes the data bus of the Slave FIFO interface to be driven by FX3. It must be asserted for performing read transfers from Slave FIFO.
GPIO[20]	H7	SLRD#	CTL3	L3	VIO1	This is the read strobe for the Slave FIFO interface. It must be asserted for performing read transfers from Slave FIFO.
GPIO[21]	G7	FLAGA	CTL4	L18	VIO1	These are the FLAG outputs from FX3. The FLAGs indicate the availability of an FX3 socket. FLAGA is configured as Current_thread_DMA_RDY. FLAGB is configured as Current_thread_DMA_watermark.
GPIO[22]	G6	FLAGB	CTL5	L16	VIO1	
GPIO[23]	K6	FLAGC	CTL6	P2	VIO1	
GPIO[25]	G5	FLAGD	CTL8	T3	VIO1	
GPIO[24]	H8	PKTEND#	CTL7	P1	VIO1	This signal is asserted to write a short packet or a zero length packet to Slave FIFO.
GPIO[28]	J5	A1	CTL11	G18	VIO1	This is the 2-bit address bus of Slave FIFO.
GPIO[29]	H5	A0	CTL12	K18	VIO1	
GPIO[0]	F10	DQ[0]	DQ[0]	H6	VIO1	This is the 16-bit data bus of Slave FIFO.
GPIO[1]	F9	DQ[1]	DQ[1]	D3	VIO1	
GPIO[2]	F7	DQ[2]	DQ[2]	M5	VIO1	
GPIO[3]	G10	DQ[3]	DQ[3]	L6	VIO1	
GPIO[4]	G9	DQ[4]	DQ[4]	T1	VIO1	
GPIO[5]	F8	DQ[5]	DQ[5]	M3	VIO1	

Table 4-1 Data Path I/O Configuration

CYUSB3014-BZX Pin		Synchronous Slave FIFO Interface with 16-bit Data Bus	Interconnect Board Net Name	FPGA Pin	EZ-USB PWB I/O Voltage	Description
Name	Location					
Data Path						
GPIO[6]	H10	DQ[6]	DQ[6]	N7	VIO1	
GPIO[7]	H9	DQ[7]	DQ[7]	T2	VIO1	
GPIO[8]	J10	DQ[8]	DQ[8]	N8	VIO1	
GPIO[9]	J9	DQ[9]	DQ[9]	H15	VIO1	
GPIO[10]	K11	DQ[10]	DQ[10]	J13	VIO1	
GPIO[11]	L10	DQ[11]	DQ[11]	H16	VIO1	
GPIO[12]	K10	DQ[12]	DQ[12]	N10	VIO1	
GPIO[13]	K9	DQ[13]	DQ[13]	N16	VIO1	
GPIO[14]	J8	DQ[14]	DQ[14]	N11	VIO1	
GPIO[15]	G8	DQ[15]	DQ[15]	N15	VIO1	
GPIO[16]	J6	PCLK	HSMC_CLKOUT_p2	U18	VIO1	Slave Interface clock 80 MHz. Note: Jumper on interconnect board from J3.44 to J2.155 needed

4B. EZ-USB Development Board Configuration

Table 4-2 EZ-USB Development Board Configuration

PWB Reference Designator	Configuration (Jumper/Switch Setting)
J40	Open
J42	2-3 shorted
J45	2-3 shorted
J47	Open
J50	Open
J52	2-3 shorted
J53	1-3 shorted
J72	1-2 shorted
J74	Open
J98	Open
J97	2-3 shorted
J96	2-3 shorted
J100	1-2 shorted
J101	2-3 shorted
J102	2-3 shorted
J104	1-2 shorted
J103	1-2 shorted
J125	1-2 shorted
J134	3-6 shorted
J135	2-4 shorted
J136	2-5 shorted
J143	2-5 shorted
J144	2-5 shorted
J145	2-5 shorted
J146	2-5 shorted
J156	Open

Table 4-2 EZ-USB Development Board Configuration	–
PWB Reference Designator	Configuration (Jumper/Switch Setting)
SW25	1 = OFF 2 = OFF 3 = ON 4 = ON
SW40	1 = ON 2 = ON 3 = ON 4 = ON

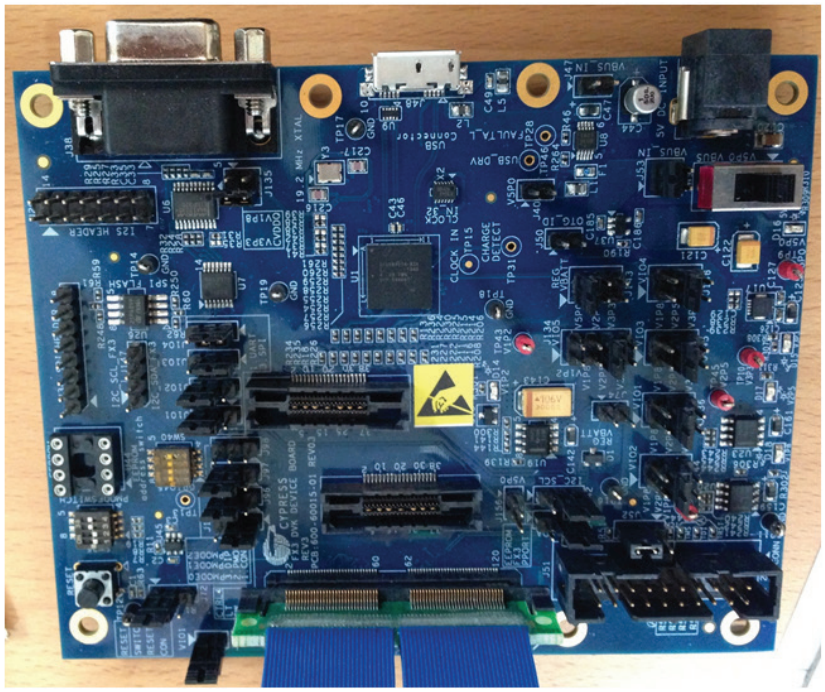


Figure 4-2: Cypress EZ-USB development board configuration

4C. FPGA/USB Processor Interface

The USB processor configures the FPGA control registers via I2C interface as described in Section 5. Once the desired configuration parameters have been set, the USB processor sets Reg_01 bits (7:6) = “11”, enabling the sensor control and the Slave FIFO operation.

The USB processor can disable the sensor control and/or Slave FIFO operation at any time in order to re-configure the control registers.

Table 5-1 Control Path I/O Configuration

CYUSB3014-BZX Pin		Synchronous Slave FIFO Interface with 16-bit Data Bus	Interconnect Board Net Name	FPGA Pin	EZ-USB PWB I/O Voltage	Description
Name	Location					
Control/Status Path						
I2C_GPIO[58]	D9	I2C_SCL	USB_SCL	L13	VIO5	SCL line of the I2C control bus Note: Jumper on interconnect board from J3.88 to J2.132 needed
I2C_GPIO[59]	D10	I2C_SDA	USB_SDA	M14	VIO5	SDA line of the I2C control bus Note: Jumper on interconnect board from J3.86 to J2.134 needed

5A. Access Protocol

1. The FPGA acts as I2C slave with address 0xAA.
2. EZ-USB FX3 processor acts as an I2C master.
3. Each I2C write transfer consists of the following:
    - 3-1. START (generated by master)
    - 3-2. Byte 1 = 0xAA
    - 3-3. ACK (generated by the slave)
    - 3-4. Byte 2 = Selected control register address
    - 3-5. ACK (generated by the slave)
    - 3-6. Byte 3 = Data byte to be written to the selected FPGA control register address
    - 3-7. ACK (generated by the slave)
    - 3-8. STOP

4. Each single byte I2C read transfer consists of the following:

    4-1. START

    4-2. Byte 1 = 0xAA

    4-3. ACK (generated by the slave)

    4-4. Byte 2 = Selected control register address

    4-5. ACK (generated by the slave)

    4-6. RESTART

    4-7. Byte 3 = 0xAB

    4-8. ACK (generated by the slave)

    4-9. Byte 4 = Data byte from the selected control register returned by the FPGA

    4-10. NACK (generated by the master)

    4-11. STOP

5. Each multi-byte I2C read transfer consists of the following:

    5-1. START

    5-2. Byte 1 = 0xAA

    5-3. ACK (generated by the slave)

    5-4. Byte 2 = Selected control register address

    5-5. ACK (generated by the slave)

    5-6. RESTART

    5-7. Byte 3 = 0xAB

    5-8. ACK (generated by the slave)

    5-9. Byte 4-1 = Data byte from the selected control register returned by the FPGA

    5-10. ACK (generated by the master)

    5-11. Byte 4-2 = Data byte from the selected control register returned by the FPGA

    5-12. ACK (generated by the master)

    5-13. Byte 4-3 = Data byte from the selected control register returned by the FPGA

    5-14. ACK (generated by the master)

    5-15. Byte 4-4 = Data byte from the selected control register returned by the FPGA

    5-16. NACK (generated by the master)

    5-17. STOP

Note: The number of bytes read is not limited to 4 (the 4 bytes read transfer is shown as an example only). As additional bytes are being read, the address pointer is auto-incrementing, starting from the address specified by Byte 2.

5B. Memory Space/Register Definitions

Table 5-2 Memory Space/Register Definitions

Address	Name	Access	Default	Bit/Field	Register Name/Field Description
0x00	Reg_00	R/O	-		FPGA F/W Revision
0x01	Reg_01	R/W	0xC0		Mode Control Register
				0	Pixel Mode: 0 = 256-pixel mode 1 = 512-pixel mode
				1	Interface Speed: 0 = Normal-speed operating mode 1 = High-speed operating mode
0x02	Reg_02	R/W	0x0A		Pixel Clock Control Low
				7:0	*Clock_Divider[7:0]*
0x03	Reg_03	R/W	0x00		Pixel Clock Control High
				0	*Clock_Divider[8]* Note: Writing to Reg_0x03 sets the *Clock_Divider[9:0]* to the values of registers Reg_02 and Reg_03. Writing to Reg_02 alone does not change the value of the *Clock_Divider[8:0]*. *Clock_Divider[8:0]* is used to set the internal FPGA clock enable used to generate odd and even clocks. The relationship between the divider value and the odd/even clocks is: 60MHz/(2 x *Clock_Divider).* The *Clock_Divider[8:0]* valid values are in the range from 1 to 511. Refer to Table 5-3 for select clock divider settings.
				7:1	Reserved
0x04	Reg_04	R/W	0x39		Integration Time Byte 0
				7:0	*Integration_Time[7:0]*
0x05	Reg_05	R/W	0x03		Integration Time Byte 1
				7:0	*Integration_Time[15:8]*
0x06	Reg_06	R/W	0x00		Integration Time Byte 2
				7:0	*Integration_Time[23:16]*
0x07	Reg_07	R/W	0x00		Integration Time Byte 3
				7:0	*Integration_Time[31:24]* Note: Writing to Reg_0x07 sets the *Integration_Time[31:0]* to the values of registers Reg_07, Reg_06, Reg_05 and Reg_04. Writing to Reg_05, Reg_06 or Reg_07 alone does not change the value of the *Integration_Time[31:0].* The units are periods of Odd/Even Clock Rate in accordance with Table 5-3.

Table 5-2 Memory Space/Register Definitions

Address	Name	Access	Default	Bit/Field	Register Name/Field Description
0x08	Reg_08	R/W	0xD1		AD5235 Digital Potentiometer 1 Low write data
				7:0	*Digital_Pot1[7:0]*
0x09	Reg_09	R/W	0x00		AD5235 Digital Potentiometer 1 High write data
				1:0	*Digital_Pot1[9:8]*
				7:2	Reserved
0x0A	Reg_0A	R/W	0xD9		AD5235 Digital Potentiometer 2 Low write data
				7:0	*Digital_Pot2[7:0]*
0x0B	Reg_0B	R/W	0x00		AD5235 Digital Potentiometer 2 High write data
				1:0	*Digital_Pot2[9:8]*
				7:2	Reserved
0x0C	Reg_0C	R/O	-		AD7991 A/D Ch0 Low
				7:0	*ADC_Ch0[7:0]*
0x0D	Reg_0D	R/O	-		AD7991 A/D Ch0 High
				3:0	*ADC_Ch0[11:8]*
				7:4	Reserved
0x0E	Reg_0E	R/O	-	3:0	AD7991 A/D Ch1 Low
				7:0	*ADC_Ch1[7:0]*
0x0F	Reg_0F	R/O	-		AD7991 A/D Ch1 High
			-	3:0	*ADC_Ch1[11:8]*
			-	7:4	Reserved
0x10	Reg_10	R/O	-		AD7991 A/D Ch2 Low
				7:0	*ADC_Ch2[7:0]*
0x11	Reg_11	R/O	-		AD7991 A/D Ch2 High
				3:0	*ADC_Ch2[11:8]*
				4:2	Reserved
0x12	Reg_12	R/W	0x00		Pattern Generator Control
				0	1 = Pattern Generator Enable 0 = Pattern Generator Disable

Table 5-2 Memory Space/Register Definitions

Address	Name	Access	Default	Bit/Field	Register Name/Field Description
				7:1	Reserved
0x13	Reg_13	R/W	0x03		Slave FIFO Control Register
				3:0	The number of 16-bit words output to the USB processor following the falling edge of FLAGB. The number of words resultant = 1 + the value of this field.
				7:4	Reserved
0x14	Reg_14	R/W	0x5A		AD5627 DAC Low
				7:0	*DAC[7:0]*
0x15	Reg_15	R/W	0x04		AD5627 DAC High
				3:0	*DAC[11:8]*
				7:4	*Reserved*
0x16	Reg_16	R/O	-		AD5235 Digital Potentiometer 1 Low read data
				7:0	*Digital_Pot1_Read[7:0]*
0x17	Reg_17	R/O	-		AD5235 Digital Potentiometer 1 High read data
				1:0	*Digital_Pot1_Read[9:8]*
				7:2	Reserved
0x18	Reg_18	R/O	-		AD5235 Digital Potentiometer 2 Low read data
				7:0	*Digital_Pot2_Read[7:0]*
0x19	Reg_19	R/O	-		AD5235 Digital Potentiometer 2 High read data
				1:0	*Digital_Pot2_Read[9:8]*
				7:2	Reserved
0x1A-0xFF					Reserved

Table 5-3 Select Clock Divider Settings

Clock_Divider[8:0]		Odd/Even Clock Rate (KHz)	Actual Pixel Rate (KHz)
Decimal	Binary		
511	111111111	58.708	7.339
500	111110100	60.000	7.500
200	011001000	150.000	18.750
100	001100100	300.000	37.500
67	001000011	447.761	55.970
50	000110010	600.000	75.000
40	000101000	750.000	93.750
37	000100101	789.474	98.684
38	000100110	810.811	101.351
33	000100001	909.091	113.636
29	000011101	1034.483	129.310
25	000011001	1200.000	150.000
22	000010110	1363.636	170.455
20	000010100	1500.000	187.500
18	000010010	1666.667	208.333
17	000010001	1764.706	220.588
15	000001111	2000.000	250.000

Table 5-3 Select Clock Divider Settings			
Clock_Divider[8:0]		Odd/Even Clock Rate (KHz)	Actual Pixel Rate (KHz)
Decimal	Binary		
14	000001110	2142.857	267.857
13	000001101	2307.692	288.462
12	000001100	2500.000	312.500
11	000001011	2727.273	340.909
10	000001010	3000.000	375.000
8	000001000	3750.000	468.750
7	000000111	4285.714	535.714
1	000000001	30000.000	3750.000

Clock Divider

Odd/Even Clock rate is derived from 60MHz clock as follows:
Odd/Even Clock Rate = 60MHz ÷ (2×Clock_Divider)

The table above provides some examples of the clock divider settings. All values of Clock_Divider[8:0] in the range from 1 to 511 are valid, resulting in the achievable pixel clock in the range from 7.339KHz to 3.750MHz, and the corresponding Odd/Even Clock Rate in the range from 58.708KHz to 30.000MHz. *Note: Selecting a clock divider value outside of the valid range will result in the value being ignored. The maximum operation frequency of the sensor is specified as 4MHz.*

Integration Time

Integration time is derived based on Odd/Even Clock Rate and the Integration_Time value as follows:
Actual Integration Time = Integration_Time ÷ Odd/Even Clock Rate

Setting Integration_Time[31:0] to 0x00000008 with Clock_Divider[8:0] = 511_{10} results in integration time being: 8 / 58.708KHz = 136.266µsec

Max Integration Time

Setting Integration_Time[31:0] to 0xFFFFFFFF and Clock_Divider[8:0] = 1 results in integration time being: $(2^{32}-1)$ / 30.000MHz = 143.166sec

Min Integration Time

Setting Integration_Time[31:0] to 0x00000001 and Clock_Divider[8:0] = 511_{10} results in integration time being: 1 / 58.708KHz = 136.267µsec

FPGA Design Description

Section 6

The FPGA design is based on Intel (Altera) Cyclone III device EP3C25F324C8. The FPGA design is written in VHDL. The design consists of the major blocks listed in Table 6-1.

Table 6-1 List of the FPGA Modules

#	Module Name	Source File Name	Description
1	top	top.vhd	Top level of the design hierarchy
2	clock_reset_gen	clock_reset_gen.vhd	Clock and reset generator
3	control	control.vhd	I2C slave control interface
4	regs	regs.vhd	Control register bank
5	detector_con	detector_con.vhd	Sensor controller
6	adc_con	adc_con.vhd	Controller responsible for U1 (ADAQ7980BCCZ) control and for U4 (AD5235BRUZ25) digital potentiometer control via SPI bus on the ADI circuit card
7	slavefifo2b_streamin	slavefifof2b_streamin.vhd	Slave FIFO interface to EZ-USB device
8	ad5627_con	ad_5627_con.vhd	Controller responsible for temperature set-point DAC U9 (AD5627RBRMZ-1) on the ADI circuit card
9	ad7991_con	ad_7991_con.vhd	Controller responsible for temperature monitoring ADC U10 (AD7991YRJZ-1500) on the ADI circuit card
10	mpac	mpac.vhd	Multi-Port Access Controller (MPAC) provides shared access of ad_5627_con and ad7991_con to the common I2C bus through I2C_master module
11	I2C_master	I2C_master.vhd	Provides I2C bus master functionality needed to access I2C slave peripherals on the ADI circuit card
12	adc_xf	adc_xf.vhd	Provides low-level control of the U1 (ADAQ7980BCCZ) ADC
13	ddr	ddr.vhd	Altera Megafunction IP providing DDR I/O functionality
14	I2C_xf	I2C_xf.vhd	I2C Slave interface, used as a part of control module
15	cbus_con	cbus_con.vhd	As a part of control module, this serves as read and write access bridge between the I2C slave interface and the register bank module (regs)
16	tx_fifo	tx_fifo.vhd	This is a FIFO that receives its data from adc_con and makes it available to the slavefifof2b_streamin module for transfer to the USB processor.
17	pll	pll.vhd	Altera Megafunction IP providing PLL functionality used within clock_reset_gen module
18	N/A	custom.vhd	This serves as the design library. It contains package "custom" which captures commonly used functions, constants and data types.

The structure of the FPGA design is shown in Figure 6-1.

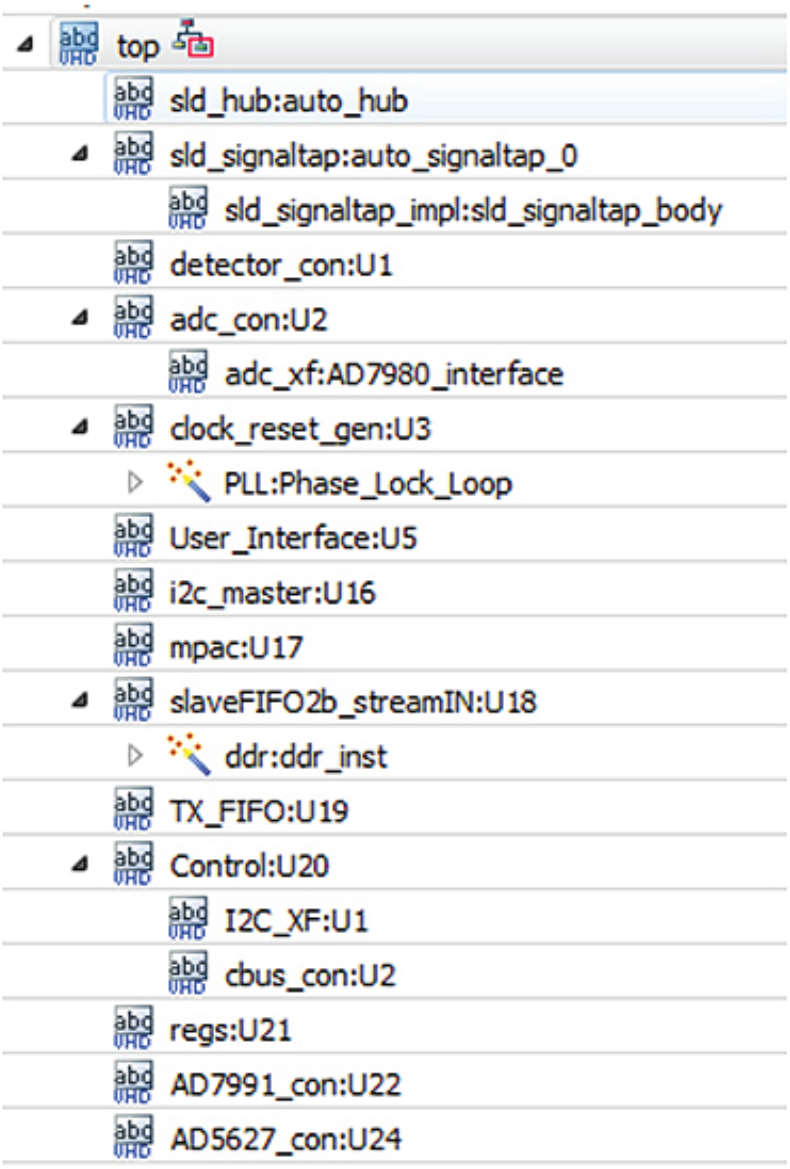


Figure 6-1: FPGA design structure

6A. clock_reset_gen.vhd

Module Inputs and Outputs

Table 6-2 Module I/Os

Port Name	Type	Direction	Description
clk_in	std_logic	in	50 MHz clock input
reset_n	std_logic	in	Active-low, asynchronous reset input
Clock_Divider	std_logic_vector (8 down to 0)	in	Controls the rate at which clk_en_1x and clk_en_2x are generated
rst_n	std_logic	out	Active-low, reset output, synchronously de-asserted, asynchronously asserted
clk	std_logic	out	60 MHz clock
clk_del	std_logic	out	60 MHz clock, synchronous to clk, 0 delayed
clk2x	std_logic	out	120 MHz clock, synchronous and aligned to clk
clk_en_1ms	std_logic	out	Active-high, single clk cycle wide pulse, every 1 msec
clk_en_10ms	std_logic	out	Active-high, single clk cycle wide pulse, every 10 msec
clk_en_100ms	std_logic	out	Active-high, single clk cycle wide pulse, every 100 msec
clk_en_1x	std_logic	out	Active-high, single clk cycle wide pulse, occurring at a rate of 60 MHz/(2×Clock_Divider)
clk_en_2x	std_logic	out	Active-high, single clk cycle wide pulse, occurring at a rate of 60 MHz/(1×Clock_Divider)

Module IP

Altera ALTPLL Megafunction: PLL, shown in Figure 6-2, generates 60MHz, 120MHz, and 60MHz clocks based on 50MHz clock input.

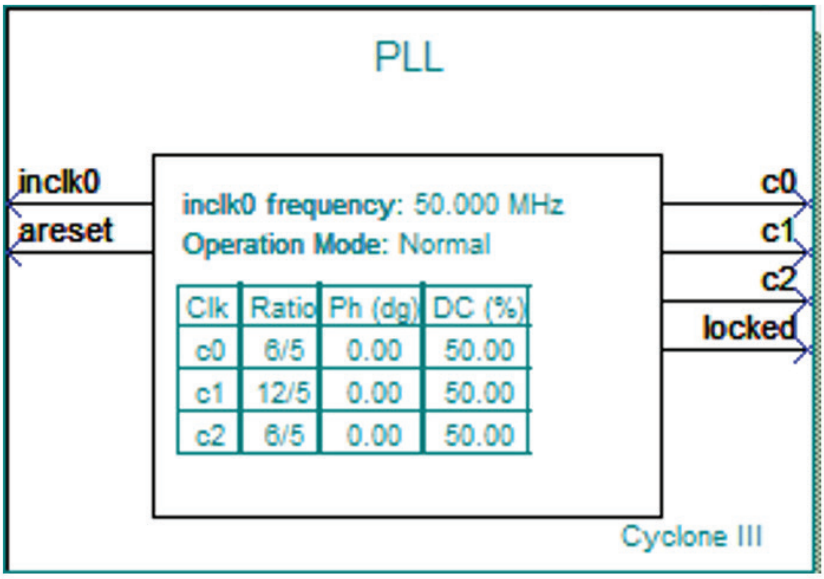


Figure 6-2: Altera ALTPLL megafunction IP providing PLL functionality

Module Description

This module receives 50MHz clock from the Altera Cyclone-III Starter Board (DK-START-3C25N). The received clock is provided to the PLL based on ALTPLL Megafunction IP provided by Altera. The resultant clocks clk, clk_2x and clk_del are generated.

The module receives an asynchronous reset signal “reset_n” and creates asynchronously asserted, synchronously to “clk” de-asserted, active-low reset output “rst_n”.

The module generates various clock enable outputs (programmable ones based on Clock_Divider: clk_en_1x and clk_en_2x; fixed ones: at 1msec, at 10msec, at 100msec intervals).

6B. control.vhd

Module Inputs and Outputs

Table 6-3 Module I/Os

Port Name	Type	Direction	Description
clk_sys	std_logic	in	60 MHz clock input
rst_n	std_logic	in	Active-low, asynchronously reset, synchronously set reset input
CBUS_CON_Di	std_logic_vector (7 down to 0)	in	Read Data bus
SCL	std_logic	in	I2C clock
SDA	std_logic	in, out	I2C data
CBUS_CON_A	std_logic_vector (7 down to 0)	out	Register address
CBUS_CON_Do	std_logic_vector (7 down to 0)	out	Register write data
CBUS_CON_WRS	std_logic	out	Register write strobe, active-high
CBUS_Read	std_logic	out	Register read strobe, active-high

Module Description

The module combines an I2C slave module (I2C_xf) with cbus_con module allowing to perform a write or a read access to a bank of registers connected external to control module. Refer to Figures 6-3 and 6-4.

During I2C write transfers the I2C_xf receives one byte at a time. A typical transfer consists of 2 payload bytes: register address byte, followed by register data byte.

Each received data payload byte is presented at recv_data(7:0) output of the I2C_xf module, while being accompanied by ser_load_en active-high single clock cycle pulse. This indicates to the CBUS controller (cbus_con) that the data is to be stored.

first_byte output from I2C_xf indicates whether the data byte presented to cbus_con represents register address or register data. If first_byte is asserted (active-high single clock cycle pulse) concurrently with data_vld output of I2C_xf, then the data byte is register address, else the data byte is register data to be stored at the respective address location.

During I2C read transfers the I2C_xf module transfers the data presented at xmit_dat(7:0) input onto the I2C bus.

A typical register write transfer is shown in Figure 6-3.

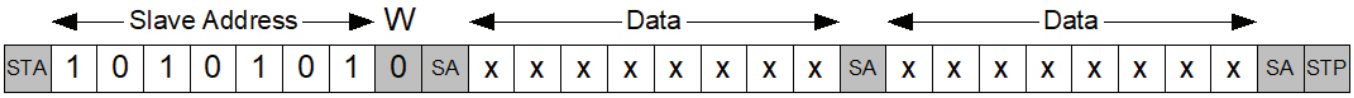


Figure 6-3: Typical I2C write transfer. STA = Start, SA = Slave ACK, STP = Stop, W = Write, X = any value (1 or 0)

The first data byte serves as register address, while the second data byte serves as register write data. A typical register read transfer is shown in Figure 6-4.

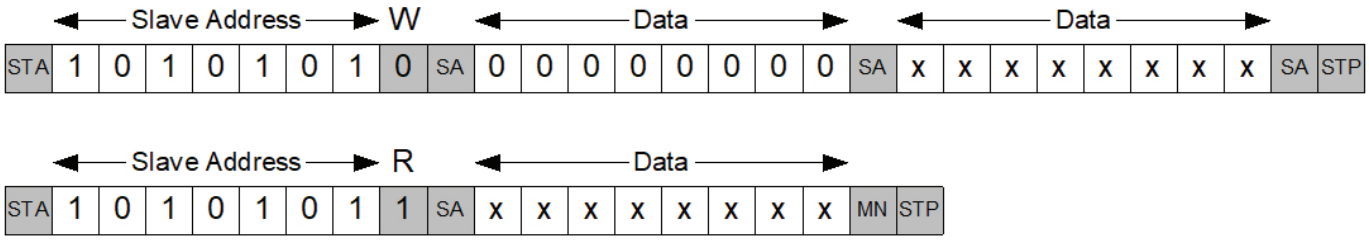


Figure 6-4: Typical I2C read transfer. MN = Master NACK, R = Read

A read transaction consists of two separate transfers:

1) The first is a write transfer with first data byte having all zeros, and the second data byte used to indicate the register address to be accessed for a subsequent read transfer.
2) The second is a read transfer with data byte returned by the slave_xf containing the data corresponding to the register address presented during the first step.

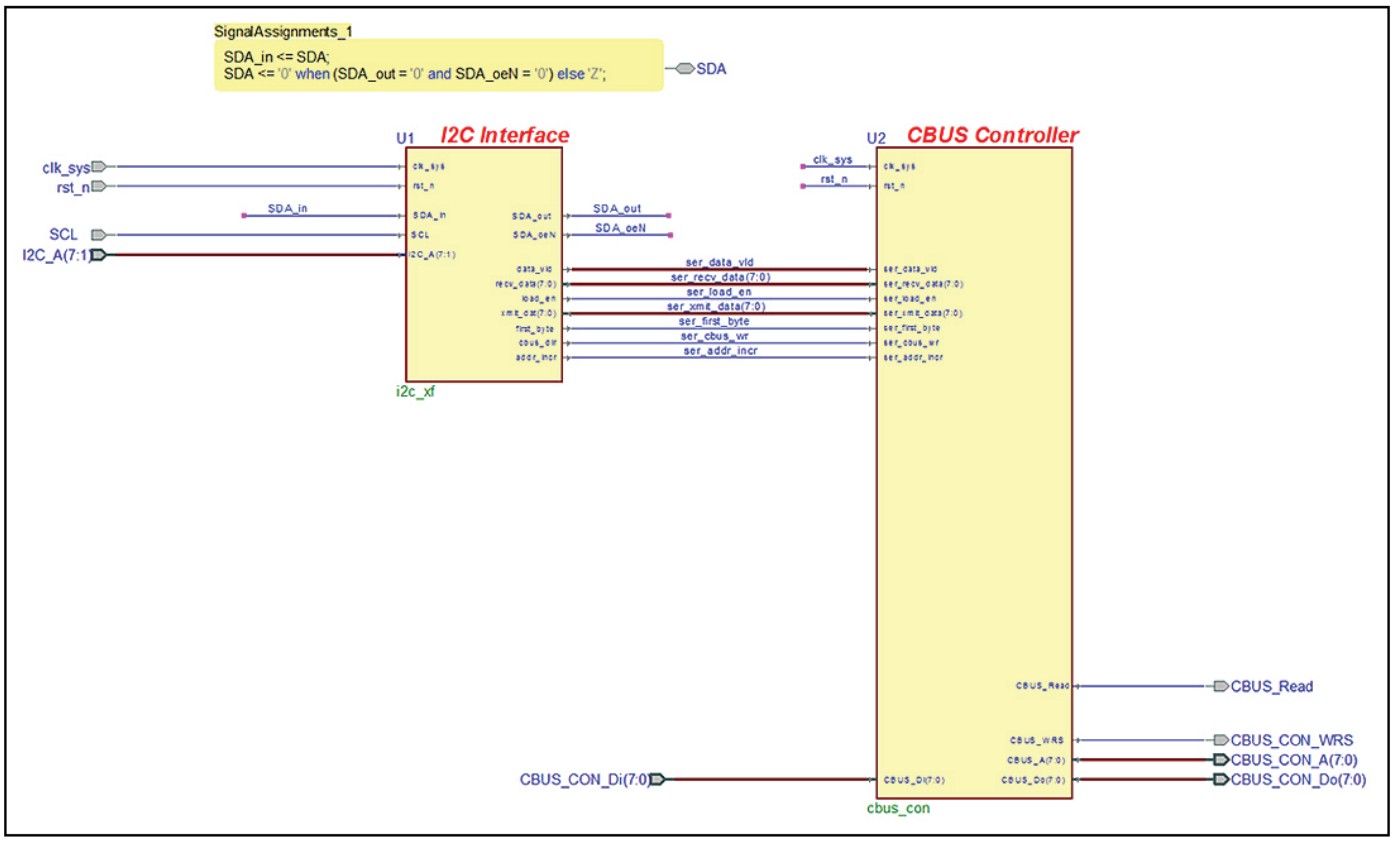


Figure 6-5: Control module architecture

6C. I2C_xf.vhd

Module Inputs and Outputs

Table 6-4 Module I/Os

Port Name	Type	Direction	Description
clk_sys	std_logic	in	60 MHz clock input
rst_n	std_logic	in	Active-low, asynchronously reset, synchronously set reset input
SDA_in	std_logic	in	I2C SDA input
SDA_out	std_logic	in	I2C SDA output
SDA_oeN	std_logic	in	I2C SDA output enable, active-low
SCL	std_logic	in	I2C SCL input
I2C_A	std_logic_vector (7 down to 1)	in	I2C Slave address
data_vld	std_logic	out	Active-high, single clk_sys wide pulse, indicating that recv_data(7:0) is valid
recv_dat	std_logic_vector (7 down to 0)	out	Data payload byte received via I2C
load_en	std_logic	out	Indicates that the data byte has been loaded into transmit register (used for read transfers)
xmit_dat	std_logic_vector (7 down to 0)	in	Data to be transmitted on I2C bus
first_byte	std_logic	out	Indicates the first data payload byte within an I2C write stream
cbus_dir	std_logic	out	CBUS direction transfer: 0 = Write, 1 = Read
addr_incr	std_logic	out	Used during read transfers, this active-high signal produces a single clk_sys cycle wide pulse each time a payload byte has been transferred via I2C interface.

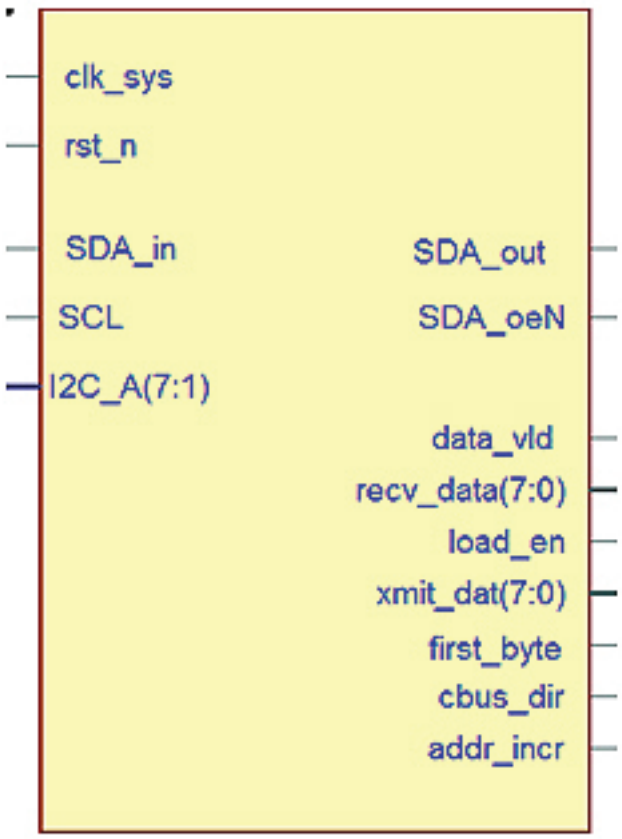


Figure 6-6: I2C module I/Os

Module Description

This module implements the physical access to the I2C bus. The module acts as an I2C slave with slave address passed via I2C_A port. The module responds to 8-bit slave address of 0xAA/0xAB. The I2C interface supports I2C clock rates of up to 400KHz.

The I2C module design is based on a finite state machine (FSM). The FSM is shown in Figure 6-7.

The FSM dwells in IDLE state until start is detected on the I2C bus. The FSM proceeds through A7_ST to A0_ST states receiving one address bit at a time with each falling edge of SCL. Upon arriving to ADDR state the received address is compared to the device address (0xAA/0xAB). If the upped 7 bits match, the slave asserts acknowledge and proceeds to receive a data byte (states D7_ST through D0_ST). Each bit of the data byte is received on the falling edge of the SCL line.

SCL_fedge represents a falling edge of I2C SCL clock received by the FPGA. All other conditional signals are self-explanatory.

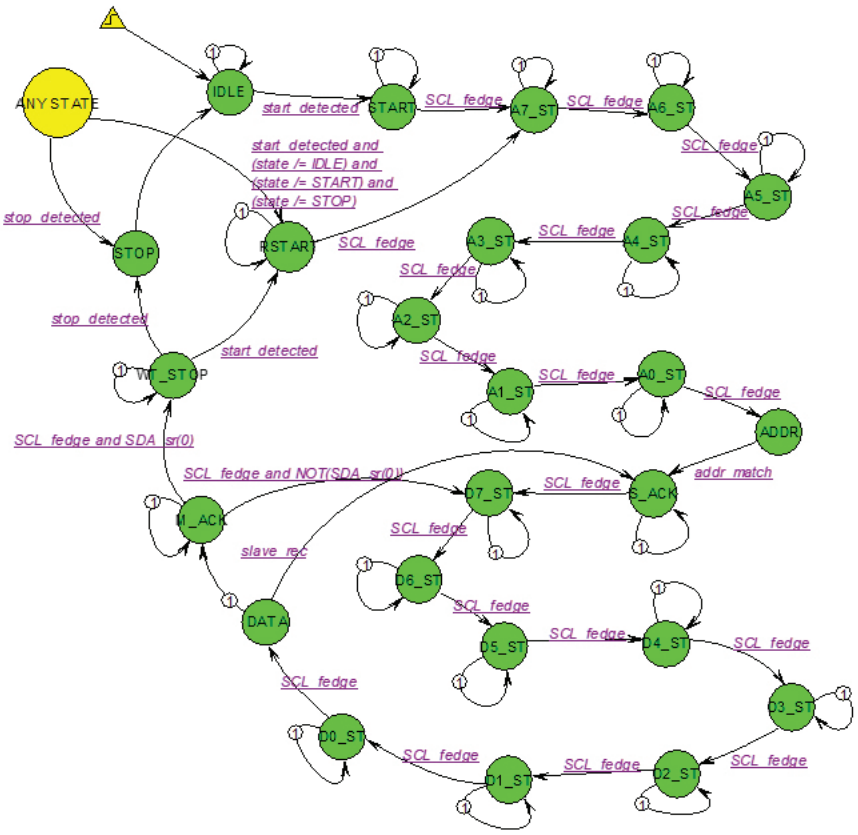


Figure 6-7: I2C state machine

6D. cbus_con.vhd

Module Inputs and Outputs

Table 6-5 Module I/Os

Port Name	Type	Direction	Description
clk_sys	std_logic	in	60 MHz clock input
rst_n	std_logic	in	Active-low, asynchronously reset, synchronously set reset input
ser_data_vld	std_logic	in	Single clock wide, active-high, signal indicating that a serial byte has been received and it is ready to be stored
ser_addr_incr	std_logic	in	For multi-byte transfers (write or read), this single clock cycle wide, active-high signal indicates that the next register address access should be started
ser_first_byte	std_logic	in	Single clock wide, active-high, signal indicating that the byte being received is the first byte, which represents CRDR address, while the second byte is the read register address. This signal is used for read transfers only.
ser_cbus_wr	std_logic	in	Single clock wide, active-high, signal indicating a CBUS write transaction
ser_recv_data	std_logic_vector (7 down to 1)	in	A data byte received from the I2C interface
ser_load_en	std_logic	in	Active-high, single clk_sys wide pulse, indicating used in conjunction with ser_addr_inc signal to cause an address on the CBUS to be incremented
ser_xmit_data	std_logic_vector (7 down to 0)	out	Data to be transmitted on the I2C bus
CBUS_Read	std_logic	out	Active-high, single clk_sys wide pulse, used a read enable on the CBUS
CBUS_WRS	std_logic	out	Active-high, single clk_sys wide pulse, used a write enable on the CBUS
CBUS_A	std_logic_vector (7 down to 0)	out	CBUS address
CBUS_Do	std_logic_vector (7 down to 0)	out	CBUS write data
CBUS_Di	std_logic_vector (7 down to 0)	in	CBUS read data

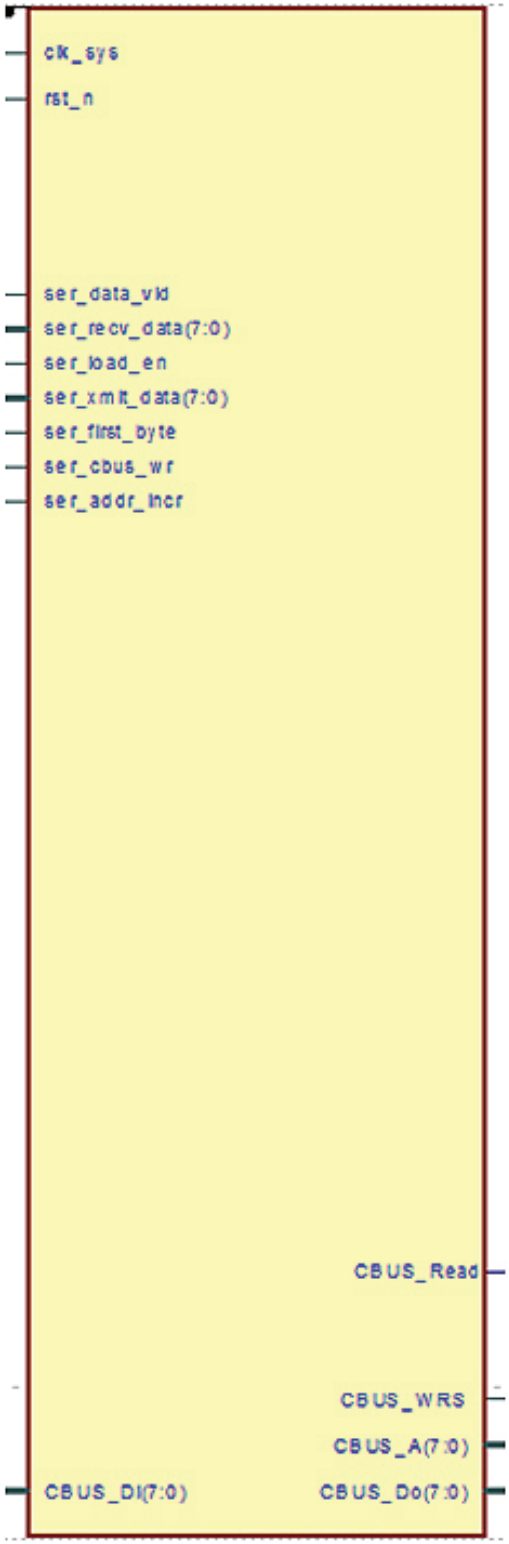


Figure 6-8: CBUS_CON module I/Os

cbus_con module receives raw bytes from the I2C_xf module and interprets them into register read and write accesses. Once interpreted, the module produces address (CBUS_A), write data (CBUS_Do) and write strobe (CBUS_WRS) signals to the downstream register bank for a write transaction; or (CBUS_A) and read strobe (CBUS_Read) for a read transaction. The read data is CBUS_Di.

The module also transfers the read data back to I2C_xf module for a transfer back to I2C master via the serial bus.

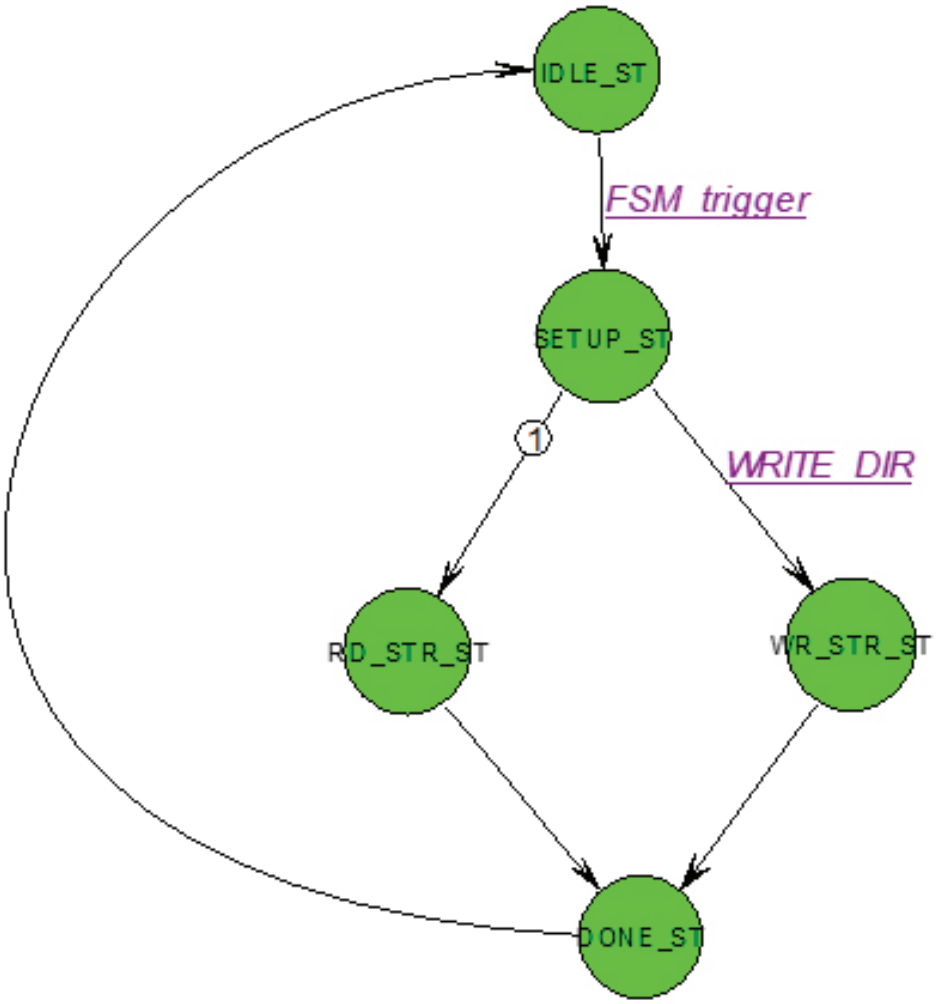


Figure 6-9: CBUS_CON state machine

6E. regs.vhd

Module Inputs and Outputs

Table 6-6 Module I/Os

Port Name	Type	Direction	Description
clk_sys	std_logic	in	60 MHz clock input
rst_n	std_logic	in	Active-low, asynchronously reset, synchronously set reset input
CBUS_Read	std_logic	in	Active-high, single clk_sys wide pulse, used a read enable on the CBUS
CBUS_WRS	std_logic	in	Active-high, single clk_sys wide pulse, used a write enable on the CBUS
CBUS_A	std_logic_vector (7 down to 0)	in	CBUS address
CBUS_Do	std_logic_vector (7 down to 0)	in	CBUS write data
CBUS_Di	std_logic_vector (7 down to 0)	out	CBUS read data
iREG_0C through iREG_19	std_logic_vector (7 down to 0)	in	These represent inputs for the readable registers at addresses 0x0C through 0x19
REG_00 through REG_2F	std_logic_vector (7 down to 0)	out	These mirror the contents of the writable registers at addresses 0x00 through 0x2F

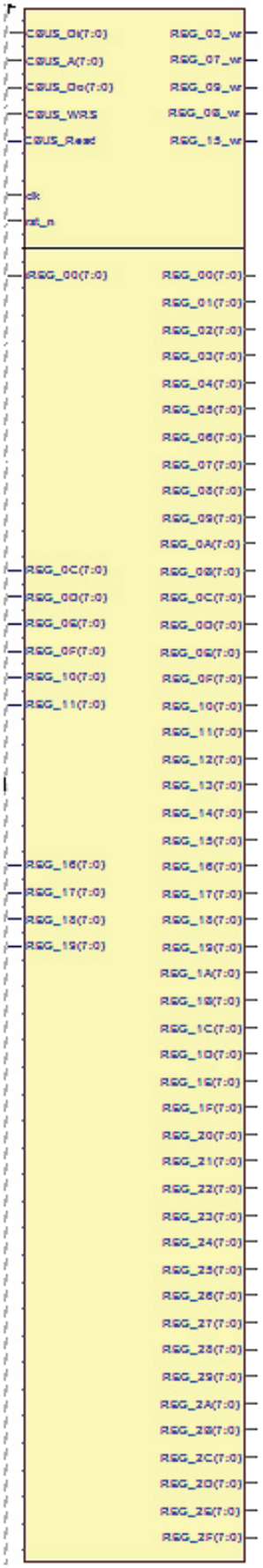


Figure 6-10: regs module I/Os

Module Description

regs module provides access to the contents of the registers at addresses 0x00 through 0x2F.

When a read transfer in the range from 0x0C to 0x19 is requested, the contents of iREG_0C through iREG_19 is used.

Read access to the address range 0x0C to 0x11 is mapped to AD7991 ADC channel 0 through 2.

REG_00 to REG_2F provide control over the various functions of the FPGA.

6F. detector_con.vhd

Module Inputs and Outputs

Table 6-7 Module I/Os

Port Name	Type	Direction	Description
clk_sys	std_logic	in	60 MHz clock input
rst_n	std_logic	in	Active-low, asynchronously reset, synchronously set reset input
clk_en_2x	std_logic	in	Active-high, clock enable at a rate equal to 60 MHz/Clock_Divider (refer to Table 6-2) The default value of Clock_Divider is 10, resulting in 6 MHz rate of clk_en_2x
clk_en_1x	std_logic	in	Active-high, clock enable at a rate equal to 60 MHz/(2 x Clock_Divider) (refer to Table 6-2) The default value of Clock_Divider is 10, resulting in 3 MHz rate of clk_en_1x
op_en	std_logic	in	Active-high signal, when set enable the operation of the sensor controller
Integration_Time	std_logic_vector (31 down to 0)	in	This input controls the integration time in units of 60 MHz/(2 x Clock_Divider). The default value is 0x00000339, resulting in 275 µsec default integration time.
reset_odd	std_logic	out	Odd reset output to the image sensor
clk_odd	std_logic	out	Odd clock output to the image sensor
adtrig_odd	std_logic	in	ADC odd pixel trigger input from the image sensor
reset_even	std_logic	out	Even reset output to the image sensor
clk_even	std_logic	out	Even clock output to the image sensor
adtrig_even	std_logic	in	ADC even pixel trigger input from the image sensor
adtrig_odd_test	std_logic	out	Test output. Emulates the corresponding trigger input.
adtrig_even_test	std_logic	out	Test output. Emulates the corresponding trigger input.
pix_mode	std_logic	in	Pixel mode: 0 = 256 pixel mode (default) 1 = 512 pixel mode
hs_mode	std_logic	in	Sensor interface speed: 0 = Regular speed mode (default) 1 = High speed mode
par_mode	std_logic	in	Parallel mode: 0 = Sequential op mode (default) 1 = Parallel op mode (odd and even pixels are clocked concurrently and slower)

Table 6-7 Module I/Os

Port Name	Type	Direction	Description
mux_con_inv	std_logic	in	Pixel multiplexor control: 1 = Invert even_odd_n 0 = Do not invert (default)
even_odd_n	std_logic	out	Multiplexor control: 0 = Odd data select 1 = Even data select
sync	std_logic_vector (1 down to 0)	out	This signal is used to indicate to Slave FIFO interface the beginning of the integration phase of sensor control "01": Integration started "10": Integration started 1 cycle ago "00": Integration not started "11": Reserved/Invalid
adc_trigger	std_logic	out	Test output for ADC controller (emulates that of the image sensor)
last_xfer	std_logic	out	Active-high, single clk wide signal: 1: Marks the moment when the last pixel burst is starting
integr_start	std_logic	out	Active-high, single clk wide signal: 1: Marks the beginning of the integration interval
integr_end	std_logic	out	Active-high, single clk wide signal: 1: Marks the end of the integration interval

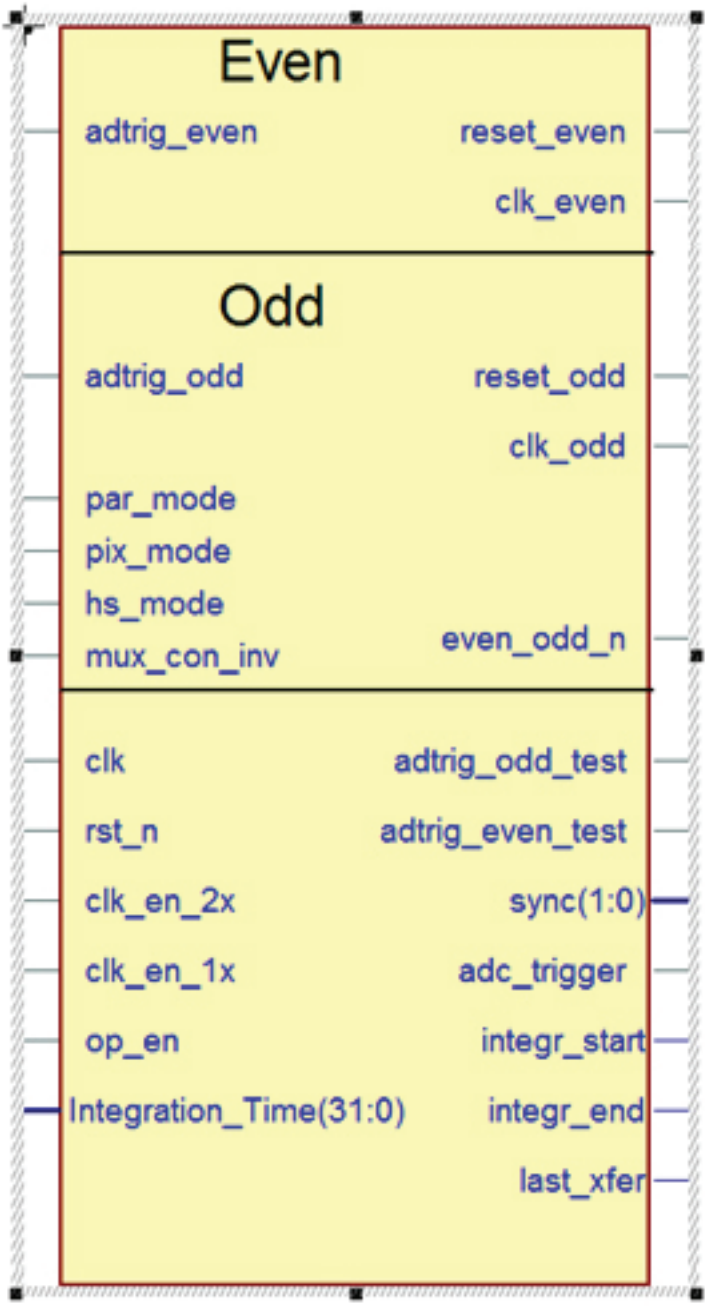


Figure 6-11: detector_con module I/Os

The detector controller module provides the logic and timing for the control signals needed to properly control a detector device under test.

The controller supports the following operating modes:

- 256 or 512 pixels
- Regular or High-speed operation
- Normal (Sequential) or Parallel operating mode

Figure 6-12 through Figure 6-28 demonstrate the relationship between the detector control signals in all possible operating modes.

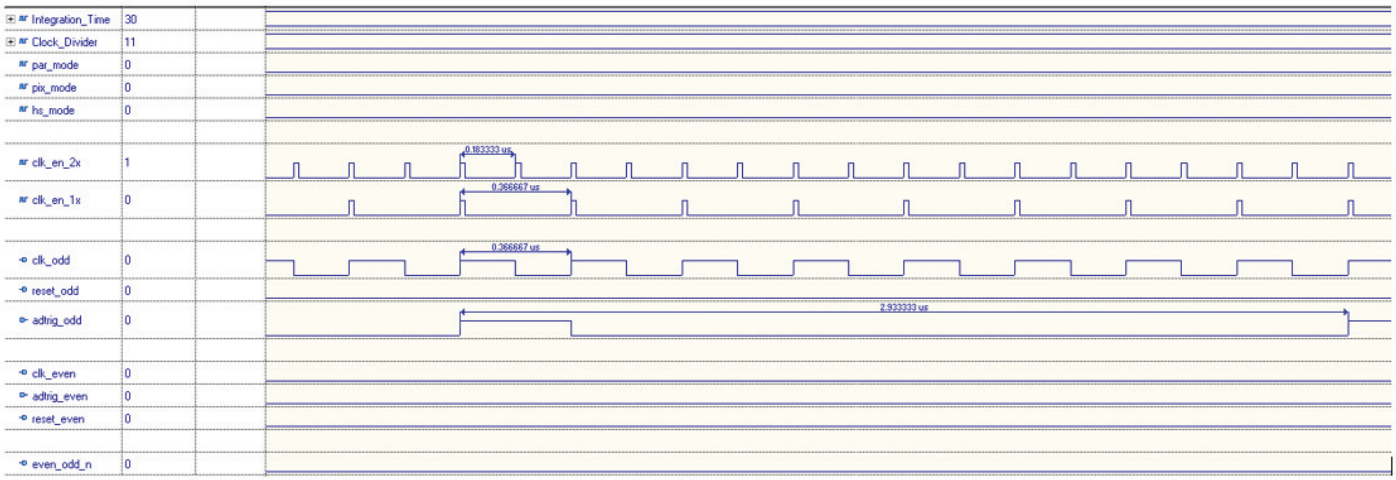


Figure 6-12: 256-pixel, regular speed, sequential mode timing

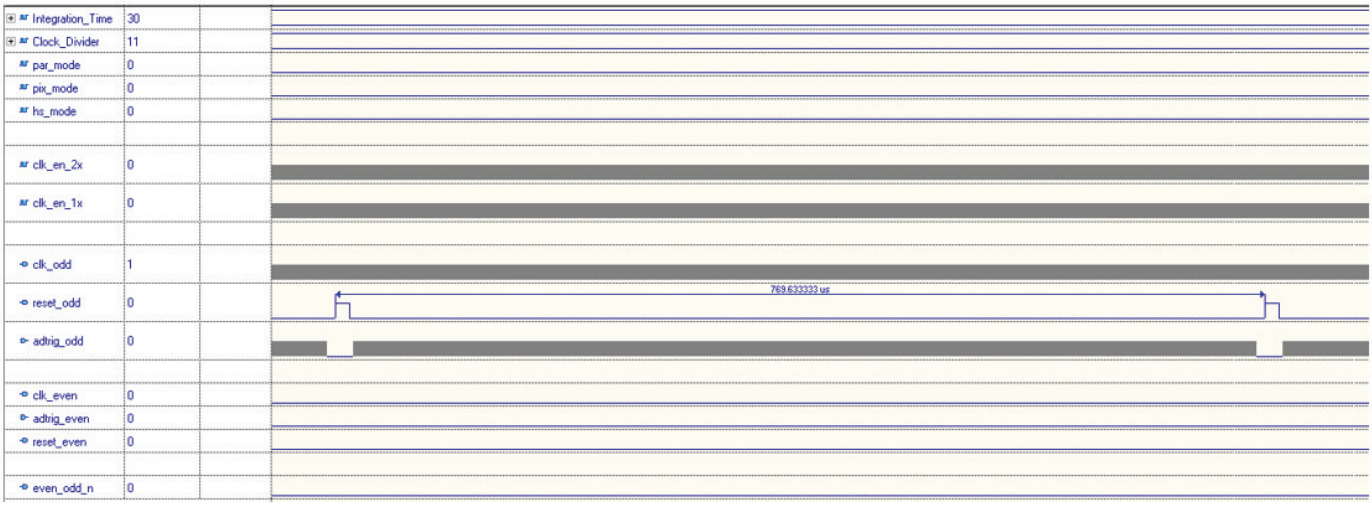


Figure 6-13: 256-pixel, regular speed, sequential mode operation

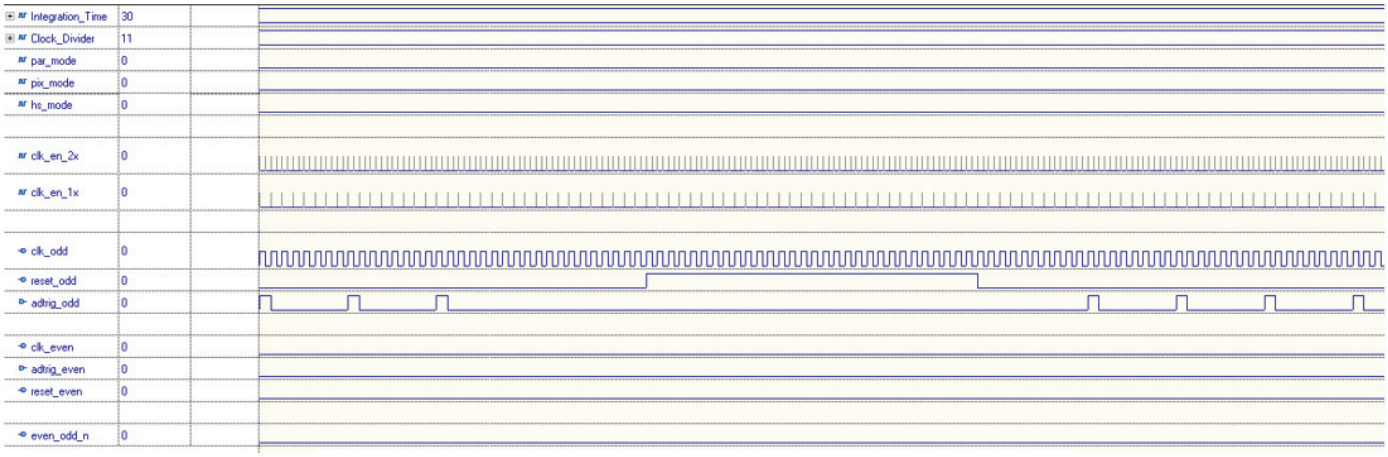


Figure 6-14: 256-pixel, regular speed, sequential mode operation around reset

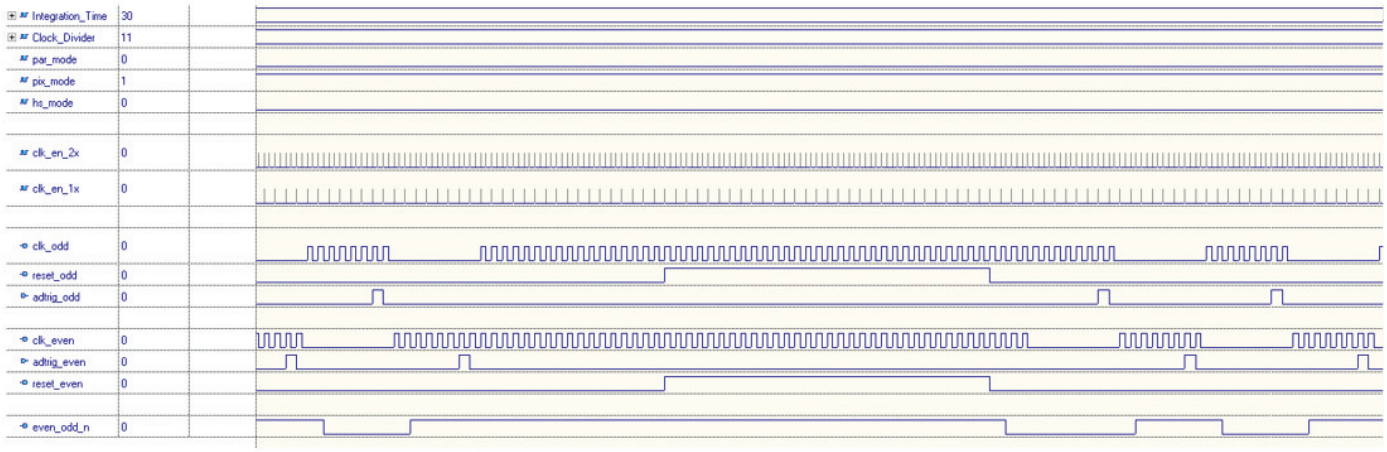


Figure 6-15: 512-pixel, regular speed, sequential mode operation around reset

Figure 6-16: 512-pixel, regular speed, sequential mode operation overview

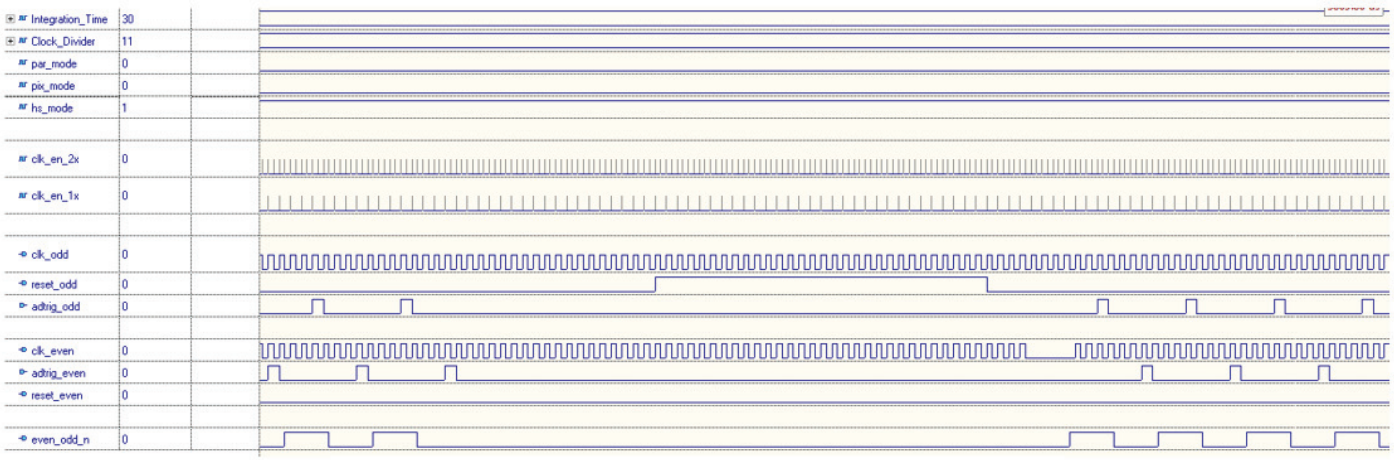


Figure 6-17: 256-pixel, high speed, sequential mode operation around reset

Figure 6-18: 256-pixel, high speed, sequential mode operation overview

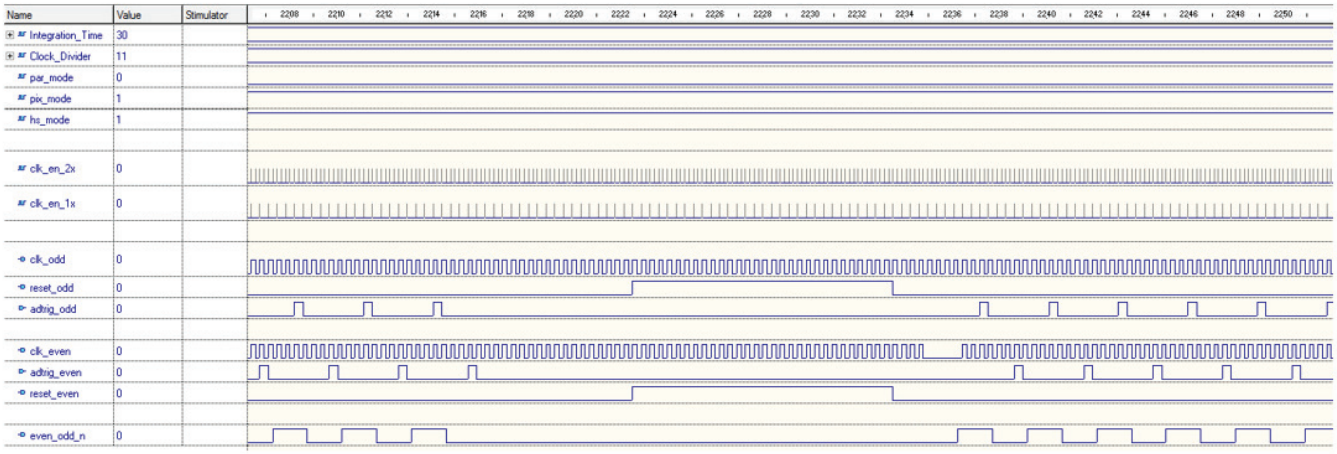


Figure 6-19: 512-pixel, high speed, sequential mode operation around reset

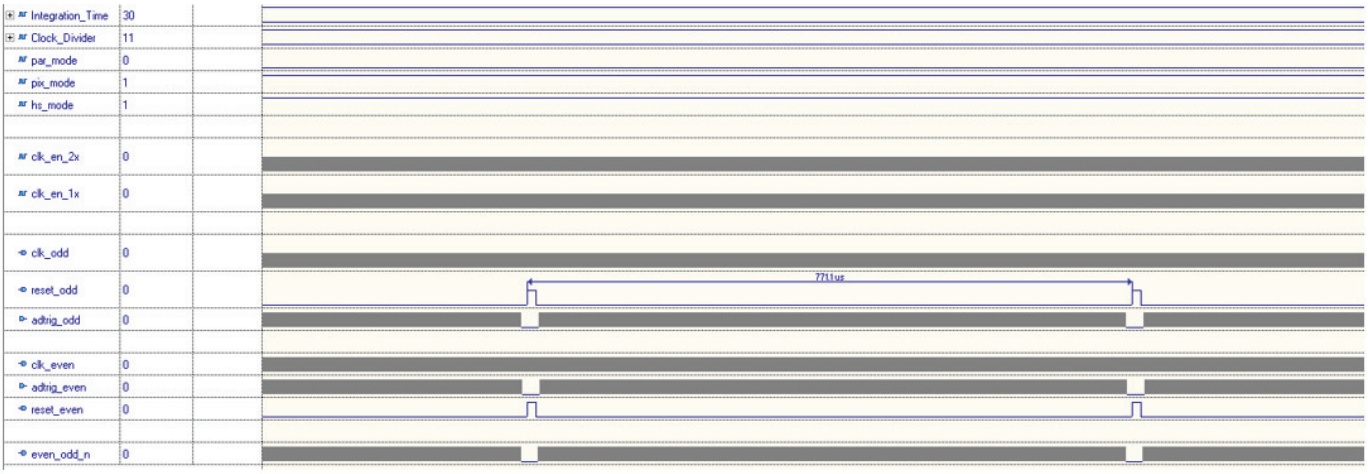


Figure 6-20: 512-pixel, high speed, sequential mode operation overview

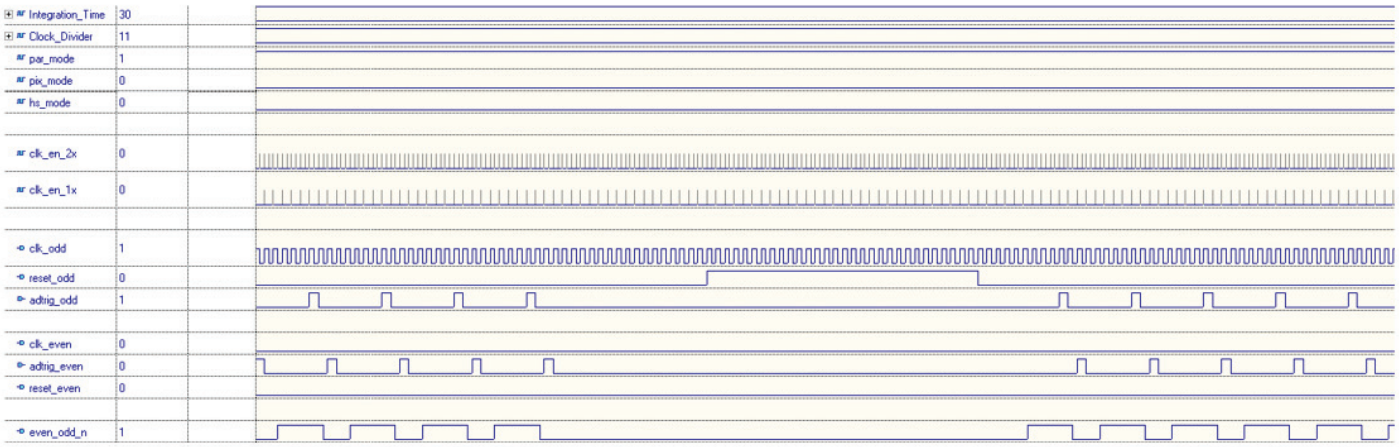


Figure 6-21: 256-pixel, regular speed, parallel mode operation around reset

Figure 6-22: 256-pixel, regular speed, parallel mode operation overview

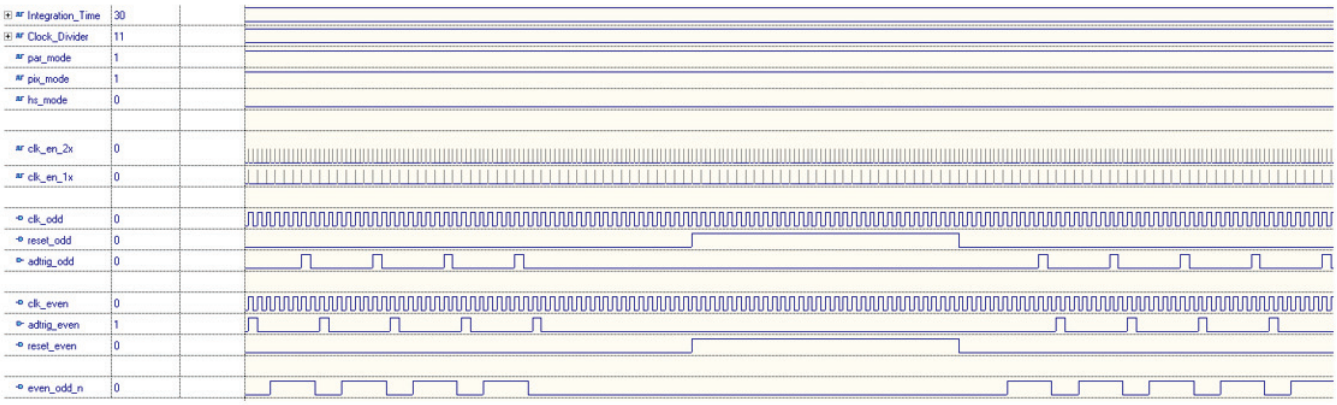


Figure 6-23: 512-pixel, regular speed, parallel mode operation around reset

Figure 6-24: 512-pixel, regular speed, parallel mode operation overview

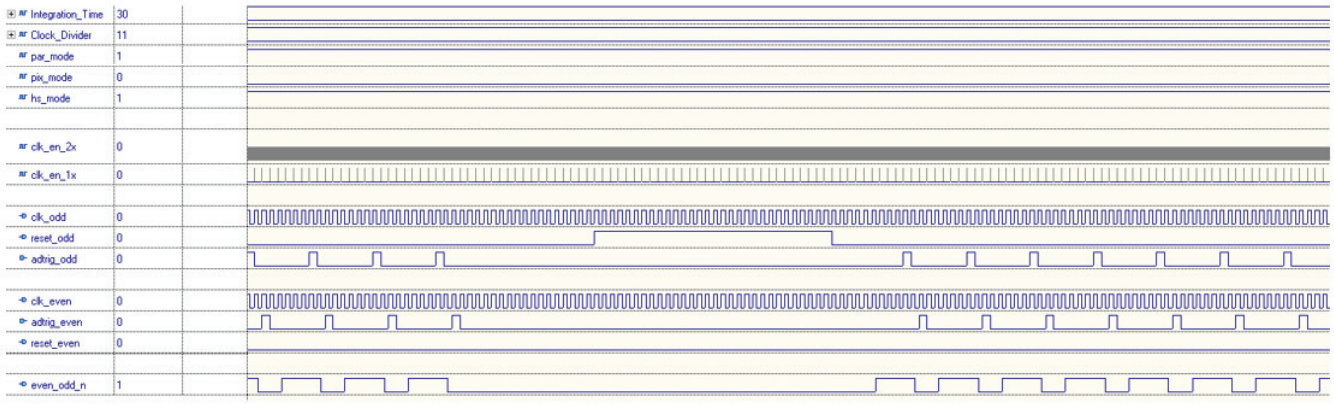


Figure 6-25: 256-pixel, high speed, parallel mode operation around reset

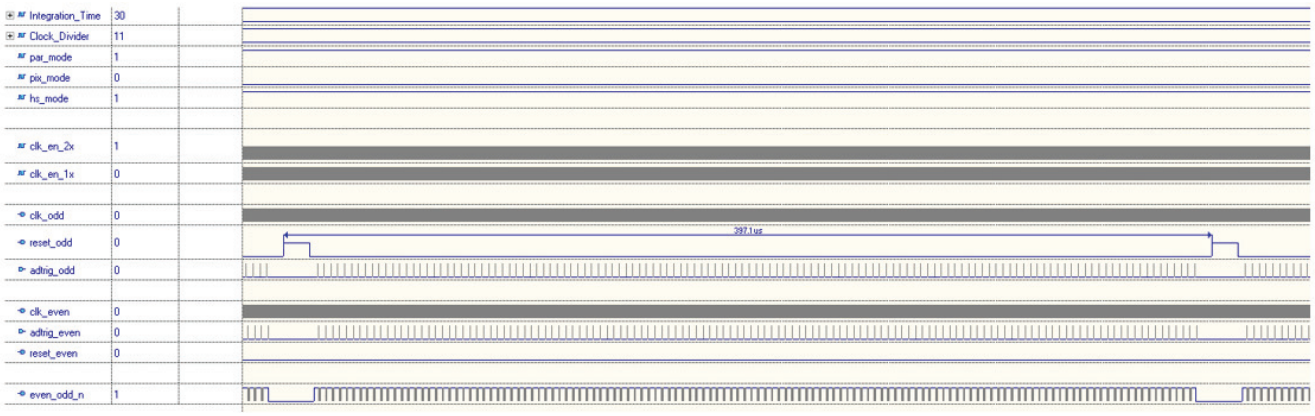


Figure 6-26: 256-pixel, high speed, parallel mode operation overview

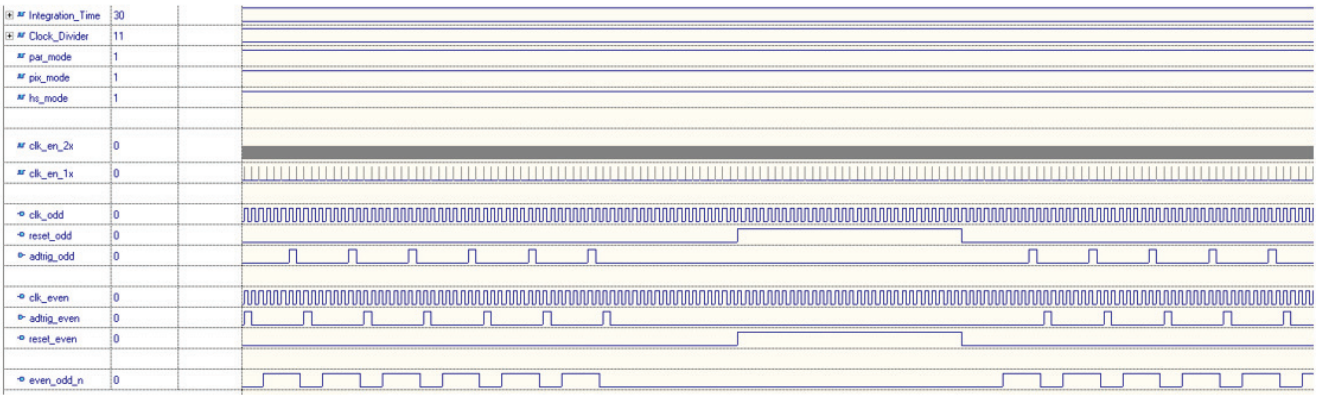


Figure 6-27: 512-pixel, high speed, parallel mode operation around reset

Figure 6-28: 512-pixel, high speed, parallel mode operation overview

The detector controller module design is based on an FSM (Finite State Machine) shown in Figure 6-29.

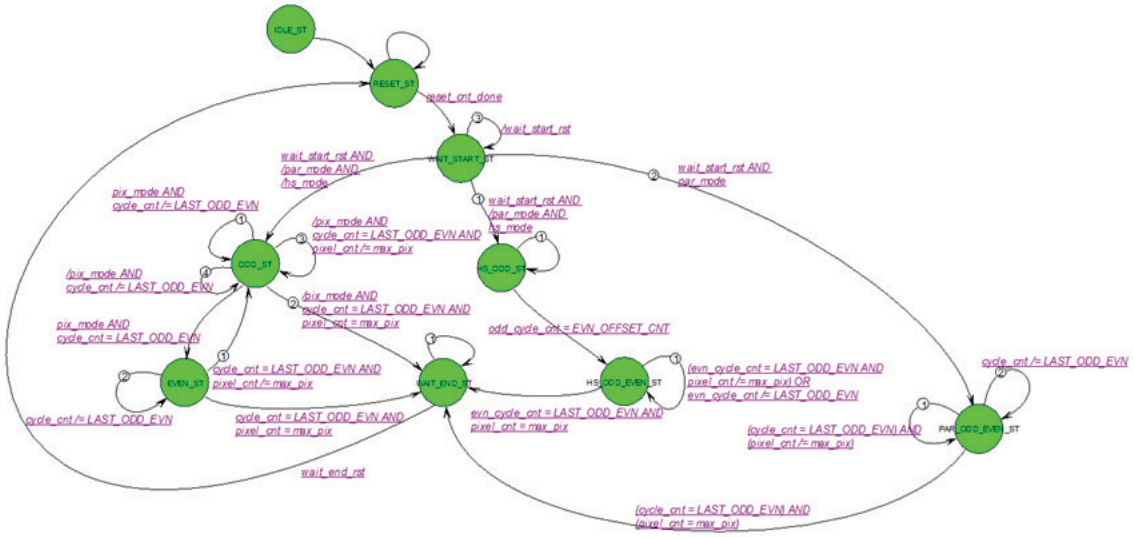


Figure 6-29: Detector controller state machine

6G. adc_con.vhd

Module Inputs and Outputs

Table 6-8 Module I/Os

Port Name	Type	Direction	Description
clk	std_logic	in	60 MHz clock input
clk_del	std_logic	in	60 MHz clock input
clk_2x	std_logic	in	60 MHz clock phase shifted by 0 deg from clk
rst_n	std_logic	in	Active-low, asynchronously reset, synchronously set reset input
clk_en	std_logic	in	100 msec clock enable, active-high, 1 clk long
mosi	std_logic	out	SPI data output (Master Out Slave In)
miso	std_logic	in	SPI data input (Master In Slave Out)
sclk	std_logic	out	SPI clock output
adc_conv	std_logic	out	ADAQ7980 CNV signal, active-high
digipot_cs_n	std_logic	out	Digital potentiometer AD5235 Chip Select, active-low
digipot_rdy	std_logic	in	Digital potentiometer AD5235 read data ready, active-high, 1 clk wide
digipot1_wr	std_logic	in	AD5235 digital potentiometer 1 write command, active-high, 1 clk wide
digipot2_wr	std_logic	in	AD5235 digital potentiometer 2 write command, active-high, 1 clk wide
RDAC1	std_logic_vector(9:0)	in	AD5235 digital potentiometer 1 write data
RDAC2	std_logic_vector(9:0)	in	AD5235 digital potentiometer 2 write data
RDAC1_rd_data	std_logic_vector(9:0)	out	AD5235 digital potentiometer 1 read data
RDAC2_rd_data	std_logic_vector(9:0)	out	AD5235 digital potentiometer 2 read data
pg_en	std_logic	in	Pattern Generator Enable, active-high
last_xfer	std_logic	in	Last Transfer indicator, active-high
integr_end	std_logic	in	Integration End indicator, active-high, 1 clk wide
integr_start	std_logic	in	Integration Start indicator, active-high, 1 clk wide
adc_data	std_logic_vector(15:0)	out	ADAQ7980 data output
data_rdy	std_logic	out	ADAQ7980 data ready, active-high, 1 clk wide
adc_trigger	std_logic	in	Trigger for ADAQ7980 ADC conversion, active-high, 1 clk wide

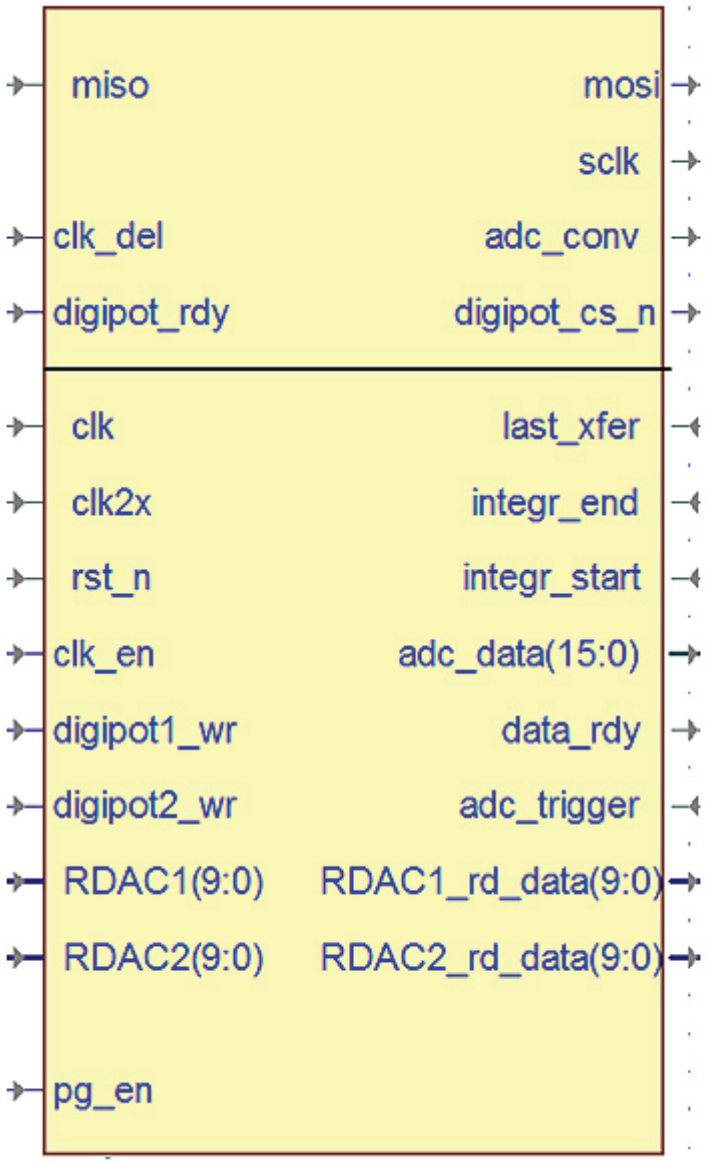


Figure 6-30: adc_con module I/Os

Module Description

adc_con module serves as an interface between the FPGA and the two Analog Devices ICs sharing the same SPI bus: ADAQ7980 (ADC) and AD5235 (Dual Digital Potentiometer). adc_con module includes adc_xf module, which implements the interface logic, while adc_con serves the functions of a wrapper and contains some glue logic. As the SPI interface is shared, the design of the adc_xf includes an arbiter, allowing access to both physical devices using the shared interface.

6H. adc_xf.vhd

Module Inputs and Outputs

Table 6-9 Module I/Os

Port Name	Type	Direction	Description
clk	std_logic	in	60 MHz clock input
clk_del	std_logic	in	60 MHz clock input
clk_2x	std_logic	in	60 MHz clock phase shifted by 0 deg from clk
rst_n	std_logic	in	Active-low, asynchronously reset, synchronously set reset input
clk_en	std_logic	in	100 msec clock enable, active-high, 1 clk long
mosi	std_logic	out	SPI data output (Master Out Slave In)
miso	std_logic	in	SPI data input (Master In Slave Out)
sclk	std_logic	out	SPI clock output
adc_cnv	std_logic	out	ADAQ7980 CNV signal, active-high
digipot_cs_n	std_logic	out	Digital potentiometer AD5235 Chip Select, active-low
digipot_rdy	std_logic	in	Digital potentiometer AD5235 read data ready, active-high, 1 clk wide
digipot1_wr	std_logic	in	AD5235 digital potentiometer 1 write command, active-high, 1 clk wide
digipot2_wr	std_logic	in	AD5235 digital potentiometer 2 write command, active-high, 1 clk wide
RDAC1	std_logic_vector(9:0)	in	AD5235 digital potentiometer 1 write data
RDAC2	std_logic_vector(9:0)	in	AD5235 digital potentiometer 2 write data
RDAC1_rd_data	std_logic_vector(9:0)	out	AD5235 digital potentiometer 1 read data
RDAC2_rd_data	std_logic_vector(9:0)	out	AD5235 digital potentiometer 2 read data
pg_en	std_logic	in	Pattern Generator Enable, active-high
last_xfer	std_logic	in	Last Transfer indicator, active-high
integr_end	std_logic	in	Integration End indicator, active-high, 1 clk wide
integr_start	std_logic	in	Integration Start indicator, active-high, 1 clk wide
data	std_logic_vector(15:0)	out	ADAQ7980 data output
ready	std_logic	out	ADAQ7980 data ready, active-high, 1 clk wide
adc_trigger	std_logic	in	Trigger for ADAQ7980 ADC conversion, active-high, 1 clk wide

adc_xf module serves as an interface between the FPGA and the two Analog Devices ICs sharing the same SPI bus: ADAQ7980 (ADC) and AD5235 (Dual Digital Potentiometer). adc_xf module implements the interface logic based on state machines. As the SPI interface is shared, the design of the adc_xf includes an arbiter, allowing access to both physical devices using the shared interface. The arbitration is based on the fact control coming from the detector_con module.

During the time when odd/even clocks and strobes are actively generated and odd/even trigger inputs are output by the sensor (adtrig_odd/ad_trig_even) the adc_xf communicates with ADAQ7980 to acquire the corresponding pixel data. During the reset intervals, when the sensor is performing integration, adc_xf allows the Read and Write accesses to the digital potentiometer AD5235. The accesses to the digital potentiometer are suspended while the accesses to ADC continue following the integration interval.

The ADC state machine is shown in Figure 6-31. The FSM idles in ADC_IDLE_ST state waiting for the next rising edge of adc trigger from the sensor. adc_tcnv_cnt is initialized to 60; hence the state machine waits in ADC_CNV_ST until the conversion is finished. The wait time is 60 x 60MHz periods, or 1000nsec. ADAQ7980 device conversion time ranges from 500 to 710nsec. With 1000nsec conversion time allowance, the design provides for an ample margin.

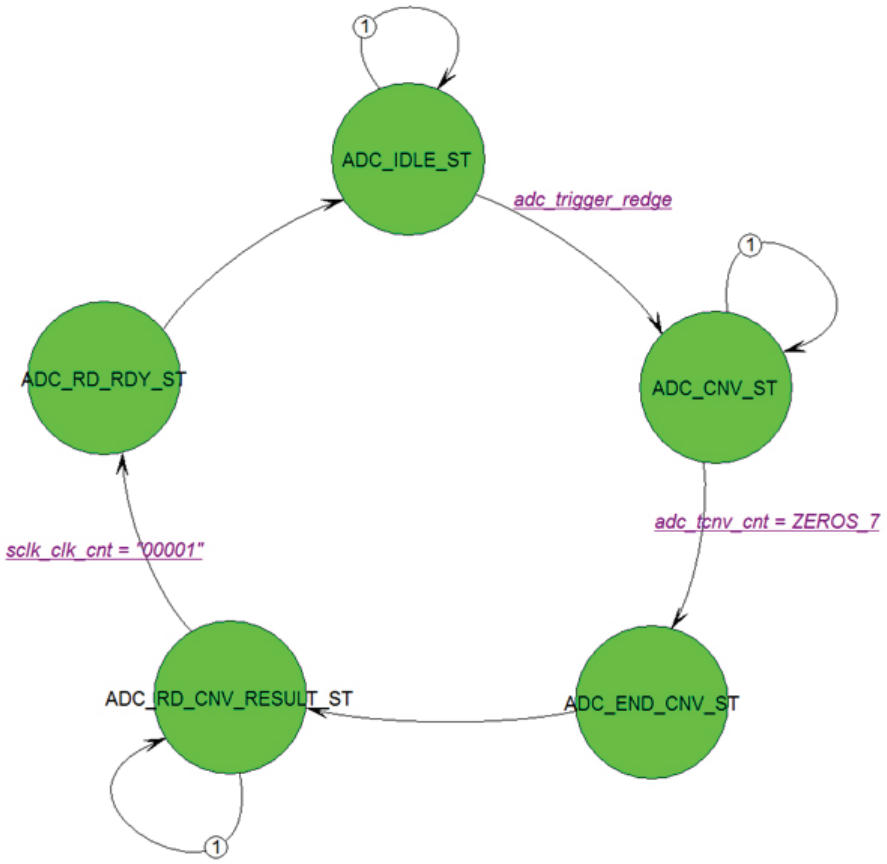


Figure 6-31: ADC state machine

The interface to ADC consists of the SPI (with MOSI and Chip-Select not connected) and CNV signal (adc_cnv). During ADC_RD_SNV_RESULT state the 16-bit ADC data is serially captured, while the adc_xf pulses SPI clock SCLK.

The digital potentiometer state machine is shown in Figure 6-32.

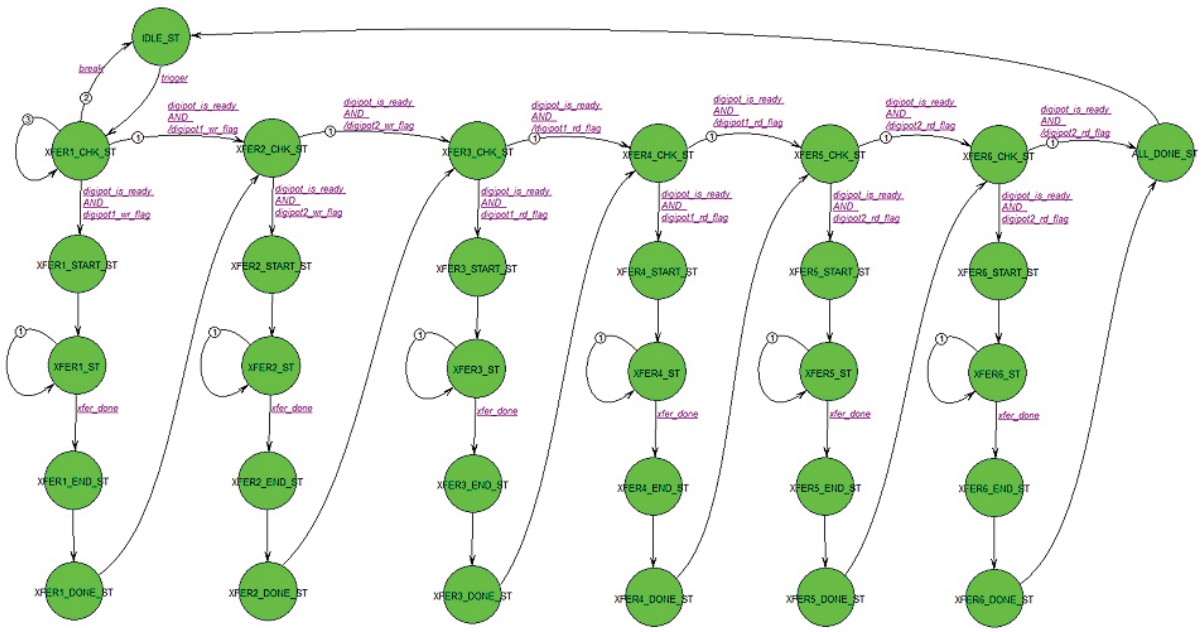


Figure 6-32: Digital potentiometer state machine

The digital potentiometer read and write requests are received via I2C and result in the corresponding trigger requests being generated. In response to the trigger requests, while the detector controller is in the reset (integration) phase, the accesses to the digital potentiometer are performed.

6l. AD7991_con.vhd

Module Inputs and Outputs

Table 6-10 Module I/Os

Port Name	Type	Direction	Description
clk_sys	std_logic	in	60 MHz clock input
rst_n	std_logic	in	Active-low, asynchronously reset, synchronously set reset input
clk_en	std_logic	in	100 msec clk enable, 1 clk wide, active-high
xfer_ack_n	std_logic	in	Transfer acknowledge, 1 clk wide, active-low
rd_data	array (0 to 15) of std_logic_vector (7 down to 0)	in	Data returned by the AD7991 device
error	std_logic	in	Error indicator, returned by the MPAC
xfer_req_n	std_logic	out	Transfer request, 1 clk wide, active-low
xfer_rw	std_logic	out	Transfer type select, 0 = Write, 1 = Read
xfer_size	natural	out	Indicates the number of payload bytes to be transferred
xfer_addr	std_logic_vector (7 down to 1)	out	Indicates the I2C slave address to be accessed In the case of AD7991, this is set to "0101001"
wr_data	array (0 to 15) of std_logic_vector (7 down to 0)	out	Payload data to be written out to the slave device
skip_wr	std_logic	out	Setting this to 1 results in I2C read access performed without write access being done first.
ADC_CH0	std_logic_vector (11 down to 0)	out	Data read from Channel 1 of the ADC
ADC_CH1	std_logic_vector (11 down to 0)	out	Data read from Channel 2 of the ADC
ADC_CH2	std_logic_vector (11 down to 0)	out	Data read from Channel 3 of the ADC

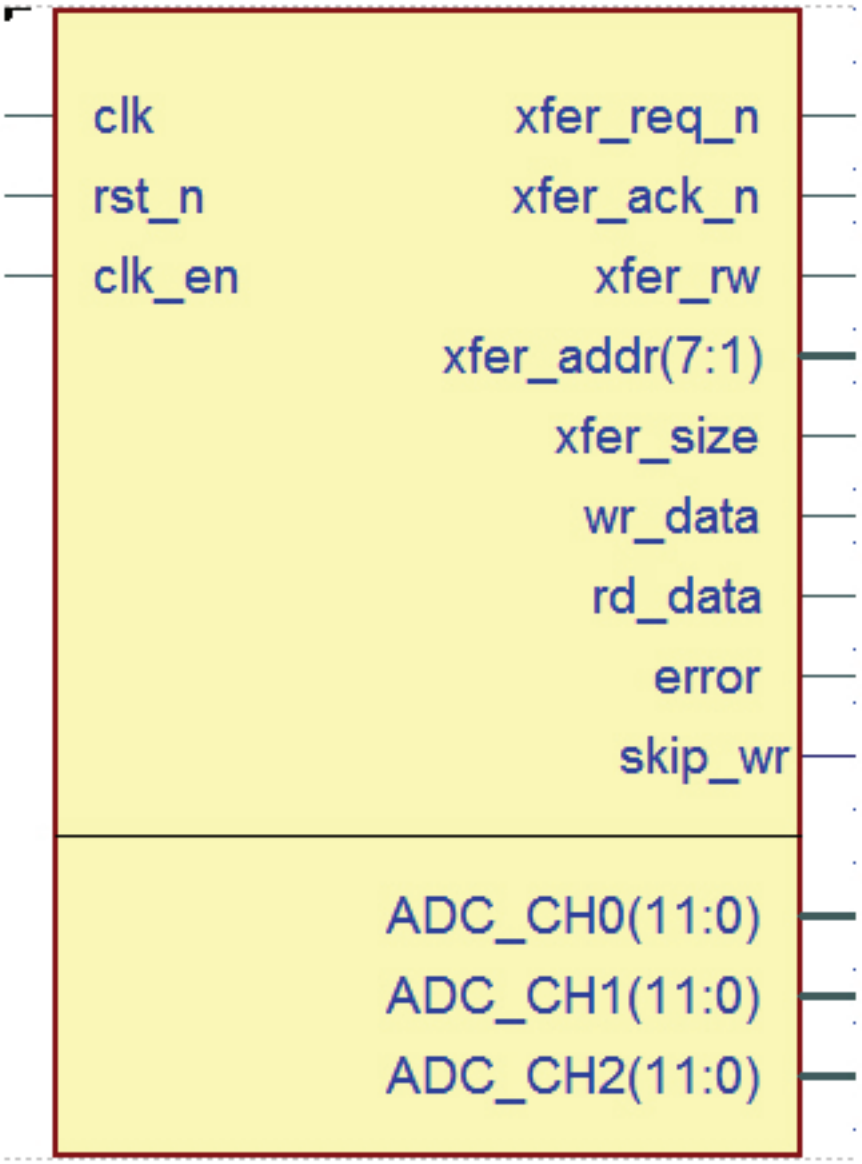


Figure 6-33: AD7991_con module I/Os

Module Description

AD7991_con module is responsible for high-level control of the AD7991 I2C ADC. The system contains 2 I2C devices: AD7991 12-bit ADC and AD5627 12-bit DAC. Both devices share the same I2C bus. To allow for the FPGA to access both devices the design includes a Multi-Port Access Controller (MPAC), connected to I2C Master at one side and to the two I2C controllers (AD7991_con and AD5627_con) on the other side.

AD7991_con controls the high-level I2C commands and data issued to and received from the DAC. The control is based on a state machine shown in Figure 6-34.

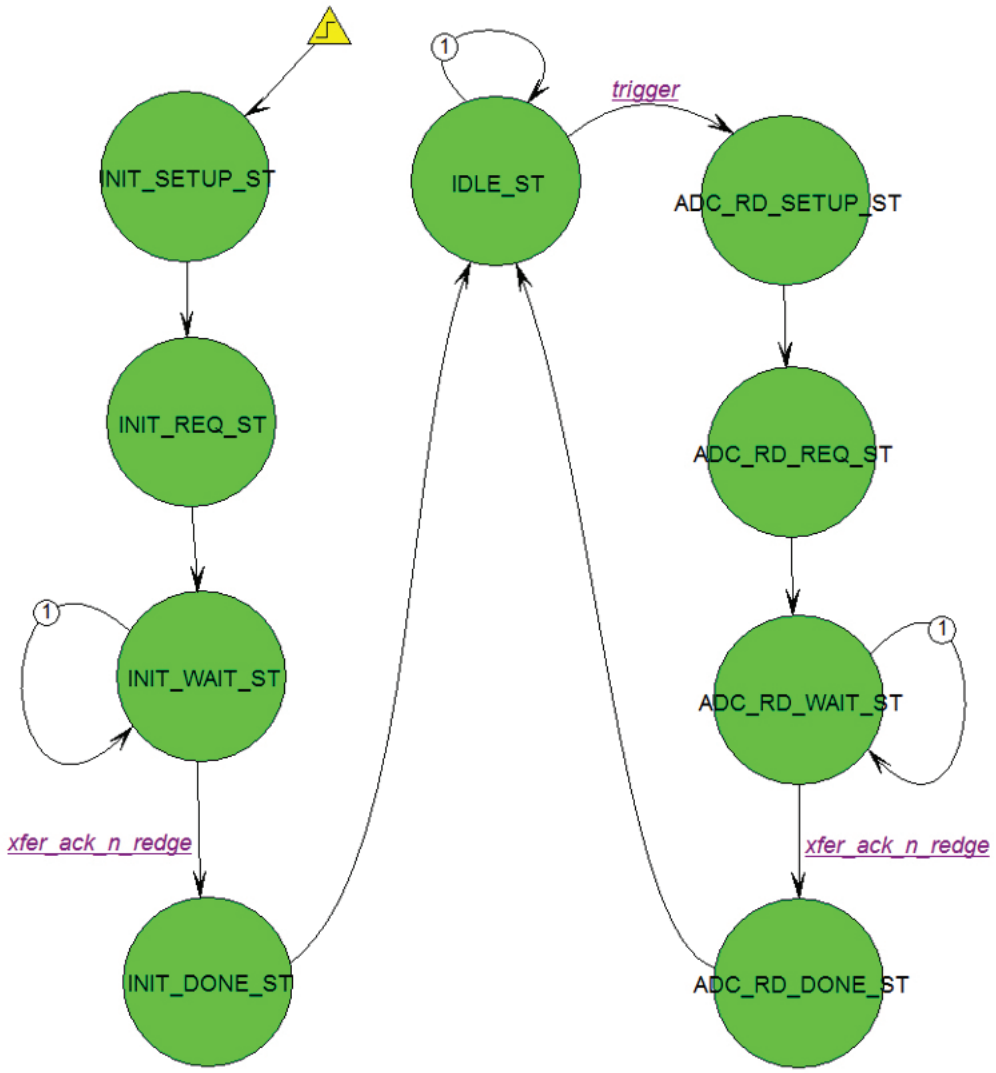


Figure 6-34: AD7991_con state machine

During the initialization phases of the state machine (INIT_SETUP_ST through INIT_DONE_ST), the AD7991 is initialized by performing the following transactions:
Write 00111000 to enable reading Ch0 and Ch1, Select External REF, Enable I2C filtering, Enable bit try and Sample Delay.

Upon completion of the initialization the controller is ready to access the ADC. The ADC read requests are performed via slave I2C interface, when the external USB controller accesses the respective FPGA slave registers. In response, the controller state machine traverses states ADC_RD_SETUP_ST through ADC_RD_DONE_ST and reads the three available analog channels.

6J. AD5627_con.vhd

Module Inputs and Outputs

Table 6-11 Module I/Os

Port Name	Type	Direction	Description
clk_sys	std_logic	in	60 MHz clock input
rst_n	std_logic	in	Active-low, asynchronously reset, synchronously set reset input
clk_en	std_logic	in	100 msec clk enable, 1 clk wide, active-high
DAC	std_logic_vector(11:0)	in	Data to be written to DAC
DAC_update	std_logic	in	Trigger, active-high, 1 clk wide resulting in data being sent to the DAC
xfer_req_n	std_logic	out	Transfer request, 1 clk wide, active-low
xfer_ack_n	std_logic	in	Transfer acknowledge, 1 clk wide, active-low
xfer_rw	std_logic	out	Transfer type select, 0 = Write, 1 = Read
xfer_addr	std_logic_vector (7 down to 1)	out	Indicates the I2C slave address to be accessed In the case of AD5627, this is set to "0001110"
xfer_size	natural	out	Indicates the number of payload bytes to be transferred
wr_data	array (0 to 15) of std_logic_vector (7 down to 0)	out	Payload data to be written out to the slave device
rd_data	array (0 to 15) of std_logic_vector (7 down to 0)	out	Data read out from the slave device
error	std_logic	in	Error indicator, returned by the MPAC
skip_wr	std_logic	out	Setting this to 1 results in I2C read access performed without write access being done first.

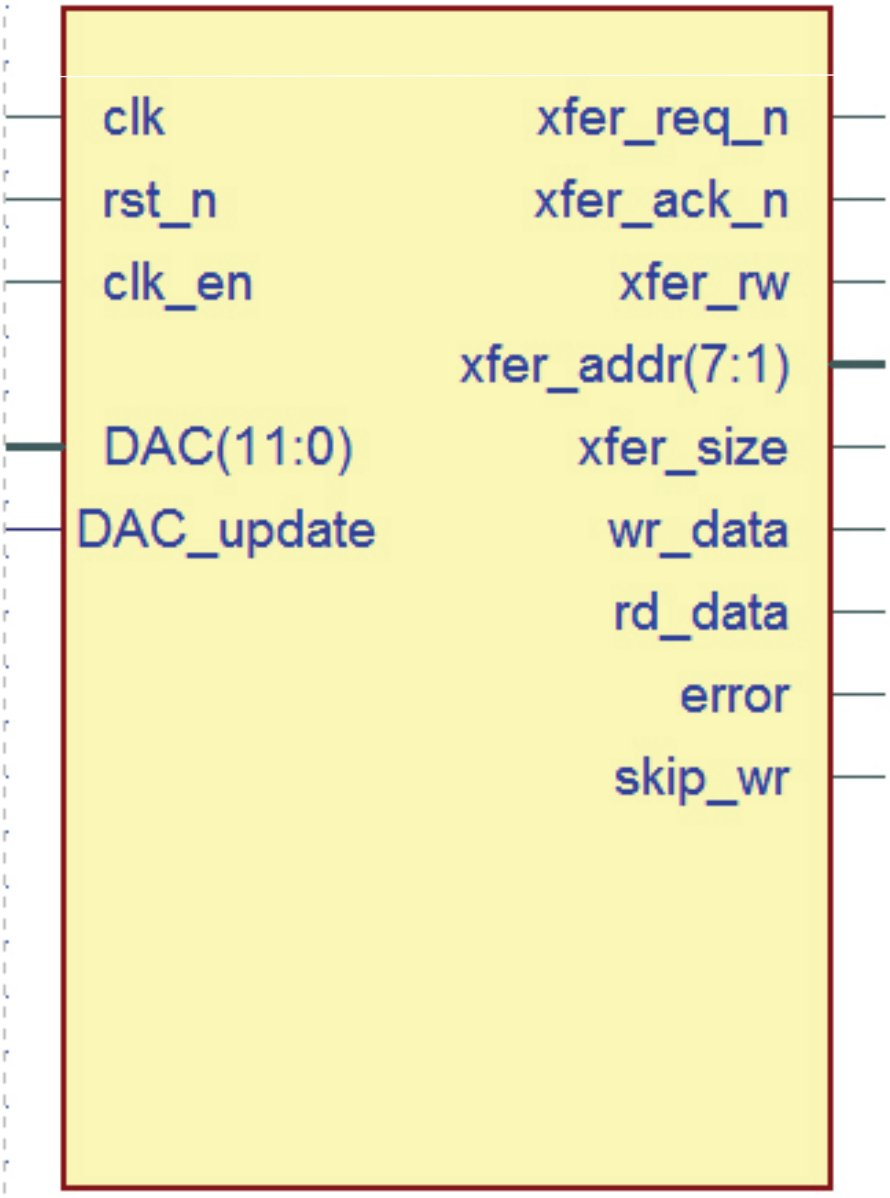


Figure 6-35: AD5627_con module I/Os

Module Description

AD5627_con module is responsible for the initialization and high-level control, including data reads and writes of the AD5627 I2C DAC. All accesses to the physical DAC are performed via I2C bus, using I2C master and a Multi-Port Access Controller (MPAC). The accesses to AD5627 are shared with I2C transfers performed in the course of accesses to AD7991, and therefore are arbitrated by the MPAC. The controller is based on a state machine shown in Figure 6-36.

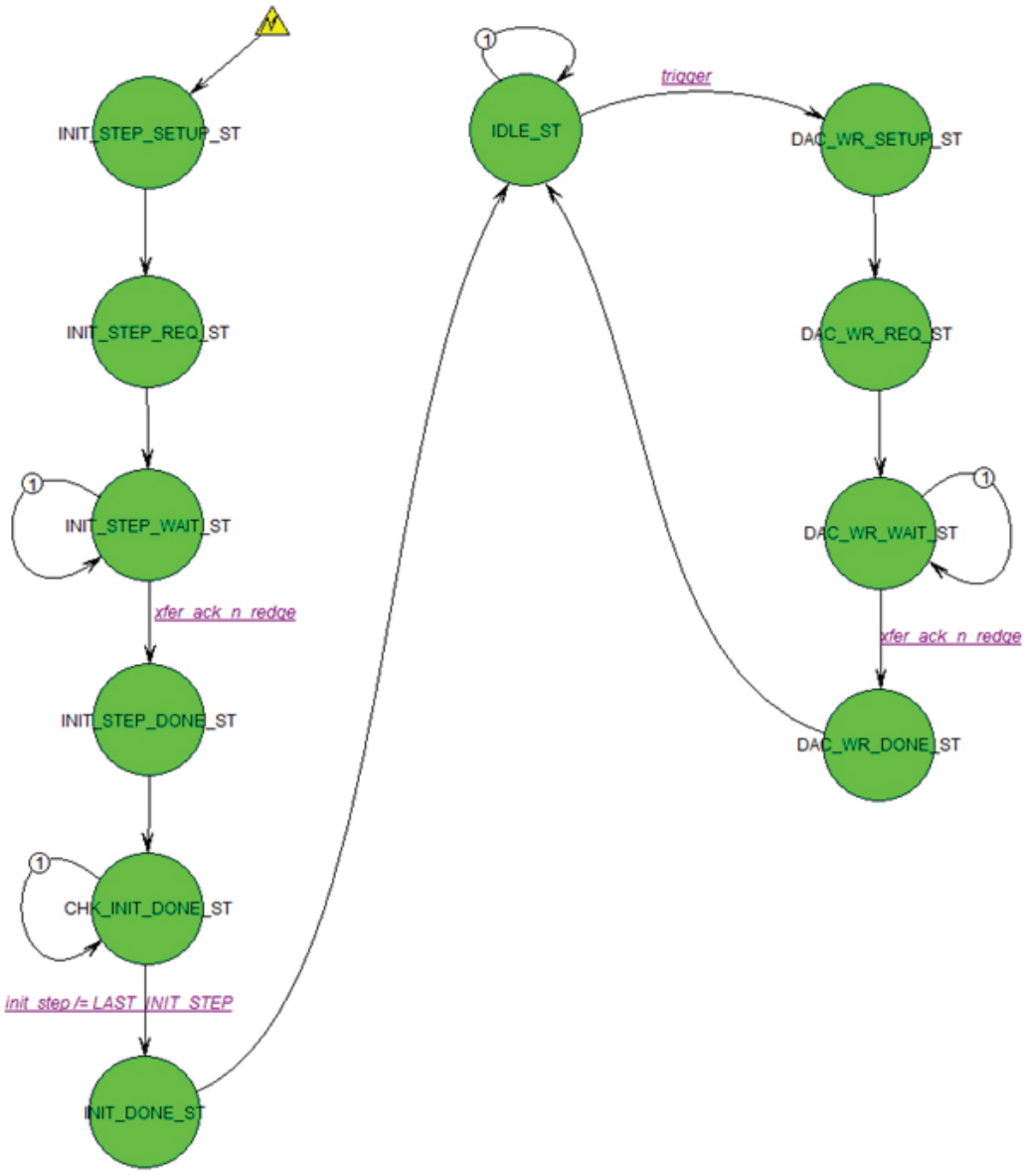


Figure 6-36: AD5627_con state machine

AD5627 initialization consists of 4 steps:

1. LDAC Setup:
a. Data Byte 1 = “00110000” (Command = “110”)
b. Data Byte 2 = “00000000” (Don’t Care)
c. Data Byte 3 = “00000001” (DAC B LDAC pin enabled, DAC A LDAC pin disabled)

2. Reference Setup:
a. Data Byte 1 = “00111000” (Command = “111”)
b. Data Byte 2 = “00000000” (Don’t Care)
c. Data Byte 3 = “00000001” (Internal Reference ON)

3. Load Input Shift Register:
a. Data Byte 1 = “01011000” (Byte Selection (S) = 1, Command = “011” (Write to and Update DAC Channel n), DAC Address = “000” (DAC A))
b. Data Byte 2 = DAC(11:4)
c. Data Byte 3 = DAC(3:0) & “0000”

4. Power-up:
a. Data Byte 1 = “00100000” (Command = Power-up)
b. Data Byte 2 = “00000000” (Don’t Care)
c. Data Byte 3 = “00000001” (Normal Operation (5:4) = “00”, Select DAC A (bit 0 = ‘1’))

For all subsequent accesses DAC write is performed whenever a trigger is received. Upon a trigger, which is an I2C slave write to the FPGA from the USB sub-system, the state machine sends the 12-bit DAC input to AD5627 device.

6K. mpac.vhd

Module Inputs and Outputs

Table 6-12 Module I/Os

Port Name	Type	Direction	Description
clk_sys	std_logic	in	60 MHz clock input
rst_n	std_logic	in	Active-low, asynchronously reset, synchronously set reset input
xfer_req_n	std_logic_vector(7:0)	in	Active-low transfer request. The request bit is asserted until a corresponding acknowledgement response xfer_ack_n bit is received.
xfer_rw	std_logic_vector(7:0)	in	Transfer direction: Read = 1, Write = 0
xfer_addr	array (7:0) of std_logic (7:0)	in	I2C slave address, one for each corresponding port number
wr_data	array (7:0) of array (15:0) of std_logic (7:0)	in	I2C payload, 16x8 array for each port
xfer_size	array (7:0) of natural	in	Transfer size (number of payload bytes), one value for each respective port
skip_wr	std_logic_vector(7:0)	in	If write is not needed before reads take place, this bit indicates that when set. One bit for each of the 8 ports.
xfer_ack_n	std_logic_vector(7:0)	in	Transfer acknowledge, active-low, 1 clk wide. Indicates that the requested transfer has been completed.
rd_data	array (15:0) of std_logic (7:0)	in	Data read from the I2C slave device
error	std_logic_vector(7:0)	in	Error indication, one for each port
I2C_xfer_req_n	std_logic	out	Output to the I2C master, active-low, 1 clk wide, requesting a data transfer
I2C_addr	std_logic_vector(7:1)	out	Address of the I2C slave device to be accessed
I2C_rw	std_logic	out	Transfer direction: 0 = Write, 1 = Read
I2C_xfer_size	natural	out	Number of payload bytes to be transferred
I2C_wr_data	array (15:0) of std_logic (7:0)	out	Payload bytes to be transferred
I2C_rd_data	array (15:0) of std_logic (7:0)	in	Data read as a result of the I2C read transfer
I2C_rd_vld	std_logic	in	Active-high, 1 clk wide, indicates that I2C data read is valid
I2C_busy	std_logic	in	Indication from the I2C master that a transfer is in progress, active-high
I2C_error	std_logic	in	I2C transfer error indication
I2C_skip_wr	std_logic	out	If write is not needed before reads take place, this bit indicates that when set.

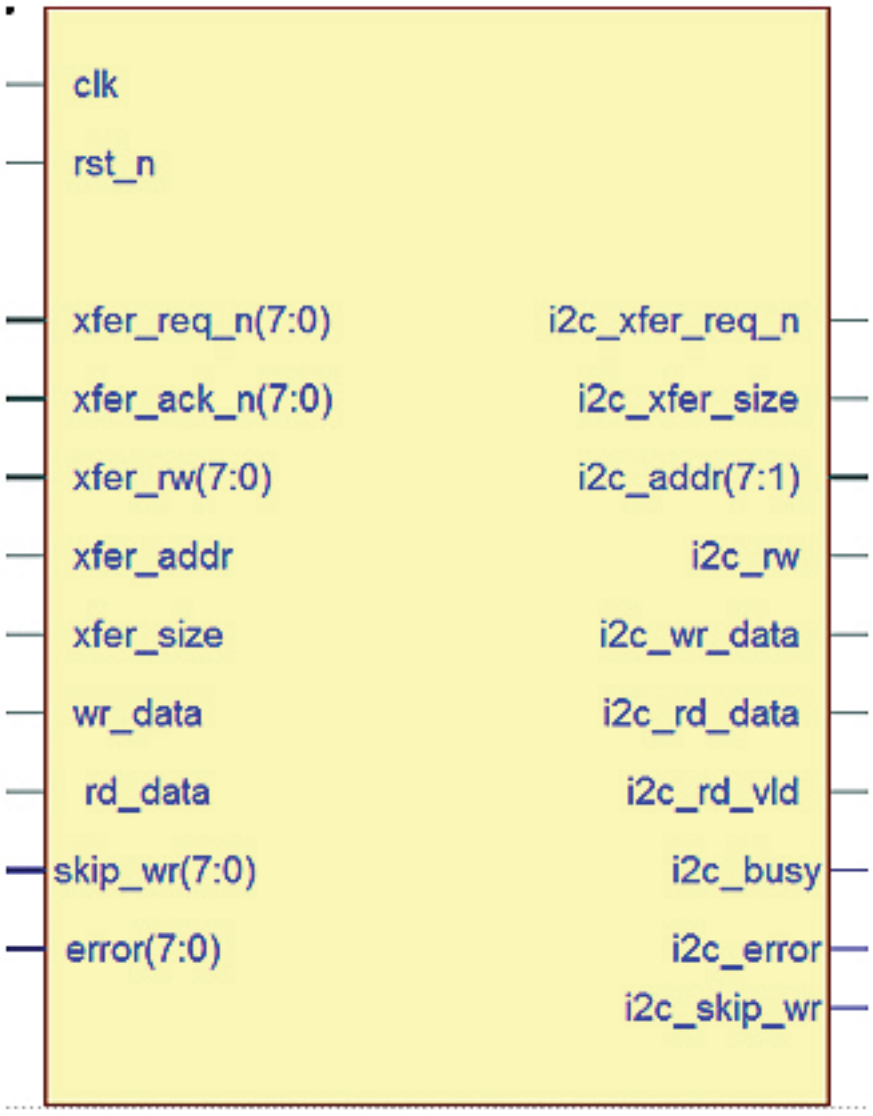


Figure 6-37: MPAC module I/Os

Module Description

The Multi-Port Access Controller links the high-level controllers needing to perform high-level I2C transfers with a single I2C Master controller available on the FPGA. AD7991_con and AD5627_con controllers utilize 2 of the 8 available ports on the MPAC. The MPAC performs round-robin access arbitration between all 8 ports. In reality, since only 2 ports are utilized the available bandwidth is split ~50/50 between the two controllers.

The MPAC design is based on 2 state machines: one state machine is used for round-robin access arbitration, while the second state machine is used for control of the interface with the I2C master module I2C_master.

The arbitration state machine is shown in Figure 6-38. A request is checked one at a time. If a request is asserted, the corresponding port is allowed access to the I2C Master interface via the I2C Master Interface state machine shown in Figure 6-39.

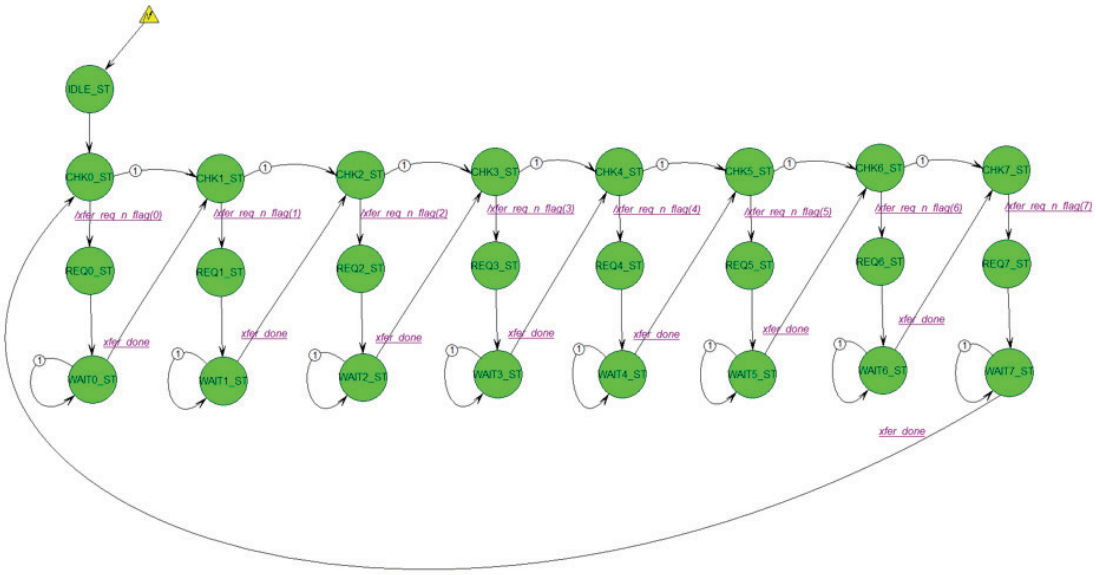


Figure 6-38: MPAC arbitration state machine

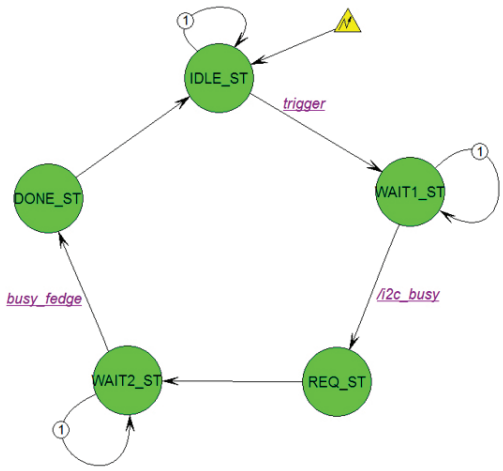


Figure 6-39: MPAC I2C master interface state machine

6L. I2C_master.vhd

Module Inputs and Outputs

Table 6-13 Module I/Os

Port Name	Type	Direction	Description
clk	std_logic	in	60 MHz clock input
rst_n	std_logic	in	Active-low, asynchronously reset, synchronously set reset input
xfer_req_n	std_logic	in	Active-low, 1 clk wide, request to perform a transfer
xfer_size	natural range 1 to 16	in	Payload size in number of bytes
addr	std_logic_vector(7:1)	in	Address of the slave device to be accessed
rw	std_logic	in	Read/write indication: 0 = Write, 1 = Read
data_wr_arr	array (0 to 15) of std_logic_vector(7 down to 0)	in	Array containing the payload data
skip_wr	std_logic	in	Set to 1 when a read transfer does not need to be preceded by a write.
scl	std_logic	in, out	I2C Clock
sda	std_logic	in, out	I2C Data
busy	std_logic	out	1 = I2C Master is busy 0 = I2C Master is ready for the next transfer request
ack_error	std_logic	out	1 = Indicates slave did not acknowledge the transfer
data_rd_vld	std_logic	out	1 = Indicates that the data_rd_arr contains the data
data_rd_arr	array (0 to 15) of std_logic_vector (7 down to 0)	out	Data returned during a read transfer

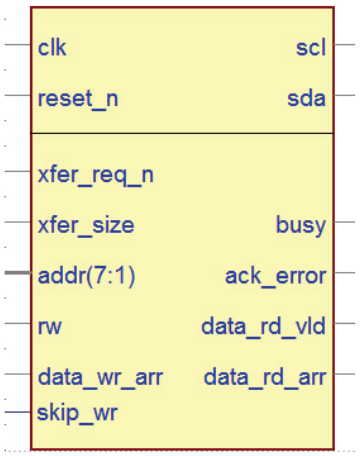


Figure 6-40: I2C master module I/Os

Module Description

This module interfaces to MPAC module and provides for master access to the I2C bus. The two slave devices accessed by the master are AD7991 and AD5627. The module design is based on a state machine shown in Figure 6-41.

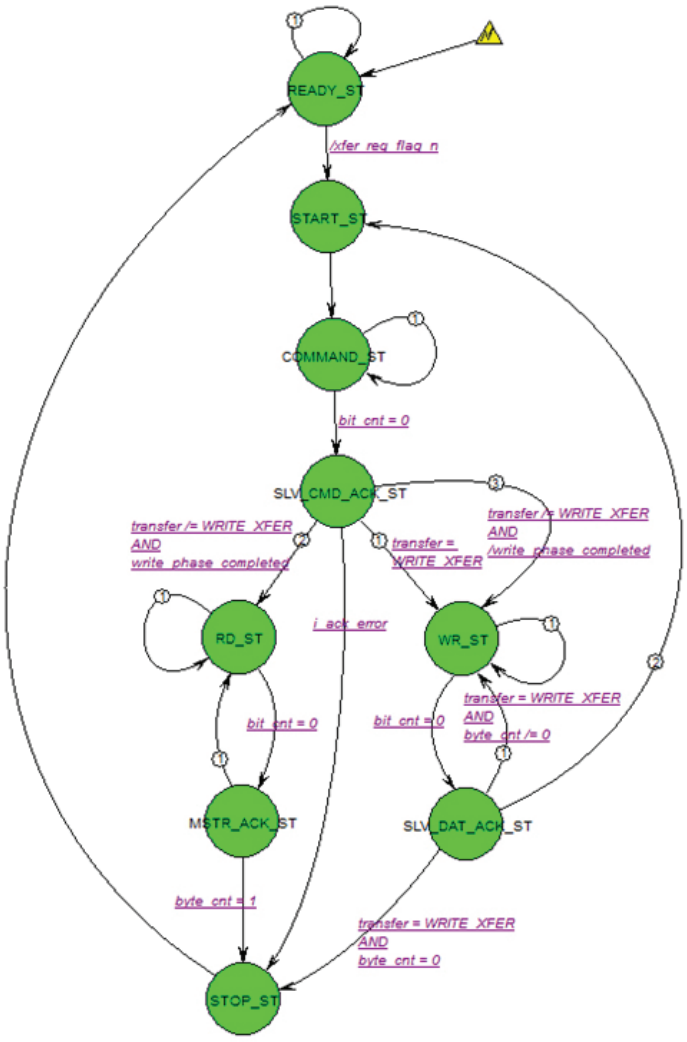


Figure 6-41: I2C master state machine

6M. tx_fifo.vhd

Module Inputs and Outputs

Table 6-14 Module I/Os

Port Name	Type	Direction	Description
clk	std_logic	in	60 MHz clock input
rst_n	std_logic	in	Active-low, asynchronously reset, synchronously set reset input
din	std_logic_vector(15:0)	in	16-bit pixel data from the A/D converter, FIFO data input
wr_en	std_logic	in	FIFO write enable, active-high
sync	std_logic_vector(1:0)	in	"01" indicates the beginning of the reset state (integration start)
dout	std_logic_vector(15:0)	out	FIFO data output
rd_en	std_logic	in	FIFO read enable, active-high
rd_vld	std_logic	out	FIFO read data valid, active-high
empty	std_logic	out	FIFO empty indication 1 = FIFO is empty
full	std_logic	out	FIFO full indication 1 = FIFO is full
flag	std_logic	out	FIFO data ready flag 1 = Data is ready for reading from FIFO
data_cnt	std_logic	out	FIFO fullness count: indicates the number of 16-bit words available for reading

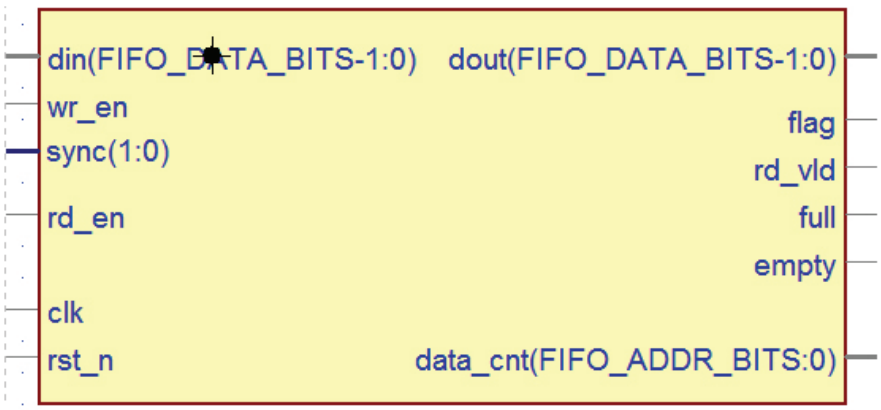


Figure 6-42: tx_fifo module I/Os

Module Description

This module serves the purpose of storing the data received from the image sensor prior to having that data transferred to the USB microprocessor via slavefifo2b_streamin module. The FIFO size is 256 words deep, with each word being 16-bits wide.

6N. slavefifo2b_streamin.vhd

Module Inputs and Outputs

Table 6-15 Module I/Os

Port Name	Type	Direction	Description
clk	std_logic	in	60 MHz clock input
rst_n	std_logic	in	Active-low, asynchronously reset, synchronously set reset input
op_en	std_logic	in	Operation Enable Connected to REG01 bit 6 1 = Operation enabled
tx_data	std_logic_vector(15:0)	in	Output of tx_fifo
clk_out	std_logic	out	Clock output, 60 MHz
fdata	std_logic_vector(15:0)	in, out	Slave interface data bus, bi-directional
faddr	std_logic_vector(1:0)	out	Slave FIFO address output, fixed at '00'
slcs_n	std_logic	out	Slave chip select, fixed at '0'
slrd_n	std_logic	out	Slave interface read enable, active-low
sloe_n	std_logic	out	Slave output enable, fixed at '1'
slwr_n	std_logic	out	Slave interface write enable, active-low
flaga	std_logic	in	Current Thread DMA Ready (active-high)
flagb	std_logic	in	Partial flag, DMA watermark value (active-high)
pktend_n	std_logic	out	Slave packet end, fixed at '1'
wr_end_wd_cnt	std_logic_vector(3:0)	in	Connected to REG13 bits (3:0)
fifo_rd_en	std_logic	out	Output to tx_fifo, read enable, active-high
fifo_data_rdy	std_logic	in	Output from tx_fifo (flag). Indicates that FIFO data is ready.
test	std_logic	out	For test purposes, not used
state_decode	std_logic_vector(2:0)	out	For test purposes, connected to the on-board LEDs
test_trig	std_logic	out	For test purposes, not used

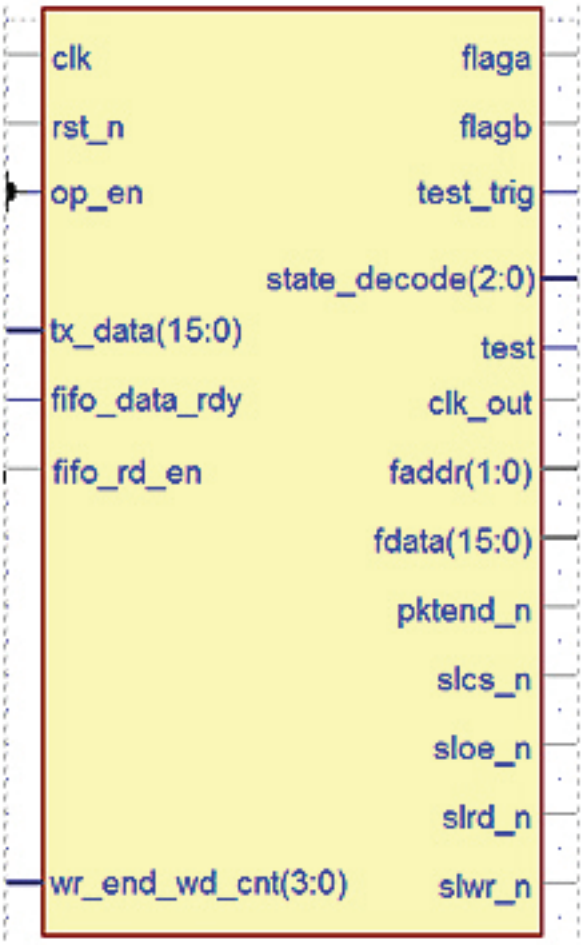


Figure 6-43: slavefifo2b_streamin module I/Os

Module Description

This module implements Synchronous Slave FIFO Interface in accordance with Cypress CYUSB301X datasheet. The synchronous slave FIFO interface is used for transferring pixel data to the USB microprocessor and onto the PC. The slave FIFO interface is based on Cypress Application Note AN65974. The interface design from the application note has been revised and further elaborated and modified for this present application. The implementation is centered on the state machine shown in Figure 6-44.

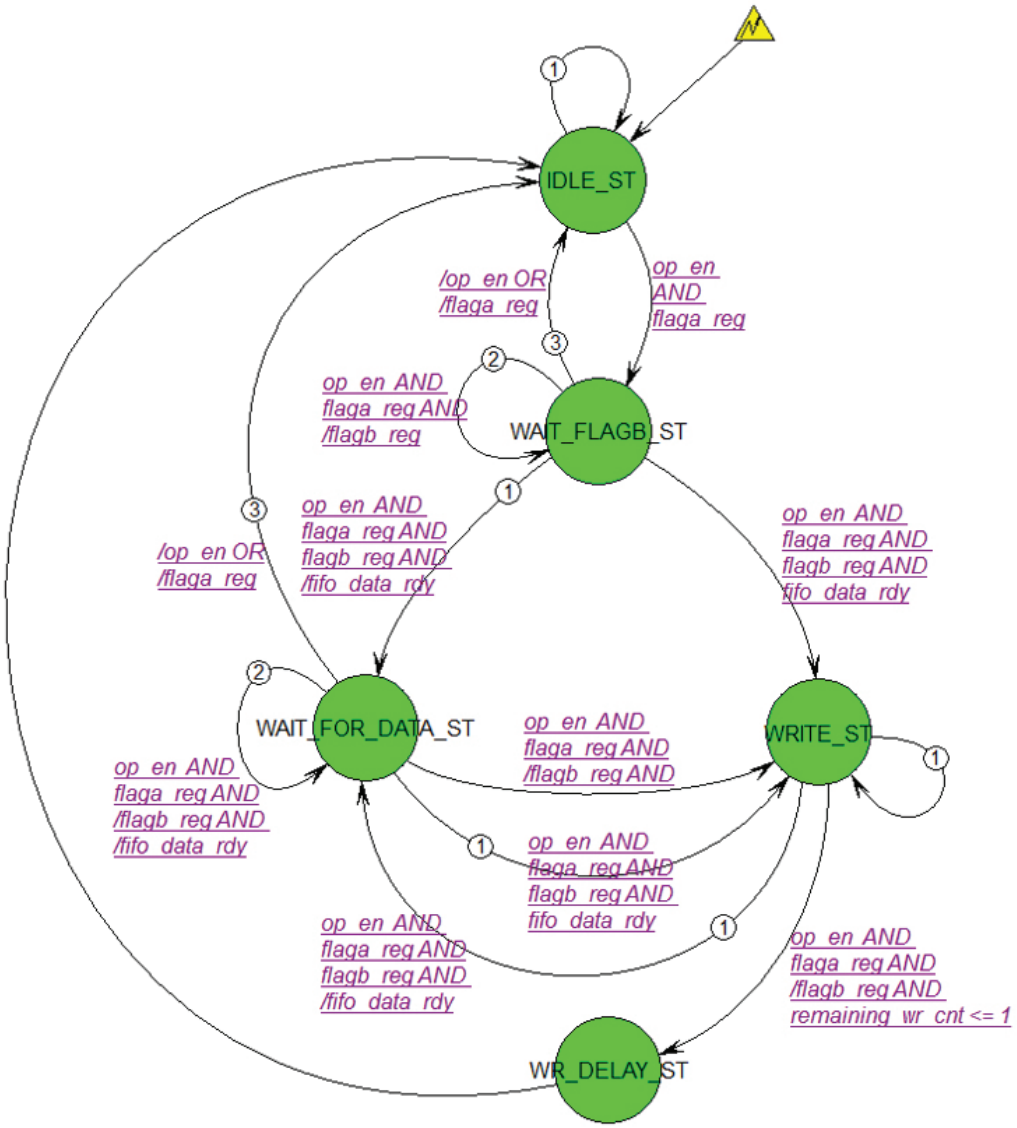


Figure 6-44: Slave FIFO interface state machine

The USB interface sub-system is based upon Cypress EZ-USB FX3 development kit. At the heart of the Cypress platform is USB microprocessor CYUSB301X supporting USB3.0 interface.

The USB microprocessor design is based on Cypress Synchronous Slave FIFO. SlaveFIFOSync project is provided as a part of this system. The project is comprised of the following files:

1. cyfx_gcc_startup.S:
Start-up code for the ARM-9 core on the FX3 device. This assembly source file follows the syntax for the GNU assembler.

2. cyfxslfifosync.h:
C header file that defines constants used by this example implementation. Can be modified to select USB connection speed, endpoint numbers and properties, etc.

3. cyfxslfifousbdscr.c:
C source file that contains USB descriptors used by this example. VID and PID are defined in this file.

4. cyfxgpif_syncsf.h:
C header file that contains the data required to configure the GPIF interface to implement the Sync Slave FIFO protocol.

5. cyfxtx.c:
C source file that provides ThreadX RTOS wrapper functions and other utilities required by the FX3 firmware library.

6. cyfxslfifosync.c:
Main C source file.

7. makefile:
GNU make compliant build script for compiling this project.

This project configures and uses the GPIF II interface on the FX3 device in synchronous slave FIFO mode. The FPGA acts as a master device that implements the Cypress-defined Sync Slave FIFO.

This project implements the following functions:
1. Configuration of the GPIF II interface to implement the Sync Slave FIFO protocol.
2. Enumeration as a vendor specific USB device with two bulk endpoints (1-OUT and 1-IN).
3. Creation of MANUAL DMA channels to enable the following data paths:
   a. All data received from the USB host through the 1-OUT endpoint is forwarded to the master device on the slave port through socket 3.
   b. All data received from the master device on the slave port through socket 0 is forwarded to the USB host through the 1-IN endpoint.
4. When any data packet is received through one of the ingress sockets, the application is notified and forwards the data to the recipient through a DMA callback function.

The output of the compilation is SlaveFifoSync.img, which is loaded into the Cypress EZ-USB board using USB Control Center Application, part of EZ-USB FX3 SDK Software tools.

To load the file into the target perform the following steps (1-5):
1) Connect EZ-USB FX3 board to the PC.
2) Once the drivers for the USB device are installed, the screen shown in Figure 7-1 will be seen.

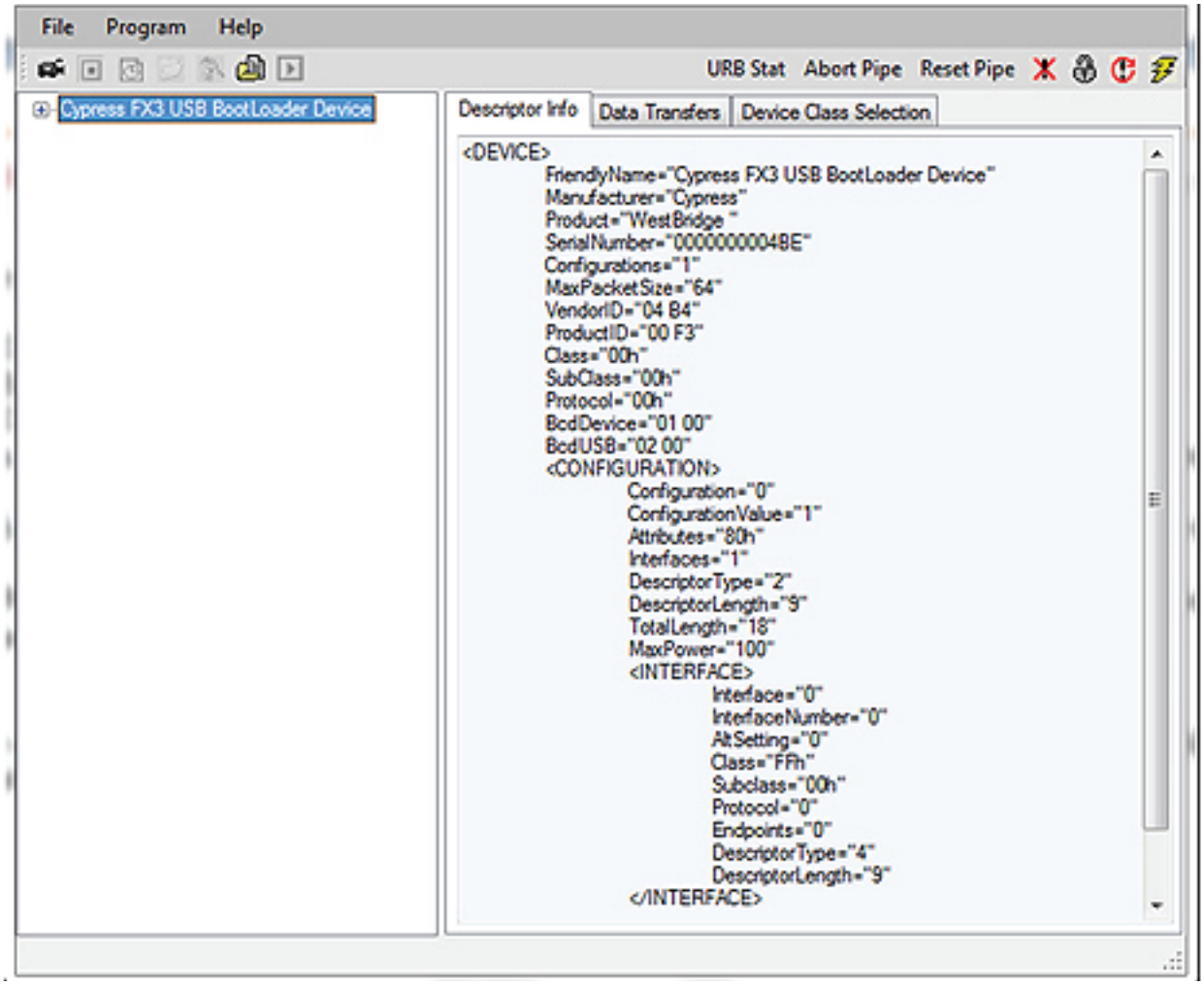


Figure 7-1: USB control center initial screen

3) Select the device (Cypress FX3 USB BootLoader Device), then click on Program -> FX3 -> RAM (refer to Figure 7-2).

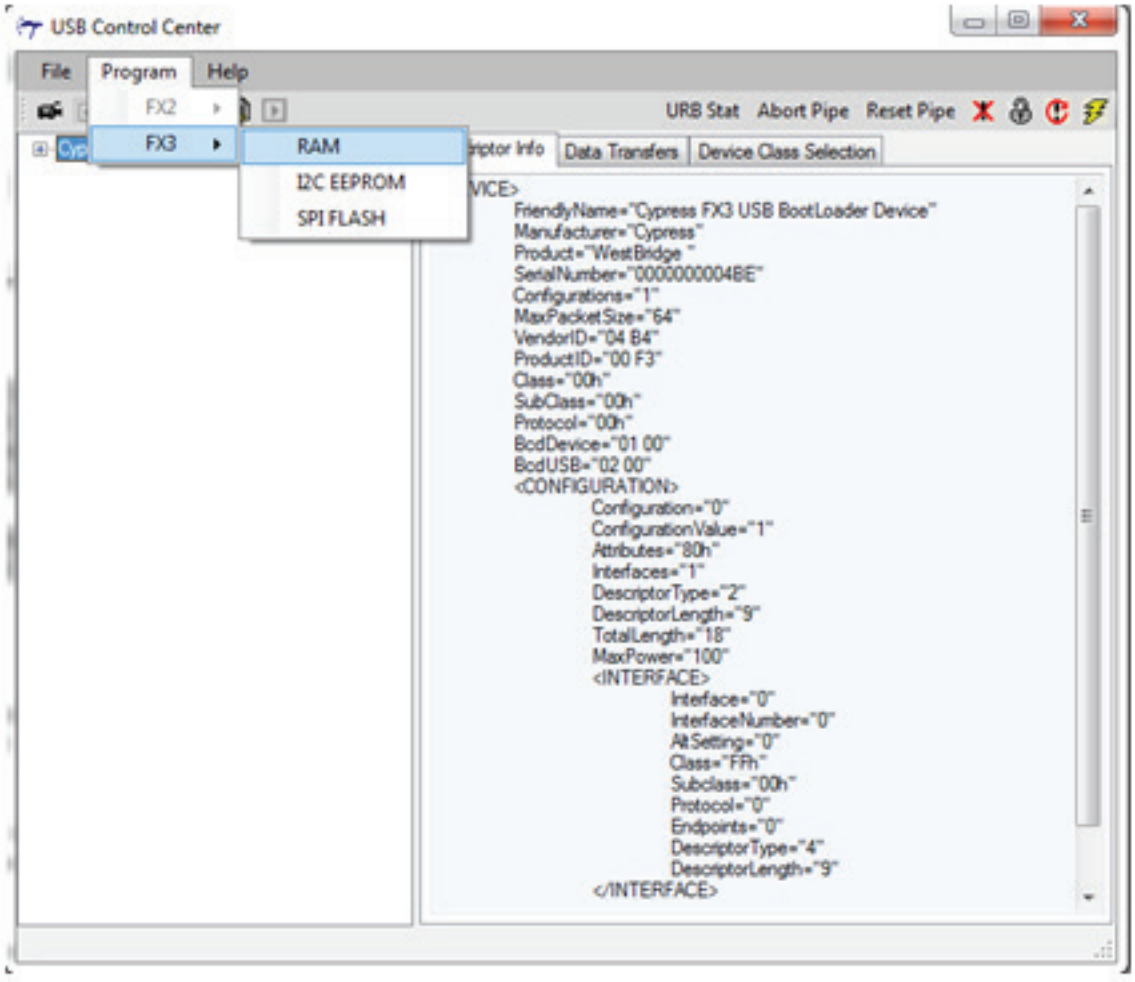


Figure 7-2: Program FX3 device

4) Navigate to the location of “SlaveFifoSync.img” and select “Open.”
5) The new driver will be loaded by the operating system, resulting in the following screen shown in Figure 7-3.

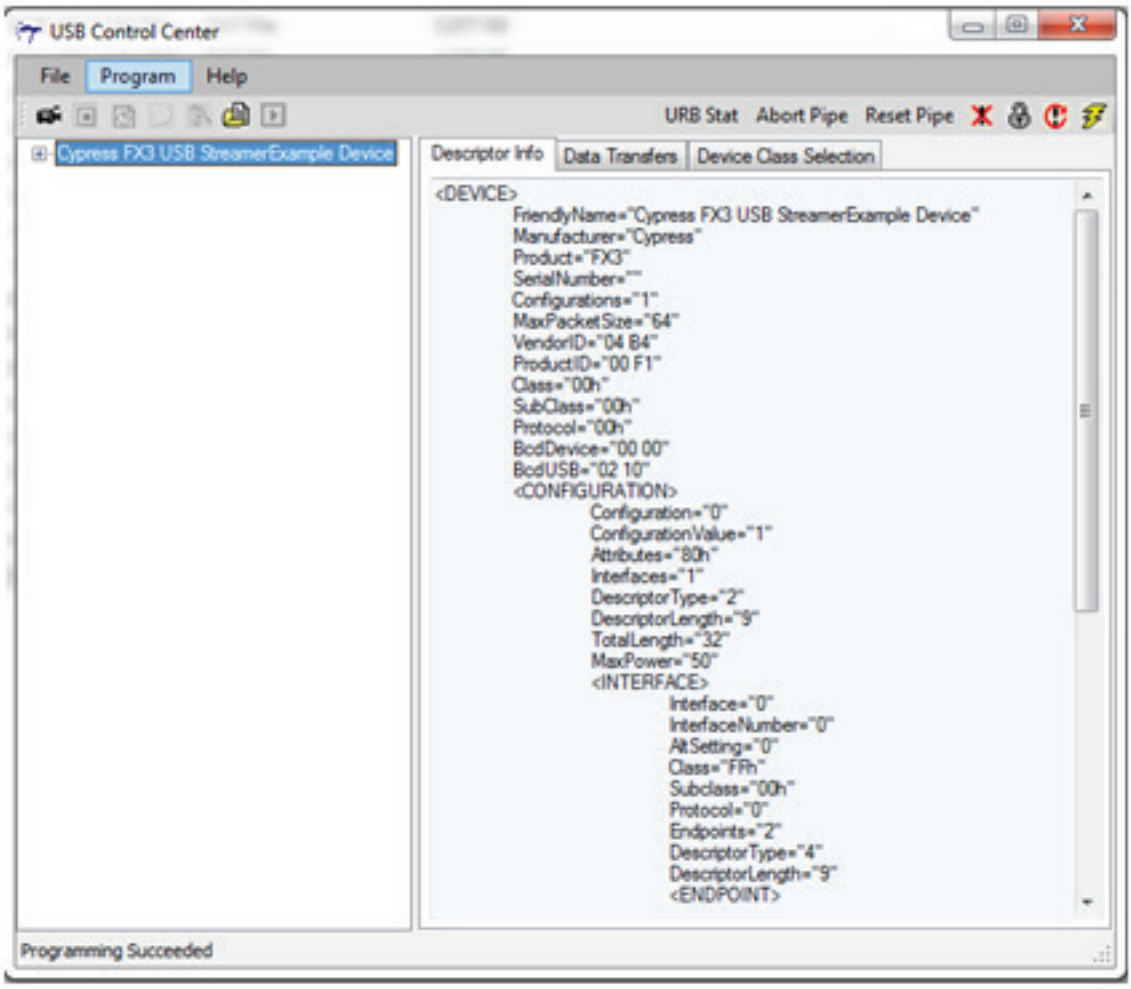


Figure 7-3: FX3 device has been programmed

Now, the USB development board is ready to perform data transfers.

Notes:
1) The firmware has been loaded into RAM; hence should the power to the board be turned off (or USB cable unplugged from the PC), the programming steps would have to be repeated.
2) It is important to use a high quality USB 3.0 cable and a corresponding USB 3.0 port on the PC.

PC Software and IGAA Driver Library

Section 8

The PC software uses the IGAA Driver Library for control, monitoring and data acquisition based on the development system described herein. Please refer to “Hamamatsu Driver Circuit for Image Sensor Control Library” document for the detailed information on the available functions.

9A. Timing Operations and Analog I/O Signals

This is the timing diagram of the G9201~8 and G9211~4 sensors. The integration time is set by the Reset input pulse width. Each pixel is being read out every 8 clocks.

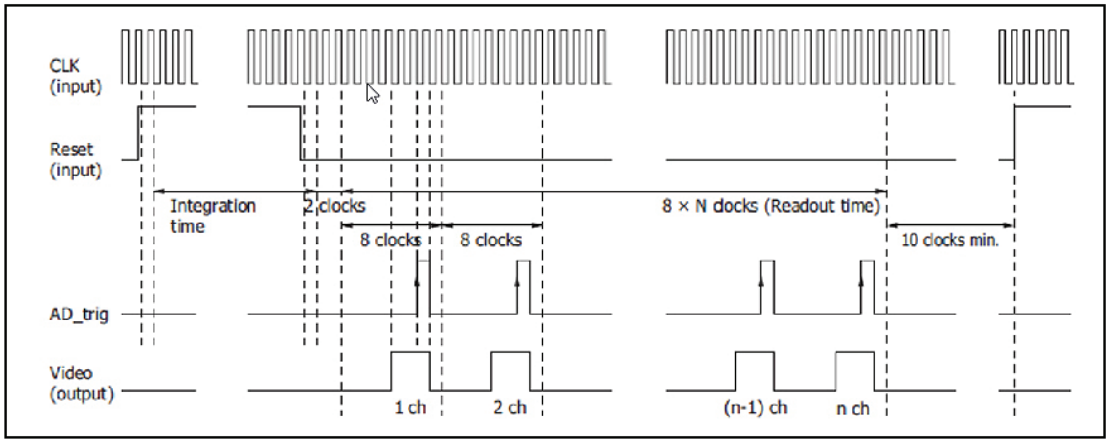


Figure 9-1: The G920x sensor timing diagram

In the software interface, the pixel format, readout mode, integration time, temperature, bias voltages, and the number of line data to be written can be defined.

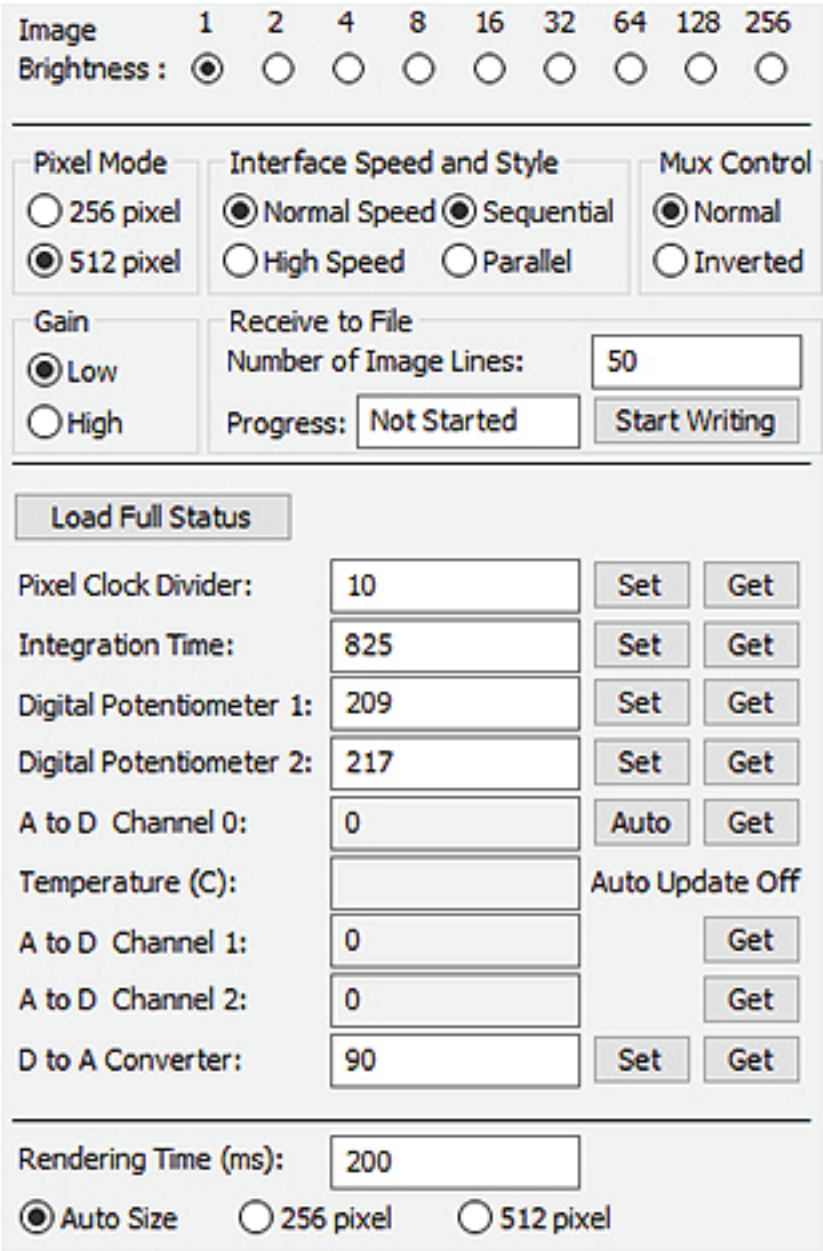


Figure 9-2: Parameter setting in the software interface

Pixel Number Select: 256 or 512.

For a 512-pixel array, there are two video output ports on the sensor chip: even and odd. Three readout modes are designed to read out from the sensor depending on the different multiplexer timing:
Multiplexer Timing (1) – Even/Odd simultaneous (parallel)
Multiplexer Timing (1a) – Non-Return to Zero (stagger)
Multiplexer Timing (2) – Clock burst with return to zero between pixel reads (bursts) – the same as C8062, the standard InGaAs multichannel detector head offered by Hamamatsu

The three different timing modes are tested, and the timing diagrams and test results are shown in Figures 9-3 to 9-18.

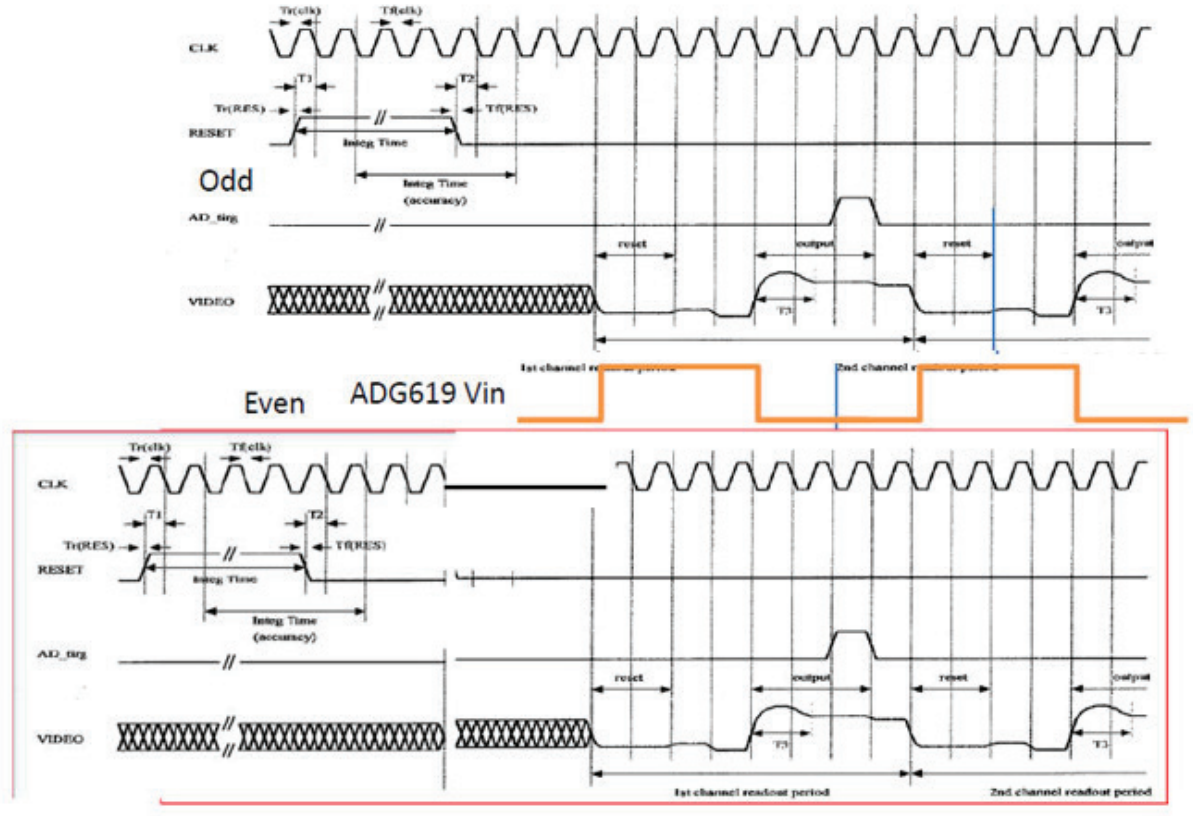


Figure 9-3: Timing diagram of MUX Timing (1) - Parallel clocking

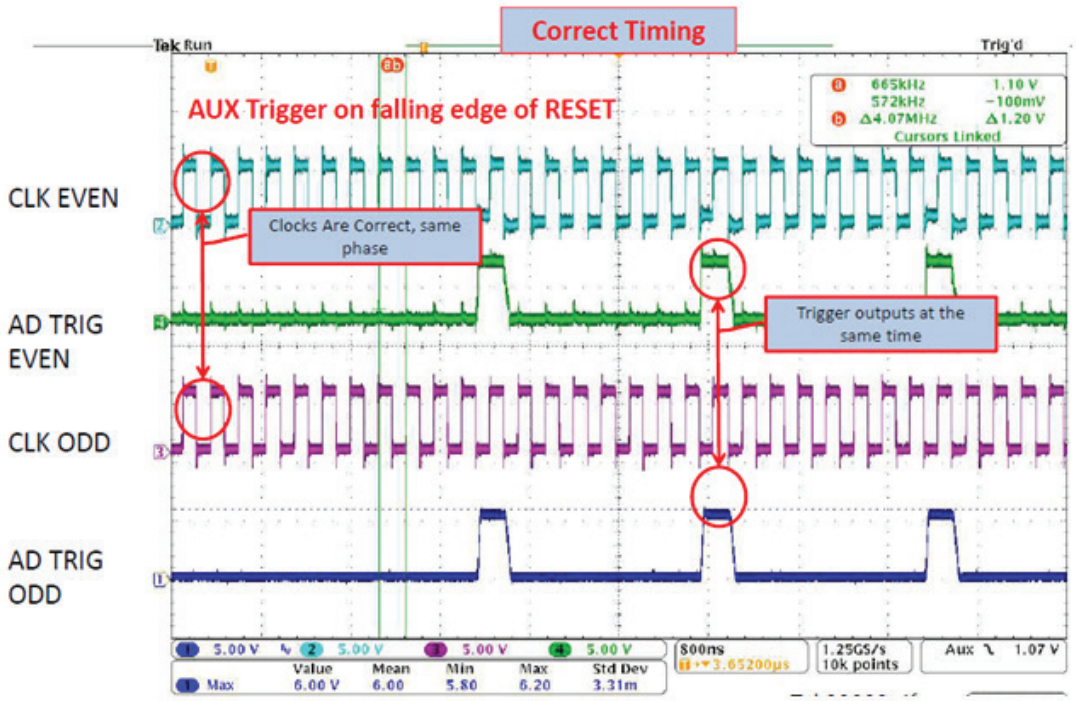


Figure 9-4: Test results of CLK and AD_Trig signals at MUX Timing (1)

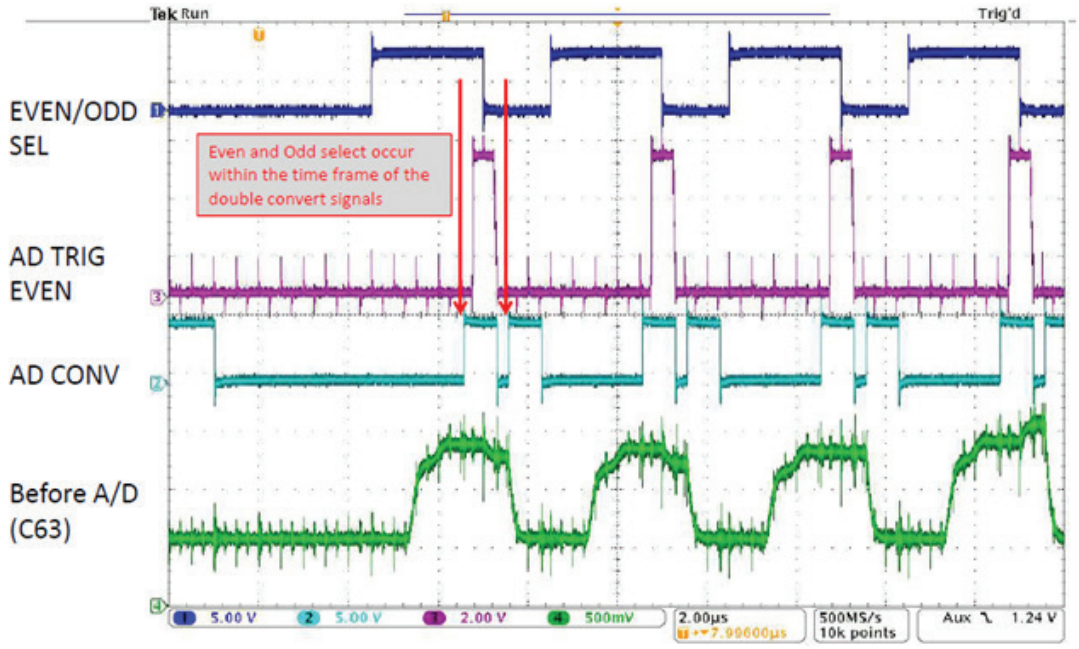


Figure 9-5: Test results of video output signal at MUX Timing (1)

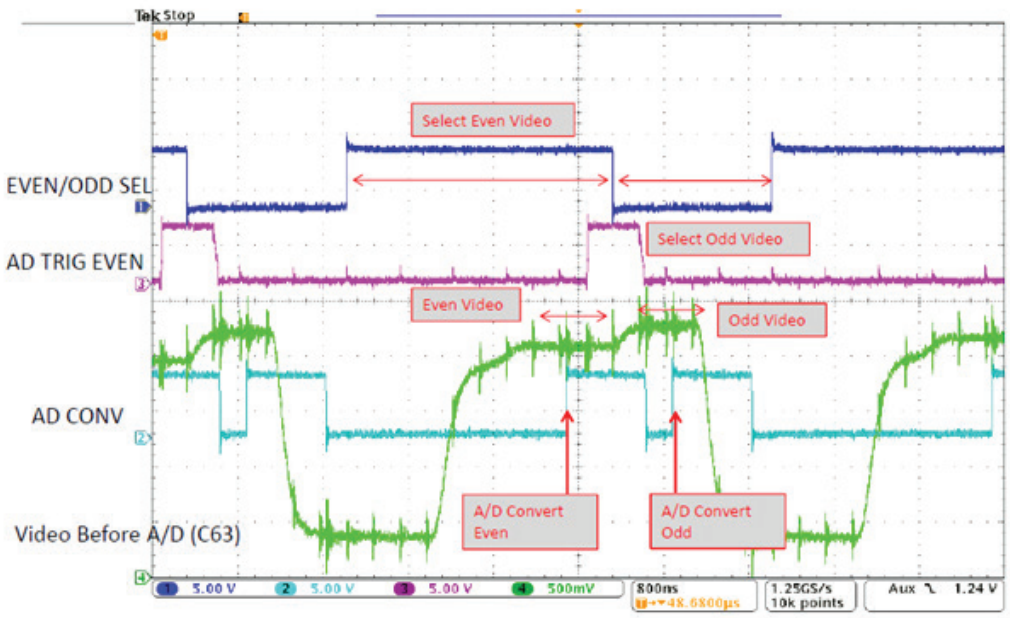


Figure 9-6: Test results of Even/Odd SEL and video output signals at MUX Timing (1)

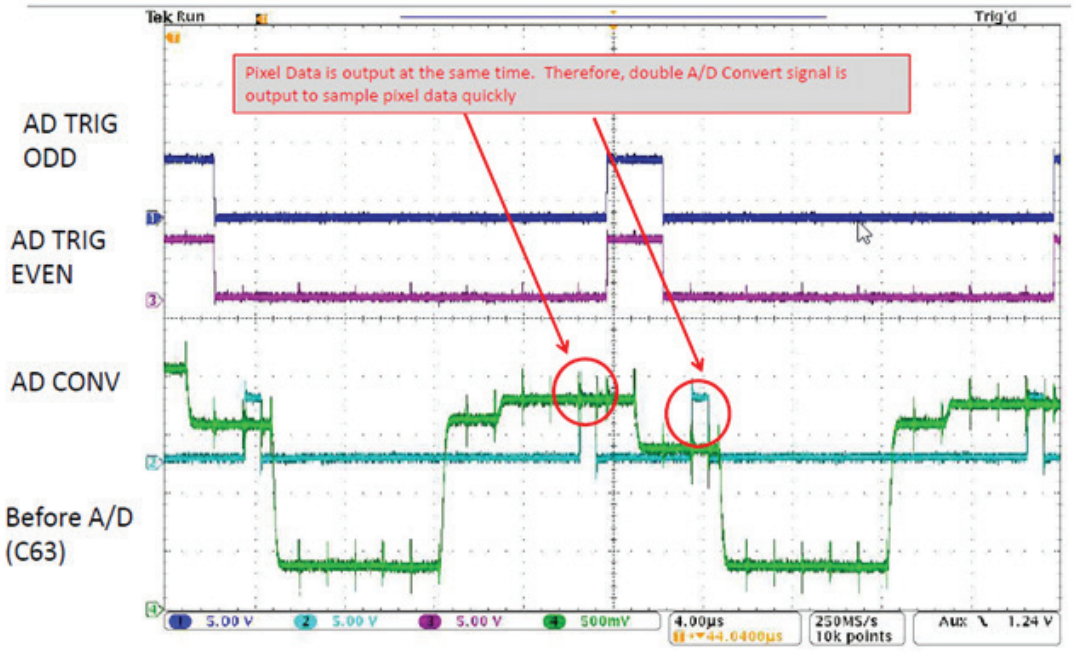


Figure 9-7: Test results of AD_CONV and video output signals at MUX Timing (1)

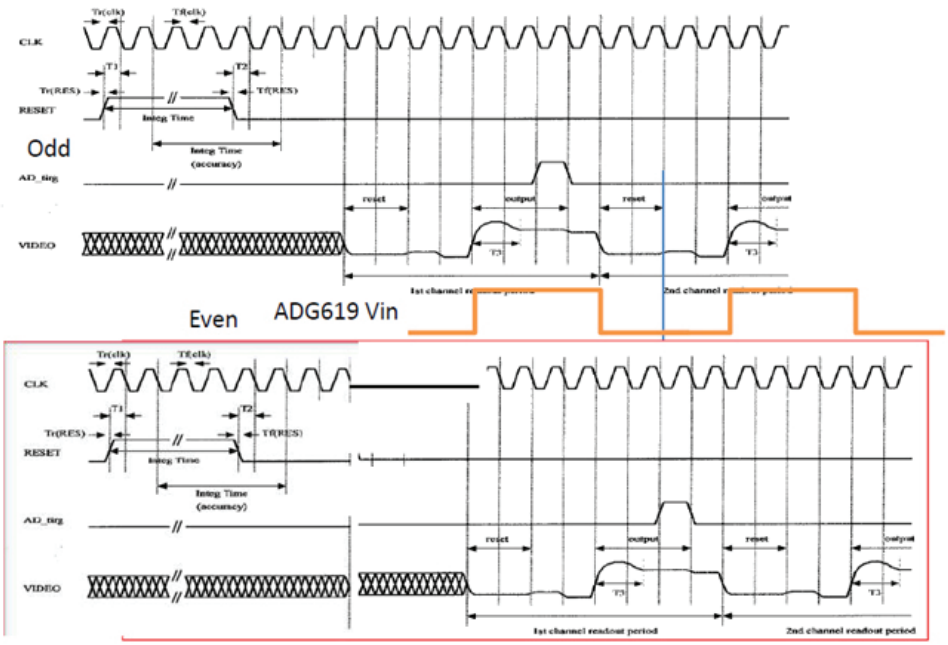


Figure 9-8: Timing diagram of MUX Timing (1a)

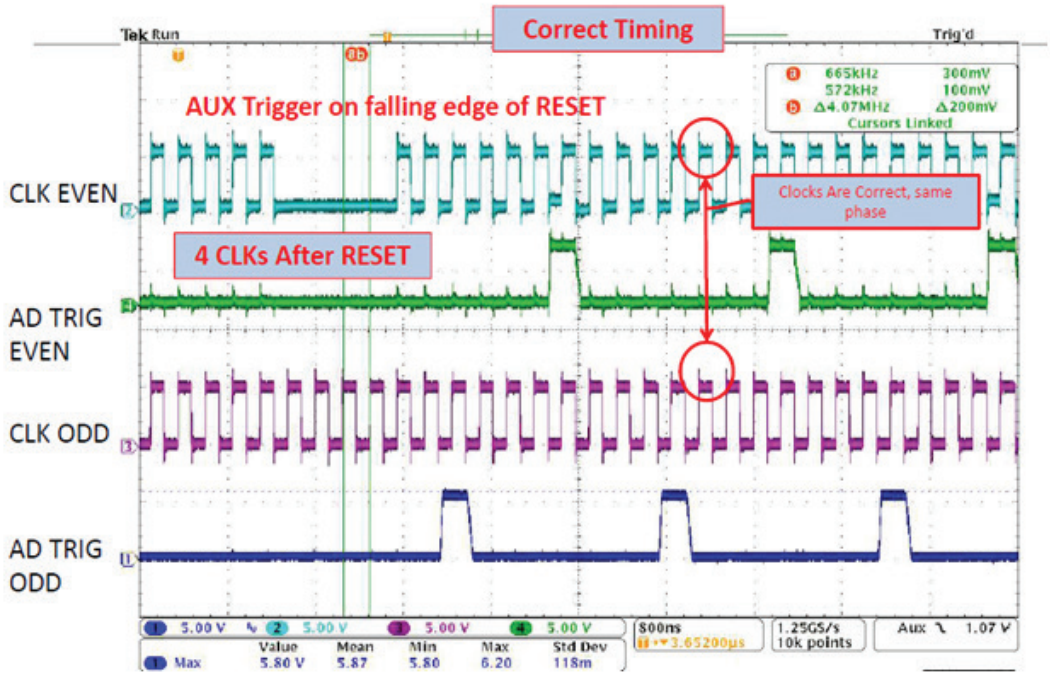


Figure 9-9: Test results of CLK and AD_Trig signals at MUX Timing (1a)

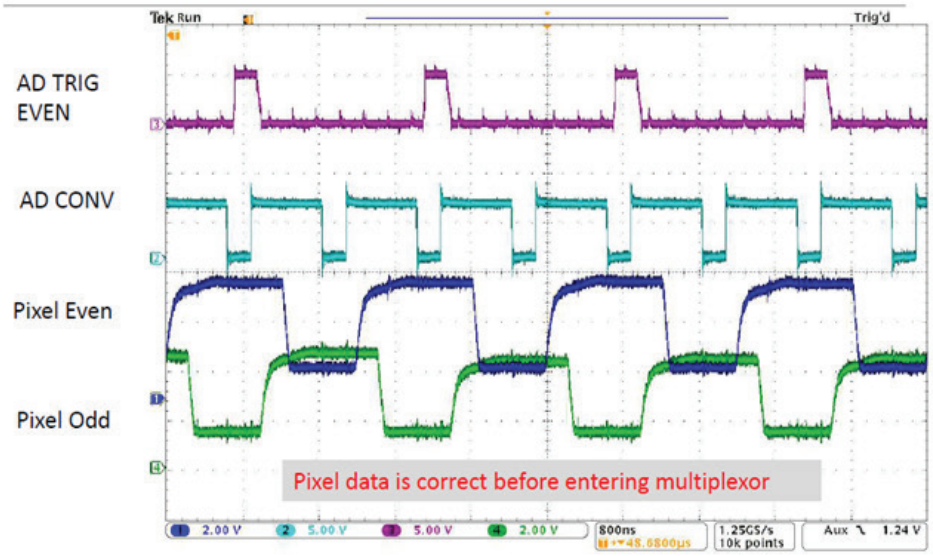


Figure 9-10: Test results of AD_CONV and video signals at MUX Timing (1a)

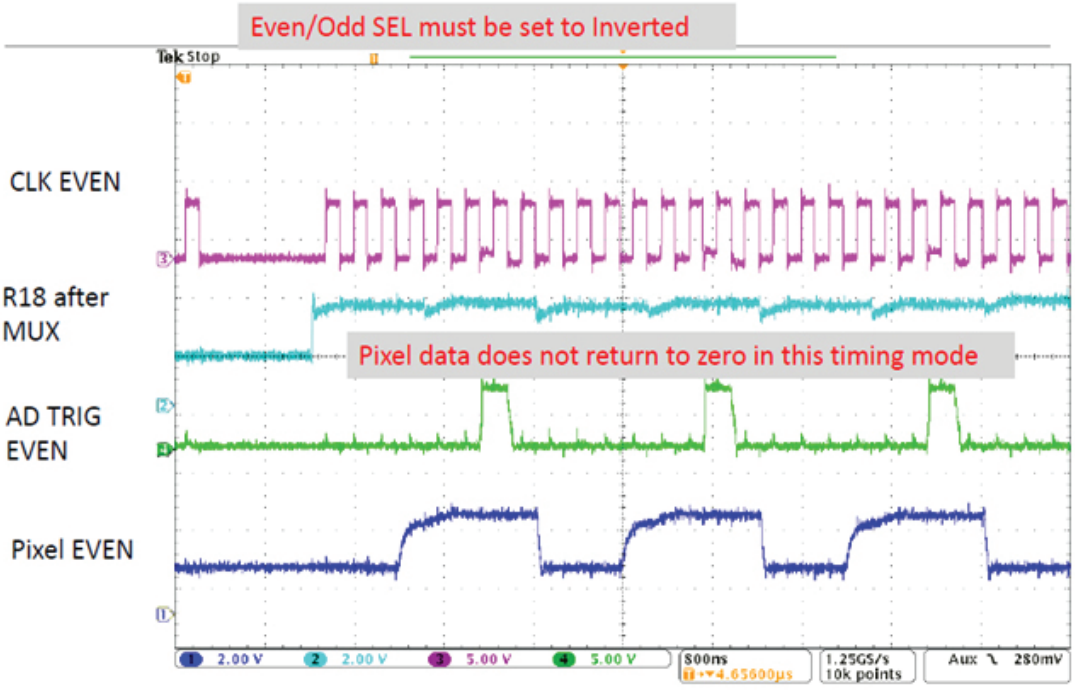


Figure 9-11: Test results of CLK and MUX output signals at MUX Timing (1a)

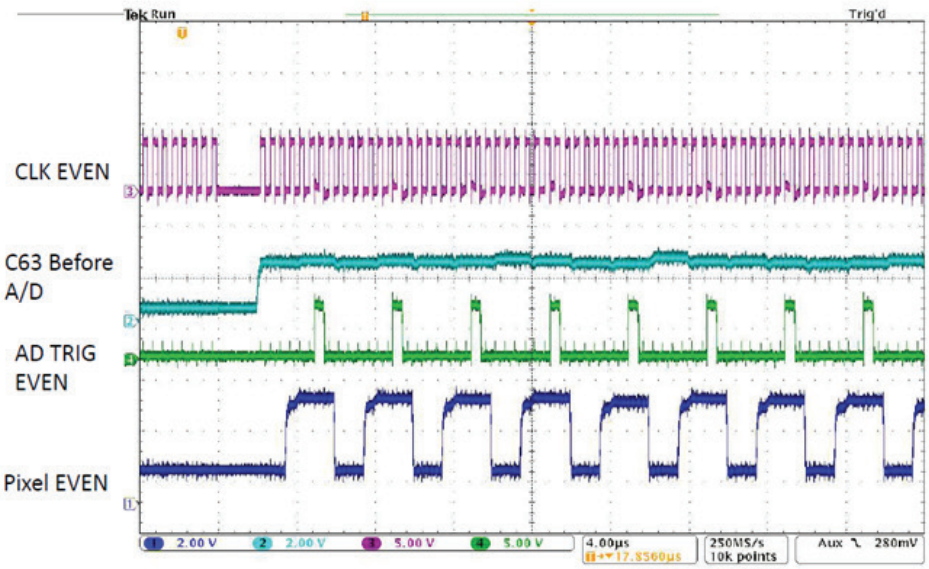


Figure 9-12: Test results of CLK, AD_Trig and video output signals at MUX Timing (1a)

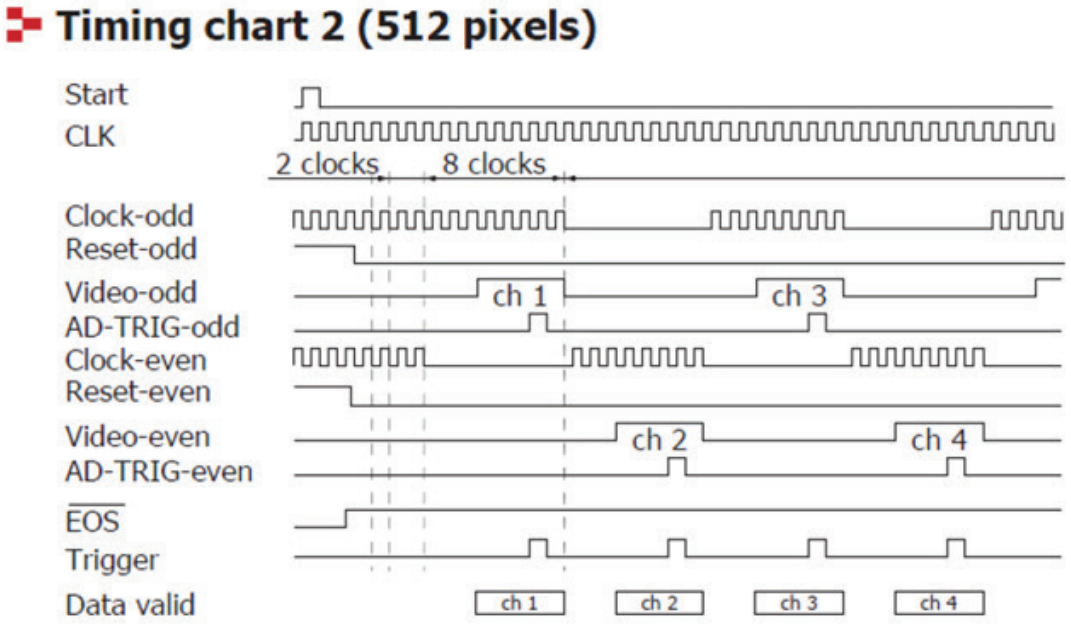


Figure 9-13: Timing diagram of MUX Timing (2)

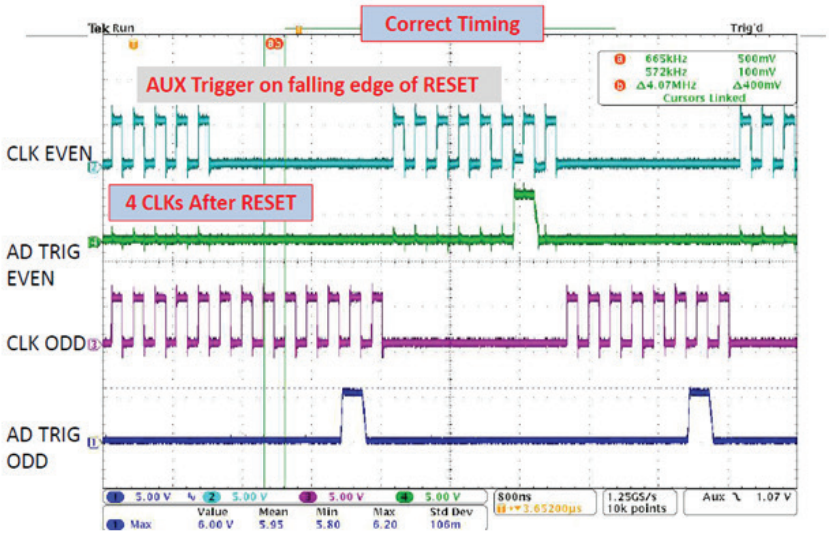


Figure 9-14: Test results of CLK and AD_Trig signals at MUX Timing (2)

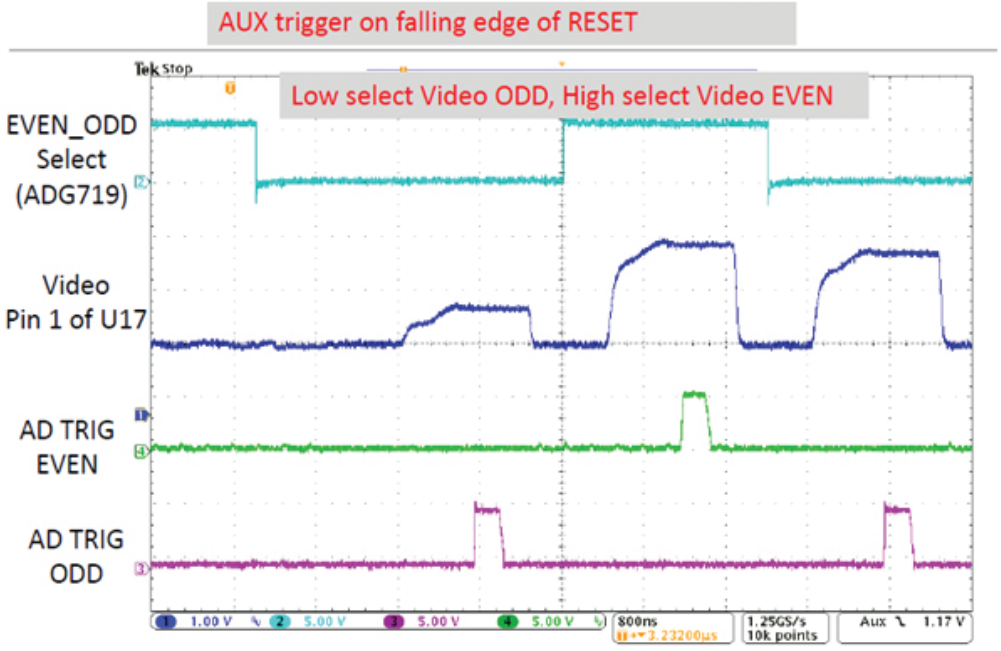


Figure 9-15: Test results of Even/Odd SEL and video output signals at MUX Timing (2)

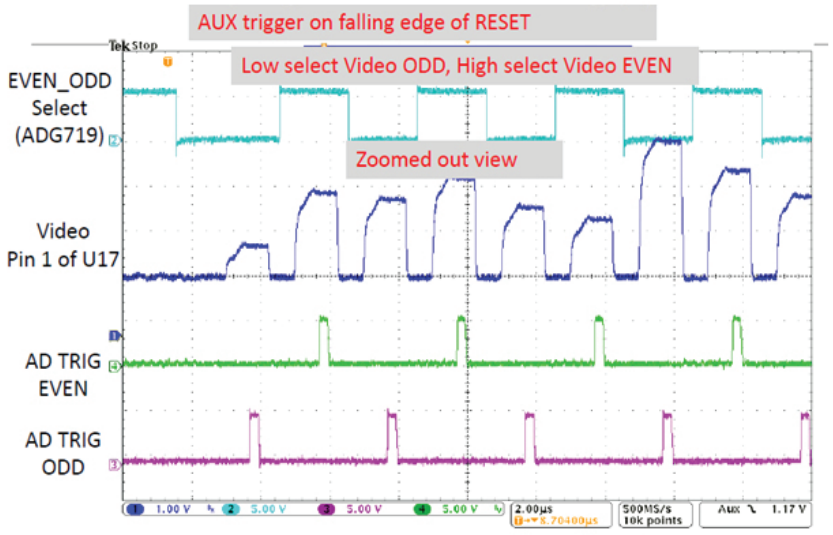


Figure 9-16: Zoom-out view of Figure 9-15

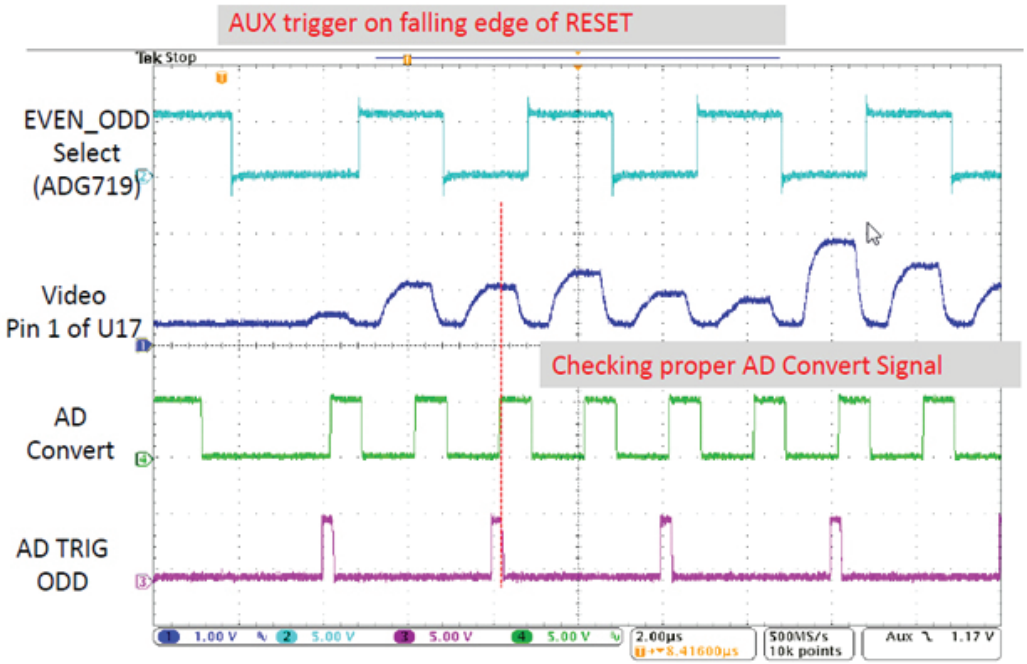


Figure 9-17: Test results of AD_CONV and video output signals at MUX Timing (2)

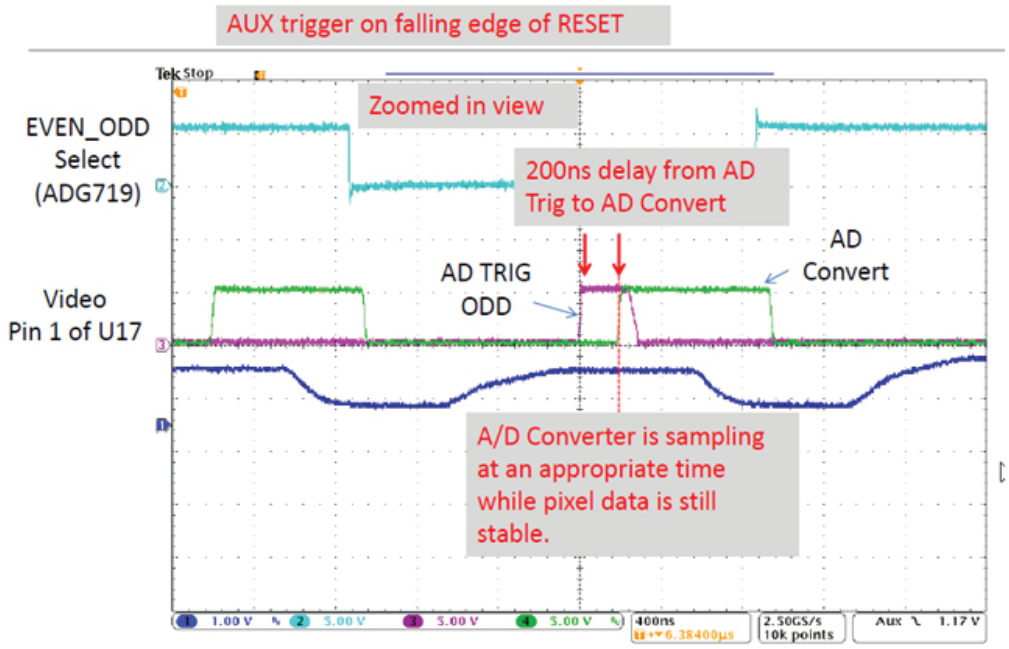


Figure 9-18: Zoom-in view of Figure 9-17

9B. Dark Stability

To verify the dark output is stable over time, indicating the InGaAs sensor cold-side temperature is stable at 0 deg. C., 600 scans are collected at 1 second integration time in the dark and the standard deviation is calculated for each pixel. *Note: The level grouping is caused by Even/Odd CMOS ROIC variation. This is normal Even/Odd output pattern.*

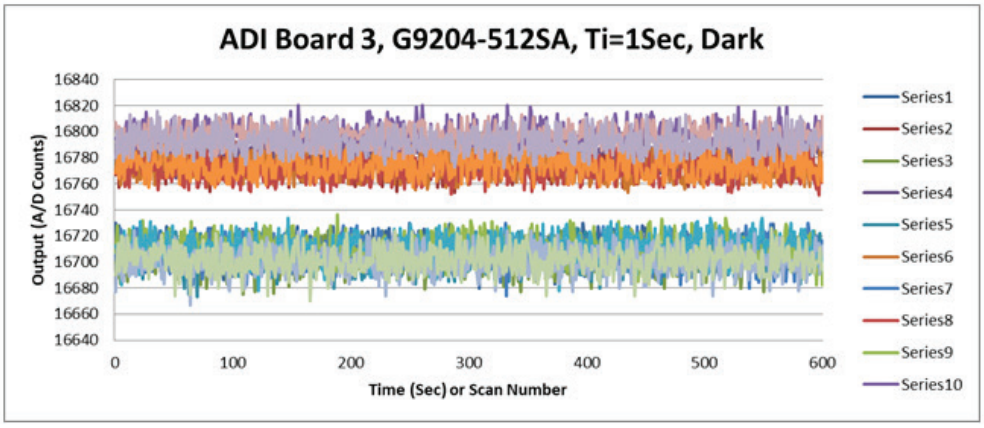


Figure 9-19: Dark measurement at 1 sec integration time

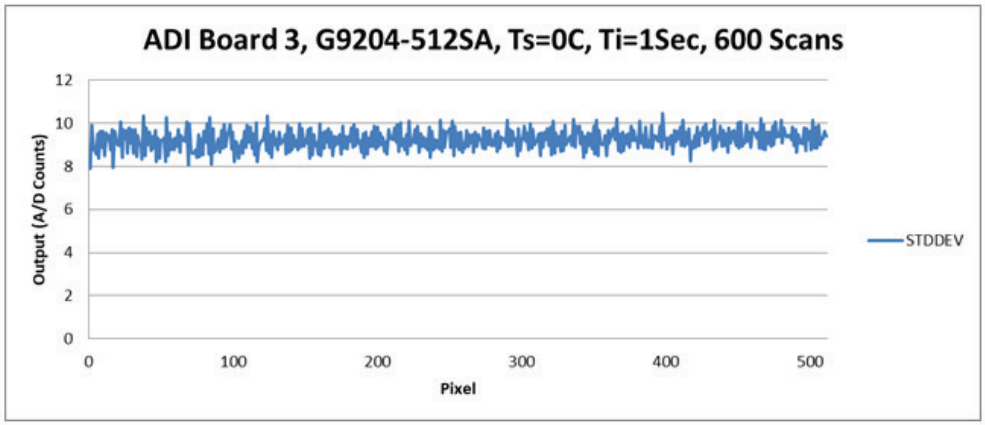


Figure 9-20: Standard deviation of 600 dark scans

9C. Noise

The readout noise was measured at 1 millisecond (msec) integration time.

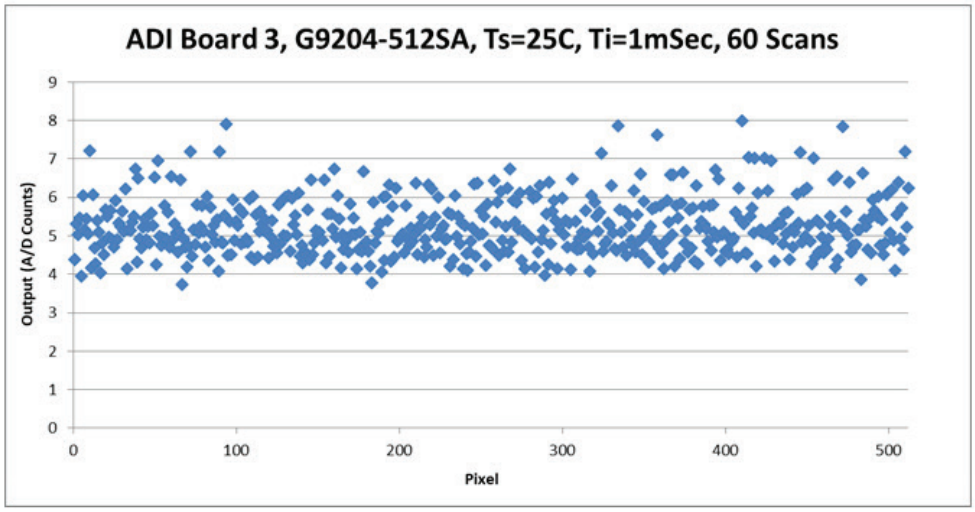
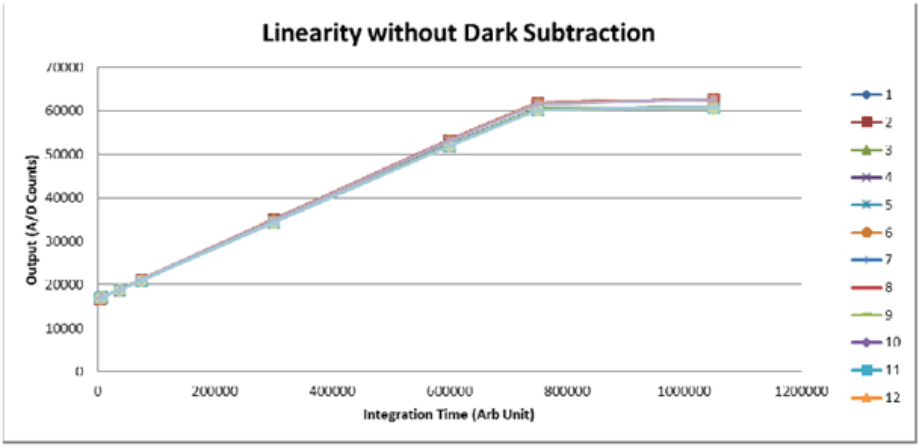


Figure 9-21: Readout measurement

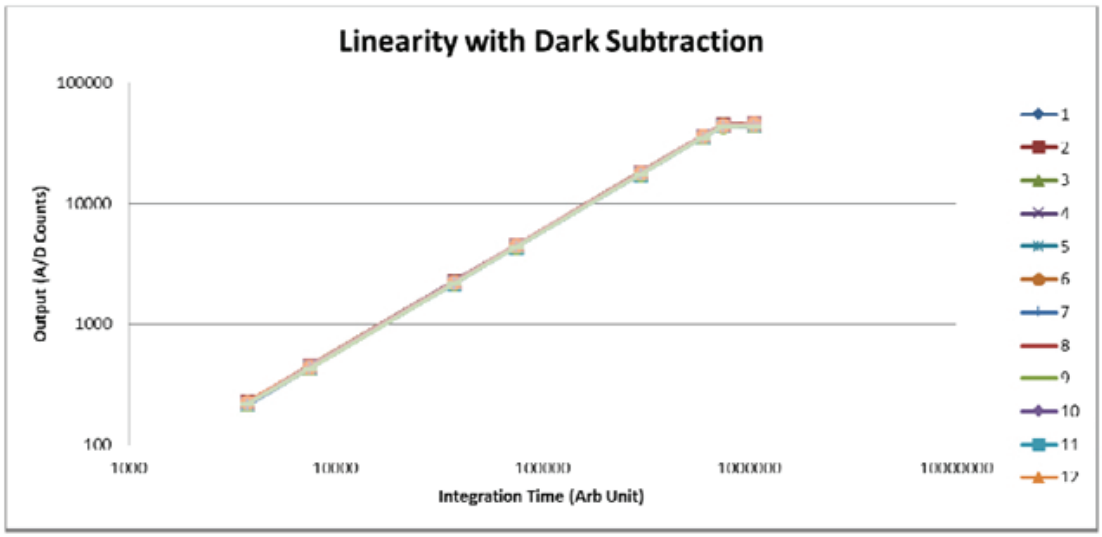
Since the unity gain and the 5V reference voltage of ADC are used, one LSB represents 76.3μV. The readout noise is in the range of 300μV~600μV.

9D. Linearity



LED light intensity is fixed, LED stability was verified. Vary the integration time to determine the linearity (double integration and signal should exactly double).

Figure 9-22: Linearity measurement with varying integration time



Subtract the dark signal at each integration setting. Individual pixel linearity plotted on log / log scale.

Figure 9-23: Linearity measurement with dark subtraction