Spring 5-2020

A High Frequency Photoacoustic System for Colorectal Cancer Imaging

Kexin Huang
Washington University in St. Louis

Washington University in St. Louis

McKelvey School of Engineering

Department of Biomedical Engineering

Thesis Examination Committee:

Quing Zhu, Chair

Song, Hu

Chao, Zhou

A High Frequency Photoacoustic System for Colorectal Cancer Imaging

By

Kexin Huang

May 2020

St. Louis, Missouri

Acknowledgements

I am grateful to everyone I worked with at the Ultrasound and Optical Imaging lab for their help and support. I would like to especially thank my thesis adviser, Professor Quing Zhu, who has provided help and guidance during my entire master's research. During my learning process, she guided me in overcoming academic challenges, analyzing my experimental results, and helping me review my thesis. Her help has been invaluable in accomplishing my thesis project.

I also would like to thank all the lab mates for their kindness and support. Thanks to Guang Yang and Hongbo Luo, with whom I always performed experiments, especially for their research ideas and critical feedback on my projects.

Finally, I appreciate the Washington University School of Engineering for giving me this platform to conduct research that I am so interested in.

Kexin Huang

Washington University in St. Louis

May 2020

Table of Contents

List of Figures

Abstract

A High Frequency Photoacoustic System for Colorectal Cancer Imaging

By

Kexin Huang

Master of Science in Biomedical Engineering

Washington University in St. Louis, 2020

Research Advisor: Professor Quing Zhu

While colorectal cancer is the second largest cause of cancer-related deaths in the United States, early detection is a key factor in its survival rate. Compared to conventional imaging modalities, photoacoustic imaging offers benefits in providing angiographic images which are valuable for early-stage tumor detection. This thesis presents the design of a 32-channel 80 MHz photoacoustic image system, whose relatively high frequency offers particular advantages.

The system comprises several modules, including a laser system, ultrasound probe, AD convertor, microcontroller (FPGA), and a computer. The system requires programs for the FPGA and the data receiver on the computer. The data transportation accuracy, signal-noise ratio, and transmission speed are analyzed here to understand the system's performance.

The system was tested by standard phantoms, like carbon fiber and black tape, and images were reconstructed by the typical delay-and-sum algorithm. As important parameters of this system, the spatial resolution and signal-to-noise ratio were analyzed.

In the future, to improve the lateral resolution of this system and broaden its imaging window, we can expand the 32-channel system up to 256 channels simply by duplicating the 32-

channel data acquisition module. This improved system can provide detailed ex-vivo and in-vivo information about colorectal cancer.

Chapter 1: Introduction

This chapter provides background information on colorectal cancer and photoacoustic imaging to clarify the motivation for building a high frequency photoacoustic system.

1.1 Colorectal Cancer

In the United States, colorectal cancer (CRC) is currently the third most commonly diagnosed cancer, with the second highest cancer-related death rate. However, the mortality of CRC has declined over the past three decades due to treatment development (12%), changing patterns in CRC risk factors (35%), and screening (53%). [1-3] The advancement of imaging technology plays an essential role in the improving the post-treatment outcomes.

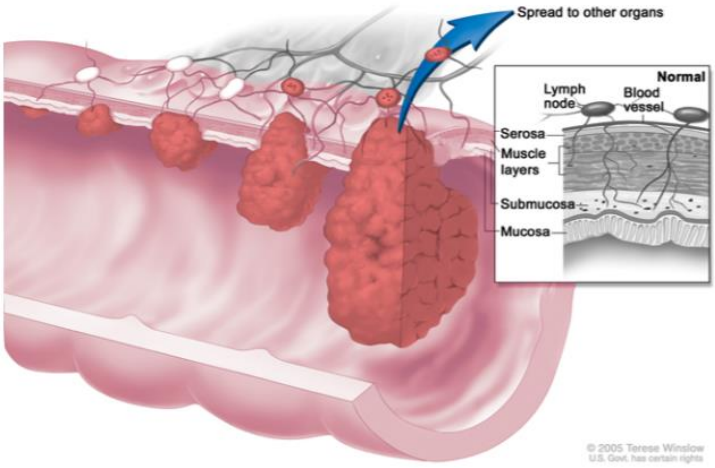

Figure 1 Colorectal Cancer Staging [4]

CRC statistics show that early detection is key to increasing the survival rate, and the survival is highly correlated with the CRC stage. [4, 5] Figure 1 displays how the malignant cancer progresses through different stages. From stage 0 to stage III, a clump of abnormal cells starts at the mucosa layer, which is the inside surface of the colon or rectum, then gradually and successively invades the submucosa, the muscle, pericolorectal tissues, nearby organs or

structures, and finally grows close to lymph nodes. In stage IV, the cancer cells spread to distant organs or lymph nodes.

The 5-year survival rate of CRC is about 90% if the cancer is diagnosed at an early stage where it stays at the original site (stage 0 to stage III), and, in general, the 5-year survival rate of CRC cases is 64%. [4, 5] From these statistics, we can see the motivation for developing an effective early-stage tumor detection modality to optimize CRC treatment outcomes.

1.2 Photoacoustic Imaging

Photoacoustic imaging is based on the photoacoustic effect, first reported by Alexander Graham Bell in 1880 [6]. Figure 2 illustrates photoacoustic phenomenon [7]. In photoacoustic imaging, the target is always irradiated by a short-pulsed laser beam. During this process, the target absorbs light fluence and converts a part of this energy into heat, which is further converted into mechanical energy by a pressure rise caused by thermoelastic expansion. This mechanical energy is propagated in the form of an ultrasonic wave, called a photoacoustic wave [8].

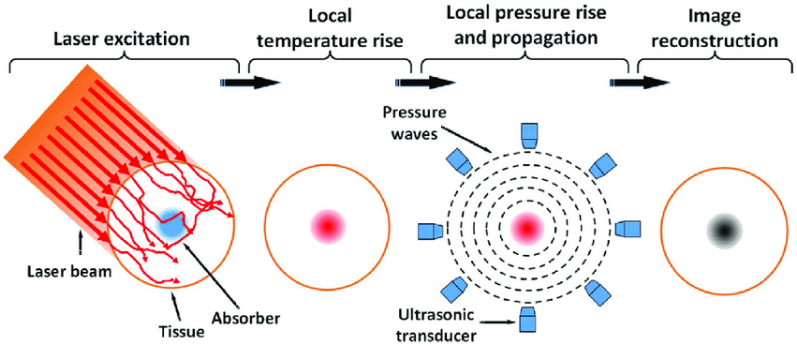

Figure 2. Photoacoustic Effect [7]

Before discussing the properties of photoacoustic imaging further, we need to review the pros and cons of optical imaging and acoustic imaging. Optical imaging is always limited by light diffusion, a fundamental constraint on spatial resolution and signal penetration, although it is

sensitive to endogenous oxy- and deoxy-hemoglobin, which provides rich information about vascular structures. Meanwhile, ultrasound has the advantages of deep penetration and good resolution in the optical quasidiffusive or diffusive regimes, but it provides only mechanical property information, which is comparatively limited for early-stage cancer diagnosis. [8, 9]

However, photoacoustic imaging combines the benefits of optical imaging and ultrasound imaging and overcomes the limitations of each, allowing researchers to obtain high resolution angiographic images in optical diffusive regimes, which could be critical information for early-stage cancer diagnosis [10-13]. With all the advantages above, photoacoustic imaging technology has already entered a period of rapid evolution in different medical areas [13-18].

1.3 Significance of High Frequency Photoacoustic Imaging

For in vitro experiments, the spatial resolution and sensitivity of a photoacoustic imaging system are essential to fulfill the requirements of an early-stage colorectal cancer study. By using a high frequency ultrasound probe in conjunction with a high sampling rate data acquisition system, high resolution photoacoustic images can be obtained to display the vascular structure in detail, which allows early-stage colorectal cancer to be distinguished by the formation of abnormal vessels.

This thesis describes a new 80 MHz 32-channel high frequency photoacoustic image system. This system is based on a high-frequency transducer array (Vevo 2100 MS 200, 15 MHz central frequency, 9-18 MHz bandwidth) with an 80 MHz high frequency AD convertor (AFE5807EVM) to improve the spatial resolution of the photoacoustic system. A detailed system design strategy and testing results are discussed, and future directions for pre-treatment human colorectal studies and system evaluations are proposed.

Chapter 2. High Frequency Photoacoustic Imaging System

In this chapter, the system architecture design is explained, including the hardware connection of the different modules, the programming structure of the FPGA and the programming structure of the data receiver on the computer. Furthermore, system testing results of the data transport accuracy, the signal-noise ratio, and the data transmission speed are analyzed to exhibit the system's performance.

2.1 Photoacoustic Imaging Theory

A short-pulsed laser beam that illuminates tissue generates an acoustic wave due to the photoacoustic effect, and such an acoustic signal can be captured by an ultrasound probe for image reconstruction. During the laser heating process, two important timescales need to be considered: the thermal relaxation time and the stress relaxation time. To set the heat conduction and the stress propagation to a negligible level, the laser pulse width has to be much shorter than the thermal relaxation time and the stress relaxation time so that the excitation could be said to be in thermal confinement and stress confinement. In this case, the photoacoustic wave equation can be expressed as follows [19]:

$$\left(\nabla^2 - \frac{1}{{v_s}^2}\frac{\partial^2}{\partial t^2}\right) p(\boldsymbol{r}, t) = -\frac{\beta}{C_p}\frac{\partial H}{\partial t}, \qquad (1)$$

where v_s is the speed of sound, β is the thermal coefficient of volume expansion, C_p denotes the specific heat capacity at constant pressure, H is the heating function defined as the thermal energy converted per unit volume and per unit time, and $p(\boldsymbol{r}, t)$ is a pressure propagation function. Applying a Green's function approach to the equation above yields the following pressure propagation solution $p(\boldsymbol{r}, t)$ [8]:

$$p(\boldsymbol{r},t)=\frac{\beta}{4\pi C_p}\frac{\partial}{\partial t}\int d\boldsymbol{r}'\frac{1}{|\boldsymbol{r}-\boldsymbol{r}'|}H\left(\boldsymbol{r}',t-\frac{|\boldsymbol{r}-\boldsymbol{r}'|}{v_s}\right), \qquad (2)$$

If three conditions are satisfied—the object motion within the pulse duration is negligible, the profile of the light pulse received at each point can be regarded as identical, and the photothermal conversion efficiency at each point is identical—then the heating function $H(\boldsymbol{r},t)$has spatiotemporal separability, which means it can be decomposed as following s[8]:

$$H(\boldsymbol{r},t)=H_s(\boldsymbol{r}')H_t(t'). \qquad (3)$$

Under thermal confinement and stress confinement, the initial pressure response due to the laser beam heating can be expressed as follows [8]:

$$p_0(\boldsymbol{r})=\mathrm{r}H_s(\boldsymbol{r}'), \qquad (4)$$

where r is the Grueneisen parameter, defined as $\mathrm{r}=\beta v_s^2/Cp$. Thus, equation (2) can be simplified as [8]

$$p(\boldsymbol{r},t)=\frac{1}{4\pi {v_s}^2}\frac{\partial}{\partial t}\left[\frac{1}{v_s t}\int d\boldsymbol{r}'p_0(\boldsymbol{r}')H_t\left(t-\frac{|\boldsymbol{r}-\boldsymbol{r}'|}{v_s}\right)\right]. \qquad (5)$$

2.2 System architecture

The structure of the high frequency photoacoustic system designed in this project can be roughly divided into three major parts: (1) the hardware modules, including the laser, ultrasound probe, AD convertor, microcontroller, and computer; (2) the programming of the field programmable gate arrays (FPGA); (3) the data receiver program on the computer. The detailed information about each part will be given in the following sections. Furthermore, testing results for this prototype will also be provided in this chapter so that an accurate data acquisition process can be guaranteed.

2.2.1 Hardware Design

Figure 3 shows the entire structure of this 32-channel 80 MHz high frequency photoacoustic prototype. First, the laser system (a Ti:sapphire laser (Symphotics TII, LS-2134, Symphotics, Camarillo, California), optically pumped with a Q-switched Nd:YAG laser (Symphotics TII, LS-2122), delivers a light pulse into a simple optical delivery system to an optical fiber tip [20, 21]. Then, the phantom illuminated by the laser beam generates an acoustic wave due to the photoacoustic effect, and this acoustic wave is captured by the high frequency ultrasound probe array (Vevo 2100 MS 200, 15 MHz central frequency, 9-18 MHz bandwidth) and converted into the electrical signal. Next, the signal is passed to the AD converters (AFE5807EVM), with an 80 MHz sampling frequency. Finally, the signal goes to an FPGA (Xilinx Artix 7) programmed to save the data synchronously with the laser trigger and transport the data package back to the computer upon command.

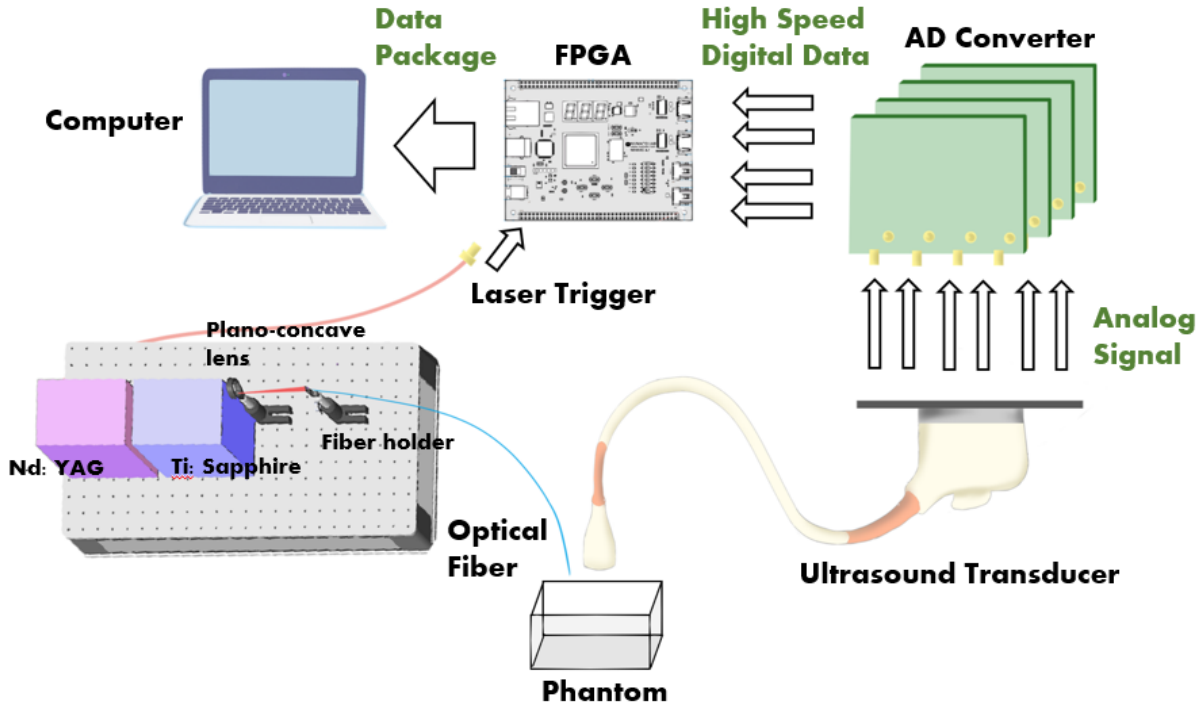

Figure 3. Structure of the high frequency photoacoustic system

(1) Laser system setting

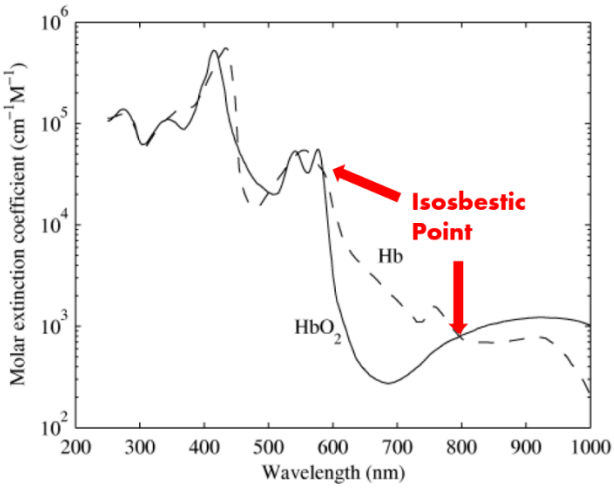

Figure 4. Molar Extinction Spectra of Hemoglobin [22]

In this prototype, a Ti:sapphire laser (Symphotics TII, LS-2134, Symphotics, Camarillo, California) is optically pumped with a Q-switched Nd:YAG laser (Symphotics TII, LS-2122), and then an optical fiber-based light delivery system carries the light to the target position. As shown on the right side of Figure 4, to take advantage of the anatomical and functional contrasts provided by endogenous oxy- and deoxy-hemoglobin, the wavelength of pulsed laser light should be placed between two isosbestic points to obtain high-resolution anatomical/ vascular images. In this paper, we selected a 750 nm wavelength laser beam to produce images (10 ns pulse duration, 15 Hz pulse repetition rate, 20 mJ/pulse at 750 nm wavelength).

(2) Transducer pin map decoding

To achieve accurate imaging reconstruction, the correct order of the pins on the ultrasound transducer needs to be figured out in order to match each channel data with the correct element. This probe contains 360 pins, and 256 of them corresponding with 256 piezoceramic elements arranged in a line. Figure 5 depicts how the order of the 256 pins in this linear transducer was correctly assigned.

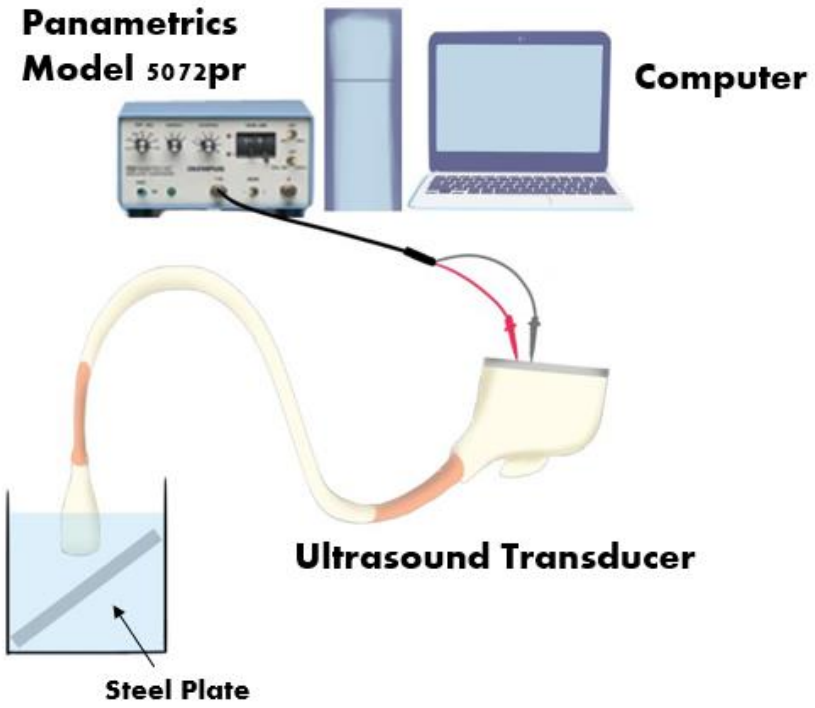

Figure 5. Setup of pin map decoding experiment

A steel plate was submerged in water at an angle to the ultrasound probe width, so the distances from the probe to the steel plate surface differed for each element. To visualize this distance variation, we used an ultrasonic pulser-receiver (Panametrics Model 5072pr) and the computer to fire each channel of this probe and record the echo that bounced back from the steel plate. By calculating the time lapse between the fired pulse and the received pulse, data obtained from all the 256 pins could be arranged in the order of distance values, which was in proportion to this time lapse. For our prototype, we selected channel 1 to channel 32 to connect with our high frequency photoacoustic system. The pin map decoding results are in Appendix 1.

(3) Analog/Digital Convertor

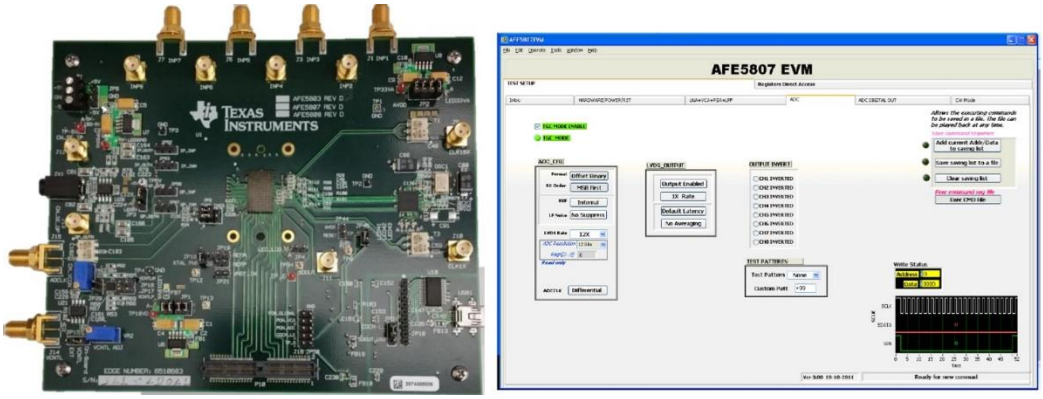

Figure 6 ADC module (left) and the control panel (right) [23]

For this system, we selected a commercial AFE5807EVM board as our AD converter, as displayed in Figure 6. This board has eight analog signal input channels and eight digital output channels. Four boards were needed to achieve 32-channel data acquisition. The sampling rate of this board can be varied from 40 MHz to 80 MHz, according to the input ADC clock. This board includes eight channels of voltage-controlled amplifier (VCA) for the 12-bit Analog-to-Digital Converter (ADC) that can be adjusted by the control panel. The VCA contains four major programmable parts: a low noise amplifier (LNA) that supports 250 mVPP to 1 VPP input signals, a voltage controlled attenuator (VCAT), a programmable gain amplifier (PGA) that can be configured as 24 dB or 30 dB, and a low-pass filter (LPF) with four cutoff options, 10 MHz, 15 MHz, 20 MHz, or 30 MHz [23] .

For an 80 MHz sampling frequency, this ADC board needs an external clock to trigger the sampling. This clock plays a critical role in data sampling and LVDS stream transmssion, so it has to be fairly precise or severe data shifting will occur. We employed another FPGA (Cyclone V) to generate four clocks with exactly the same 80 MHz clock speed.

(4) Related circuit design

Three circuit boards are employed in this prototype, two for analog signal transport and one for digital data transport.

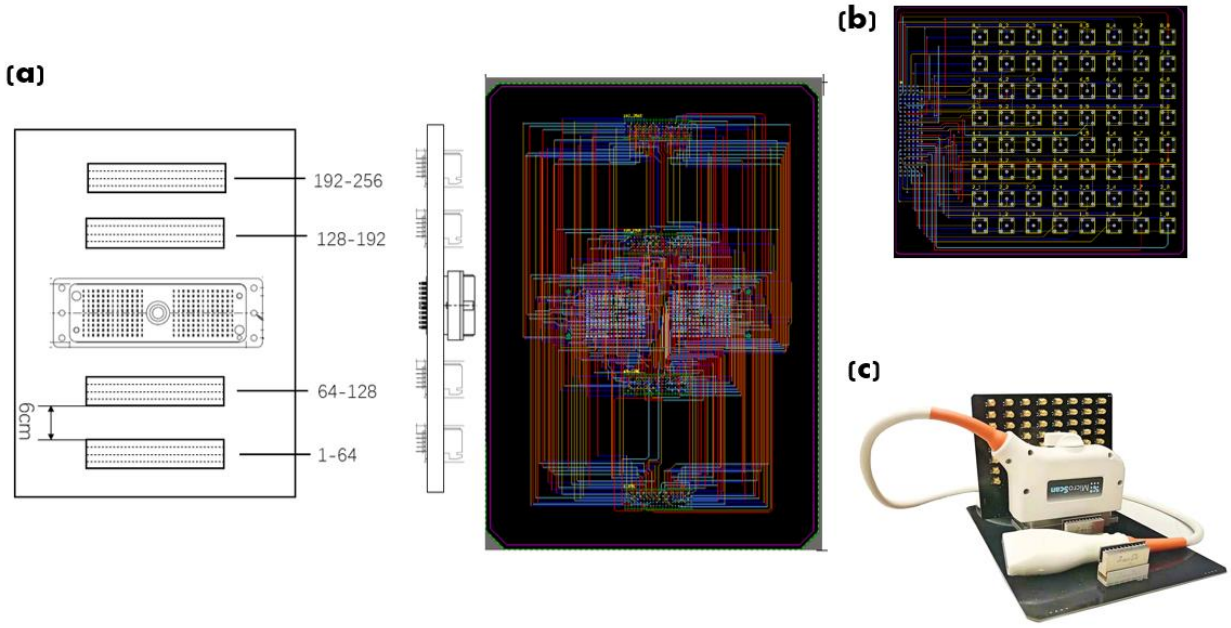

Figure 7. Circuit board design for analog signal transport: (a) motherboard, (b) adapter board, (c) system photo

The analog signal circuits include one motherboard and four adapter boards, as shown in Figure 7. On the motherboard (Figure 7a), all 256 channels are connected to output connectors and divided into four parts, with 64 channels on each part. Plugged into the motherboard, each adapter board (Figure 7b) converts the 64 channel outputs into SMA ports so they can be connected with the ADC modules. Since we designed only 32 channels for this prototype, only one adapter board is used in the following system design, as Figure 7c shows. Other unused pins are grounded in order to eliminate the noise generated by the suspend pins. The motherboard is an 8-layer circuit board with four signal layers and four internal planes. All the internal planes are grounded, and one is placed between every two signal layers. Furthermore, to avoid parallel signal crosstalk, the wire space is at least twice the wire width, and for adjacent signal layers, the wire directions are perpendicular to each other to guarantee satisfactory signal quality.

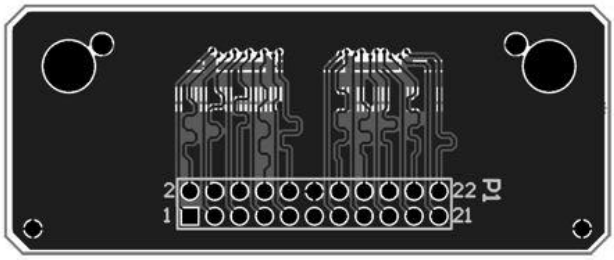

Figure 8 Circuit board for digital data transportation

The design of the digital converter is much simpler than that of the other circuit boards because each converter here contains only eight channels of digital data and two clock channels. Since the ADC module outputs data as a low-voltage differential signaling (LVDS) stream, each channel must be designed to carry the differential signal pairs. To obtain accurate digital data, all these pairs have to be length-matched and impedance-matched. However, the traces here have already been designed to be as short as possible (less than 2 cm) to eliminate impedance mismatch. Thus, we length-matched only the signal wires in this circuit. Four converters are needed in the high frequency photoacoustic system to support 32 channels of data transfer.

2.2.2 FPGA programming

We used a Mimas V2 FPGA Development board based on the Xilinx Artix 7 FPGA (Mimas A7) , chosen for three of its components: 40 length-matched differential pairs, the Xilinx Artix 7 FPGA, and the USB 2.0 host interface based on an FT2232H module. [24]

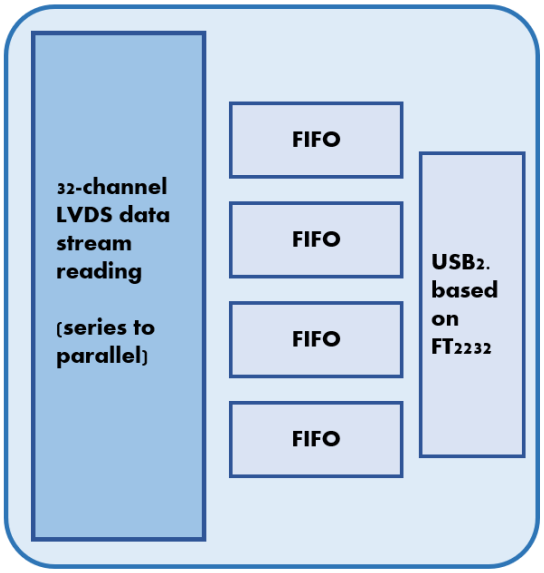

Figure 9. Block diagram of FPGA coding structure

Figure 9 represents the coding structure of the FPGA, including the LVDS data stream acquisition, FIFO reading/writing process, and the USB 2.0 host interface based on a FT2232H module.

(1) LVDS data stream acquisition

Figure 10 displays the time sequence of the LVDS data stream. There are a slow frame clock coming from the input ADC clock (f_{ADC}) and a fast bit clock defined by the ADC resolution (b_r). The frequency of fast bit clock can be calculated as follows:

$$Freq = b_r \times f_{ADC} \div 2. \quad (6)$$

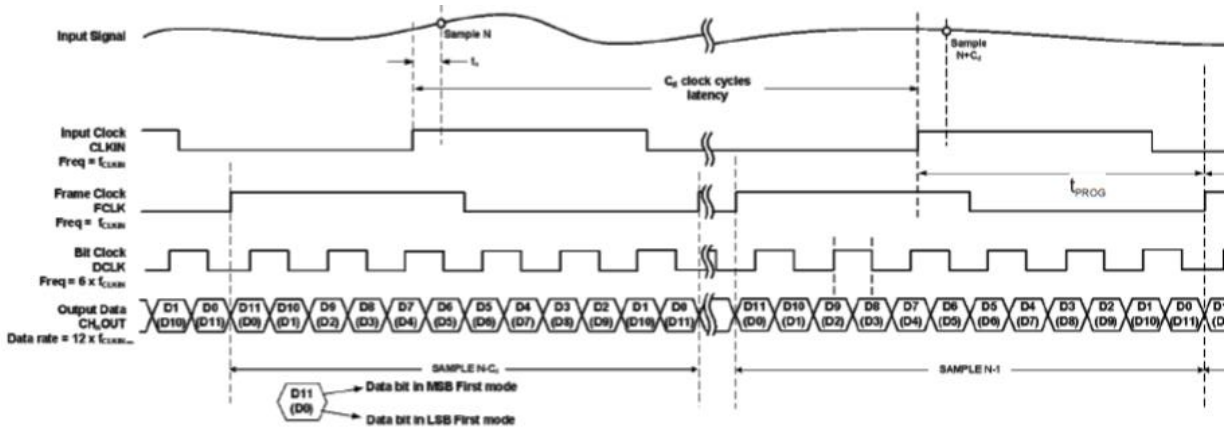

Figure 10. 12-Bit 6x serialization mode [25]

According to Equation (6), for 10-bit and 12-bit resolution, the bit clock should be 400 MHz and 480 MHz, respectively, at a 80 MHz sampling rate. From the diagram above, the data acquisition process is triggered by the dual edge of the fast bit clock, and the phase of this clock is 90 degree shifted compared to the slow frame clock. For the SelectIO IP core provided by Xilinx, we found that the internal PLL clock triggering mode was more stable than the external clock mode, because the fast clock signals from outside are less reliable than the clock signals generated inside the FPGA PLL. Therefore, to trigger the data reading process, we used the PLL IP core to generate this fast bit clock with an accurate phase shift.

(2) FIFO reading and writing

The data writing process of this system is controlled by the laser trigger. Once the rising edge of the laser trigger is detected, the wr_rst pin input of the FIFO is driven high and the module starts to reset, which clears all the data in the FIFO and prepares for writing new data. After the resetting process finishes, the wr_en pin is pulled up and the writing process is enabled. A certain length of data is then pumped into this FIFO. The calculation of this length will be introduced in the following section. Once the full signal is pulled up, the FIFO stops writing data until it is reset by the next laser trigger.

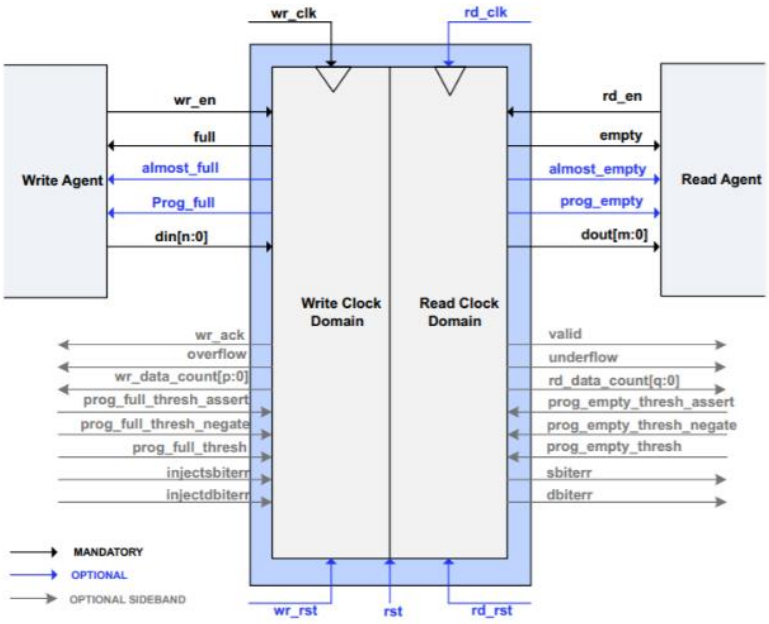

Figure 11. Native Interface FIFOs Signal Diagram [26]

A read command sent from the computer through the FT2232 controls the FIFO readout process. After receiving the read command, the read enable pin is pulled up to start the reading process. During the reading process, the laser trigger reading process is programmed to suspend, so that data overwriting can be prevented. After all the data in the FIFO has been read out, the now-empty pin is pulled up, terminating the reading process until the next read command from the computer is received.

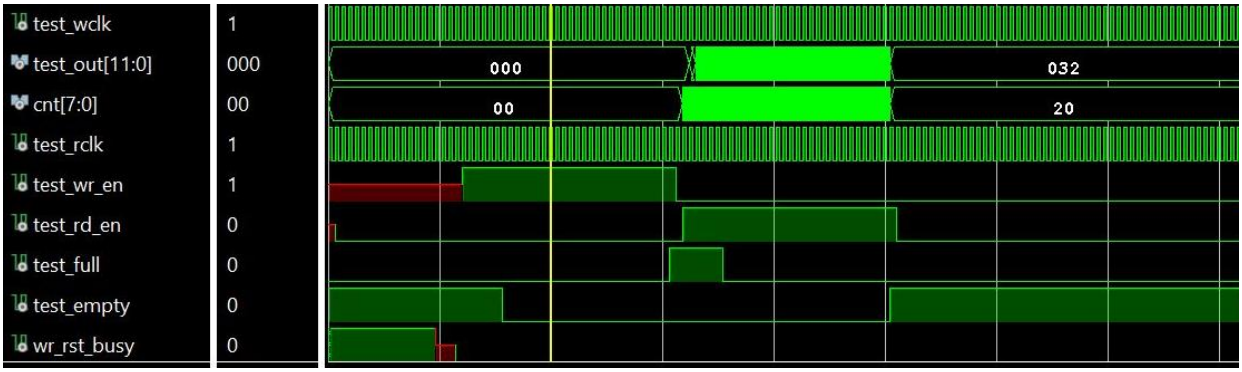

Figure 12. FIFO simulation result

In Figure 12, a FIFO working pattern is simulated with 32 reading and writing lengths. As shown, both the writing and reading process are successful. Since the FPGA is connected to four ADC modules, four FIFOs like this one are programmed inside the FPGA. All the FIFOs begin to empty and write data at the rising edge of the laser trigger, so the data inside the FIFOs is being

sampled at the same time. However, during the reading process, the four FIFOs' reading is enabled in sequence to avoid a data traffic jam.

(3) The USB 2.0 host interface based on the FT2232H

The read and write signal waveforms of the FT2232H in the asynchronous mode are displayed in Figure 13. The data read/write process starts after the falling edge of pin RXF/TXE, which is pulled up or down by commands from the computer. In this mode, the data transfer speed is about 8MB/s.

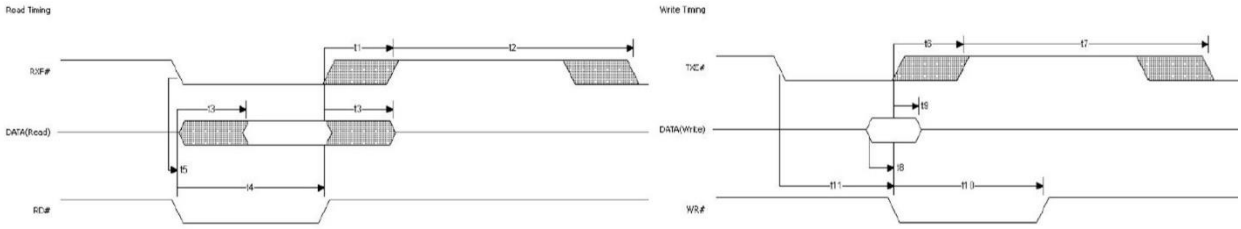

Figure 13. FT245 asynchronous FIFO Interface READ Signal Waveforms(left) and WRITE Signal Waveforms (right) [27]

2.2.3 Computer programming

To send a command from the computer, we can directly call the dynamic link library of the FT2232, using code written in Python. The code for receiving data has five simple steps: (1) Open the corresponding USB port, (2) send a command to the device to suspend trigger reading, (3) send four sequential read commands to read out four data packages written by the four ADC modules, (4) send a command to resume laser trigger reading, and (5) calculate the transfer speed, close the USB port, and write all the data into four TXT files.

2.3 System Testing

The relationship between the data length (N) and imaging depth (D) can be calculated by

$$D = \frac{v_s \times N}{f_{ADC}}, \quad (7)$$

where the v_s is the speed of sound (about 1540 m/s in the water) and f_{ADC} is the sampling frequency (80 MHz). In this system, the data length is set to be 4096, which means the imaging depth should be approximately 7.7 cm.

2.3.1 Ramp pattern testing

A ramp patten is a testing pattern that can be configured in the ADC module by the control panel. In the ramp pattern, the ADC outputs a repeating full-scale code in steps of 1LSB every clock cycle. Passing this test is critical for successful data acquisition because the correct ramp pattern can prove that every number received is accurate with regard to the current resolution and that no data is missing or has shifted during transmission from the ADC to the computer.

Figure 14 shows the ramp test results for four channels randomly selected from the total 32 channels. The resolution here is 10-bit, and the ramp should increase from 0 to 1024 in binary code, which the following results confirm.

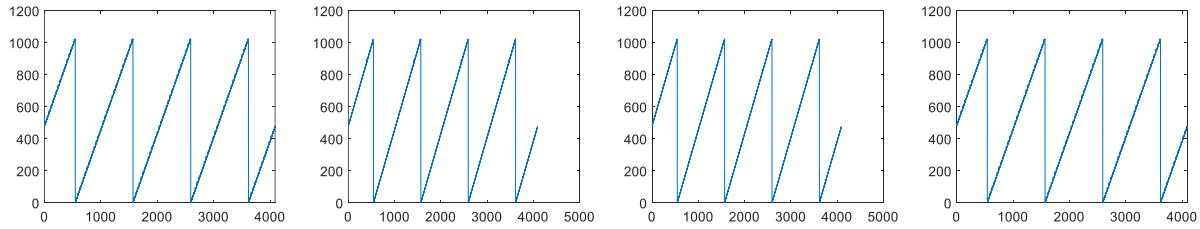

Figure 14. Ramp test results for four randomly selected channels

2.3.2 Signal to noise ratio (SNR) testing with sinusoidal inputs

Sine waves with frequencies of 3 MHz, 6 MHz, and 9 MHz were used to measure the SNR of this system. The peak-to-peak voltage of the sine waves was 70 mV. Figure 15 shows the curves

of the signal data in the frequency domain after fast Fourier transformation (FFT). The SNR of this system is about 40dB. For each frequency, the SNR was calculated by dividing the signal peak value by the standard deviation of the background signal amplitudes. The SNR here is for the specific testing results, so it may vary for different inputs to different channels.

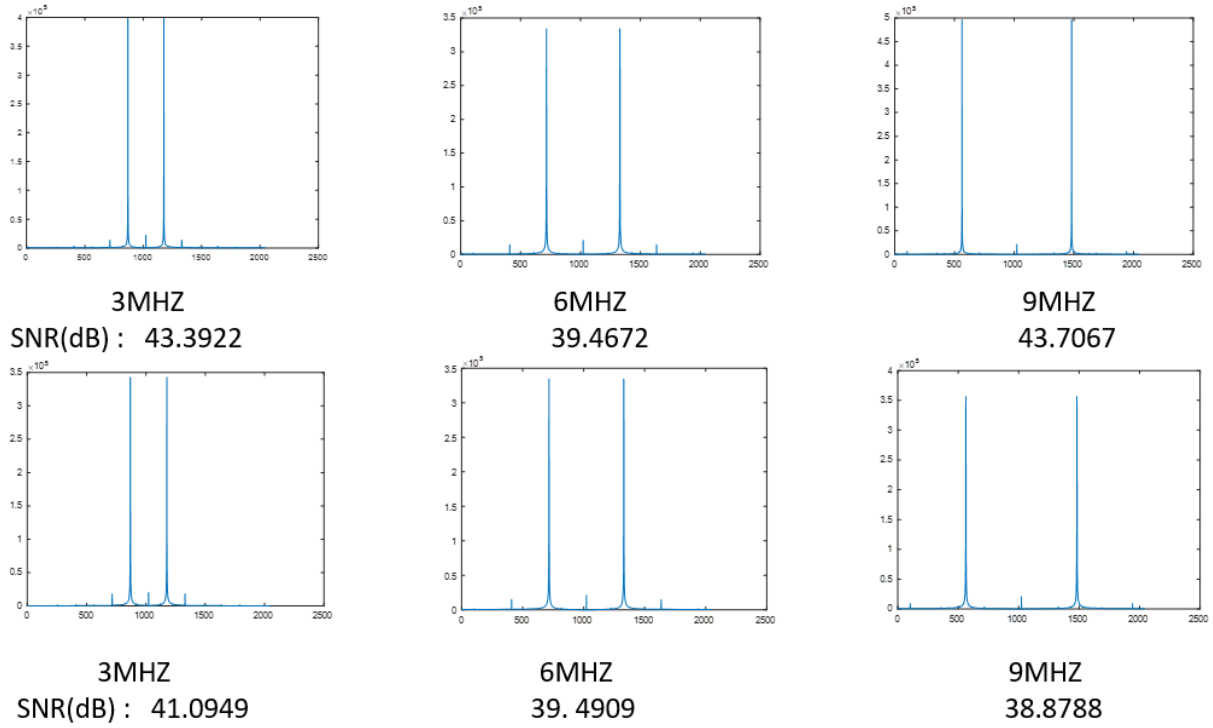

Figure 15. FFT of sine waves (70 mV Vpp) sampled by the ADC

2.3.3 Data transfer speed

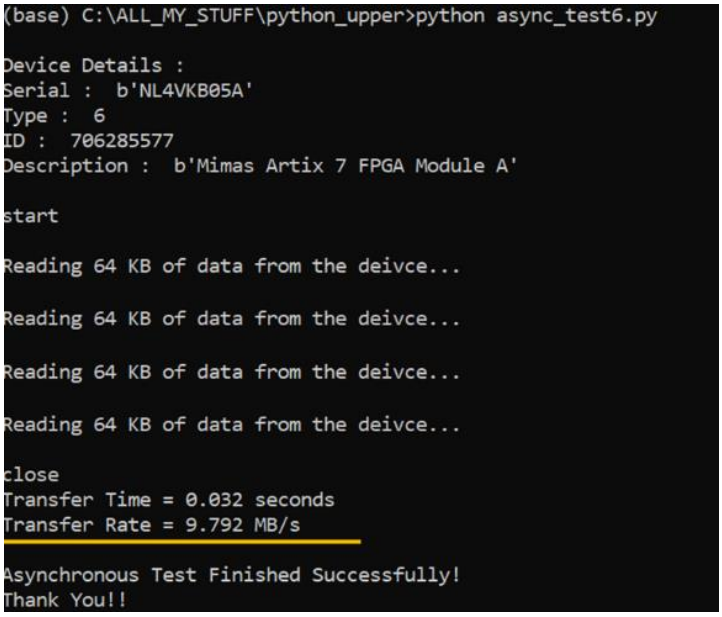

Figure 16. Data transportation speed

The data transfer speed is also an important property of this high frequency photoacoustic system. By setting the start point and the end point of the timer respectively at the of opening of the USB port and the closing of the USB port, we can calculate the data transfer speed from its elapsed time. In this case, the speed is about 9.8 MB/s (Figure 16).

Chapter 3. Imaging Reconstruction

In this chapter, the imaging reconstruction method is introduced, and the system testing results of standard phantoms are analyzed. Some important parameters of this system will be calculated in the following content.

3.1 Receiving Beamforming Algorithm

An acoustic wave generated from the illuminated target along a line at angle θ will reach different elements with slightly different time delays because of the distance variations. In this system, the ultrasound probe (Vevo 2100 MS 200) houses a linear array. To construct a 2D photoacoustic image, we apply a polar coordinate system to specify every point (R, θ) of this 2D image, as shown in Figure 17 [28]:

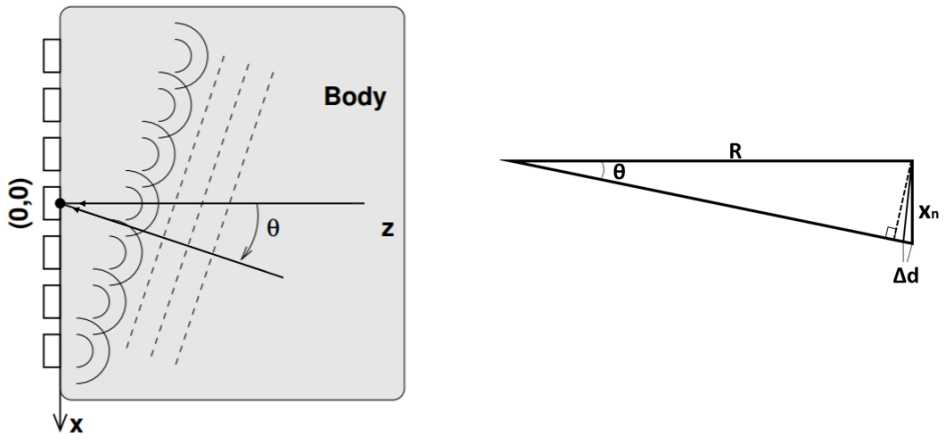

Figure 17 Time delay calculation [28]

The zero point is placed at the center of the elements in the linear array. x_n denotes the x position of the center of the n-th transducer element ($x_n = nd$, where d is the transducer pitch spacing). The space Δd can be calculated by

$$\Delta d = \frac{x_n}{sin\theta} - R \approx x_n sin\theta. \quad (8)$$

In practice, the width of the linear array is usually much smaller than the depth of focus, as shown in the right drawing of Figure 17, which implies that x_n << R. In this case, Δd can be approximated as $x_n sin\theta$, which leads to the time delay $\tau(x_n, R, \theta)$ equation:

$$\tau(x_n, R, \theta) = \frac{\frac{x_n}{sin\theta} - R}{v_s} \approx \frac{x_n sin\theta}{v_s}, (9)$$

where v_s is the speed of sound in the tissue or phantom.

The output image can be approximated by the following coherent summation:

$$O(t) = \sum_{i=0}^{N} S_n\big(t - \tau(x_n, R, \theta)\big), (10)$$

where $S_n(t)$ is the received signal on the n-th element, and $O(t)$ is the value of output image point.

3.2 Spatial Resolution

The spatial resolution of the 2D imaging system is determined by the axial resolution (AR) and the lateral resolution (LR). Axial resolution refers to the smallest distance between two objects that can be distinguished in the longitudinal direction. Likewise, lateral resolution refers to the same distance in the transverse direction. For acoustic imaging, the axial resolution should be the wavelength of the acoustic wave, λ, while the lateral resolution varies dynamically with the focal point depth. The axial and lateral resolutions are given by the following equations [29]:

$$AR = \lambda = \frac{v_s}{f} \qquad (11)$$

$$LR = 0.4 \times \lambda \times \frac{F}{L}, \qquad (12)$$

where f is the central frequency of the ultrasound probe (15 MHz), F is the focal depth (25.8mm in this case), and L is the active length of the aperture (32 channel elements, 4.96 mm in this case). The calculated AR is 102.67μm, and the LR is 213.62 μm.

To test this high frequency photoacoustic system's spatial resolution, we imaged a thin carbon fiber. This carbon fiber was illuminated by a 10 mJ/cm^2 light beam and imaged by the ultrasound probe, placed perpendicularly to this fiber to obtain a cross section image. The captured image is displayed in Figure 18.

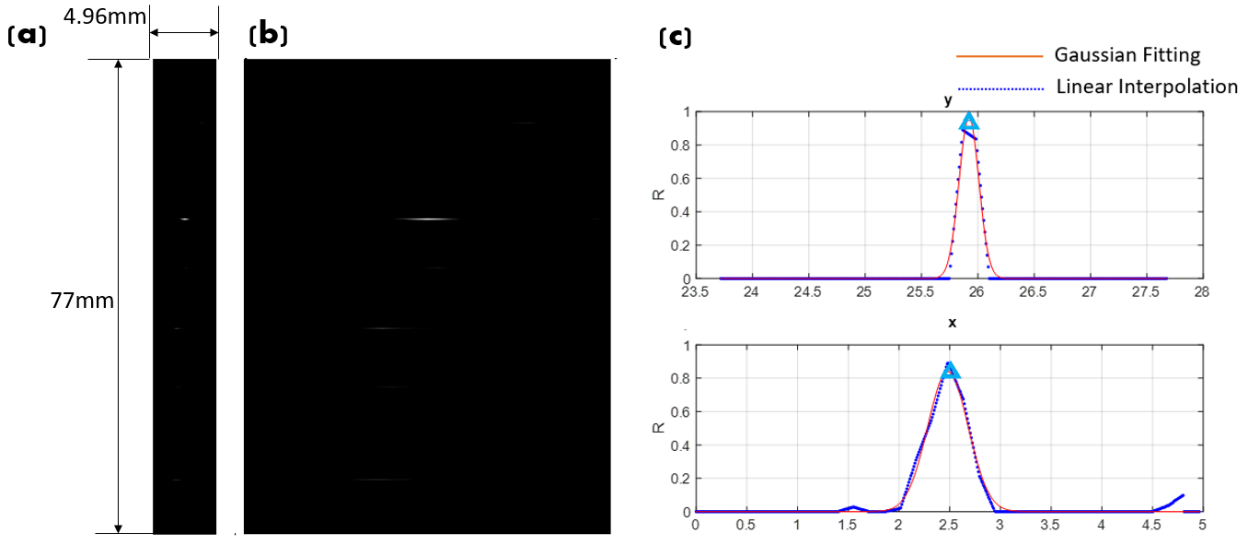

Figure 18. Spatial resolution test: (a)Original PA image showing the cross section of carbon fiber (b) Original Image expanded in x dimension to show the detail (c) resolution calculation in x and y direction

In Chapter 2.3, we calculated the image depth of this high frequency photoacoustic system, which is about 77 mm, and the image width is the same as the aperture size of this 32-channel array, which is 4.96 mm. From the position of this carbon fiber, we find the sharpest peaks in the longitudinal (y) and transverse directions (x) to calculate the resolution, shown in the left side of the figure and marked by the little pale blue triangles in the right side figure. The blue dashed curves are the outcomes after linear interpolation of the original image, and the red curves show Gaussian fits for the blue dashed curves. The spatial resolution is approximated by the full width

at half maximum (FWHM) of these two curves. In this experiment, the lateral resolution obtained is 467.2 μm and the axial resolution is 227.4 μm.

The spatial resolution we obtained here is different from the theoretical value we calculated before, but it is still a reasonable value because the cross section of a carbon fiber depends on the thickness of the carbon fiber. The thinner the fiber is, the better the resolution value we could obtain, as long as the diameter of this fiber is larger than the spatial resolution.

3.3 Phantom Image

A piece of black tape was the image phantoms used to test this system. The laser energy level was about 10 mJ/cm^2. Two images of this black tape are shown in Figure 19. In the left image, the light beam irradiated the edge of the black tape; in the right-side image, the light beam irradiation point moved a little bit toward the center area. The dynamic range here to form these images is -10dB.

Figure 19. Black tape images

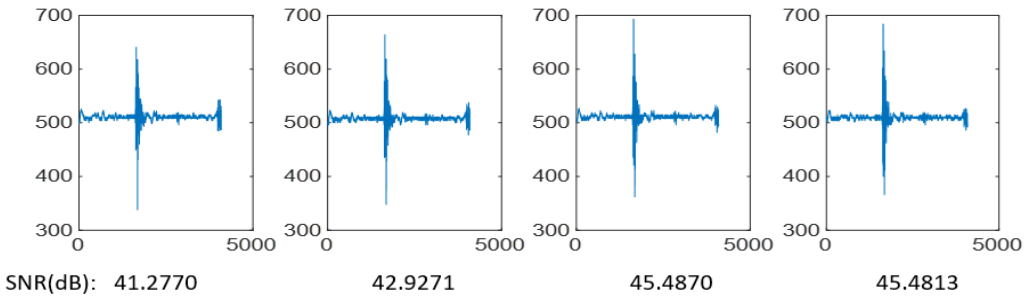

Figure 20. SNR of photoacoustic signals

To assess the image quality of this high frequency photoacoustic system, we randomly selected four channels' raw data to calculate the SNR, with the results seen in Figure 20. The peaks are clean and neat, which indicating the good quality SNR. The SNR values were calculated as the signal peak value divided by the standard deviation of the background noise. In this figure, the SNR varies from 40 to 46 dB, which is ideal for our colorectal cancer imaging requirement.

Chapter 4. Discussion and Future Work

To summarize, the 32-channel 80MHz high frequency photoacoustic image system presented here consists of a laser (Symphotics TII), an ultrasound probe (Vevo 2100 MS 200), adapter circuits, an ADC module (AFE5807), an FPGA development board (Mimas A7, based on Xilinx Artix 7), and a computer. It can form 2D images 4.96 mm wide and 77 mm deep. The lateral resolution of this system is 467.2 μm (theoretically 213.62 μm) at 25.8 mm deep and the axial resolution is 227.4 μm (theoretically 102.67 μm). With a laser energy of around 10 mJ/cm^{2,} the SNR is about 40 dB. The ADC module has 10-bit resolution, and the sampling frequency is 80 MHz, with a 15 Hz laser trigger frequency. The input peak-to-peak voltage ranges from 0.25V to 1V. The data transfer speed is about 9MB/s.

Although the design and testing of this 32-channel system are complete, the system could potentially be expanded up to 256 channels. All the elements of this ultrasound probe are connected to the mother board, so by adding multiples of this 32-channel system and synchronizing them with the same laser trigger, we can produce a prototype with more channels to enhance the lateral resolution and expand the width of the image window.

Such an expanded system would be useful in ex-vivo and in-vivo experiments involving human colorectal cancer. The 80 MHz sampling frequency guarantees a sufficient sampling rate to provide detailed information during experiments.

References

[1]. Siegel, Rebecca L., et al. "Colorectal cancer statistics, 2017." CA: a cancer journal for clinicians 67.3 (2017): 177-193.

[2]. American Cancer Society. Cancer Facts & Figures 2020. Atlanta: American Cancer Society; 2020.

[3]. Edwards BK, Ward E, Kohler BA, et al. Annual report to the nation on the status of cancer, 1975-2006, featuring colorectal cancer trends and impact of interventions (risk factors, screening, and treatment) to reduce future rates. Cancer. 2010;116: 544-573.

[4]. Garcia, M., et al. "Global Cancer Facts & Figures American Cancer Society." Atlanta, GA, USA (2007).

[5]. Howlader N, Noone AM, Krapcho M, et al (eds). SEER Cancer Statistics Review, 1975-2016, National Cancer Institute, Bethesda, MD

[6]. L V Wang and H Wu, Biomedical Optics: Principles and Imaging, John Wiley and Sons Inc., 2007.

[7]. Yao, Junjie, and Lihong V. Wang. "Sensitivity of photoacoustic microscopy." *Photoacoustics* 2.2 (2014): 87-101.

[8]. Lihong V. Wang, H.-i. W. (2007). Biomedical Optics: Principles and Imaging. John Wiley & Sons, Inc.

[9]. Xia J, Yao J, Wang LV. Photoacoustic tomography: principles and advances. Electromagn Waves (Camb). 2014;147:1–22.

[10]. Wang LV. Multiscale photoacoustic microscopy and computed tomography. Nat Photon. 2009 ;3:503–509.

[11]. Wang LV, Hu S. Photoacoustic Tomography: In Vivo Imaging from Organelles to Organs. Science. 2012 Mar 23;335:1458–1462. 2012.

[12]. Oladipupo S, Hu S, Kovalski J, Yao JJ, Santeford A, Sohn RE, Shohet R, Maslov K, Wang LHV, Arbeit JM. VEGF is essential for hypoxia-inducible factor-mediated neovascularization but dispensable for endothelial sprouting. Proceedings of the National Academy of Sciences of the United States of America. 2011 Aug 9;108:13264–13269.

[13]. Bitton R, Zemp R, Yen J, Wang LV, Shung KK. A 3-D high-frequency array based 16 channel photoacoustic microscopy system for in vivo micro-vascular imaging. IEEE Trans Med Imaging. 2009 Aug;28:1190–1197.

[14]. Staley J, Grogan P, Samadi AK, Cui H, Cohen MS, Yang X. Growth of melanoma brain tumors monitored by photoacoustic microscopy. Journal of Biomedical Optics. 2010 Jul-Aug;15:040510.

[15]. Hu S, Maslov K, Tsytsarev V, Wang LV. Functional transcranial brain imaging by optical-resolution photoacoustic microscopy. Journal of Biomedical Optics. 2009 Jul-Aug;14:040503.

[16]. Subach FV, Zhang LJ, Gadella TWJ, Gurskaya NG, Lukyanov KA, Verkhusha VV. Red Fluorescent Protein with Reversibly Photoswitchable Absorbance for Photochromic FRET. Chemistry & Biology. 2010 Jul 30;17:745–755.

[17]. Favazza CP, Cornelius LA, Wang LHV. In vivo functional photoacoustic microscopy of cutaneous microvasculature in human skin. Journal of Biomedical Optics. 2011 Feb;16:026004.

[18]. Yao J, Maslov KI, Puckett ER, Rowland KJ, Warner BW, Wang LV. Double-illumination photoacoustic microscopy. Optics Letters. 2012;37:659–661.

[19]. Lihong V Wang, Photoacoustic Imaging and Spectroscopy, CRC Press, New York, 4-5 (2009).

[20]. Yang, Guang, et al. "Optimized light delivery probe using ball lenses for co-registered photoacoustic and ultrasound endo-cavity subsurface imaging." Photoacoustics 13 (2019): 66-75.

[21]. Yang, Guang, et al. "Co-registered photoacoustic and ultrasound imaging of human colorectal cancer." Journal of biomedical optics 24.12 (2019): 121913.

[22]. Mohammadi-Nejad, Ali-Reza, et al. "Neonatal brain resting-state functional connectivity imaging modalities." Photoacoustics 10 (2018): 1-19.

[23]. Afe5807 Evaluation Module. Retrieved March 17, 2020, from http://www.ti.com/tool/AFE5807EVM?keyMatch=AFE5807EVM&tisearch=Search-EN-everything

[24]. Mimas A7 – Artix 7 Fpga Development Board. Retrieved March 17, 2020, from https://numato.com/product/mimas-a7-artix-7-fpga-development-board-with-ddr-sdram-and-gigabit-ethernet

[25]. Fully Integrated, 8-Channel Ultrasound Analog Front End with Passive CW Mixer, 1.05 nV/rtHz, 12-Bit, 80 MSPS, 117 mW/CH. SLOS703C –SEPTEMBER 2010–REVISED MAY 2013

[26]. FIFO Generator v13.2 LogiCORE IP Product Guide Vivado Design Suite PG057 October 4, 2017

[27]. Future Technology Devices International Ltd FT2232H Dual High Speed USB to Multipurpose UART/FIFO IC. Datasheet Version 2.5. Document No.: FT_000061 Clearance No.: FTDI#77

[28]. Ultrasound Array. Retrieved March 17, 2020, from https://web.eecs.umich.edu/~fessler/course/516/l/ca-array-1-8.pdf

[29]. Swarup Bhunia, S. M. (2015). Implantable Biomedical Microsystems: Design Principles and Applications (1st Edition). Elsevier.

Appendix

1. Pin map decoding result

	1	2	3	4	5	6	7	8	9	10	11	12
1\12	**115**	**179**	**127**	**255**	**123**	**251**	**108**	**172**	**128**	**192**	**124**	**188**
2\18	✕	**243**	✕	**191**	✕	**187**	✕	**236**	✕	**256**	✕	**252**
3\29	**51**	✕	**63**	**253**	**59**	**233**	**44**	**170**	**64**	**174**	**60**	**178**
4\34	✕	✕	**189**	✕	**169**	✕	**106**	✕	**238**	✕	**114**	✕
5\44	✕	✕	**125**	**61**	**105**	**41**	**234**	**42**	**110**	**46**	**242**	**50**
6\54	**119**	**55**	✕	✕	**231**	**167**	**112**	**48**	**244**	**180**	**184**	**120**
7\59	✕	**247**	✕	✕	✕	**39**	✕	**176**	✕	**52**	✕	**56**
8\69	**183**	**101**	✕	**97**	**103**	**173**	**240**	**166**	**116**	✕	**248**	**126**
9\74	**37**	✕	**33**	✕	**237**	✕	**38**	✕	✕	✕	**190**	✕
10\84	**229**	**165**	**225**	**161**	**45**	**109**	**230**	**102**	✕	✕	**254**	**62**
11\94	**35**	**99**	**47**	**111**	**235**	**171**	**36**	**164**	**40**	**104**	✕	✕
12\99	✕	**163**	✕	**175**	✕	**107**	✕	**100**	✕	**168**	✕	✕
13\110	**227**	**113**	**239**	**117**	**43**	**185**	**228**	**58**	**232**	**118**	✕	**226**
14\116	**177**	✕	**181**	✕	**249**	✕	**122**	✕	**182**	✕	**34**	✕
15\128	**241**	**49**	**245**	**53**	**121**	**57**	**186**	**250**	**54**	**246**	**98**	**162**
	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
1\140	**83**	**147**	**95**	**223**	**91**	**219**	**76**	**140**	**96**	**160**	**92**	**156**
2\146	✕	**211**	✕	**159**	✕	**155**	✕	**204**	✕	**224**	✕	**220**
3\157	**19**	✕	**31**	**221**	**27**	**201**	**12**	**138**	**32**	**142**	**28**	**146**
4\162	✕	✕	**157**	✕	**137**	✕	**74**	✕	**206**	✕	**82**	✕
5\172	✕	✕	**93**	**29**	**73**	**9**	**202**	**10**	**78**	**14**	**210**	**18**
6\182	**87**	**23**	✕	✕	**199**	**135**	**80**	**16**	**212**	**148**	**152**	**88**
7\187	✕	**215**	✕	✕	✕	**7**	✕	**144**	✕	**20**	✕	**24**
8\197	**151**	**69**	✕	**65**	**71**	**141**	**208**	**134**	**84**	✕	**216**	**94**
9\202	**5**	✕	**1**	✕	**205**	✕	**6**	✕	✕	✕	**158**	✕
10\212	**197**	**133**	**193**	**129**	**13**	**77**	**198**	**70**	✕	✕	**222**	**30**
11\222	**3**	**67**	**15**	**79**	**203**	**139**	**4**	**132**	**8**	**72**	✕	✕
12\227	✕	**131**	✕	**143**	✕	**75**	✕	**68**	✕	**136**	✕	✕
13\238	**195**	**81**	**207**	**85**	**11**	**153**	**196**	**26**	**200**	**86**	✕	**194**
14\246	**145**	✕	**149**	✕	**217**	✕	**90**	✕	**150**	✕	**2**	✕
15\256	**209**	**17**	**213**	**21**	**89**	**25**	**154**	**218**	**22**	**214**	**66**	**130**

2. FPGA Code

```
`timescale 1ns / 1ps
module mimas_ADC_top(
    input [39:0] P13,
    input [39:0] P12,
    input CLK,
    input RESET,

    input TXE_N,
    input RXF_N,
    inout [7:0] DATA,
```

```
    output WR_N,
    output RD_N,
    output SIWUA,
    output [7:0] LED
    );

wire wr_en;
wire rd_en;
wire r_clk;
wire [11:0] data_out_board1;
wire wr_en_board1;
wire rd_en_board1;
wire empty1;
wire [11:0] data_out_board2;
wire wr_en_board2;
wire rd_en_board2;
wire empty2;
wire [11:0] data_out_board3;
wire wr_en_board3;
wire rd_en_board3;
wire empty3;
wire [11:0] data_out_board4;
wire wr_en_board4;
wire rd_en_board4;
wire empty4;
wire status_board1;
wire status_board2;
wire status_board3;
wire status_board4;
assign
wr_en=wr_en_board1|wr_en_board2|wr_en
_board3|wr_en_board4;
assign
rd_en=rd_en_board1|rd_en_board2|rd_en_b
oard3|rd_en_board4;
assign LED[0]=empty1;
assign LED[2]=empty2;
assign LED[4]=empty3;
assign LED[6]=empty4;
trigger_control inst0(
    .clk(CLK),
    .fifo_wr_en(wr_en),
    .fifo_rd_en(rd_en),
    .trigger(RESET),
    .trig(trig)
    );

series_to_parallel_1 inst_board1(
    .FCLK_p(P13[2]),
    .FCLK_n(P13[3]),
    .channel_ch1_p(P13[22]),
    .channel_ch1_n(P13[23]),
    .channel_ch2_p(P13[4]),
    .channel_ch2_n(P13[5]),
    .channel_ch3_p(P13[6]),
    .channel_ch3_n(P13[7]),
    .channel_ch4_p(P13[8]),
    .channel_ch4_n(P13[9]),
    .channel_ch5_p(P13[16]),
    .channel_ch5_n(P13[17]),
    .channel_ch6_p(P13[18]),
    .channel_ch6_n(P13[19]),
    .channel_ch7_p(P13[20]),
    .channel_ch7_n(P13[21]),
    .channel_ch8_p(P13[0]),
    .channel_ch8_n(P13[1]),
    .sw(trig),
    .r_clk(r_clk),
    .status(status_board1),
    .wr_en(wr_en_board1),
    .rd_en(rd_en_board1),
    .empty(empty1),
    .p(LED[1]),
    .data_fifo(data_out_board1)
    );

series_to_parallel_2 inst_board2(
    .FCLK_p(P13[12]),
    .FCLK_n(P13[13]),
    .channel_ch1_p(P13[24]),
    .channel_ch1_n(P13[25]),
    .channel_ch2_p(P13[26]),
    .channel_ch2_n(P13[27]),
    .channel_ch3_p(P13[28]),
    .channel_ch3_n(P13[29]),
    .channel_ch4_p(P13[30]),
    .channel_ch4_n(P13[31]),
    .channel_ch5_p(P13[32]),
    .channel_ch5_n(P13[33]),
    .channel_ch6_p(P13[34]),
    .channel_ch6_n(P13[35]),
```

```
    .channel_ch7_p(P13[36]),
    .channel_ch7_n(P13[37]),
    .channel_ch8_p(P13[38]),
    .channel_ch8_n(P13[39]),
    .sw(trig),
    .r_clk(r_clk),
    .status(status_board2),
    .wr_en(wr_en_board2),
    .rd_en(rd_en_board2),
    .empty(empty2),
    .p(LED[3]),
    .data_fifo(data_out_board2)
    );

series_to_parallel_3 inst_board3(
    .FCLK_p(P12[12]),
    .FCLK_n(P12[13]),
    .channel_ch1_p(P12[24]),
    .channel_ch1_n(P12[25]),
    .channel_ch2_p(P12[26]),
    .channel_ch2_n(P12[27]),
    .channel_ch3_p(P12[28]),
    .channel_ch3_n(P12[29]),
    .channel_ch4_p(P12[30]),
    .channel_ch4_n(P12[31]),
    .channel_ch5_p(P12[34]),
    .channel_ch5_n(P12[35]),
    .channel_ch6_p(P12[32]),
    .channel_ch6_n(P12[33]),
    .channel_ch7_p(P12[36]),
    .channel_ch7_n(P12[37]),
    .channel_ch8_p(P12[22]),
    .channel_ch8_n(P12[23]),
    .sw(trig),
    .r_clk(r_clk),
    .status(status_board3),
    .wr_en(wr_en_board3),
    .rd_en(rd_en_board3),
    .empty(empty3),
    .p(LED[5]),
    .data_fifo(data_out_board3)
    );

series_to_parallel_3 inst_board4(
    .FCLK_p(P12[0]),
    .FCLK_n(P12[1]),
    .channel_ch1_p(P12[2]),
    .channel_ch1_n(P12[3]),
    .channel_ch2_p(P12[4]),
    .channel_ch2_n(P12[5]),
    .channel_ch3_p(P12[8]),
    .channel_ch3_n(P12[9]),
    .channel_ch4_p(P12[10]),
    .channel_ch4_n(P12[11]),
    .channel_ch5_p(P12[14]),
    .channel_ch5_n(P12[15]),
    .channel_ch6_p(P12[16]),
    .channel_ch6_n(P12[17]),
    .channel_ch7_p(P12[18]),
    .channel_ch7_n(P12[19]),
    .channel_ch8_p(P12[6]),
    .channel_ch8_n(P12[7]),
    .sw(trig),
    .r_clk(r_clk),
    .status(status_board4),
    .wr_en(wr_en_board4),
    .rd_en(rd_en_board4),
    .empty(empty4),
    .p(LED[7]),
    .data_fifo(data_out_board4)
    );

usb_reading inst_usb(
    .CLK(CLK),
    .TXE_N(TXE_N),
    .RXF_N(RXF_N),
    .data_in_board1(data_out_board1),
    .data_in_board2(data_out_board2),
    .data_in_board3(data_out_board3),
    .data_in_board4(data_out_board4),
    .board1_wr_en(wr_en_board1),
    .board2_wr_en(wr_en_board2),
    .board3_wr_en(wr_en_board3),
    .board4_wr_en(wr_en_board4),
    .empty1(empty1),
    .empty2(empty2),
    .empty3(empty3),
    .empty4(empty4),

    .DATA(DATA),
    .r_clk(r_clk),
    .WR_N(WR_N),
```

```
    .RD_N(RD_N),
    .status_board1(status_board1),
    .status_board2(status_board2),
    .status_board3(status_board3),
    .status_board4(status_board4),
    .SIWUA(SIWUA)
   );
Endmodule

`timescale 1ns / 1ps

module trigger_control(
   input clk,
   input fifo_wr_en,
   input fifo_rd_en,
   input trigger,
   output trig
   );

reg trig_reg;
assign trig = trig_reg;
always @(posedge clk)begin
      if(fifo_wr_en == 1'b0 && fifo_rd_en
== 1'b0)
        trig_reg <= trigger;
end
endmodule

`timescale 1ns / 1ps
module series_to_parallel_1(
   input FCLK_p,
   input FCLK_n,
   input channel_ch1_p,
   input channel_ch1_n,
   input channel_ch2_p,
   input channel_ch2_n,
   input channel_ch3_p,
   input channel_ch3_n,
   input channel_ch4_p,
   input channel_ch4_n,
   input channel_ch5_p,
   input channel_ch5_n,
   input channel_ch6_p,
   input channel_ch6_n,
   input channel_ch7_p,
   input channel_ch7_n,
   input channel_ch8_p,
   input channel_ch8_n,
   input sw,
   input r_clk,
   input status,
   output wr_en,
   output rd_en,
   output empty,
   output p,
   output [11:0] data_fifo
   );

wire dclk_pll;
reg rst;
reg [11:0] data_ch1_reg;
reg [11:0] data_ch2_reg;
reg [11:0] data_ch3_reg;
reg [11:0] data_ch4_reg;
reg [11:0] data_ch5_reg;
reg [11:0] data_ch6_reg;
reg [11:0] data_ch7_reg;
reg [11:0] data_ch8_reg;

wire FCLK;
IBUFDS #(
     .DIFF_TERM("FALSE"),      //
Differential Termination
     .IBUF_LOW_PWR("TRUE"),     // Low
power="TRUE", Highest
performance="FALSE"
     .IOSTANDARD("DEFAULT")     //
Specify the input I/O standard
   ) IBUFDS_inst1 (
     .O(FCLK),  // Buffer output
     .I(FCLK_p),  // Diff_p buffer input
(connect directly to top-level port)
     .IB(FCLK_n) // Diff_n buffer input
(connect directly to top-level port)
   );

clk_wiz_0 inst_clk
 (
  // Clock out ports
  .clk_out1(dclk_pll),
  // Status and control signals
  .reset(1'b0),
```

```
  .locked(locked0),
 // Clock in ports
  .clk_in1(FCLK)
 );


//wire [7:0] group_bit;
wire [79:0] data_out;
wire [7:0] channel_p;
wire [7:0] channel_n;
assign
channel_p={channel_ch8_p,channel_ch7_p,
channel_ch6_p,channel_ch5_p,channel_ch4
_p,channel_ch3_p,channel_ch2_p,channel_c
h1_p};
assign
channel_n={channel_ch8_n,channel_ch7_n,
channel_ch6_n,channel_ch5_n,channel_ch4
_n,channel_ch3_n,channel_ch2_n,channel_c
h1_n};
//assign
group_bit={bitslip,bitslip,bitslip,bitslip,bitsli
p,bitslip,bitslip,bitslip};
selectio_wiz_0 inst_selectio
 (
  // From the system into the device
  .data_in_from_pins_p(channel_p),
  .data_in_from_pins_n(channel_n),
  .data_in_to_device(data_out),
  .bitslip(8'b00000000),       // Bitslip module
is enabled in NETWORKING mode
                               // User should tie it to
'0' if not needed
  .clk_in(dclk_pll),        // Fast clock input
from PLL/MMCM
  .clk_div_in(~FCLK),    // Slow clock input
from PLL/MMCM
  . io_reset(rst));

wire [9:0] data_out_ch1;
wire [9:0] data_out_ch2;
wire [9:0] data_out_ch3;
wire [9:0] data_out_ch4;
wire [9:0] data_out_ch5;
wire [9:0] data_out_ch6;
wire [9:0] data_out_ch7;
wire [9:0] data_out_ch8;
always @(negedge FCLK)begin
   rst<=sw;
   data_ch1_reg[9:0]<=data_out_ch1[9:0];
   data_ch2_reg[9:0]<=data_out_ch2[9:0];
   data_ch3_reg[9:0]<=data_out_ch3[9:0];
   data_ch4_reg[9:0]<=data_out_ch4[9:0];
   data_ch5_reg[9:0]<=data_out_ch5[9:0];
   data_ch6_reg[9:0]<=data_out_ch6[9:0];
   data_ch7_reg[9:0]<=data_out_ch7[9:0];
   data_ch8_reg[9:0]<=data_out_ch8[9:0];
end
//MSB
assign
data_out_ch1={data_out[7],data_out[15],dat
a_out[23],data_out[31],data_out[39],data_o
ut[47],data_out[55],data_out[63],data_out[7
1],data_out[79]};
assign
data_out_ch2={data_out[6],data_out[14],dat
a_out[22],data_out[30],data_out[38],data_o
ut[46],data_out[54],data_out[62],data_out[7
0],data_out[78]};
assign
data_out_ch3={data_out[5],data_out[13],dat
a_out[21],data_out[29],data_out[37],data_o
ut[45],data_out[53],data_out[61],data_out[6
9],data_out[77]};
assign
data_out_ch4={data_out[4],data_out[12],dat
a_out[20],data_out[28],data_out[36],data_o
ut[44],data_out[52],data_out[60],data_out[6
8],data_out[76]};
assign
data_out_ch5={data_out[3],data_out[11],dat
a_out[19],data_out[27],data_out[35],data_o
ut[43],data_out[51],data_out[59],data_out[6
7],data_out[75]};
assign
data_out_ch6={data_out[2],data_out[10],dat
a_out[18],data_out[26],data_out[34],data_o
ut[42],data_out[50],data_out[58],data_out[6
6],data_out[74]};
assign
data_out_ch7={data_out[1],data_out[9],data
_out[17],data_out[25],data_out[33],data_out
```

```
[41],data_out[49],data_out[57],data_out[65],
data_out[73]};
assign
data_out_ch8={data_out[0],data_out[8],data
_out[16],data_out[24],data_out[32],data_out
[40],data_out[48],data_out[56],data_out[64],
data_out[72]};

reg bitslip;
reg [3:0] cnt;
always @(posedge FCLK)begin
      if(rst)
        cnt<=0;
      else begin
         if(cnt<3)begin
           cnt<=cnt+1;
           bitslip<=1;
         end
         else
           bitslip<=0;
     end
end

wire [95:0] data_ch;
assign data_ch[11:0]=data_ch1_reg;
assign data_ch[23:12]=data_ch2_reg;
assign data_ch[35:24]=data_ch3_reg;
assign data_ch[47:36]=data_ch4_reg;
assign data_ch[59:48]=data_ch5_reg;
assign data_ch[71:60]=data_ch6_reg;
assign data_ch[83:72]=data_ch7_reg;
assign data_ch[95:84]=data_ch8_reg;

data_saving inst_fifo(
   .w_clk(FCLK),
   .r_clk(r_clk),
   .data_ch(data_ch),
   .rst(sw),
   .status(status),
   .wr_en(wr_en),
   .rd_en(rd_en),
   .empty(empty),
   .p(p),
   .data_out(data_fifo)
   );
endmodule
```

```
`timescale 1ns / 1ps
module series_to_parallel_2(
   input FCLK_p,
   input FCLK_n,
   input channel_ch1_p,
   input channel_ch1_n,
   input channel_ch2_p,
   input channel_ch2_n,
   input channel_ch3_p,
   input channel_ch3_n,
   input channel_ch4_p,
   input channel_ch4_n,
   input channel_ch5_p,
   input channel_ch5_n,
   input channel_ch6_p,
   input channel_ch6_n,
   input channel_ch7_p,
   input channel_ch7_n,
   input channel_ch8_p,
   input channel_ch8_n,
   input sw,
   input r_clk,
   input status,
   output wr_en,
   output rd_en,
   output empty,
   output p,
   output [11:0] data_fifo
   );

wire dclk_pll;
reg rst;
reg [11:0] data_ch1_reg;
reg [11:0] data_ch2_reg;
reg [11:0] data_ch3_reg;
reg [11:0] data_ch4_reg;
reg [11:0] data_ch5_reg;
reg [11:0] data_ch6_reg;
reg [11:0] data_ch7_reg;
reg [11:0] data_ch8_reg;

wire FCLK;
IBUFDS #(
     .DIFF_TERM("FALSE"),      //
Differential Termination
```

```
    .IBUF_LOW_PWR("TRUE"),     // Low
power="TRUE", Highest
performance="FALSE"
    .IOSTANDARD("DEFAULT")     //
Specify the input I/O standard
  ) IBUFDS_inst1 (
    .O(FCLK),  // Buffer output
    .I(FCLK_p),  // Diff_p buffer input
(connect directly to top-level port)
    .IB(FCLK_n) // Diff_n buffer input
(connect directly to top-level port)
  );

clk_wiz_1 inst_clk
 (
  // Clock out ports
  .clk_out1(dclk_pll),
  // Status and control signals
  .reset(1'b0),
  .locked(locked0),
 // Clock in ports
  .clk_in1(FCLK)
 );



//wire [7:0] group_bit;
wire [79:0] data_out;
wire [7:0] channel_p;
wire [7:0] channel_n;
assign
channel_p={channel_ch8_p,channel_ch7_p,
channel_ch6_p,channel_ch5_p,channel_ch4
_p,channel_ch3_p,channel_ch2_p,channel_c
h1_p};
assign
channel_n={channel_ch8_n,channel_ch7_n,
channel_ch6_n,channel_ch5_n,channel_ch4
_n,channel_ch3_n,channel_ch2_n,channel_c
h1_n};
//assign
group_bit={bitslip,bitslip,bitslip,bitslip,bitsli
p,bitslip,bitslip,bitslip};
selectio_wiz_0 inst_selectio
 (
  // From the system into the device
  .data_in_from_pins_p(channel_p),
  .data_in_from_pins_n(channel_n),
  .data_in_to_device(data_out),
  .bitslip(8'b00000000),       // Bitslip module
is enabled in NETWORKING mode
                               // User should tie it to
'0' if not needed
  .clk_in(dclk_pll),        // Fast clock input
from PLL/MMCM
  .clk_div_in(~FCLK),    // Slow clock input
from PLL/MMCM
  . io_reset(rst));

wire [9:0] data_out_ch1;
wire [9:0] data_out_ch2;
wire [9:0] data_out_ch3;
wire [9:0] data_out_ch4;
wire [9:0] data_out_ch5;
wire [9:0] data_out_ch6;
wire [9:0] data_out_ch7;
wire [9:0] data_out_ch8;
always @(negedge FCLK)begin
    rst<=sw;
    data_ch1_reg[9:0]<=data_out_ch1[9:0];
    data_ch2_reg[9:0]<=data_out_ch2[9:0];
    data_ch3_reg[9:0]<=data_out_ch3[9:0];
    data_ch4_reg[9:0]<=data_out_ch4[9:0];
    data_ch5_reg[9:0]<=data_out_ch5[9:0];
    data_ch6_reg[9:0]<=data_out_ch6[9:0];
    data_ch7_reg[9:0]<=data_out_ch7[9:0];
    data_ch8_reg[9:0]<=data_out_ch8[9:0];
end
//MSB
assign
data_out_ch1={data_out[7],data_out[15],dat
a_out[23],data_out[31],data_out[39],data_o
ut[47],data_out[55],data_out[63],data_out[7
1],data_out[79]};
assign
data_out_ch2={data_out[6],data_out[14],dat
a_out[22],data_out[30],data_out[38],data_o
ut[46],data_out[54],data_out[62],data_out[7
0],data_out[78]};
assign
data_out_ch3={data_out[5],data_out[13],dat
a_out[21],data_out[29],data_out[37],data_o
```

```
ut[45],data_out[53],data_out[61],data_out[6
9],data_out[77]};
assign
data_out_ch4={data_out[4],data_out[12],dat
a_out[20],data_out[28],data_out[36],data_o
ut[44],data_out[52],data_out[60],data_out[6
8],data_out[76]};
assign
data_out_ch5={data_out[3],data_out[11],dat
a_out[19],data_out[27],data_out[35],data_o
ut[43],data_out[51],data_out[59],data_out[6
7],data_out[75]};
assign
data_out_ch6={data_out[2],data_out[10],dat
a_out[18],data_out[26],data_out[34],data_o
ut[42],data_out[50],data_out[58],data_out[6
6],data_out[74]};
assign
data_out_ch7={data_out[1],data_out[9],data
_out[17],data_out[25],data_out[33],data_out
[41],data_out[49],data_out[57],data_out[65],
data_out[73]};
assign
data_out_ch8={data_out[0],data_out[8],data
_out[16],data_out[24],data_out[32],data_out
[40],data_out[48],data_out[56],data_out[64],
data_out[72]};

reg bitslip;
reg [3:0] cnt;
always @(posedge FCLK)begin
      if(rst)
        cnt<=0;
      else begin
          if(cnt<3)begin
            cnt<=cnt+1;
            bitslip<=1;
          end
          else
            bitslip<=0;
     end
end

wire [95:0] data_ch;
assign data_ch[11:0]=data_ch1_reg;
assign data_ch[23:12]=data_ch2_reg;
assign data_ch[35:24]=data_ch3_reg;
assign data_ch[47:36]=data_ch4_reg;
assign data_ch[59:48]=data_ch5_reg;
assign data_ch[71:60]=data_ch6_reg;
assign data_ch[83:72]=data_ch7_reg;
assign data_ch[95:84]=data_ch8_reg;

data_saving inst_fifo(
   .w_clk(FCLK),
   .r_clk(r_clk),
   .data_ch(data_ch),
   .rst(sw),
   .status(status),
   .wr_en(wr_en),
   .rd_en(rd_en),
   .empty(empty),
   .p(p),
   .data_out(data_fifo)
   );
endmodule


`timescale 1ns / 1ps
module series_to_parallel_3(
   input FCLK_p,
   input FCLK_n,
   input channel_ch1_p,
   input channel_ch1_n,
   input channel_ch2_p,
   input channel_ch2_n,
   input channel_ch3_p,
   input channel_ch3_n,
   input channel_ch4_p,
   input channel_ch4_n,
   input channel_ch5_p,
   input channel_ch5_n,
   input channel_ch6_p,
   input channel_ch6_n,
   input channel_ch7_p,
   input channel_ch7_n,
   input channel_ch8_p,
   input channel_ch8_n,
   input sw,
   input r_clk,
   input status,
   output wr_en,
```

```
    output rd_en,
    output empty,
    output p,
    output [11:0] data_fifo
    );

wire dclk_pll;
reg rst;
reg [11:0] data_ch1_reg;
reg [11:0] data_ch2_reg;
reg [11:0] data_ch3_reg;
reg [11:0] data_ch4_reg;
reg [11:0] data_ch5_reg;
reg [11:0] data_ch6_reg;
reg [11:0] data_ch7_reg;
reg [11:0] data_ch8_reg;

wire FCLK;
IBUFDS #(
     .DIFF_TERM("FALSE"),       //
Differential Termination
     .IBUF_LOW_PWR("TRUE"),     // Low
power="TRUE", Highest
performance="FALSE"
     .IOSTANDARD("DEFAULT")     //
Specify the input I/O standard
   ) IBUFDS_inst1 (
     .O(FCLK),  // Buffer output
     .I(FCLK_p),  // Diff_p buffer input
(connect directly to top-level port)
     .IB(FCLK_n) // Diff_n buffer input
(connect directly to top-level port)
   );

clk_wiz_2 inst_clk
 (
  // Clock out ports
  .clk_out1(dclk_pll),
  // Status and control signals
  .reset(1'b0),
  .locked(locked0),
 // Clock in ports
  .clk_in1(FCLK)
 );

//wire [7:0] group_bit;
wire [79:0] data_out;
wire [7:0] channel_p;
wire [7:0] channel_n;
assign
channel_p={channel_ch8_p,channel_ch7_p,
channel_ch6_p,channel_ch5_p,channel_ch4
_p,channel_ch3_p,channel_ch2_p,channel_c
h1_p};
assign
channel_n={channel_ch8_n,channel_ch7_n,
channel_ch6_n,channel_ch5_n,channel_ch4
_n,channel_ch3_n,channel_ch2_n,channel_c
h1_n};
//assign
group_bit={bitslip,bitslip,bitslip,bitslip,bitsli
p,bitslip,bitslip,bitslip};
selectio_wiz_0 inst_selectio
 (
  // From the system into the device
  .data_in_from_pins_p(channel_p),
  .data_in_from_pins_n(channel_n),
  .data_in_to_device(data_out),
  .bitslip(8'b00000000),       // Bitslip module
is enabled in NETWORKING mode
                               // User should tie it to
'0' if not needed
  .clk_in(dclk_pll),        // Fast clock input
from PLL/MMCM
  .clk_div_in(~FCLK),    // Slow clock input
from PLL/MMCM
  . io_reset(rst));

wire [9:0] data_out_ch1;
wire [9:0] data_out_ch2;
wire [9:0] data_out_ch3;
wire [9:0] data_out_ch4;
wire [9:0] data_out_ch5;
wire [9:0] data_out_ch6;
wire [9:0] data_out_ch7;
wire [9:0] data_out_ch8;
always @(negedge FCLK)begin
    rst<=sw;
    data_ch1_reg[9:0]<=data_out_ch1[9:0];
    data_ch2_reg[9:0]<=data_out_ch2[9:0];
    data_ch3_reg[9:0]<=data_out_ch3[9:0];
```

```
    data_ch4_reg[9:0]<=data_out_ch4[9:0];
    data_ch5_reg[9:0]<=data_out_ch5[9:0];
    data_ch6_reg[9:0]<=data_out_ch6[9:0];
    data_ch7_reg[9:0]<=data_out_ch7[9:0];
    data_ch8_reg[9:0]<=data_out_ch8[9:0];
end
//MSB
assign
data_out_ch1={data_out[7],data_out[15],dat
a_out[23],data_out[31],data_out[39],data_o
ut[47],data_out[55],data_out[63],data_out[7
1],data_out[79]};
assign
data_out_ch2={data_out[6],data_out[14],dat
a_out[22],data_out[30],data_out[38],data_o
ut[46],data_out[54],data_out[62],data_out[7
0],data_out[78]};
assign
data_out_ch3={data_out[5],data_out[13],dat
a_out[21],data_out[29],data_out[37],data_o
ut[45],data_out[53],data_out[61],data_out[6
9],data_out[77]};
assign
data_out_ch4={data_out[4],data_out[12],dat
a_out[20],data_out[28],data_out[36],data_o
ut[44],data_out[52],data_out[60],data_out[6
8],data_out[76]};
assign
data_out_ch5={data_out[3],data_out[11],dat
a_out[19],data_out[27],data_out[35],data_o
ut[43],data_out[51],data_out[59],data_out[6
7],data_out[75]};
assign
data_out_ch6={data_out[2],data_out[10],dat
a_out[18],data_out[26],data_out[34],data_o
ut[42],data_out[50],data_out[58],data_out[6
6],data_out[74]};
assign
data_out_ch7={data_out[1],data_out[9],data
_out[17],data_out[25],data_out[33],data_out
[41],data_out[49],data_out[57],data_out[65],
data_out[73]};
assign
data_out_ch8={data_out[0],data_out[8],data
_out[16],data_out[24],data_out[32],data_out
[40],data_out[48],data_out[56],data_out[64],
data_out[72]};

reg bitslip;
reg [3:0] cnt;
always @(posedge FCLK)begin
      if(rst)
        cnt<=0;
      else begin
         if(cnt<3)begin
           cnt<=cnt+1;
           bitslip<=1;
         end
         else
           bitslip<=0;
     end
end

wire [95:0] data_ch;
assign data_ch[11:0]=data_ch1_reg;
assign data_ch[23:12]=data_ch2_reg;
assign data_ch[35:24]=data_ch3_reg;
assign data_ch[47:36]=data_ch4_reg;
assign data_ch[59:48]=data_ch5_reg;
assign data_ch[71:60]=data_ch6_reg;
assign data_ch[83:72]=data_ch7_reg;
assign data_ch[95:84]=data_ch8_reg;

data_saving inst_fifo(
   .w_clk(FCLK),
   .r_clk(r_clk),
   .data_ch(data_ch),
   .rst(sw),
   .status(status),
   .wr_en(wr_en),
   .rd_en(rd_en),
   .empty(empty),
   .p(p),
   .data_out(data_fifo)
   );
endmodule

`timescale 1ns / 1ps


module data_saving(
```

```
    input w_clk,
    input r_clk,
    input [95:0] data_ch,
    input rst,
    input status,
    output wr_en,
    output rd_en,
    output empty,
    output p,
    output [11:0] data_out
    );
parameter DEPTH_W = 4096;
parameter DEPTH_R = 32768;
reg wr_reg = 0;
reg rd_reg = 0;
reg [15:0] rcnt = 0;
reg [15:0] wcnt = 0;
reg package = 0;

assign p=package;
wire wr_rst_busy;
wire rd_rst_busy;
wire full;
//wire empty;
assign wr_en = wr_reg;
assign rd_en = rd_reg;

always @(posedge w_clk)begin
      if(rst)
        wcnt <= 0;
      else begin
        if(package && full == 1'b0 &&
empty == 1'b1 && wr_rst_busy == 1'b0 &&
rd_rst_busy == 1'b0)
          wr_reg <= 1'b1;
        if(wr_en == 1'b1)
          wcnt<=wcnt+1;
        if(wcnt >= DEPTH_W - 2 && full ==
1'b1)
          wr_reg <= 1'b0;
     end
end
always @(posedge r_clk)begin
      if(rst)begin
        package <= 1'b1;
        rcnt <=0;
      end
      else begin
        if(package && status && wr_reg
== 1'b0 && empty == 1'b0 && wr_rst_busy
== 1'b0 && rd_rst_busy == 1'b0)begin
            rd_reg <= 1'b1;
            package <= 1'b0;
          end
          if(rd_en == 1'b1)
            rcnt<=rcnt+1;
          if(rcnt >= DEPTH_R - 2 && empty
== 1'b1)
            rd_reg <= 1'b0;
       end
end

fifo_generator_0 inst1(
   .rst(rst),
   .wr_clk(w_clk),
   .rd_clk(r_clk),
   .din(data_ch),
   .wr_en(wr_en),
   .rd_en(rd_en),
   .dout(data_out),
   .full(full),
   .empty(empty),
   .wr_rst_busy(wr_rst_busy),
   .rd_rst_busy(rd_rst_busy)
  );
endmodule

`timescale 1ns / 1ps


module usb_reading(
    input CLK,
    input TXE_N,
    input RXF_N,
    input [11:0] data_in_board1,
    input [11:0] data_in_board2,
    input [11:0] data_in_board3,
    input [11:0] data_in_board4,
    input board1_wr_en,
    input board2_wr_en,
    input board3_wr_en,
    input board4_wr_en,
```

```
    input empty1,
    input empty2,
    input empty3,
    input empty4,

    inout [7:0] DATA,
    output r_clk,
    output WR_N,
    output RD_N,
    output status_board1,
    output status_board2,
    output status_board3,
    output status_board4,
    output SIWUA
   );

reg wr_reg= 1'b1;
reg rd_reg= 1'b1;
reg tx_reg;
reg [7:0] command=8'b00000000;
reg [7:0] data_reg;
reg status;
assign WR_N = wr_reg;
assign DATA = (status) ? data_reg:
8'bzzzzzzzz;
assign SIWUA = 1'b0;
assign RD_N = rd_reg;

reg delay;
reg delay2;
reg delay3;
reg rd_reg_reg;
always @(posedge CLK)begin
      if(rd_reg == 1'b1)
        rd_reg <=RXF_N;
      if(rd_reg == 1'b0)begin
        delay <= ~rd_reg;
        delay2 <= delay;
        rd_reg <= delay2;
      end
end
always @(posedge CLK)begin
      delay3 <=rd_reg;
      rd_reg_reg <=delay3;
end
always @(negedge rd_reg_reg)begin
      command <= DATA;
      if(command == 8'h6F &&
board1_wr_en == 1'b0 && board2_wr_en
== 1'b0 && board3_wr_en == 1'b0 &&
board4_wr_en == 1'b0 )
        status <= 1'b1;
      if(status == 1'b1)
        status <= 1'b0;
end
assign status_board1 = status & (~empty1);
assign status_board2 = status &
(~status_board1) & (~empty2);
assign status_board3 = status &
(~status_board1) & (~status_board2) &
(~empty3);
assign status_board4 = status &
(~status_board1) & (~status_board2) &
(~status_board3) & (~empty4);

//status need to be pulled back to 0 after data
transportation
always @(posedge CLK)begin
      if(status == 1'b1)begin
        tx_reg <=TXE_N;
        wr_reg <=tx_reg;
      end
      else begin
        wr_reg= 1'b1;
        tx_reg= 1'b1;
      end
end

reg r_clk_reg=1'b0;
reg h_byte=1'b0;
reg [11:0] test= 12'b000000000000;
wire [3:0] cnt;
assign r_clk =(status) ? r_clk_reg : CLK;
assign
cnt={status_board4,status_board3,status_bo
ard2,status_board1};
always @(posedge tx_reg)begin
      r_clk_reg <= ~r_clk_reg;
      if(h_byte==0)begin
        if(cnt == 4'b0001)
```

```
            data_reg[7:0] <=
data_in_board1[7:0];
          if(cnt == 4'b0010)
            data_reg[7:0] <=
data_in_board2[7:0];
          if(cnt == 4'b0100)
            data_reg[7:0] <=
data_in_board3[7:0];
          if(cnt == 4'b1000)
            data_reg[7:0] <=
data_in_board4[7:0];
        end
        else begin
          data_reg[7:4] <= 4'b0000;
          if(cnt == 4'b0001)
            data_reg[3:0] <=
data_in_board1[11:8];
          if(cnt == 4'b0010)
            data_reg[3:0] <=
data_in_board2[11:8];
          if(cnt == 4'b0100)
            data_reg[3:0] <=
data_in_board3[11:8];
          if(cnt == 4'b1000)
            data_reg[3:0] <=
data_in_board4[11:8];
        end
end

wire r_clk2;
assign r_clk2 =(status) ? tx_reg : CLK;
always @(posedge r_clk2)begin
        if(status==0)
          h_byte <=0;
        else
          h_byte <= ~h_byte;
end

endmodule
```

3. Python Code

```
from __future__ import print_function

import time
import random
import binascii
import ftd2xx

dev = ftd2xx.open(1)
dev.setTimeouts(5000, 5000)
dev.purge(ftd2xx.defines.PURGE_TX|ftd2xx.defines.PURGE_RX)
print("\nDevice Details :")
print("Serial : " , dev.getDeviceInfo()['serial'])
print("Type : " , dev.getDeviceInfo()['type'])
print("ID : " , dev.getDeviceInfo()['id'])
print("Description : " , dev.getDeviceInfo()['description'])

BLOCK_LEN = 4096*2*8

tx_data = 'o'
written = dev.write(tx_data)
print("\nstart")
ts=time.time()
rx_data0 = dev.read(BLOCK_LEN)
```

```
print("\nReading %d KB of data from the deivce..." % (BLOCK_LEN / 1024))
rx_data1 = dev.read(BLOCK_LEN)
print("\nReading %d KB of data from the deivce..." % (BLOCK_LEN / 1024))
rx_data2 = dev.read(BLOCK_LEN)
print("\nReading %d KB of data from the deivce..." % (BLOCK_LEN / 1024))
rx_data3 = dev.read(BLOCK_LEN)
print("\nReading %d KB of data from the deivce..." % (BLOCK_LEN / 1024))
te = time.time()
p = te - ts
tx_data = 'c'
written = dev.write(tx_data)
print("\nclose" )
dev.close()

rx2data0=rx_data0.hex()
data2txt0="
for i in range(0,BLOCK_LEN):
   data2txt0=data2txt0+rx2data0[2*i]+rx2data0[2*i+1]+' '
x=len(data2txt0)
data_cut0=data2txt0[12:(x-36)]

rx2data1=rx_data1.hex()
data2txt1="
for i in range(0,BLOCK_LEN):
   data2txt1=data2txt1+rx2data1[2*i]+rx2data1[2*i+1]+' '
x=len(data2txt1)
data_cut1=data2txt1[12:(x-36)]

rx2data2=rx_data2.hex()
data2txt2="
for i in range(0,BLOCK_LEN):
   data2txt2=data2txt2+rx2data2[2*i]+rx2data2[2*i+1]+' '
x=len(data2txt2)
data_cut2=data2txt2[12:(x-36)]

rx2data3=rx_data3.hex()
data2txt3="
for i in range(0,BLOCK_LEN):
   data2txt3=data2txt3+rx2data3[2*i]+rx2data3[2*i+1]+' '
x=len(data2txt3)
data_cut3=data2txt3[12:(x-36)]

file_handle=open('rx_board1.txt',mode='w')
file_handle.write(data_cut0)
file_handle.close()
file_handle=open('rx_board2.txt',mode='w')
```

```
file_handle.write(data_cut1)
file_handle.close()
file_handle=open('rx_board3.txt',mode='w')
file_handle.write(data_cut2)
file_handle.close()
file_handle=open('rx_board4.txt',mode='w')
file_handle.write(data_cut3)
file_handle.close()



print("Transfer Time = %.3f seconds" % p)
speed = (2*BLOCK_LEN / 1024./1024 / p)
print("Transfer Rate = %.3f MB/s" % speed)

print("\nAsynchronous Test Finished Successfully!\nThank You!!")
```

4. MATLAB code to read data

```
clc;clear;
[data_ch8,data_ch7,data_ch6,data_ch5,data_ch4,data_ch3,data_ch2,data_ch1]=getdata3('rx_board1.txt');
[data_ch15,data_ch14,data_ch13,data_ch12,data_ch11,data_ch10,data_ch9,data_ch16]=getdata3('rx_board2.txt');
[data_ch22,data_ch21,data_ch20,data_ch19,data_ch18,data_ch17,data_ch24,data_ch23]=getdata3('rx_board3.txt');
[data_ch29,data_ch28,data_ch27,data_ch26,data_ch25,data_ch32,data_ch31,data_ch30]=getdata3('rx_board4.txt');

figure()
subplot(241);plot(data_ch1)
subplot(242);plot(data_ch2)
subplot(243);plot(data_ch3)
subplot(244);plot(data_ch4)
subplot(245);plot(data_ch5)
subplot(246);plot(data_ch6)
subplot(247);plot(data_ch7)
subplot(248);plot(data_ch8)
figure()
subplot(241);plot(data_ch9)
subplot(242);plot(data_ch10)
subplot(243);plot(data_ch11)
subplot(244);plot(data_ch12)
subplot(245);plot(data_ch13)
subplot(246);plot(data_ch14)
subplot(247);plot(data_ch15)
subplot(248);plot(data_ch16)
figure()
subplot(241);plot(data_ch17)
subplot(242);plot(data_ch18)
subplot(243);plot(data_ch19)
subplot(244);plot(data_ch20)
subplot(245);plot(data_ch21)
subplot(246);plot(data_ch22)
subplot(247);plot(data_ch23)
subplot(248);plot(data_ch24)
figure()
subplot(141);plot(data_ch25)
subplot(242);plot(data_ch26)
subplot(243);plot(data_ch27)
subplot(244);plot(data_ch28)
subplot(245);plot(data_ch29)
subplot(246);plot(data_ch30)
subplot(247);plot(data_ch31)
subplot(248);plot(data_ch32)

data_reorder(1,:,1)=data_ch1-511;
data_reorder(2,:,1)=data_ch2-511;
data_reorder(3,:,1)=data_ch3-511;
data_reorder(4,:,1)=data_ch4-511;
data_reorder(5,:,1)=data_ch5-511;
```

```
data_reorder(6,:,1)=data_ch5-511;
data_reorder(7,:,1)=data_ch7-511;
data_reorder(8,:,1)=data_ch7-511;
data_reorder(9,:,1)=data_ch9-511;
data_reorder(10,:,1)=data_ch10-511;
data_reorder(11,:,1)=data_ch11-511;
data_reorder(12,:,1)=data_ch12-511;
data_reorder(13,:,1)=data_ch13-511;
data_reorder(14,:,1)=data_ch14-511;
data_reorder(15,:,1)=data_ch15-511;
data_reorder(16,:,1)=data_ch16-511;

data_reorder(1,:,2)=data_ch18-511;
data_reorder(2,:,2)=data_ch18-511;
data_reorder(3,:,2)=data_ch19-511;
data_reorder(4,:,2)=data_ch20-511;
data_reorder(5,:,2)=data_ch21-511;
data_reorder(6,:,2)=data_ch22-511;
data_reorder(7,:,2)=data_ch23-511;
data_reorder(8,:,2)=data_ch24-511;
data_reorder(9,:,2)=data_ch25-511;
data_reorder(10,:,2)=data_ch26-511;
data_reorder(11,:,2)=data_ch27-511;
data_reorder(12,:,2)=data_ch28-511;
data_reorder(13,:,2)=data_ch28-511;
data_reorder(14,:,2)=data_ch30-511;
data_reorder(15,:,2)=data_ch31-511;
data_reorder(16,:,2)=data_ch31-511;
j=1;
%j=j+1;

filename=['data',num2str(j),'.mat'];
save(filename,'data_reorder')


function
[data_ch1,data_ch2,data_ch3,data_ch4,data_
ch5,data_ch6,data_ch7,data_ch8]=getdata2(f
ilename)
%80M modified data
a=textread(filename,'%s')';
alpha=hex2dec(a)';
l=length(alpha);
data=zeros(1,l/2);
for i=1:(l/2)
   data(i)=alpha((i-
1)*2+1)*64+alpha(2*i)*256;
end
data=data/64;

for i=3:(l/16-4)
   data_ch8(i-2)=data(i*8-7);
   data_ch7(i-2)=data(i*8-6);
   data_ch6(i-2)=data(i*8-5);
   data_ch5(i-2)=data(i*8-4);
   data_ch4(i-2)=data(i*8-3);
   data_ch3(i-2)=data(i*8-2);
   data_ch2(i-2)=data(i*8-1);
   data_ch1(i-2)=data(i*8);
end

modify_L=280;
modify_H=700;
for i=2:length(data_ch1)

if(data_ch1(i)<modify_L||data_ch1(i)>modif
y_H)
      data_ch1(i)=data_ch1(i-1);
   end
end
for i=2:length(data_ch2)

if(data_ch2(i)<modify_L||data_ch2(i)>modif
y_H)
      data_ch2(i)=data_ch2(i-1);
   end
end
for i=2:length(data_ch3)

if(data_ch3(i)<modify_L||data_ch3(i)>modif
y_H)
      data_ch3(i)=data_ch3(i-1);
   end
end
for i=2:length(data_ch4)

if(data_ch4(i)<modify_L||data_ch4(i)>modif
y_H)
      data_ch4(i)=data_ch4(i-1);
   end
end
```

```
for i=2:length(data_ch5)
if(data_ch5(i)<modify_L||data_ch5(i)>modify_H)
    data_ch5(i)=data_ch5(i-1);
  end
end
for i=2:length(data_ch6)
if(data_ch6(i)<modify_L||data_ch6(i)>modify_H)
    data_ch6(i)=data_ch6(i-1);
  end
end
for i=2:length(data_ch7)
if(data_ch7(i)<modify_L||data_ch7(i)>modify_H)
    data_ch7(i)=data_ch7(i-1);
  end
end
for i=2:length(data_ch8)
if(data_ch8(i)<modify_L||data_ch8(i)>modify_H)
    data_ch8(i)=data_ch8(i-1);
  end
end
end
```

5. MATLAB Code to calculate spatial resolution

```
clear;clc;
x=0:4.96/499:4.96;
y=0:77/649:77;
[P,map]=imread('3_3db.bmp');
P=im2double(P);
figure(1)
colormap(hot);
imagesc(x,y,P)
xlabel('x mm')
ylabel('y mm')
n=floor(77/6.4);
y1=0:77/649/n:77;
for i=1:500
  inter_p_B_tmp(:,i) = interp1(y,P(:,i),y1);
end
m1=max(inter_p_B_tmp);
m2=max(m1);
for i=1:500
  if m1(i)==m2
    chx=i;
  end
end
line1=inter_p_B_tmp(:,chx);
for i=1:5193
  if line1(i)==m2
    chy=i;
  end
end
line2=inter_p_B_tmp(chy,:);
figure()
subplot(211)
plot(y1,line1)
title('y')
subplot(212)
plot(line2)
title('x')

figure()
subplot(211)
plot(y1,line1)
[fitresult1, gof1] = createFit(y1(2400:2800), line1(2400:2800))
title('y')
subplot(212)
plot(x,line2)
title('x')
[fitresult2, gof2] = createFit(x, line2)
hold off

xf1=y1(2400:2800);
f1 =  fitresult1.a1.*exp(-((xf1-fitresult1.b1)/fitresult1.c1).^2);
xf2=x;
f2 =  fitresult2.a1.*exp(-((xf2-fitresult2.b1)/fitresult2.c1).^2);

m1=max(f1);p1=find(f1==m1);
m2=max(f2);p2=find(f2==m2);
```

```
for i=2:p1
    if abs(f1(i)-m2/2)<abs(f1(i-1)-m2/2)
        hy1=i;
    end
 end
 for i=p1:length(xf1)
    if abs(f1(i)-m2/2)<abs(f1(i-1)-m2/2)
        hy2=i;
    end
 end
res_y=xf1(hy2)-xf1(hy1)
%
 for i=2:p2
    if abs(f2(i)-m2/2)<abs(f2(i-1)-m2/2)
        hx1=i;
    end
 end
 for i=p2:500
    if abs(f2(i)-m2/2)<abs(f2(i-1)-m2/2)
        hx2=i;
    end
 end
res_x=xf2(hx2)-xf2(hx1)

function [fitresult, gof] = createFit(x, y)
[xData, yData] = prepareCurveData(x, y); %
Set up fittype and options.
ft = fittype( 'gauss1' );
opts = fitoptions( 'Method',
'NonlinearLeastSquares' );
opts.Display = 'Off';
opts.Lower = [-Inf -Inf 0];
opts.StartPoint = [y(ceil(length(x)/2))
x(ceil(length(x)/2)) 0.1]; % Fit model to
data.
[fitresult, gof] = fit( xData, yData, ft, opts );
% Plot fit with data.
h = plot( fitresult, xData, yData);
xlabel( '?' );
ylabel( 'R' );
legend off
grid on;hold on;
plot(fitresult.b1,fitresult.a1,'^','markersize',6)
```