Application Note:

Interfacing with the SDRAM controller and implementing a Circular Buffer

By: Aaron Arnason, Edrick de Guzman & Byron Maroney
Group 4

Assumptions:

1. You have a base project similar to that of the Introductory Labs.
2. This also means that you have included the SDRAM controller interface in your QSYS setup
3. You have a working knowledge of Quartus and Qsys.

Hardware Requirements:

1. Altera DE2
2. Speakers
3. Audio cables
4. Audio Source

Software Requirements:

1. Quartus II 32-Bit Version 12.1 SP1

Source Code:

1. The source code in order to fully build this application note can be found in the course website at https://www.ualberta.ca/~delliott/local/ece492/appnotes/2016w/g4_CircularBuffer_SDRAM/

Purpose

1. The purpose of the application note is to guide users on how to utilize the Off-chip SDRAM using a custom SDRAM core component. The SDRAM is useful in many ways.

a. First, the SDRAM has the most space out of all the off-chip memories on the FPGA. It has a whopping 8 MB of memory available to fulfill your desires.
b. Secondly, the SDRAM enables for data manipulation. This means that once you have stored information into the SDRAM, the user is provided the option to access different address spaces in the SDRAM. This was useful in our audio implementation because the manipulation of the addresses allowed us to apply delay and echo to the audio data.

SDRAM

1. Using the SDRAM gives you the ability to utilize memory up to 8MB.
2. Utilizing the SDRAM requires the SDRAM controller component in Qsys to be able to interface with it. Assuming that you have used the Introductory Labs as you base project, the SDRAM controller component should already be instantiated.
3. The SDRAM is byte-addressable, this means that each address space is able to store up to 1-byte (8-bits).
4. This space is very useful for applications such as a delay line in which the SDRAM is used as a circular buffer where data can be stored and accessed.
5. The SDRAM controller provides the following signals to use in your application. For this application note, the signals are used to implement a circular buffer to pass through audio data to demo that the SDRAM functions accordingly.

```
avm_m0_address       : out std_logic_vector(31 downto 0);                    --    m0.address
avm_m0_read          : out std_logic;                                        --      .read
avm_m0_waitrequest   : in  std_logic                     := '0';             --      .waitrequest
avm_m0_readdata      : in  std_logic_vector(15 downto 0) := (others => '0'); --      .readdata
avm_m0_write         : out std_logic;                                        --      .write
avm_m0_writedata     : out std_logic_vector(15 downto 0);                    --      .writedata
avm_m0_readdatavalid : in  std_logic                     := '0';             --      .readdatavalid
```

   a. These are the signals you use in order to access the SDRAM and implement the circular buffer.
      i. ***address*** is the address you want to access in the SDRAM
      ii. ***read*** is a signal to notify the SDRAM that we're ready to read from it
      iii. ***waitrequest*** must be checked whether it is "0" in order for you to begin reading or writing.
      iv. ***readdata*** is the data coming out of SDRAM.
      v. ***write*** is a signal to notify the SDRAM that we're writing to it.
      vi. ***writedata*** is the data you want to write into the SDRAM.
      vii. ***readdatavalid*** is not used in this app note.
6. For your application, you will need to add these signals in your custom component's VHDL code to interface with the SDRAM controller.

Circular Buffer - Custom Component

1. In this application note, we're implementing the SDRAM as a circular buffer for the audio to demonstrate the functionality of the application note.
2. The circular buffer is used by the Audio Core to store audio samples.
3. The SDRAM will also output data in it to the Audio Core, generally proving that data can be written and read from the SDRAM.
4. A buffer is considered circular when the write pointer reaches the end of the specified buffer length and loops back to the beginning of the buffer. This process overwrites the old data stored in that space.
5. The circular buffer also requires a read pointer to read data from it. This read pointer can be initialized wherever the designer wants it to be. For this application, the read pointer is lagging behind the write pointer by 2 address spaces (2 bytes, in order to accommodate for the 16-bit audio sample).

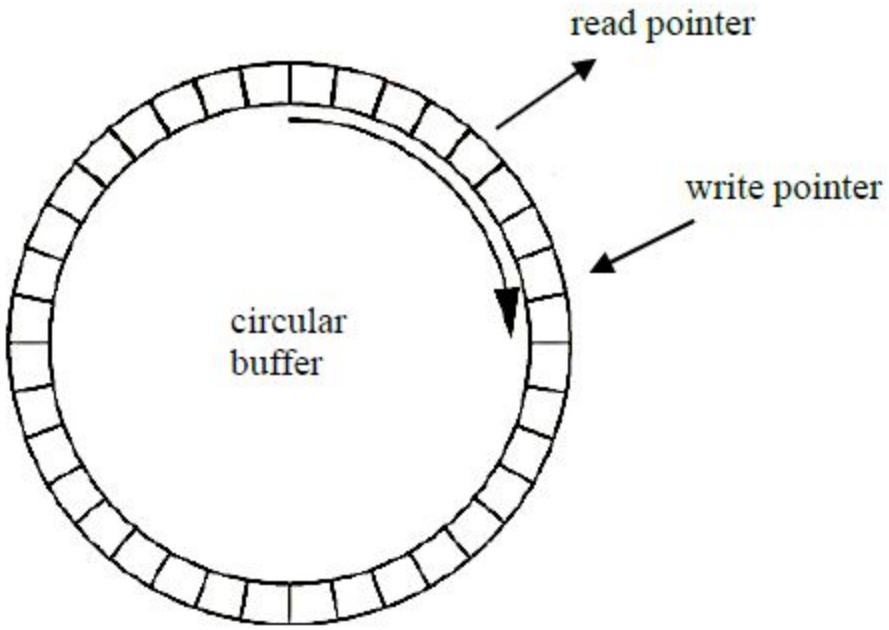

Figure 2. Circular Buffer [2]

6. Base address and buffer size are values provided by the user for the circular buffer. The base address is the starting address of the SDRAM, while the "buffer size -1" becomes the end address of the SDRAM.
7. The provided sdrambuffer.vhd acts as the memory mapped master component for the memory mapped slave port provided by the SDRAM controller core. This file uses the

signals required to use the SDRAM controller interface. In your application, your custom component will act as the memory mapped master component.

Building Instructions

1. Start off by utilizing the Introductory Lab 1 to build a base project [1].
2. Download the files necessary from the course website
    a. Sdrambuffer.vhd - custom component
    b. niosII_system.qsys
    c. niosII_system.sopcinfo
    d. new_component_hw.tcl
    e. sdram_circular_buffer.qar
    f. sdram_circular_buffer.qpf
    g. sdram_circular_buffer.vhd
3. Launch Quartus from the scripts folder of your working directory
    a. In Quartus go to File -> Open Project…
    b. Choose the sdram_circular_buffer.qar
    c. Remove the “restored” path in the Destination Folder after choosing the archive file.
    d. Open Qsys by selecting tools -> qsys (within Quartus)
        i. Open the niosII_system.qsys
        ii. At this point, everything in the project should be ready to be generated and compiled.
        iii. Go to Generation Tab and click Generate.
    e. Go back to Quartus and compile the project.
        i. Expect about 443 warnings.
    f. Once the compilation is done, program the DE2.
4. You should now be able to hear audio when you connect the speakers to the Line out and your music into Line In.

QSYS Configuration (Custom Component)

Adding the Hardware components in QSYS

- This section will describe how the sdrambuffer.vhd custom component is interfaced with the SDRAM controller so that you can follow it to do the same with your custom component.

Note:

1. **At this point, you should have already added the necessary signals that were described above in your custom component that will utilize the SDRAM.**
2. **The Avalon Memory Mapped Master utilizes those signals. If you don't provide them, the you will not be able to interface with the SDRAM.**
3. **The instructions below are for the provided custom component used to demonstrate the functionality of the SDRAM. This will be very similar for your application.**

Add SDRAM buffer custom component : New Component

On the project pane of Qsys, double click "New Component"

Component Type Tab

- Name: sdram_buffer
- Display name: sdram_buffer

Files tab

- Click "+", then search for sdramBuffer.vhd
- Once added, click "Analyze Synthesis files"
- Once completed with no errors, click on the Interfaces tab

Interfaces tab

- Under "m0"
  - Associated clock to clock
  - Associated reset to reset
- Under Clock
  - Type to Clock Input
- Under Reset
  - Type to Reset Input
- Then add the follow interfaces
  - Add Interface
    - Change type to avalon_streaming_sink
    - Change the name to streaming_in

        - Set the clock and reset
    - Add Interface
        - Change type to avalon_streaming_source
        - Change the name to streaming_out
        - Set the clock and reset

Signals Tab

- Change Interface to streaming_in and signal type to valid for “buffer_in_valid”
- Change Interface to streaming_in and signal type to data for “buffer_in”
- Change Interface to streaming_out and signal type to valid for “buffer_out_valid”
- Change Interface to streaming_out and signal type to data for “buffer_out”

Switch back to Interfaces Tab

- Remove Interfaces Without Signals

Click “Finish”

Add SDRAM controller:

On the Component Library look for the SDRAM Controller under Memories and Memory Controllers, then External Memory Interfaces, then SDRAM Interfaces, then add the SDRAM Controller.

- Set Presets to Custom
- Set data width to 16
- Click “Finish”

Connecting the SDRAM controller to the SDRAM Buffer

- “m0” of SDRAM buffer is connected to “s1” of SDRAM controller
- Connect the “clock” to the system clock
- Create Global Reset Network to connect the “reset”
- streaming_in and streaming_out will require timing adapters

References:

[1]
https://eclass.srv.ualberta.ca/pluginfile.php/2247658/mod_resource/content/5/ECE492_W16_Lab1.pdf
[2]
http://jugglingpirate.net/final-year-project-electronics/