University of Pennsylvania
Department of Electrical and System Engineering
System-on-a-Chip Architecture

ESE532, Spring 2017 HW6: Hardware Accelerator Sunday, February 18

Due: Friday, March 3, 5:00PM

In this assignment, we will accelerate the video encoder by implementing functions on the programmable fabric. To save you some time, we provide the initial SDSoC project, which can be downloaded from here.

For this homework, you will have a new partner. You can find the partner assignment in the *Files* section of Canvas.

Timeline

This homework is due in two weeks instead of the usual one week. It consists of three questions. The first two questions provide more guidance than the last question. Therefore, we suggest that you complete the first two questions (3 and 4) this week and the last question (5) next week. If you submit the answers to the first two questions before 5:00PM on February 24, we will make an effort to grade it before February 27. Should you decide to submit it later, but before the March 3 deadline, you will not be penalized, however.

Note that there will be no office hours on February 21 and February 23 due to absence of the TA. We encourage you to send your questions on Piazza instead.

Hardware Acceleration in SDSoC

The first steps towards hardware implementation of a function in SDSoC are explained in Chapter 2 of the SDSoC tutorial. There are several methods that you can use to communicate with your accelerator from the software. You can indicate with a pragma which method you would like to use. The SDSoC pragmas are described in Chapter 17 of the SDSoC manual.

Vivado HLS

Ideally, you can tell SDSoC which function to implement in hardware, add a few pragmas, and SDSoC does the rest. In practice, you will often discover that existing functions cannot be mapped directly on the hardware because they are not synthesizable or inefficient. In those cases, you will need to rewrite your code. Although you can do this in SDSoC, our experience is that the errors and warnings in SDSoC are often unclear or hidden. Moreover,

a debug cycle typically takes longer. Hence, we suggest to start the acceleration of a function in Vivado HLS, which has better debug facilities. Once your function has been verified and synthesized successfully in Vivado HLS, you can copy it to SDSoC and integrate it in the system.[1] Vivado HLS is also based on Eclipse, so most of the GUI should be similar.

Creating a new project is explained here. Make sure you enter the top-level function during the creation of the project (although you can also change it later). The top-level function is the function that will be called by the part of your application that runs in software. Vivado HLS needs it for synthesis. You can also indicate which files you want to create. It is wise to create a testbench file, too, while you are creating the project.

Although you can and should also verify your system in SDSoC like before, you will generally need more time to build your project, so we suggest to make a testbench in Vivado HLS to debug the hardware. The requirements for testbenches are not any different from other software applications written in C. Similar to them, they have a `main` function that is invoked. To the main function you can add any functionality needed to test your function. That includes calling the top function that you would like to test. When the testbench is satisfied that the function is correct, it should return 0. Otherwise, it should return another value.

You can run the testbench by selecting *Project* → *Run C Simulation* from the menu. A window should pop up. The default settings of the dialog should be fine. You can dismiss the dialog by pressing *OK*. You can see in the *Console* whether your test has passed. If your test fails, you can run the test in debug mode. This can be done by repeating the same procedure, except that you should check the box in front of *Launch Debugger* this time before you dismiss the dialog. This will take you to the *Debug* perspective, which should look familiar by now. You can go back to the original perspective by pressing the *Synthesis* button in the top, right corner. Note that in SDSoC you may be used to the fact that it builds your project automatically when you change your code and start or relaunch a debug session. Unfortunately, Vivado HLS will only build your code if you select *Run C Simulation* from our experience.

Once you are satisfied with your code, you can run *Solution* → *Run C Synthesis* → *Active Solution* from the menu to synthesize your design. You can also verify the synthesized version of your accelerator in your testbench. If you choose to do so, Vivado HLS will run your accelerator in a simulator, so this method is called C/RTL Cosimulation. Simulation can take considerably more time, so it may not be practical for every testbench. Anyway, you can start it by choosing *Solution* → *Run C/RTL Cosimulation* from the menu.

To get some insight into what the hardware that Vivado HLS creates looks like, you can read this. The different pragmas that you can use in your functions are listed in the sections for the associated TCL commands in the Vivado HLS manual. If you need information about the `PIPELINE` pragma, you can look up the `set_directive_pipeline` command.

[1] SDSoC uses Vivado HLS under the covers. Ideally, it would hide the need to use it independently, but it also hides useful errors and warnings.

Homework Submission

Please be concise in your answers.

1. **Learning and navigating the tools.**

   Document all the problems you encountered in working the assignment and how you solved them. For each, include:

   (a) Step (if applicable)

   (b) OS you were running

   (c) Error message or unexpected behavior you encounted

   (d) How the problem was resolved

   (e) How you found the solution (e.g., how did you experiment or reason through an answer, which document had the answer, where on the web did you find an answer, which student or TA was able to point out the answer)

2. **Teamwork**

   (a) Document the work sessions with your partner (where, when, duration).

   (b) How did you collaborate on the work? (pair programming is acceptable, explain)

   (c) What background basics, tricks, or techniques did you learn from your partner?

   (d) Provide highlights for cases where working together allowed you to understand deeper, combine insights, avoid pitfalls, and/or build confidence in your directions and solutions.

3. **Initial Implementation**

   (a) Adapt the `fullsearch` function such that it can be mapped onto hardware. Follow these steps:

      i. `fullsearch` reduces the search area when it extends beyond the edge of the image. This results in fewer loop iterations. Loops with varying numbers of iterations do not accelerate well in hardware, so we will only accelerate the common case. Create a new function in Vivado HLS that is a specialization of `fullsearch` for the cases in which we can use the full search area.

      ii. The input variables `org` and `blk` refer to the frame buffers, which contain much more data than needed. Some of the data transfer methods that we will explore are limited to a buffer of 16KB. Therefore, the next step is to adapt your new function such that it takes input from 2 smaller buffers that contain only the data in the search area and the current macroblock respectively. Declare the input parameters as arrays instead of pointers such that SDSoC has no trouble determining the the size of the blocks.

iii. Set the clock period to 7 ns. In the menu, go to *Solution→Solution Setting...*. That should open up the *Solution Settings* dialog. Select *Synthesis* on the left side, and enter the clock period on the right.

iv. Verify that your new function works and synthesizes. As mentioned earlier, we suggest making a testbench in Vivado HLS. You can test your motion estimator using actual data from the encoder that you extract, but it is probably easier to test with some artificial data that you prepare.

v. Integrate your new function in the encoder. You will need to adapt `frame_ME` such that it calls the old `fullsearch` or your new function depending on the search area size. Verify that the integration works.

Communicate with the first version of your accelerator via shared memory. You can instruct SDSoC to use shared memory using the `zero_copy` pragma. Show the code changes that you made so far.

(b) Report the average duration of one call to your new function in the application without hardware acceleration on the ZedBoard at optimization level `-O2`. Report also the execution time needed for the entire application.

(c) What does Amdahl's Law tell you about the maximum potential application speedup you can get from accelerating your new `fullsearch` function?

(d) Report the speedup that SDSoC expects the encoder to reach after accelerating the your new `fullsearch` in hardware. You can find this information in the *Performance Estimation Report*. How to generate this report is described in the SDSoC environment introduction.

(e) Report the expected speedup when data is transferred via DMA. You will need to add the right pragma(s) for this.

(f) Choose the implementation with the best expected speedup, and run it on the FPGA (without performance estimation). Report the average duration of a call to your new function. Report also the execution time of the entire application.

4. **Explore Accelerator**

(a) Review the resource utilization and schedules in Vivado HLS. Schedules can be very insightful for pragmas.

i. What is the number of flipflops that your function utilizes according to the synthesis report?

ii. What is the initiation interval of the innermost loop of your function? You can find it in the *Analysis* perspective.

(b) Include the *Data Motion Report* for your accelerator in your report, and answer the following related questions:

i. Describe how the pixels in the search area are transported to your accelerator.

ii. Explain why a buffer in paged memory can increase the complexity of the DMA controller.

(c) Answer the following questions about the hardware implementation. You can view the hardware by opening the project in `<Project>/SDDebug/_sds/p0/ipi/zed.xpr` in Vivado. Here, `<Project>` is the directory of the SDSoC project in which you built your hardware.

i. Explain how the motion vector is transfered from the accelerator to the processor. Mention the hardware blocks and buses that are involved.

ii. Report the address and size of the memory area that is assigned to your accelerator.

iii. Include a picture of your FPGA implementation, with your hardware function highlighted. To view the implementation, press the *Implementation→Open Implemented Design* in the *Flow Navigator* on the left side of the Vivado window. In the *Netlist* window, you can find a module with the name of your function and a `_0` suffix. Right-click on it, select *Highlight*, and choose a color.

5. **Exploring Variant Hardware Implementation Design Choices**

Improve the performance of the motion estimation.

(a) Try five different implementations of your function in which you apply one pragma that could improve the performance at a time. Show how and where you apply the pragma and report the encoder performance and hardware resource utilization for each case. Some pragmas that you could try are: the `HLS pipeline`, `HLS unroll`, `HLS array_partition`, `HLS dataflow`, `HLS inline` pragma, or `SDS data_mover`. Feel free to change the code as necessary, as long as the code is semantically identical and you show the code changes.

(b) Show the implementation of your function that yields the best performance. Report the obtained encoder performance. Limit your search to combinations of the optimizations that you tried. Trying all combinations will probably be too time-consuming, so select the best implementation based on performance estimates, reasoning, and the results above.