2015
Stefano Tabanelli

SDSoC by simple examples

Accelerating Your Success™

Agenda 1/2

- What is SDSoC
- SDSoC by simple examples, hands-on intro
- Hello world
  - L0
- One line code DMA
  - L1 no PL acceleration
  - L1_1 PL accelerated, non-contiguous memory
  - L1_2 PL accelerated, contiguous memory
  - L1_3 PL accelerated, zero copy memory
  - L1_4 PL accelerated, zero copy memory, async tasks

Agenda 2/2

- Two operands functions
  - L2 with PL acceleration
  - L2_1 PL accelerated, pipelined
  - L2_2 PL accelerated, pipelined, perf estimation
  - L2_3 PL accelerated, pipelined, loop unrolled, perf estimation

Backup slides

- Matrix floating point multiplication example on Linux
  - L3 (Xilinx UG1028)
- Demo
  - SDSoC Custom Vivado project

What is SDSoC

- SDSoC is
  - A C/C++ IDE for Zynq&MPSoC (similar to XSDK)
  - A C/C++ to VHDL/Verilog function converter (Vivado HLS invoked in background)
  - A data mover instantiator (to feed data to functions implemented in Programmable Logic)
- In essence SDSoC is a tool able to compile a C/C++ project for Cortex core, moving functions to PL on demand

What is SDSoC

- SDSoC is targeting software developers
  - No Vivado/HLS explicit installation (just SDSoC)
  - No VHDL/Verilog pre-requisites
  - Modest investment
- SDSoC can generate a complete system (bitstream+application+OS) running Linux, FreeRTOS or Baremetal in SD card format

SDSoC by simple examples, hands-on intro

- SDSoC automatically compile C/C++ but…
- …considerable amount of time needed to study #pragmas and practice different optimizations
- During this session we'll not try to obtain extreme performances
- The goal is to make you
  - Aware of SDSoC flow
  - Understand the basics of the tool

Requirements

- SDSoC 2015.2 installed and licensed
- Microzed board
  - If you miss the board
    - share with a colleague OR
    - proceed using only your laptop and SDSoC
- JTAG cable
- microUSB cable
- uSD card & card reader (L3)
- CAT5 Ethernet cable (L3)

Hello world

- Lab L0 is used to check that the SDSoC installation and the Microzed board are correctly working on your system
- Go to page 3 “L0 Hello World” and follow the steps

One line code DMA

- During lab L1 you will write a very simple C program to copy data from a source address to a destination address
- In C this function is just one code line (plus one for the loop)
- Along the lab we'll evolve the code using SDSoC directives and move the function from the Cortex A9 to the Programmable Logic (PL)

One line code DMA

- Chapter “L1 One line code DMA, no PL acceleration” will show how to implement the function using only the Cortex A9, therefore no PL will be used
- The memory will be dynamically allocated from the system heap
- Go to page 7 and follow the steps

One line code DMA

- Chapter “L1_1 One line code DMA, PL accelerated, non-contiguous memory” will show how to move the function to PL
- SDSoC will take care of transferring data to/from the PL function using a DMA
- Because we used malloc the memory can be physically non-contiguous, therefore a Scatter Gather DMA will be automatically selected and used
- Go to page 13 and follow the steps

One line code DMA

- Chapter “L1_2 One line code DMA, PL accelerated, contiguous memory” will show how to use an SDSoC C library function to allocate a physically contiguous memory area
- Because we have a physically contiguous memory area the tool will instantiate a simpler DMA (not Scatter Gather)
- Go to page 20 and follow the steps

One line code DMA

- Chapter “L1_3 One line code DMA, PL accelerated, zero copy memory” will show how to use a #pragma to prevent any dynamic memory allocation
- Because we have no memory allocation, the tool will turn our function into an AXI master, able to access the dynamic memory and collect the required data autonomously
- Go to page 26 and follow the steps

One line code DMA

- Chapter “L1_4 One line code DMA, PL accelerated, zero copy memory, async tasks” will show how to use a #pragma to make our function non-blocking
- Cortex A9 will not wait for our function to return, therefore additional tasks can be executed on the processor in parallel to our function
- Go to page 28 and follow the steps

Two operands function

- During lab L2 we'll write a simple C program to compute a mathematical function on two input vectors
- The vectors are formed by integers (32 bit)
- The example function is the multiplication
- The vector mult function will be implemented in PL
- Along the lab we'll guide SDSoC via #pragmas and build different PL implementations, trying to enhance the performances

Two operands function

- Chapter "L2 Two operands function, PL accelerated" will show a C code to multiply two integer vectors
- The vector multiply function will be implemented in PL by SDSoC
- SDSoC generates a Vivado HLS sub-project to convert our function into Vhdl/Verilog/SystemC
- We'll open Vivado HLS and see that a single multiplication is started every 9 cycles
- Go to page 33 and follow the steps

Two operands function

- Chapter “L2_1 Two operands function, PL accelerated, pipelined” will show how to optimize the performance of multiply function
- Using a #pragma we’ll instruct SDSoC to pipeline our function, therefore we’ll start a multiplication on every cycle (instead of the previous 9 cycles)
- Go to page 40 and follow the steps

Two operands function

- Chapter “L2_2 Perf estimation on two operands function, PL accelerated, pipelined” will benchmark the PL implementation vs the Cortex A9 implementation
- Using SDSoC “Performance estimation” we’ll run on Microzed the two implementations and we’ll compare the required cycles
- Go to page 42 and follow the steps

Two operands function

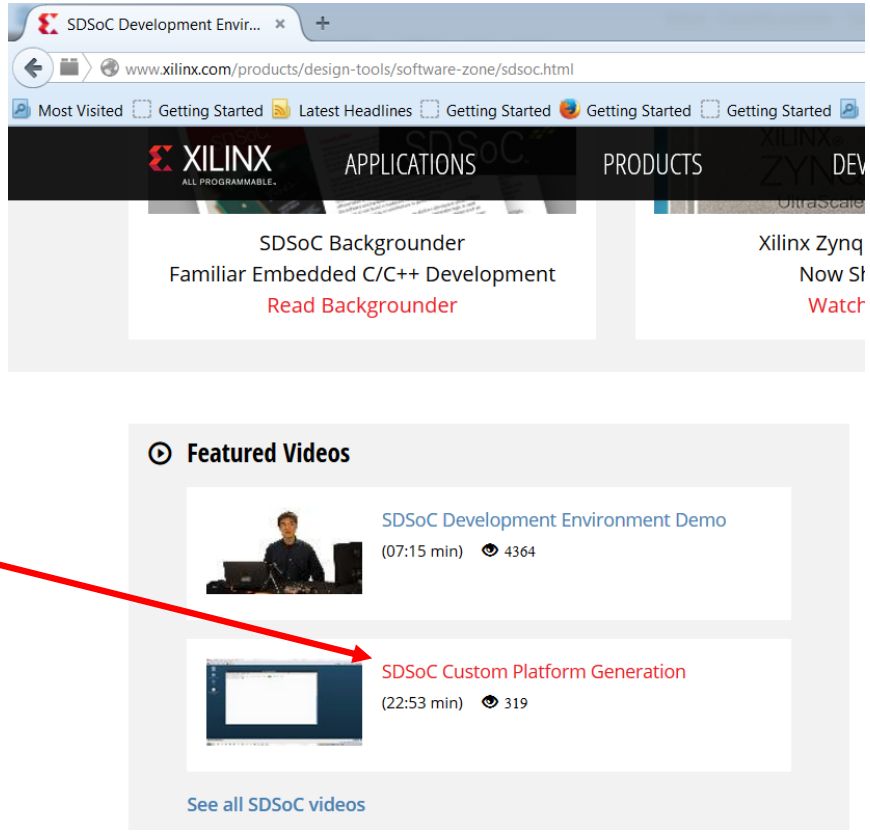
- Chapter “L2_3 Perf estimation on two operands function, PL accelerated, pipelined, loop unrolled” will exploit PL to execute several multiply functions in parallel
- We’ll use a SDSoC #pragma to instantiate two multiply function in parallel on the PL
- These 2 functions need to access the data concurrently, therefore we’ll instruct the tool to align the data structure parallelism
- Go to page 44 and follow the steps

Backup slides

- SDSoC provides the best advantages when operating on complex functions
- A good example (floating point matrix multiplication) is shown on “SDSoC Environment User Guide” UG1028 (http://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_2/ug1028-sdsoc-getting-started.pdf)
- Note that the example will be run on Linux
- Go to page 48 and follow the steps

Demo : SDSoC Custom Vivado project

http://www.xilinx.com/products/design-tools/software-zone/sdsoc.html




Thank you.