System Design Using Xilinx Tools

Rajesh Kedia

07-Dec-2015

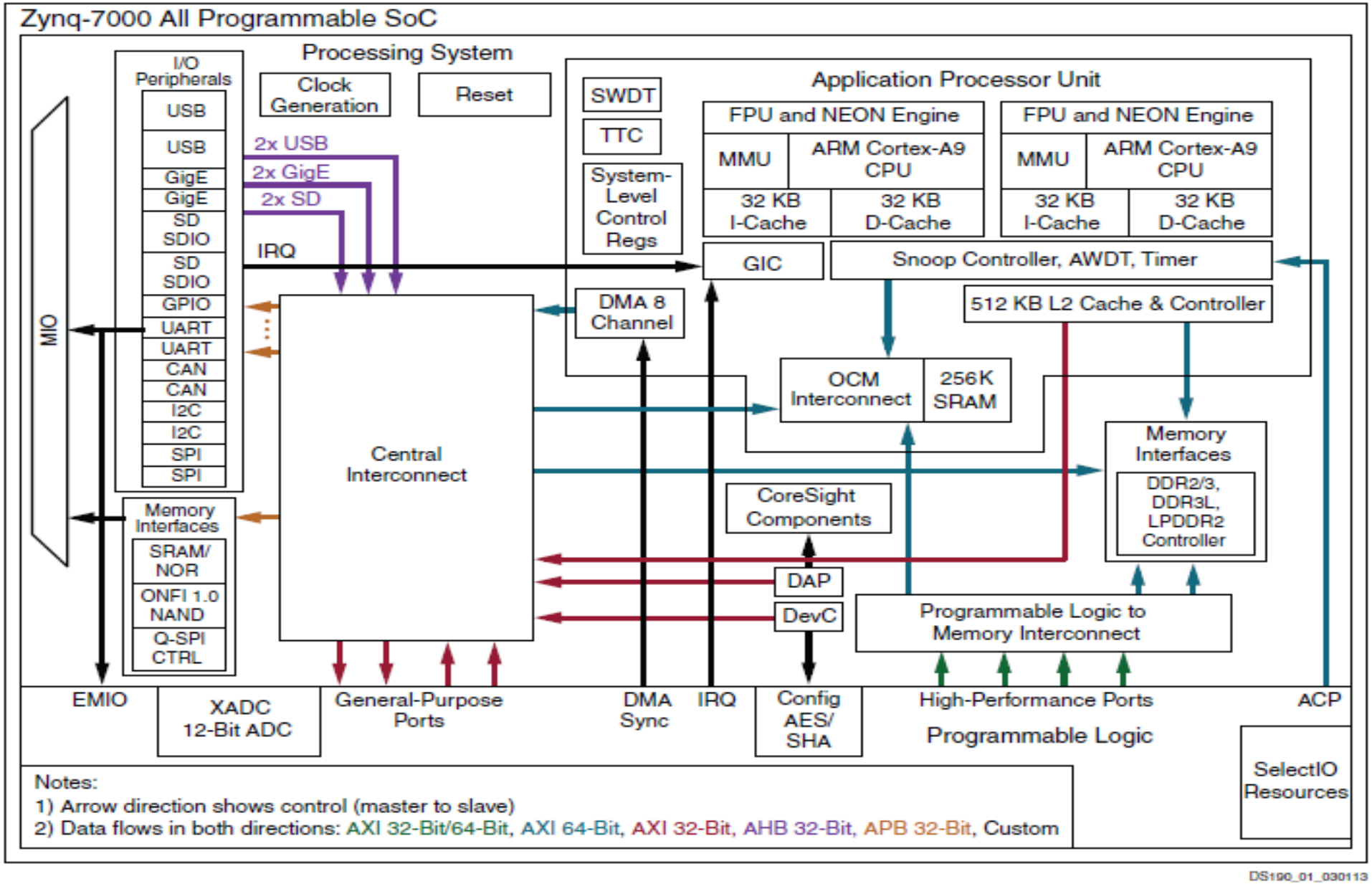


Figure 1: **Zynq-7000 All Programmable SoC Overview**

Fig. source: DS190, Zynq TRM, Xilinx.

Highlights:

PS:

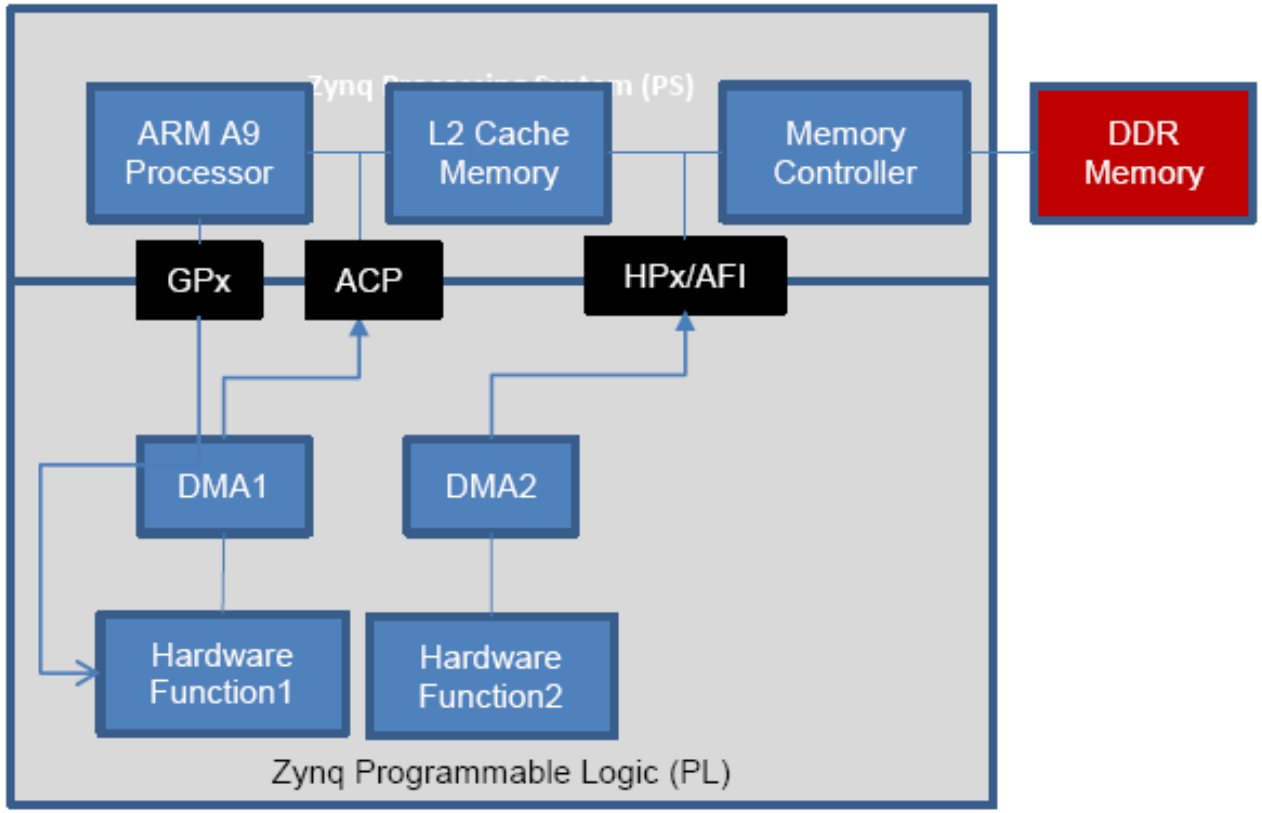
- Dual-Core ARM A9
- L1 and L2 Caches
- DDR interfaces
- DMA controller
- Peripherals
- Interconnects

PL:

- CLB: LUTs, FFs, Adders.
- DSP
- BRAM: Single or dual port

Zynq Data Flow Interface

- 3 types of system ports:
  - ACP (Accelerator Coherence Port):
    - Direct access of L2 in a coherent manner.
    - Fastest means of accessing shared data.
  - HPx (High Performance 0-3):
    - Direct access of DDR via AFI (Async FIFO)
    - Either HW or Proc. can access at a time, cache flushed before HW invocation
  - GPx (General Purpose 0/1):
    - Processor can read/write the registers in the HW module.
    - Not preferable for large data transfers.



Simplified Zynq + DDR diagram showing memory access ports and memories

Fig. source: ug1028, SDSoC documents.

Tools from Xilinx

Sl. No.	Name	Input	Output	Comment
1	ISE	RTL	Bitfile	
2	Vivado IPI	Block Diagram (+RTL)	Bitfile	Invokes ISE internally to generate bitfile
3	Vivado HLS	C/C++/SystemC	RTL	
4	SDK	C/C++	ELF	
5	SDSoC	C/C++	Bitfile + ELF	Invokes HLS, ISE internally to generate the bitfile and SDK to generate the ELF

Bitfile: Final synthesized hardware information to be loaded in FPGA

ELF: Executable software application for APU

What is High Level Synthesis (HLS)?

- Given a C/C++ function, synthesize it to an equivalent RTL description (VHDL/verilog).
- Tools provide many directives (pragmas) to control the structure of logic generated:
  - Resource usage versus speed tradeoffs.
  - Application specific constraints to optimize the hardware.

Example Directives in Vivado HLS:

- ARRAY PARTITION:
  - Partitions array into smaller arrays, each in a different BRAM. Thus providing parallel data ports.
- PIPELINE:
  - Pipelines subsequent iterations of the loop so that the overall throughput is increased.
- UNROLL:
  - Unroll for loops to create multiple independent operations.
- RESOURCE:
  - If any specific resource is to be used for a variable or array. E.g. BRAM, DSP.

How to use HDL generated by HLS?

- Just a synthesis is not enough in embedded applications.
- Need to connect the IP to the processor system.
- Need to write application code to access the IP.
- The generated IP may not be optimized as per the application needs and might require iterations of synthesis and import.

SDSoC aims to unify the above steps under a single framework.

SDSoC Overview

- Software Defined System on Chip
- C/C++ based System design tool for HW-SW co-design targeting Zynq-7000 family of devices
- Within a given application, user can specify which functions to be implemented in HW
- The tool automatically generates the equivalent HDL code for the HW portion
- Downstream flows (synthesis, bitstream generation for HW and compile/link for the SW portion) invoked automatically
- Can specify pragma for directing tool for specific optimization or user constraints

Optimization Options in SDSoC

SDSoC based System Optimizations

- Memory allocation for efficient data transfer
  - sds_alloc versus malloc
  - DMA mode can be DMA_SIMPLE or DMA_SG (scatter gather) – automatically inferred or can be forced by pragma.
- Pragma available to select ACP/HP ports for data transfer.
- Select appropriate clock sources for accelerator and data transfer:
  - –clkid n
  - –dmclkid n
- Synchronization Method:
  - Polling
  - Interrupts

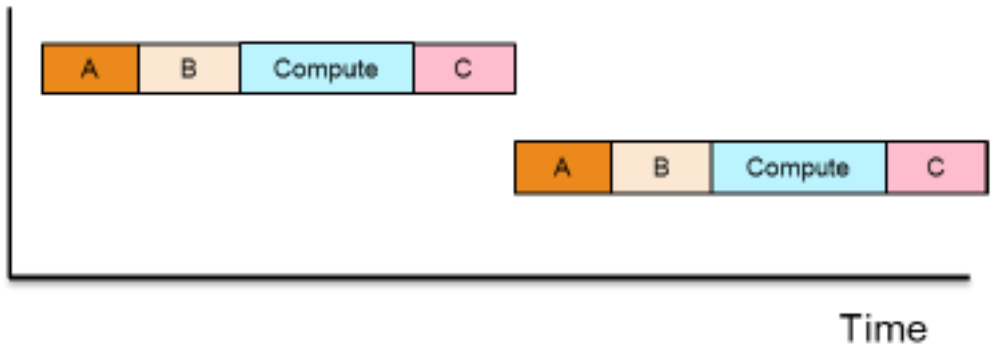
Vivado HLS based Optimizations

- Increasing parallelism:
  - #pragma HLS PIPELINE II=1 (II stands for Initiation Interval)
- Increasing memory bandwidth:
  - #pragma HLS array_partition variable=in_A block factor=16 dim=2
  - #pragma HLS array_partition variable=in_B block factor=16 dim=1

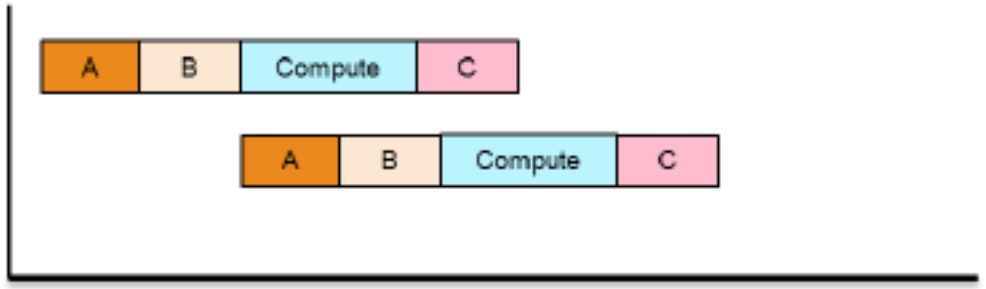
- All Vivado HLS pragma can be used at SDSoC to control the synthesis.

Task Level Pipelining

- Sequential:
  - Simpler Hardware and less logic



- Pipelined:
  - Multiple buffers to store previous data until it is processed.
  - #pragma SDS async(ID) and #pragma SDS wait(ID)



Accelerator Optimizations

- Accelerator to accelerator communication:
  - Directly connect the o/p of one acc. to next, bypassing the main memory. E.g. Multiply and Accumulate. Multiplier o/p can be fed to accumulator input directly.
  - Another example:
    - mmult_accel(tin1Buf, tin2Buf, toutBufHwInter);
    - mmult_accel(toutBufHwInter, tin2Buf, toutBufHw);
- Multiple Instances of accelerator:
  - Whether multiple calls to same function creates one instance or multiple instances of in HW.
  - Controlled by using the async pragma.
  - Also sometimes defined by the connectivity of the HW function (same example as above).

Other Features

- Available Libraries (FIR, Math, etc.)
- Existing HDL IP can also be integrated into the flow
- Dividing HW functions into partitions

References

- C:/Xilinx/SDSOC/2014.4/doc/
  - ug1027 – Introduction to SDSoC
  - ug1028 – SDSoC Getting Started
- DS190 – Zync 7000 Overview
- Vivado HLS documentation

Lab Experiments Overview

Day-1: Interface IP to Zynq.

- Integrate GPIO module to Zynq processor. GPIO module will drive LEDs on Zedboard.
- Write a C/C++ application to drive different values on the LEDs.
- Integrate an adder to the system.
- Write an application to add 2 numbers using this adder and display the sum on LEDs.
- Interface an accelerator which accepts start address, num. of words as inputs. Reads corresponding number of data from BRAM, adds them and displays the sum on LEDs.

Day-2: Explore SDSoC.

- Experience accelerator design in a system level scenario.
- Use matrix multiplication as the example.
- Explore various directives which are used to control the structure of the generated hardware.

  Steps:
  - Matrix multiplication synthesis to hardware and evaluate speedup post synthesis.
  - Matrix multiplication performance and area estimation if porting to hardware.
  - Try various system level variations and understand impact on speed/area.

Day-3: Multithreading using SDSoC.

- Write multi-threaded application for dual-core.
- Spawn some of the threads on the hardware accelerator.
- Understand directives for synchronization of threads.

Thank You

BACKUP

Comparison with Vivado HLS

Sl. No.	Vivado HLS	SDSoC
1	Generate RTL separately for each of the target function	System Level flow, generates RTL for all target functions within the system
2	Manual efforts to integrate the generated RTL with other RTLs or software	Automatic integration of the generated RTL: o connection via stubs to software o connection between accelerators
3	Cannot perform optimizations at the system level	Can optimize across blocks in the system
4	Requires understanding of C and HDL	Lesser understanding of HDL needed