SDSoC Development Environment

SDx 2018.2

XILINX.

Objectives

- **After completing this module, you will be able to:**
  - Describe the SDSoC development environment
  - List some of the benefits of using SDSoC
  - Identify some of the underlying tools the SDSoC development environment uses
  - Describe the SDSoC tool development flow
  - List steps involved in creating an SDSoC project

Outline

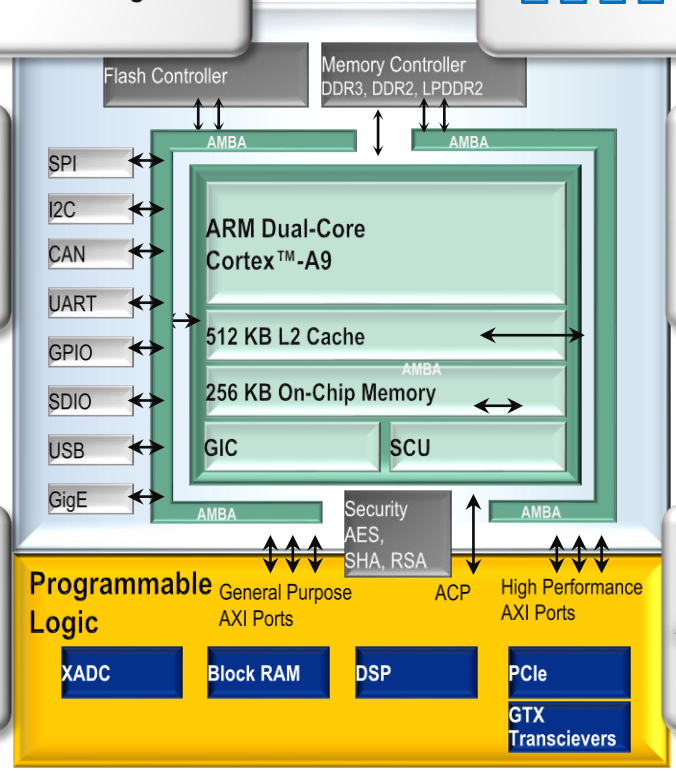
- SDSoC Development Environment
- SDSoC Development Flow
- SDSoC Project Creation
- Summary
- Lab1 Intro

Zynq-7000 All Programmable SoC Highlights

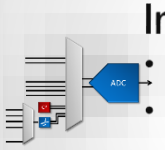
Application Processors
- Up to 1GHz
- Single & Double Floating Point

7-Series Programmable Logic
- 28K-440K Logic Cells
- Hard PCIe Gen2x8

Integrated Memory Mapped Peripherals
e.g. USB2.0, GigE, SDIO, CAN



High Bandwidth Memory & DSP
- Internal L1/L2 Cache and OCM
- DDR3, DDR2, LPDDR2 w/ ECC
- Up to 2,662 GMACs of DSP perf.



Integrated Analog
- Dual multi-channel 12-bit ADC
- Up to 1Msps
- Temp & Voltage sensors

AXI Interconnect
- Up to 100Gb of bandwidth
- Over 3000 interconnects
- ACP Acceleration Port

SDSoC™ Environment

Runtime SW & Tools
- Eclipse-based SDK for profiling and performance analysis
- Cross-trigger/Heterogeneous Debug, Run-time Libraries
- Extensive support for OS, RTOS, AMP, Hypervisor
- PetaLinux for easiest out-of-box development

SDSoC- Part of SDx

- **SDx provides SDSoC - SDAccel Development Environment Common Infrastructure**
  - Eclipse based IDE with support for project creation, emulation, performance estimation, implementation and debug
  - One-stop Reports View to access all reports
- **SDSoC supports**
  - Devices support
    - Zynq-7000 and Zynq UltraScale+ MPSoC support
  - ARM compiler tool chain support
    - Linaro-based gcc 6.2-2016.11 32-bit and 64-bit tool chains
  - Target OS support
    - Linux, bare-metal, FreeRTOS and FreeRTOS_Xilinx
  - QEMU and RTL co-simulation (Zynq target, Linux Host OS only, beta for Windows)
  - OpenCL compilation flows supported for Zynq and Zynq UltraScale+ MPSoC devices

SDSoC Development Environment

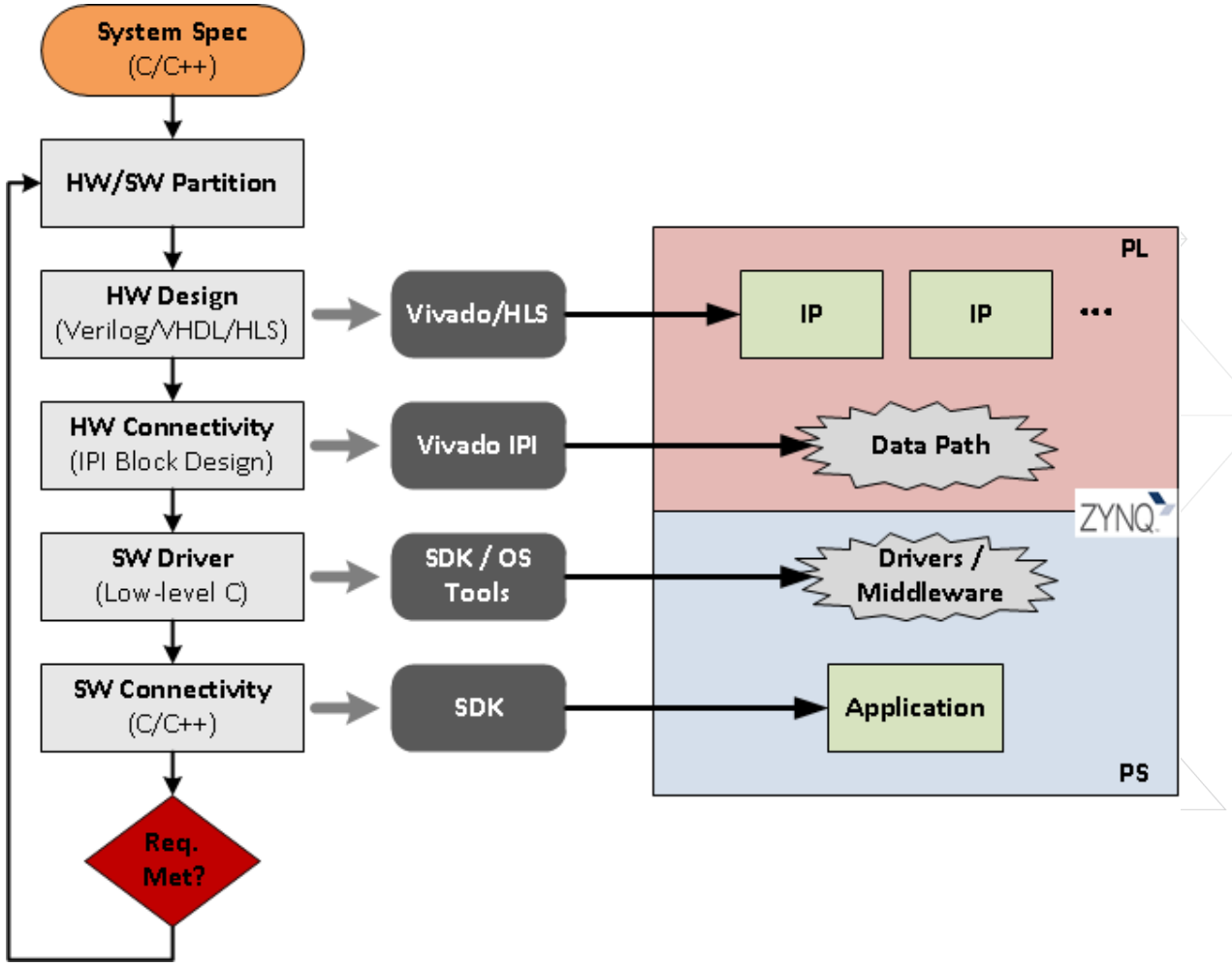
- **What is it?**
  - SDSoC provides a familiar embedded C/C++ application development experience including
    - An easy to use Eclipse IDE
    - A comprehensive design environment for heterogeneous Zynq SoC and MPSoC
    - C/C++ full-system optimizing compiler
  - SDSoC enables function acceleration in PL with a click of button using various technologies
    - Vivado IPI
    - Vivado HLS
    - SDK
    - Profiler
  - SDSoC provides infrastructure to combine processing system, accelerators, data movers, signaling, and drivers
    - Whereas the Vivado HLS tool only build the RTL; connectivity, supporting IP drivers etc. must be supplied by the designer

Benefits of SDSoC Development Environment

- **Shorter development cycles**
  - Estimation shows improvement over software-only solution for system and accelerators
  - Iterative improvement can be made at early stages of development without the need to build hardware
- **Simplified interface and partitioning between hardware and software**
  - Software-based flow vs RTL
    - User interfaces with software-only tools; hardware management is abstracted
    - Removes much of the hardware/software distinction and throw-over-the-wall effect
- **Automated initial design**
  - Users can tweak code at macro- and micro-architectural levels
  - Designers still have manual control over the constructed Vivado HLS tool and Vivado Design Suite projects

Development Flow Without SDSoC

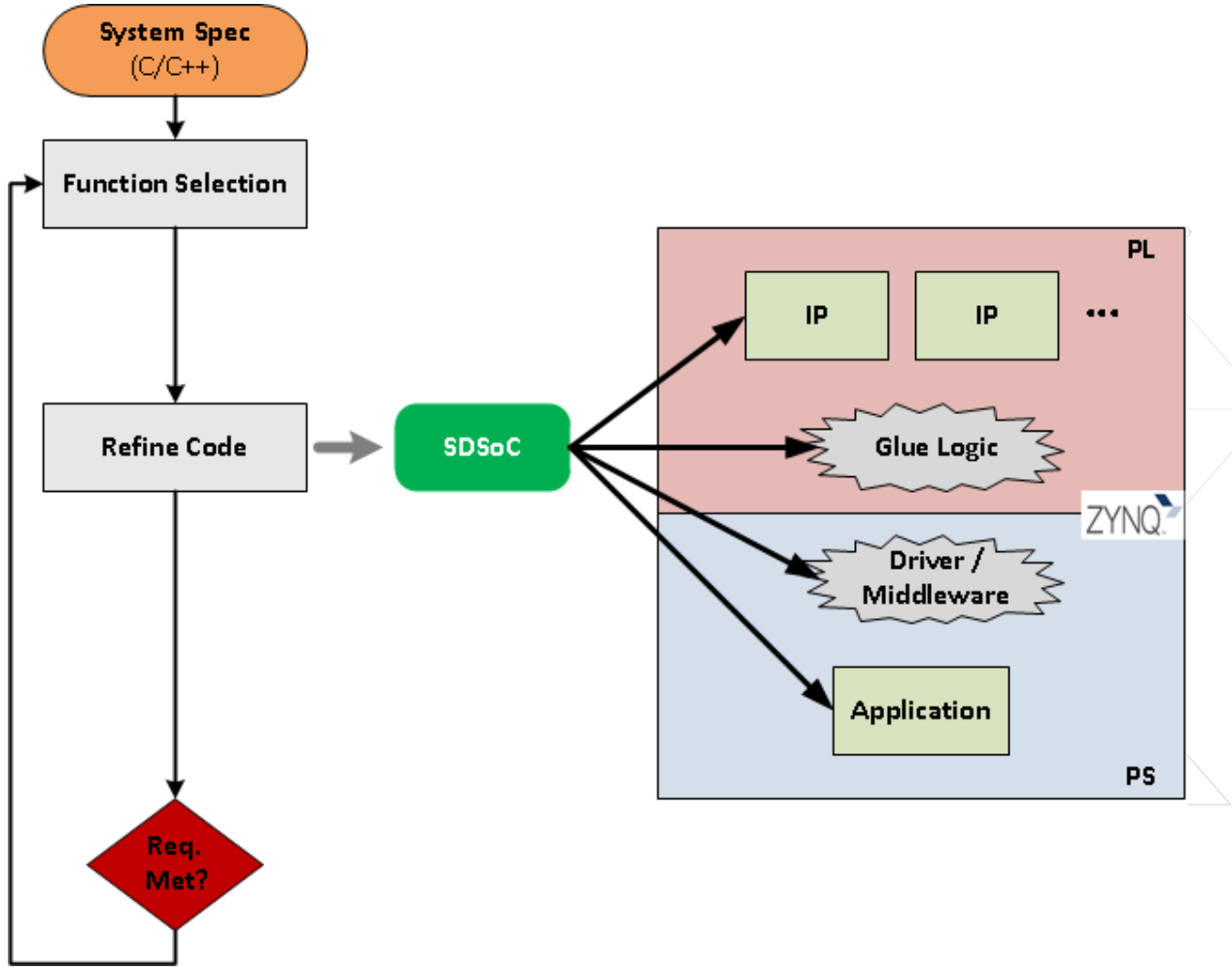
- **Overall process requires expertise at all steps in the development process**
  - Various hardware design entries
  - Hardware connectivity at system level
  - Driver development to drive custom hardware
  - Integrating with application and target OS



XILINX

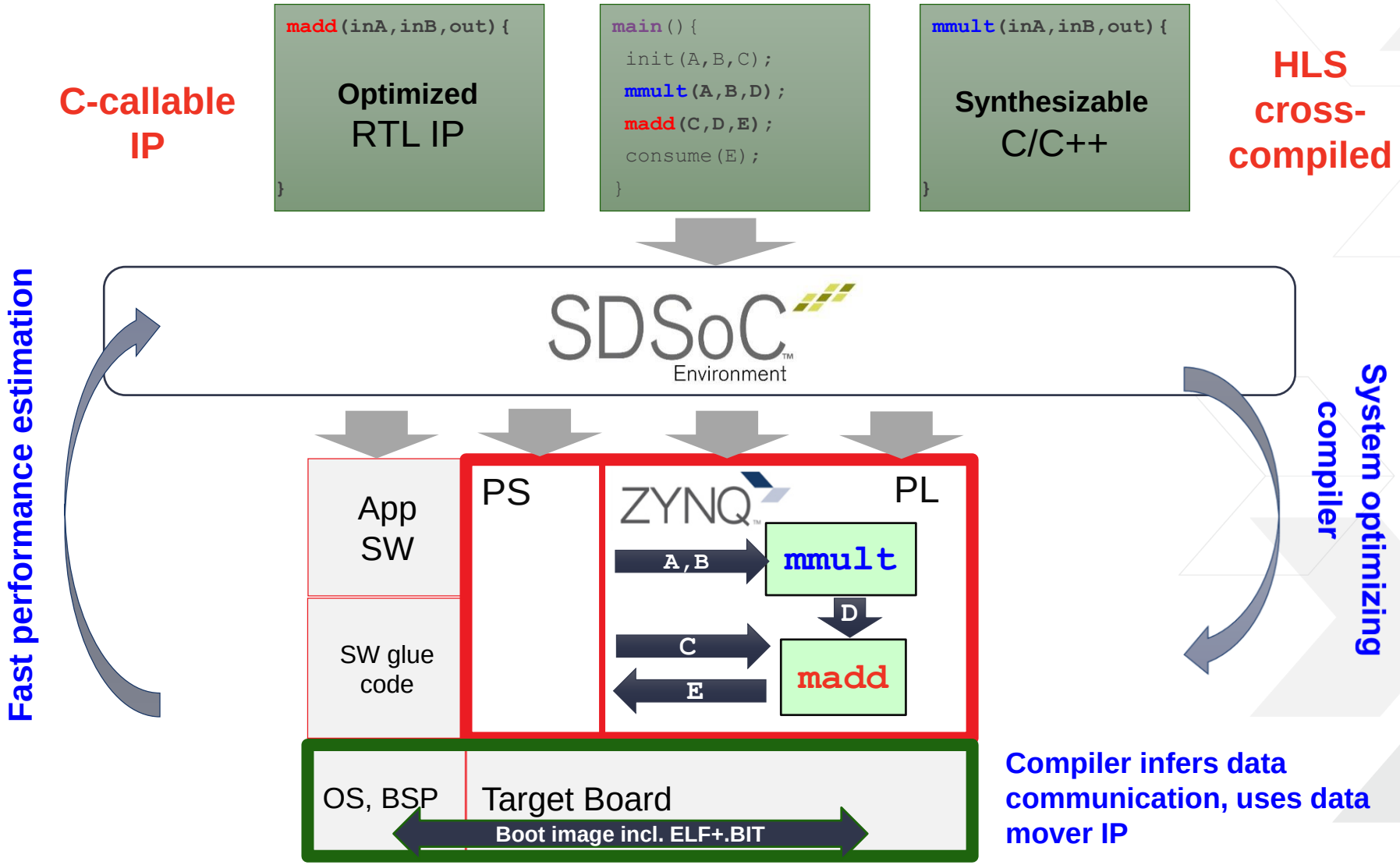
Development Flow with SDSoC

- **SDSoC development environment consolidates a multi-step/multi-tool process into a single tool and reduces hardware/software partitioning to simple function selection**
  - Code typically needs to be refined to achieve optimal results



SDSoC Development Flow

SDSoC Development Flow



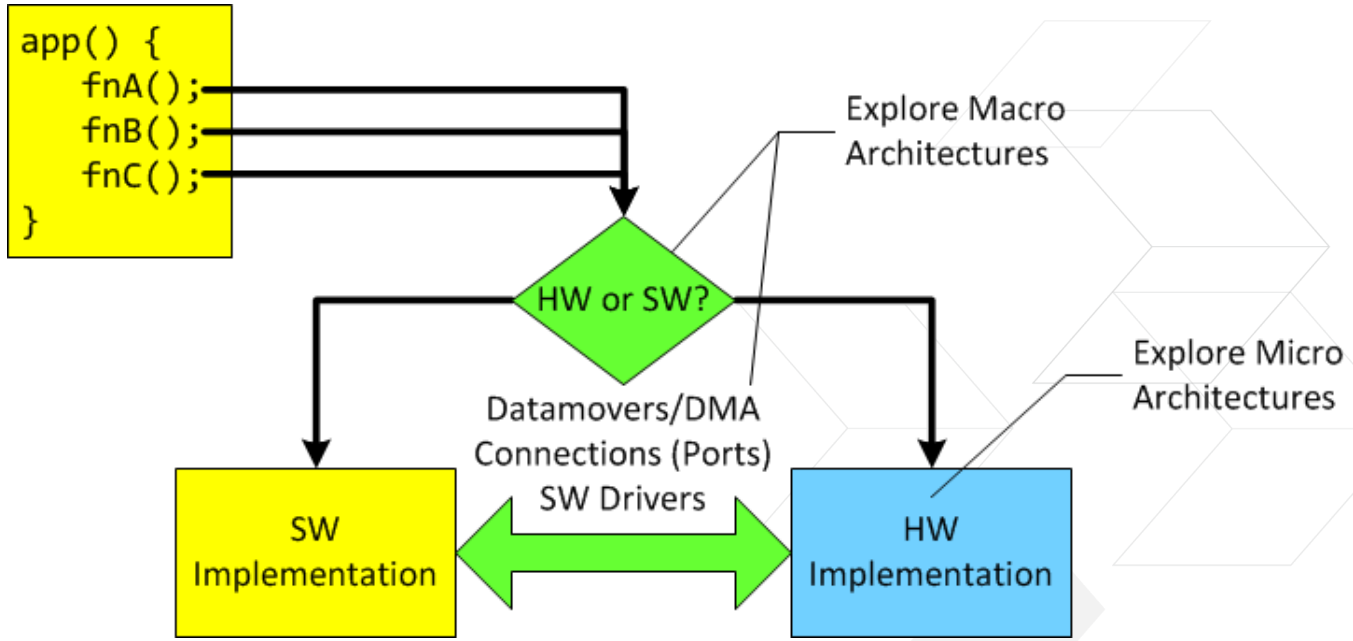
XILINX

Users must keep in mind

- **Tools abstract away many of the challenges**
  - Novices can attain some basic level of results
  - Full performance only comes with understanding
- **The Vivado HLS tool**
  - The SDSoC development environment does not perform any code optimizations
  - The Vivado HLS tool skills are very important to achieve optimal performance
  - Understanding the Zynq AP SoC architecture is also beneficial
- **C/C++**
  - Tools are GNU based
  - New C learners should follow a non-SDSoC development environment flow
  - Familiarity with refactoring for the SDSoC development environment

System-level Considerations

- What gets accelerated?
- How is software implemented in hardware?
  - Is hardware design expertise available?
- How will software and hardware talk to each other?
- Will it meet performance requirements the first try?
  - What changes are required at the macro/micro-architecture levels

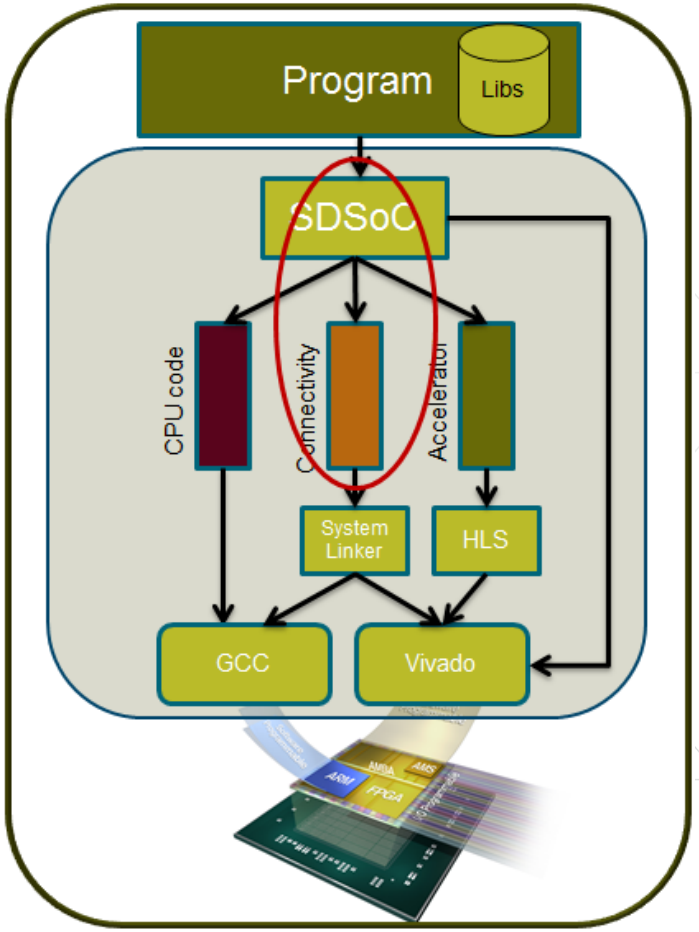


Candidate for an Accelerator

- **Not every function can be a candidate for acceleration**
  - For example, pre-compiled code, some user libraries, OS services, etc.
- **Computationally intensive algorithms are ideal candidates**
  - Profiling results help identify performance bottlenecks
    - Just because a function takes a long time, it is not automatically a candidate
  - Slight code modifications (or pragmas) effect the microarchitecture of the accelerator
- **Trade-off between data movement costs and acceleration benefits should be considered**

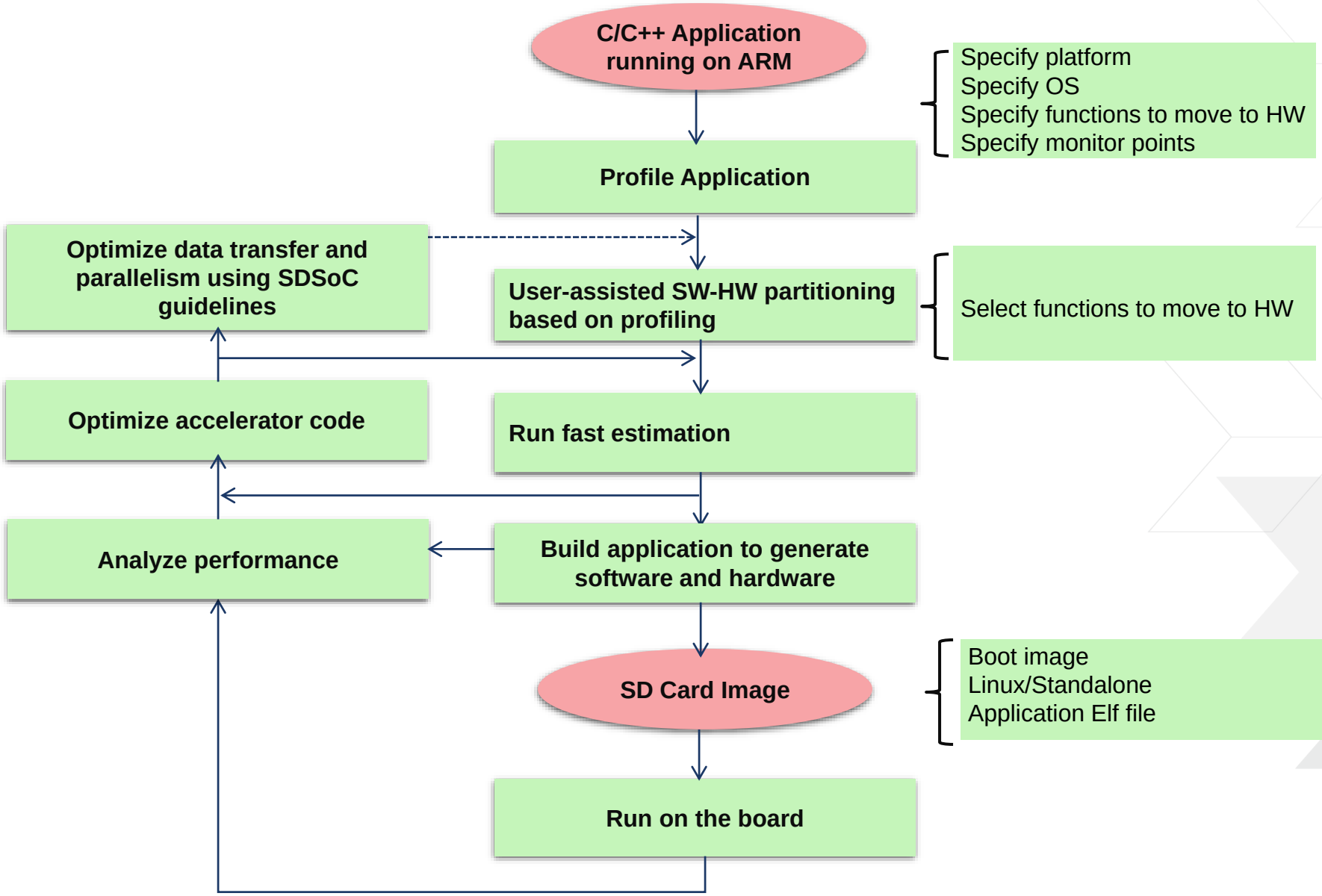
Simplifying Hardware/Software Partitioning

- **SDSoC simplifies the process of identifying and accelerating functions**
  - Integrates profiling tools
  - Provides libraries for accurate timing measurements
  - *User selects functions* to accelerate
  - Estimates hardware speed-up
- ***Automated* acceleration is achieved through**
  - Advanced analysis of code using specialized compilers
  - Assistance of user-specified pragmas
  - Extension of a predefined hardware platform
- **Hardware platforms packaged with the environment and from third parties**
  - Environment extends platform, providing additional capabilities

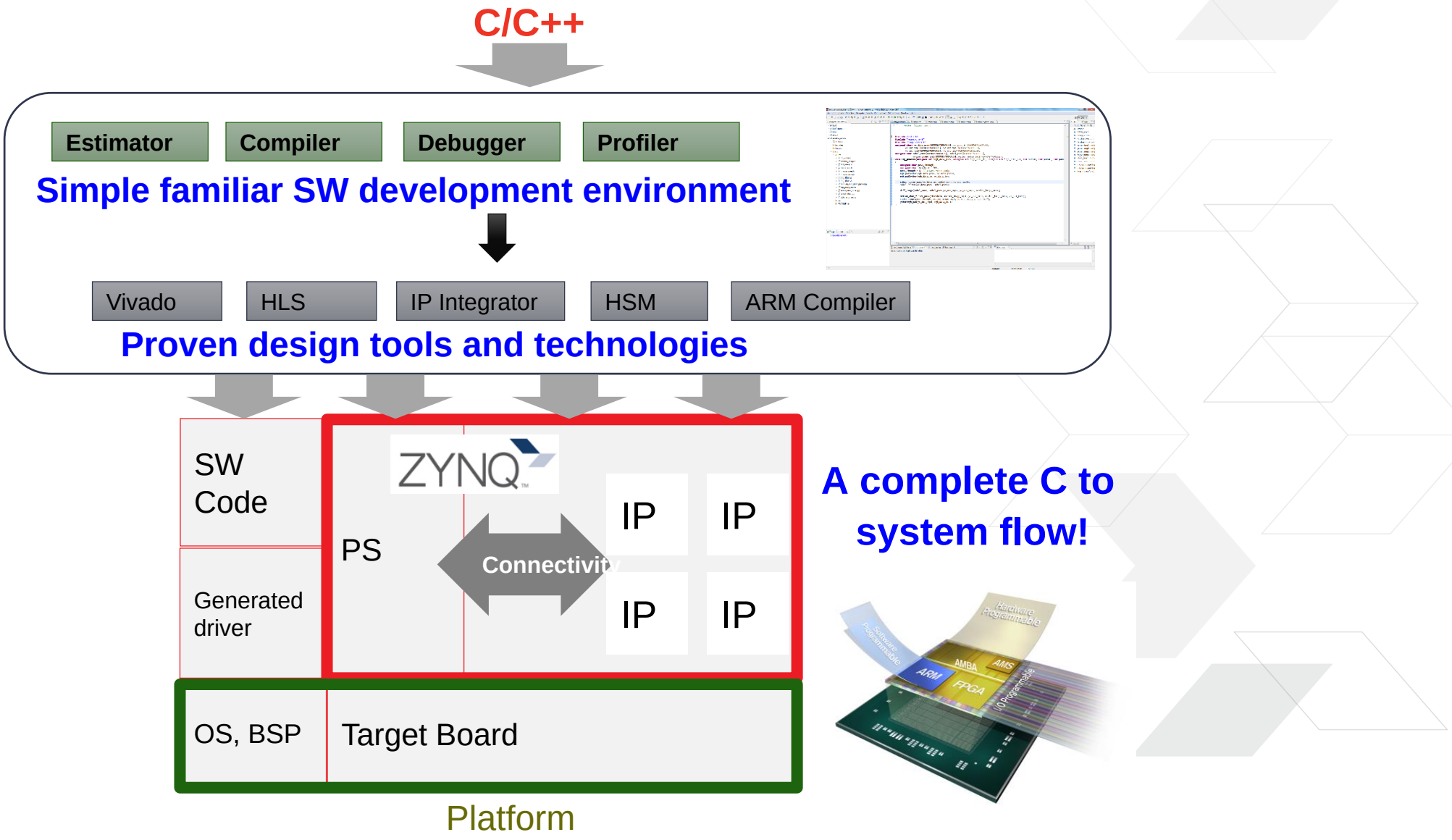


XILINX.

Development Flow



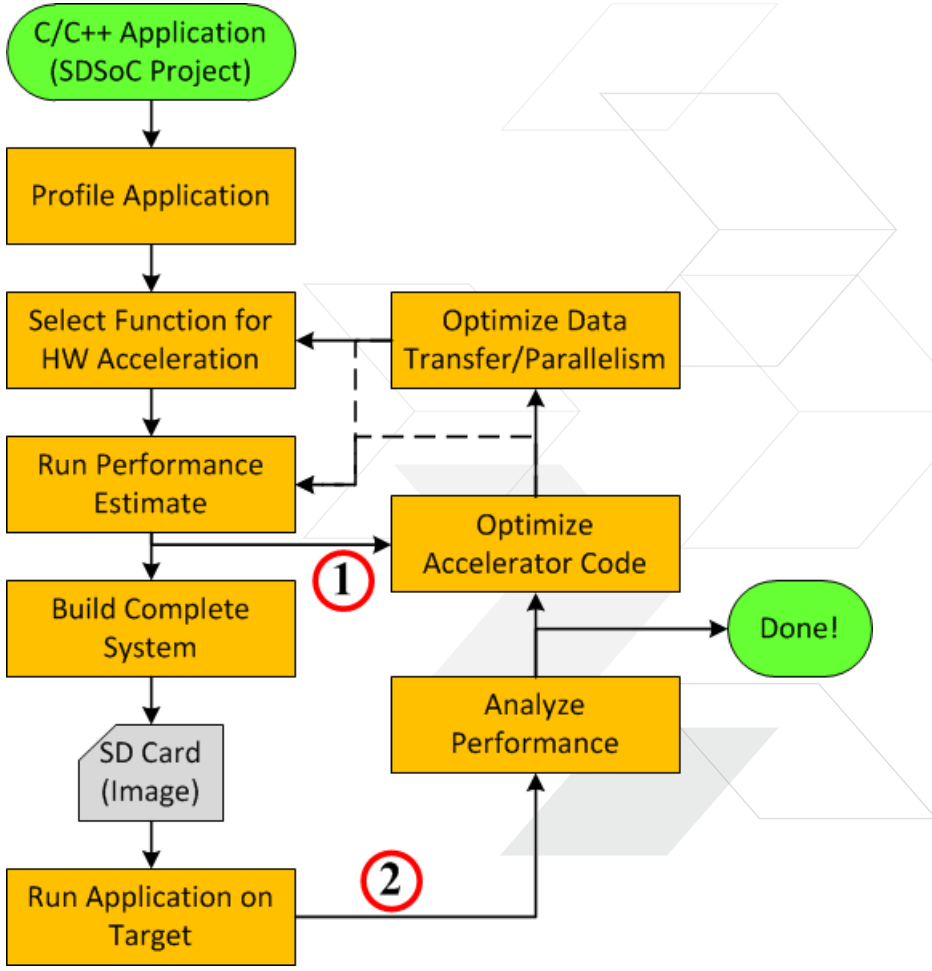
SDSoC: Complete End-to-End Flow



XILINX.

Development Flow

- **Without hardware generation, the SDSoC development environment acts like a normal C/C++ IDE**
- **With SDSoC it starts with a pure software system and ends with an accelerated SW/HW system**
  - Developers select C functions for hardware acceleration
- **Estimate provides quick-turns to get architectures close**
  - Used in early development
- **Release generates full hardware/software products**
  - For later and final stages of development



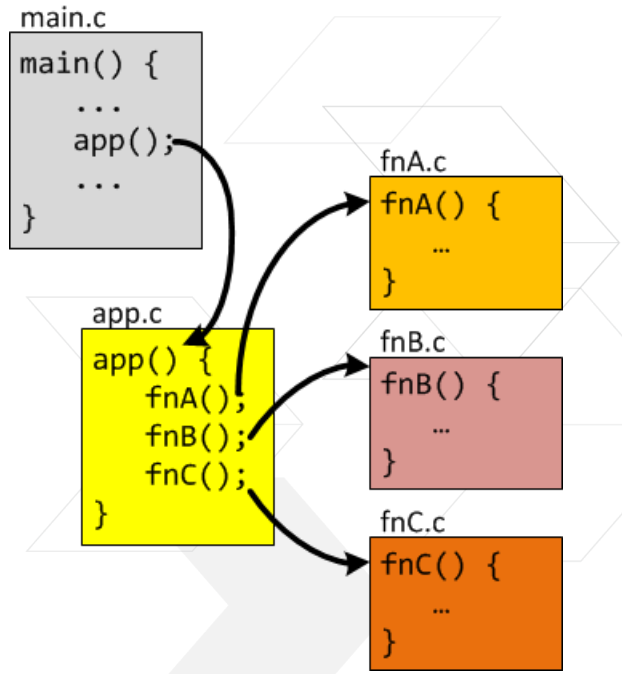
XILINX.

Profiling the Application

- **The goal of offloading software to hardware is to improve overall system performance**
- **Finding software eligible for acceleration follows general software optimization practices**
  - Profiling yields the necessary data by summarizing where execution time was spent
- **The SDSoC development environment includes the TCF profiling tool**
  - No code instrumentation or special recompiling required
  - Profiling results show percentage of time spent in functions relative to a total execution time
- **For absolute timing measurements *sds_lib* offers some functions for collecting system timer values**
  - Timer resource must be available and code must be changed to incorporate timer functions

Accelerating Software Functions

- **Individual functions must be marked for acceleration**
  - Functions selected for acceleration are "hardware functions“
    - Hardware functions can contain sub-functions
  - Refactoring required if critical points are not isolated as functions
  - Each function becomes an individual piece of IP
- **OS-based platforms have only one master thread communicates with all cores**
- **Restrictions**
  - Only one function per file scope
    - Splitting of files may be required to accelerate multiple functions
  - Call to a hardware function must reside in a different file scope than the hardware function
  - Other tool limitations may apply
    - See Vivado HLS tool C/C++ guidelines

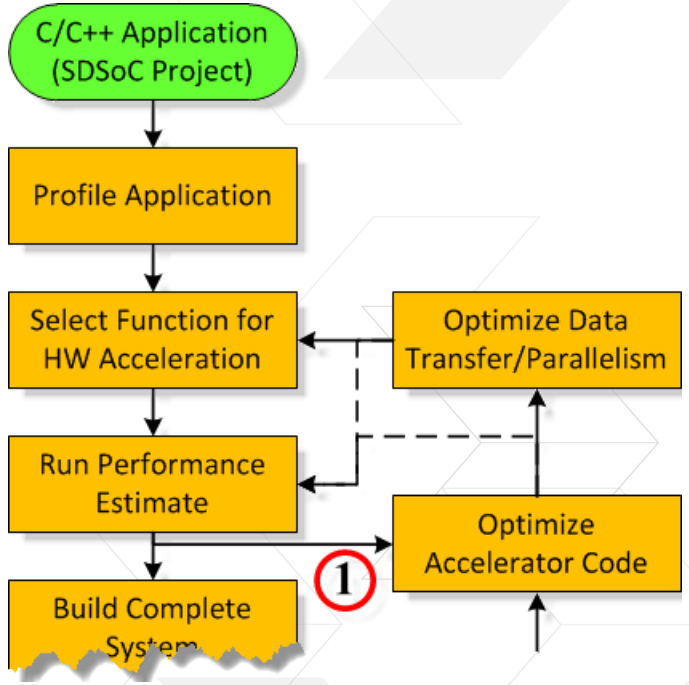


XILINX.

Estimating Performance

- **Enabling the Estimate Performance option builds a project only up to a point that performance estimates can be determined**
  - A complete Release or Debug build can take a **long** time
  - The Estimate Performance option shows the impact of moving functions to hardware in only a few minutes
- **Comparison of hardware-accelerated system with original software-only system is accomplished by producing two separate sets of data, one for each scenario**
  - The Vivado HLS tool is used to produce data for hardware estimates
  - Generated ELF must be run on the actual target to generate data for software estimates
  - IDE provides a Report viewer that compares and illustrates the data

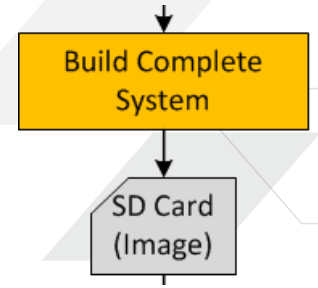
Initial Development Flow



- **Building with the Estimate Performance option yields enough information to fine-tune optimization**
- **First attempts often produce poorer performance than software alone**
  - Typical when code has not been refactored and/or annotated for acceleration
- **Recommendation: Initial design iterations should use the "short" path until performance improvements are seen**
  - Begin by optimizing code to improve accelerator performance, parallelism, and data flow
  - Rebuild to observe impact of changes on estimates
  - Repeat cycle until satisfactory results are achieved
- **Detailed system analysis and fine-tuning should be performed next**
  - Building and analyzing the full HW/SW system

Building the Complete System

- **Building a project is where SDSoC does all the heavy lifting**
  - Three basic methods to run the build process
    - Through the IDE, using a makefile, directly from the command line
  - It is during this step that SDSoC calls upon its internal framework and several sub-executables to accomplish
    - (1) system compilation, (2) system analysis, (3) system generation
- **Activate the Release build configuration for a fully optimized build**
- **A successful build will result in several output products including the following key items**
  - Application program in ELF format
  - Bitstream (optional)
  - SD card image (optional)
  - Design projects (e.g. Vivado Design Suite and Vivado HLS tool)

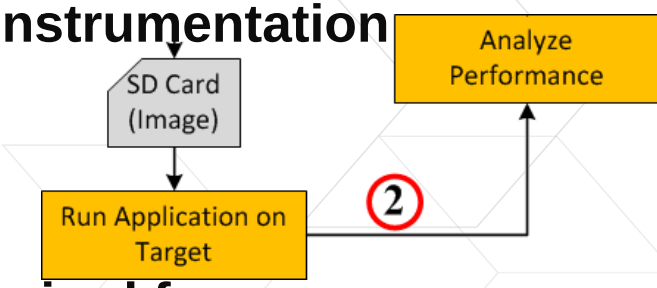


Running the Application

- **Standalone applications do not require but can use an SD card**
  - When SD card is used, the SSBL in the binary file is replaced by the application ELF
    - Boots directly to the application ELF
  - Applications can be run from the IDE using an SDSoC tool Run configuration
- **SDSoC development environment requires use of an SD card to run applications for Linux**
  - Files required to run the application on hardware are grouped together as an SD card image
    - First stage bootloader (FSBL), bitstream, and second stage bootloader (SSBL) embedded in a single binary file (BOOT.BIN)
    - Linux kernel
    - Application ELF
    - Other miscellaneous files
  - ELF must be explicitly run from the mounted SD card directory

Analyzing the Built System

- **The goal of this step is to analyze the performance of the updated hardware/software system**
- **The same tools and techniques used initially apply here: profiling, instrumentation**
  - Profiler tool for relative software execution times
  - Time-stamping functions provided through SDSoC API (sds_lib)



- **System consists of hardware and software with additional tools required for complete analysis**
  - Vivado HLS tool provides reports detailing generated hardware (included with output products)
  - Vivado Design Suite can open the generated system hardware design to review additional reports, modify hardware
  - SDSoC tool compiler can optionally insert an AXI performance monitor (APM) into the generated system
    - APM monitors dataflow between the PS-PL

Optimizing the System

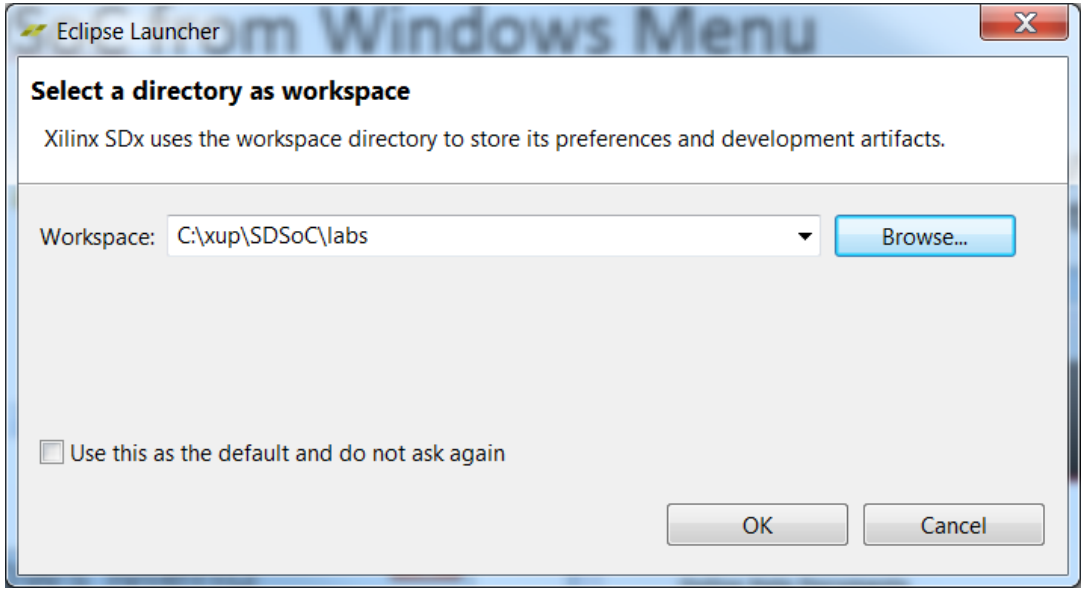
- **Optimizing the system is broken down into two steps: accelerator and then data flow and parallelism**
- **Optimizing the accelerator consists of refining and/or annotating the hardware function (micro-architecture)**
  - Re-factoring code to be more HLS-friendly
  - Adding code annotations (i.e., pragmas) that HLS can interpret and use to better optimize the accelerator
- **For system-level optimizations (data-flow/parallelism) there are several options (macro-architecture) including**
  - Using physically contiguous memory
  - Using shared memory
  - Using multiple instances of the same hardware
  - Overriding HLS default behavior

SDSoC Project Creation

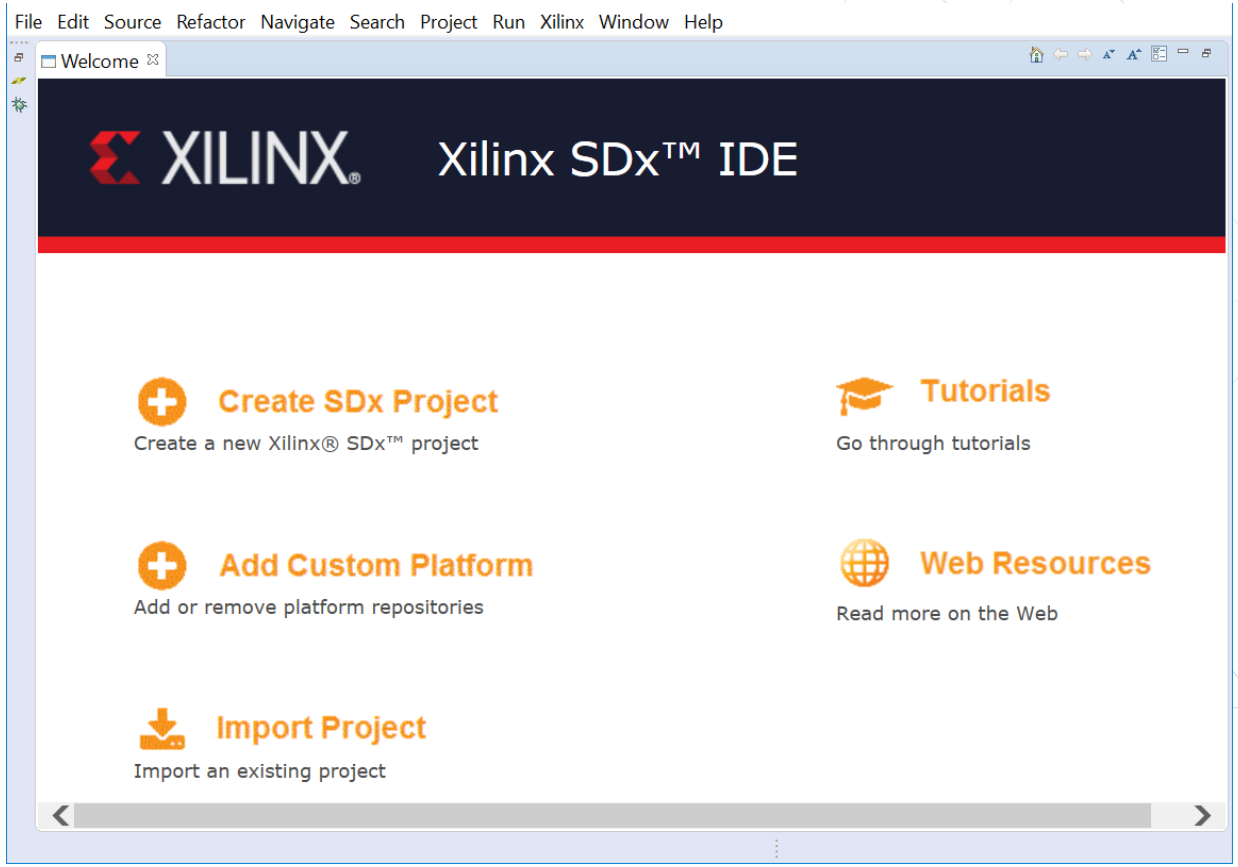
Invoke SDx from Windows Menu

1. Start SDx

2. Select a workspace



3. If workspace empty then Welcome screen appears otherwise projects stored in the workspace are displayed



XILINX

Creating, Importing, and Opening Projects

- **You can create a new project in the selected workspace either by clicking on the Create SDx Project link or by selecting File > New > Xilinx SDx Project**
  - Select project type
    - Application
    - Platform
  - Identify project name
  - Select target platform
  - Select target OS
  - Select either available template or an empty project option
- **You can import an existing project**
  - Select archive file if the project is in an archived form
  - Identify path if the project is saved in a hierarchical directory structure

Project Definition

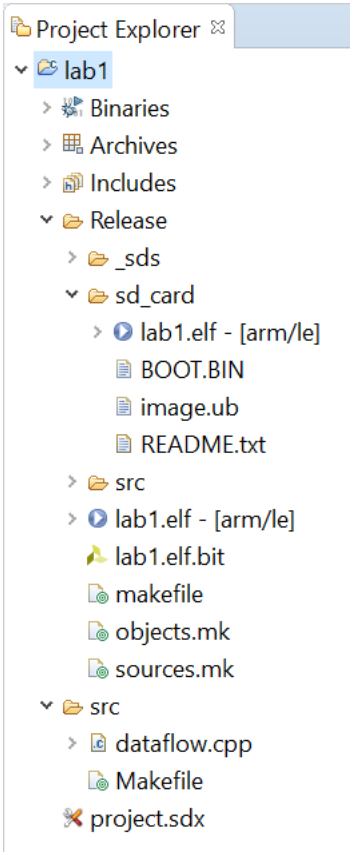
- **An SDSoC project is a folder in a workspace that contains all project files and tool metafiles**
- **SDSoC projects carry some additional properties beyond these general Eclipse-based concepts**
  - Associated with a specific SDSoC development environment platform
  - Built for a specific type of operating system (OS)
- **Built-in OS support: Linux (default), Standalone (a.k.a bare-metal), and FreeRTOS**

Operating System Support

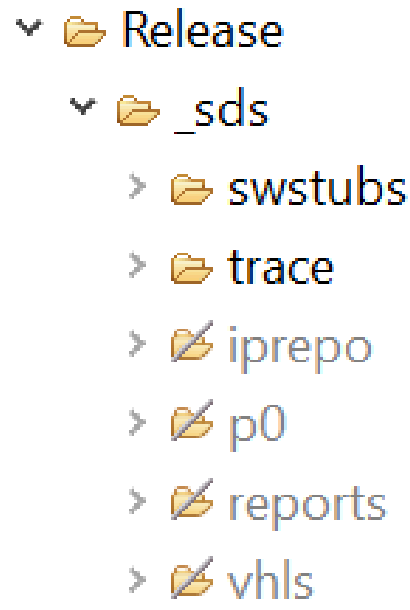
- **The output products of an SDSoC project vary depending on the targeted OS**
  - For standalone projects
    - *swstubs* folder will include the Xilinx standalone library (*libxil*)
  - For Linux projects
    - SD card image will include a pre-compiled Linux kernel and root file system
    - Pre-compiled kernel contains drivers to communicate with accelerators
  - For FreeRTOS projects
    - Projects will link to a FreeRTOS library supplied by the SDSoC development environment as an extension to the GNU toolchain

Project Hierarchy

- **Project Explorer View**
  - The root folder of a project hierarchy is named after the project itself (*lab1*)
  - Three virtual branches are provided: Binaries, Archives, Includes
    - These list executables, libraries, and header files found in or used by the project
  - Other branches represent actual folders on the file system: build outputs and source files
  - The *src* folder holds source files
  - The final element at root level is the actual SDSoC project file: *project.sdx*

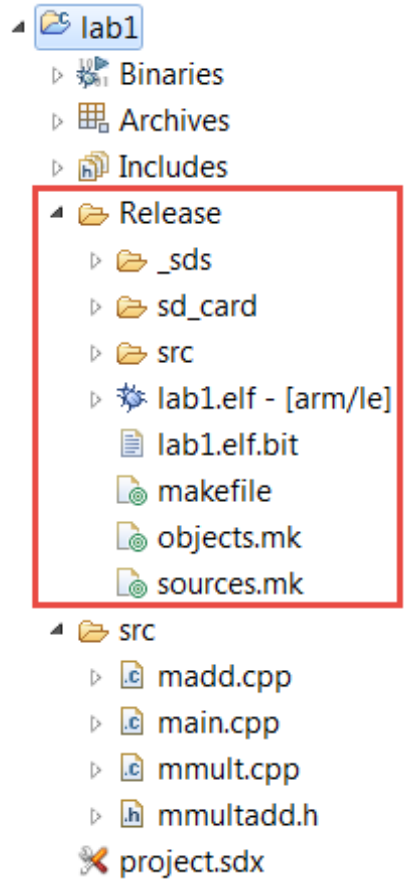


- **Windows Explorer View**
  - Hosts real files and directories



Project Hierarchy (2)

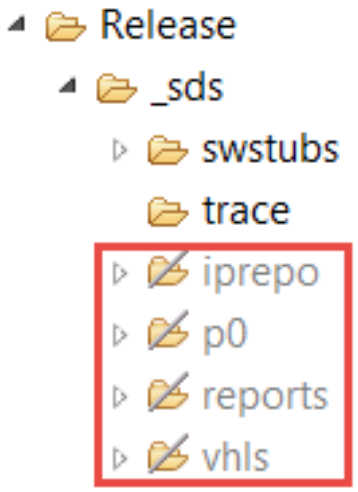
- **Common to every build folder are several important components**
  - The compiled application executable is placed at the root of the build folder (*lab1.elf*)
  - Several auto-generated make files used by the IDE to initiate the build process
  - The *_sds* folder is a catch all for everything else
    - Reports, files generated by back-end tools (e.g. Vivado Design Suite project), software stubs, etc.
  - An *sd_card* folder that contains the SD card image
    - Not strictly an image but rather a collection of files and folders
  - A *src* folder that contains compiled source files (i.e., object files)



XILINX.

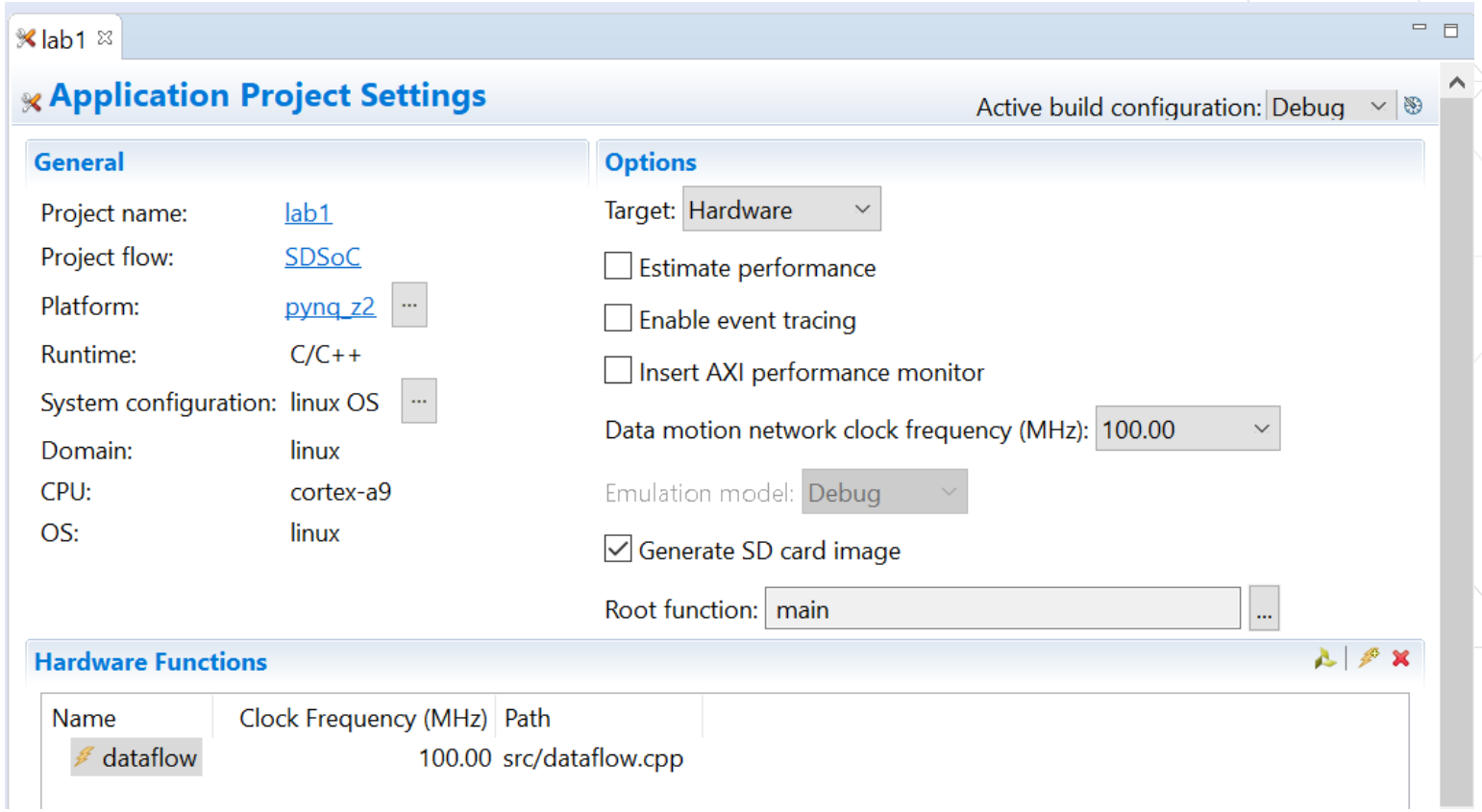
Project Hierarchy (3)

- **The _*sds* folder contains several sub-folders**
  - *swstubs*
    - Software files and stubs generated and supplied by the SDSoC development environment
    - Some original source code modified to interface with accelerators
  - *iprepo*
    - Repository of cores generated by the Vivado HLS tool (i.e., accelerators)
  - *p0*
    - Vivado IPI project and related files, can invoke Vivado by double-clicking on the *.xpr entry
  - *reports*
    - Various reports including data motion network
  - *vhls*
    - Vivado hls project files for each of the targeted accelerators
- **Grayed out folders indicate output products generated by independent back-end tools**



SDx Project Settings View

- Select functions which are hardware target
- Select Data Motion network clock frequency
- Other options



XILINX.

Summary

Summary

- **The SDSoC development environment provides a familiar embedded C/C++ application development experience**
  - It provides infrastructure to combine processing system, accelerators, data movers, signaling, and drivers
- **Benefits of using SDSoC include**
  - Shorter development cycles
  - Simplified interface and partitioning between hardware and software
- **SDSoC uses Vivado HLS, Vivado IPI, SDK tools**
- **SDSoC project creation involves identifying workspace, entering project name, selecting target platform, selecting OS, and either selecting pre-defined application templates or creating empty application**

Lab1 Intro

Lab1 Intro

- **Introduction**
  - This lab guides you through the process of using SDSoC to create a new project using available templates, mark a function for hardware implementation, build a hardware implemented design, and run the project either on ZedBoard or Zybo board
- **Objectives**
  - Create a new SDSoC environment project for your application from a number of available platforms and project templates
  - Mark a function for hardware implementation
  - Build your project to generate a bitstream containing the hardware implemented function and an executable file that invokes this hardware implemented function
  - Test the design in hardware

Adaptable.
Intelligent.

XILINX