Estimation and Events Tracing

SDx 2018.2

XILINX.

Objectives

- **After completing this module, you will be able to:**
  - Describe what estimation is in SDSoC and how it works
  - Distinguish between profiling and estimation
  - Evaluate estimation reports for hardware acceleration effectiveness
  - Understand various events, including synchronization, taking place when accelerators are in the design

Outline

- Introduction
- Estimation in SDSoC
- Accelerator events tracing
- Summary
- Lab4 Intro

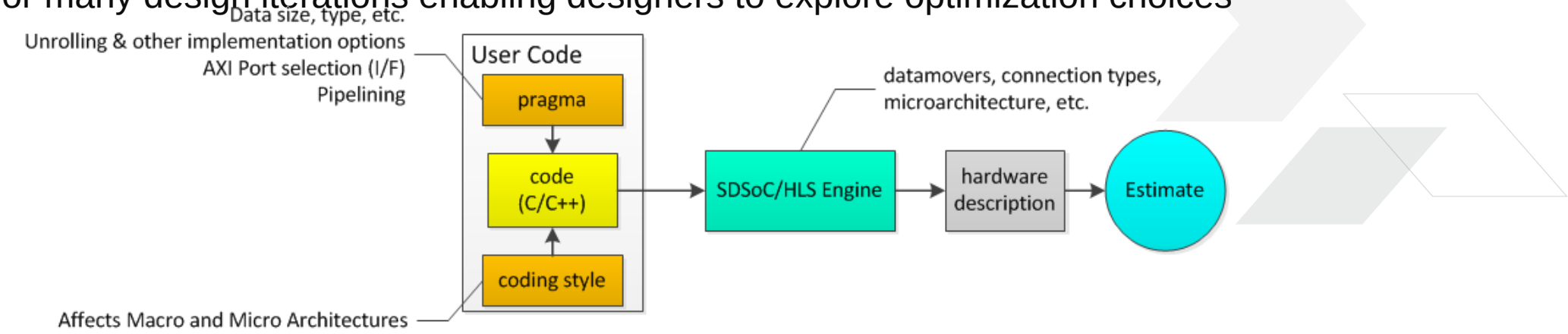
What is Performance Estimation?

- **Performance estimation is a faster way to determine system speed up by targeting functions in hardware**
  - Quick turn around as it does not require full bitstream generation
  - Provides more design iterations to explore various accelerator optimizations
  - Uses Estimate Performance option with Debug configuration
    - Release and Debug configurations (without Estimate option enabled) require full bitstream generation which would be time consuming
- **Restrictions**
  - Performance estimation flow is not supported in the presence of asynchronous calls

Estimation in SDSoC

How Does the Estimation Flow Work?

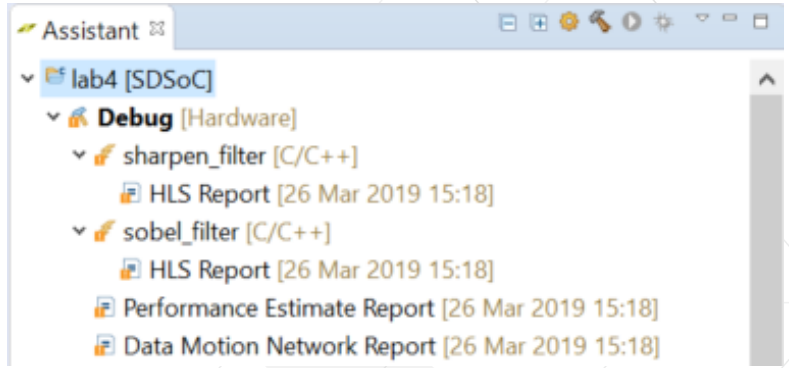
- **SDSoC development environment uses HLS engine**
- **HLS engine converts selected C/C++ functions into hardware *fragments***
  - Each fragment has approximated performance and resource usage
- **Software performance determined by running on hardware**
  - Hardware performance estimated from HLS reports
- **Since a final bitstream (with accelerators) is not required, this is a fairly quick process**
  - Allows for many design iterations enabling designers to explore optimization choices



XILINX.

Under the Hood Estimation

- **Estimation performs a number of tasks**
  - Pulls the function marked for hardware from the code and replaces it with HW configuration functions
  - Passes software function to the Vivado HLS tool and compiles it
    - Each accelerator has its own project (defined by its name)
    - Vivado HLS tool results found under *Assistant* > *labx* > *Debug*
    - Synthesis and implementation files are under each accelerator
    - Report showing resources and timing estimates found

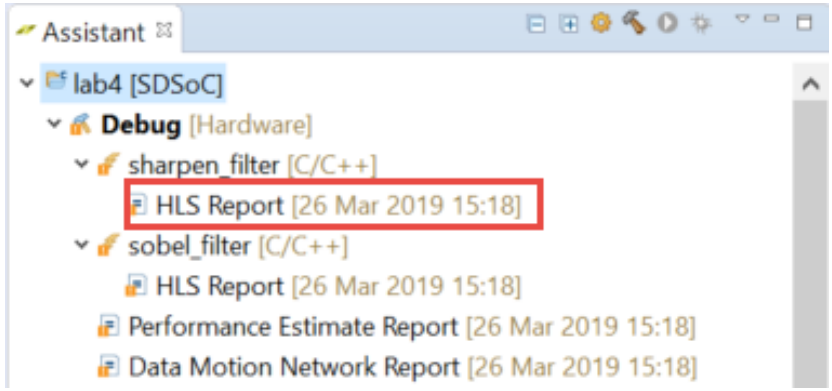


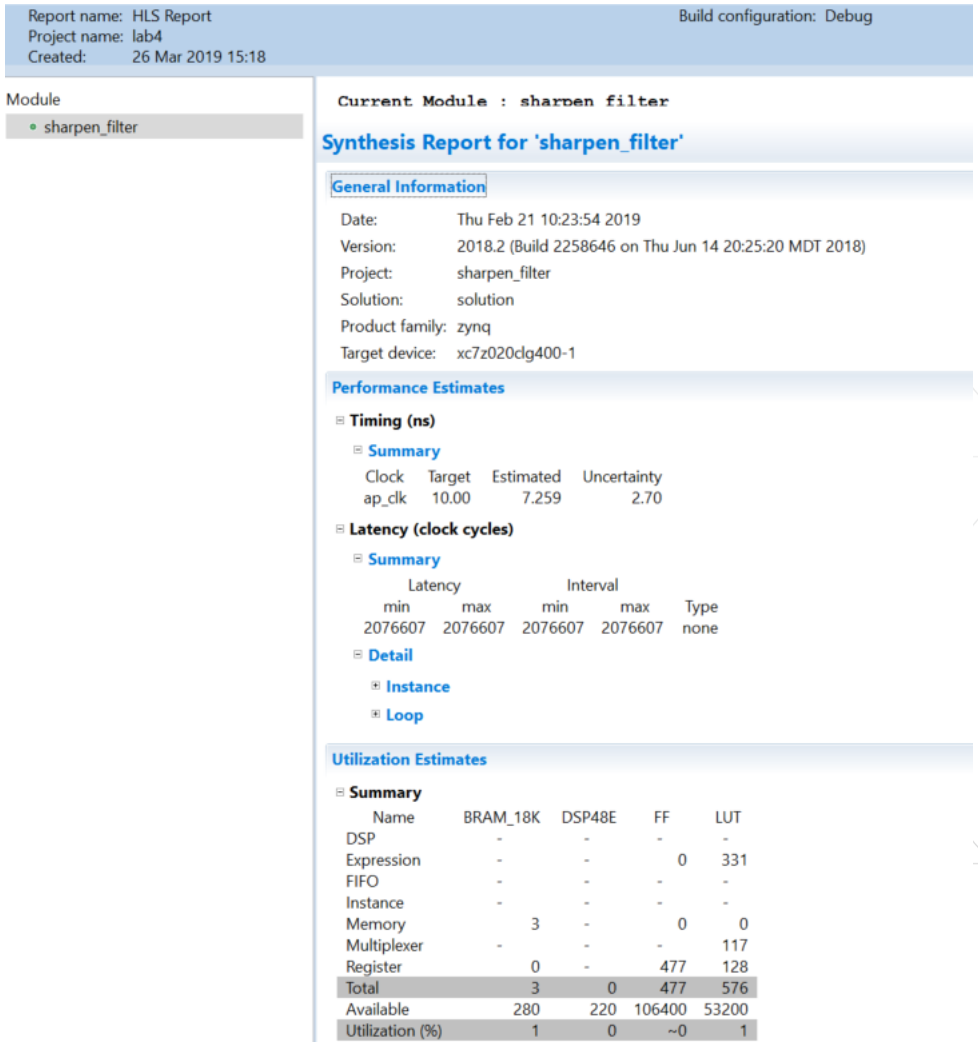
- **Software performance and comparative performances measured on processor**
- **Resource and timing estimates reports provided in an easy-to-read report**

XILINX.

GUI Integration of HLS Reports

- **HLS report can be accessed in the SDSoC GUI**
  - Synthesis
    - Performance estimates
    - Utilization estimates
    - Interface summary





XILINX.

Scope of Overall Speedup

- **In the Performance Estimation Report's first line shows the estimated speedup for the top-level function (referred to as perf root)**
  - This function is set to "main" by default
  - However, there might be code that you would like to exclude from this comparison, for example allocating buffers, initialization and setup
  - If you wish to see the overall speedup when considering some other function, you can do this by specifying a different function as the root for performance estimation flow
  - The flow works with the assumption that all functions selected for hardware acceleration are children of the root

Performance Estimation Flow with Linux

- **Similar to Standalone OS based flow except**
  - Select Linux as the target OS while creating the project
- **Select the functions you want to target**
- **Set Estimate option in Debug build configuration**
- **Build project**
- **Copy the contents of the `sd_card` folder under Estimate to an sd card and boot up the board**
  - Ensure that the board is connected to an Ethernet router using an Ethernet cable
  - Ensure that a serial terminal is connected
- **Power on the board and notice the leased IP address in the Linux boot log**

Performance Estimation Flow with Linux (2)

- **If you don’t see the bootup assigning any ip address then do the following**
  - Execute `ifconfig` command in the Linux console and see if any ip address is assigned to eth0
  - If not assigned then execute the following command

    ```
    ifconfig eth0 192.168.0.10
    ```
  - Make sure that the PC has the ip address assigned as 192.168.0.1
- **In the SDSoC environment click on the Estimate Performance Speedup link**
- **Click on the *Click Here* link of the SDSoC Report Viewer**
- **Click on the New button and set up the IP address (192.168.0.10) and the port (1534)**
- **Click OK which will execute the application on the board and the speedup results will be displayed**

After Estimation

- **If you have achieved the desired performance**
  - Build the application for the Release configuration
  - Copy the generated SD card image to an SD card if required
    - Required for Linux, optional for Standalone
  - Run the application on the board to obtain the actual performance of the generated system
- **If you have not achieved the desired performance**
  - Review the reports to identify what is slowing down
  - Use SDSoC tool and/or HLS pragmas to improve performance
  - Review your coding style
  - Leverage more efficient libraries if available
  - Repeat the above steps until you get what you need

Differences Between Profiling and Estimation

- **Profiling is used to determine where CPU spends most of its cycles and the order of functions being called**
  - Results identify software "bottlenecks“
  - It is not helpful for identifying accelerator performance improvement
    - Profiling generally does not show actual performance
- **Determining performance by instrumenting in the Release/Debug configurations is time consuming**
  - Involves bitstream generation for complete hardware, including hardware accelerators
    - Time consuming task
    - Restricts design exploration
- **Estimation executes software on hardware**
  - However, for hardware accelerator performance estimate it uses HLS reports, eliminating actual need of generating full bitstream

Accelerator Events Tracing

Tracing Feature

- **Tracing provides visibility into the duration of “higher level events” during the execution of a program, with finer granularity than overall run time**
  - Accelerator functions
  - Data transfers between accelerators and PS
- **Tracing helps debug execution, showing what happened when, and how much data was transferred**
- **Trace points are application-specific, depending on accelerators**
- **Tracing is intrusive, but minimize its impact on execution time and PL area**

Tracing Capabilities

- **Tracing is restricted to calls into hardware functions**
  - Must have at least 1 function marked for hardware
- **Automatic hardware and software instrumentation**
  - Accelerator control code (stub code) in software
  - State of standard HLS-produced hardware accelerator (start/stop)
  - State of AXI-Stream interfaces into/out-of accelerator (start/stop)
- **Must be connected to a board to run SDSoC tracing**
  - Board connected to host PC (JTAG/USB only for standalone, both JTAG/USB and TCP/TCF for Linux)
- **Post processing-like action**
  - First run application and buffer up traces in PL, then program finishes, finally read out traces

Stub Code for the Accelerators

> Example: Matrix Multiplication and Add

Original Code

main function

```
int main(int argc, char* argv[]) {
   float *A, *B, *C, *D, tmp1;
   init(A, B, C, D);

   mmult(A, B, tmp1);
   madd(tmp1, C, D);

   check(D);
}
```

mmult function

```
void mmult(float *A, float *B, float *C) {
   for (int a=0; a<A_NROWS; a++)
      for (int b=0; b<B_NCOLS; b++) {
         float result = 0;
         for (int c=0; c<A_NCOLS; c++)
            result += A[a][c]*B[c][b];
         C[a][b] = result;
      }
}
```

madd function

```
void madd(float *A, float *B, float *C) {
   for (int i=0; i<NROWS; i++)
      for (int j=0; j<NCOLS; j++)
         C[i][j] = A[i][j] + B[i][j];
}
```

Stub Code

main function

```
int main(int argc, char* argv[]) {
   float *A, *B, *C, *D, tmp1;
   init(A, B, C, D);

   _p0_mmult_0(A, B, tmp1);
   _p0_madd_0(tmp1, C, D);

   check(D);
}
```

mmult function

```
void _p0_mmult_0(float *A, float *B, float *C) {
   cf_send_cmd();

   cf_send_i(req2, A);
   cf_send_i(req3, B);
}
```

madd function

```
void _p0_madd_0(float *A, float *B, float *C) {
   cf_send_cmd();

   cf_send_i(req0, B);
   cf_receive_i(req1, C);

   cf_wait(req0);
   cf_wait(req1);
   cf_wait(req2);
   cf_wait(req3);
}
```

Tracing Capabilities (Stub Tracing)

Without Trace

With Trace

main function

```
int main(int argc, char* argv[]) {
   float *A, *B, *C, *D, tmp1;
   init(A, B, C, D);

   _p0_mmult_0(A, B, tmp1);
   _p0_madd_0(tmp1, C, D);

   check(D);
}
```

```
int main(int argc, char* argv[]) {
   float *A, *B, *C, *D, tmp1;
   init(A, B, C, D);

   _p0_mmult_0(A, B, tmp1);
   _p0_madd_0(tmp1, C, D);

   check(D);
}
```

No Change

mmult function

```
void _p0_mmult_0(float *A, float *B, float *C) {
   cf_send_cmd();

   cf_send_i(req2, A);
   cf_send_i(req3, B);
}
```

```
void _p0_mmult_0(float *A, float *B, float *C) {
   sds_trace(EVENT_START);
   cf_send_cmd();
   sds_trace(EVENT_STOP);

   sds_trace(EVENT_START);
   cf_send_i(req2, A);
   sds_trace(EVENT_STOP);
   sds_trace(EVENT_START);
   cf_send_i(req3, B);
   sds_trace(EVENT_STOP);
}
```

madd function

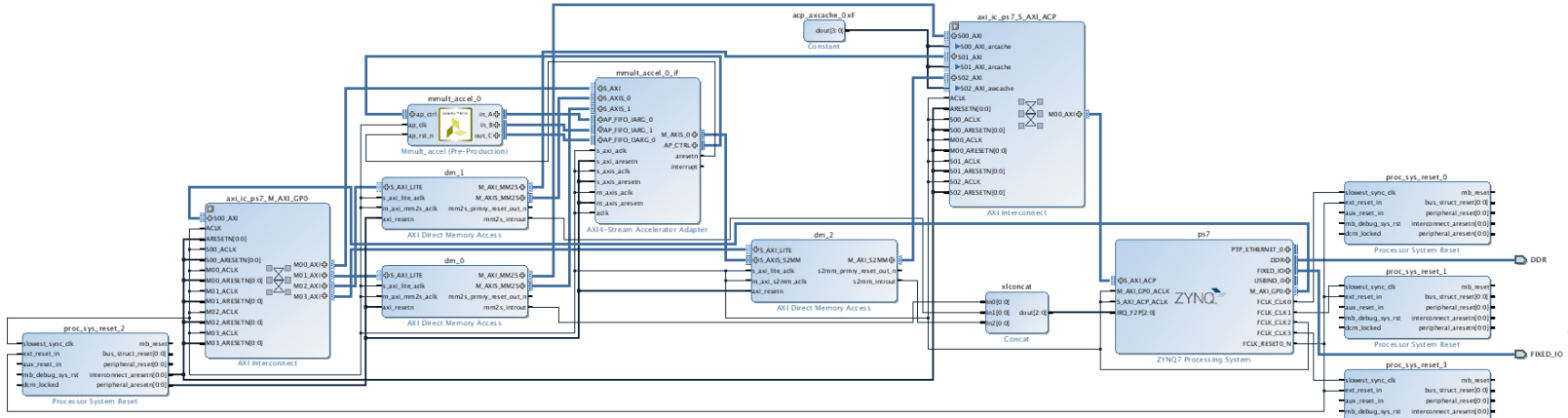
```
void _p0_madd_0(float *A, float *B, float *C) {
   cf_send_cmd();

   cf_send_i(req0, B);
   cf_reveive_i(req1, C);

   cf_wait(req0);
   cf_wait(req1);
   cf_wait(req2);
   cf_wait(req3);
}
```

```
void _p0_madd_0(float *A, float *B, float *C) {
   sds_trace(EVENT_START);
   cf_send_cmd();
   sds_trace(EVENT_STOP);

   sds_trace(EVENT_START);
   cf_send_i(req0, B);
   sds_trace(EVENT_STOP);
   sds_trace(EVENT_START);
   cf_reveive_i(req1, C);
   sds_trace(EVENT_STOP);

   sds_trace(EVENT_START);
   cf_wait(req0);
   sds_trace(EVENT_STOP);
   sds_trace(EVENT_START);
   cf_wait(req1);
   sds_trace(EVENT_STOP);
   sds_trace(EVENT_START);
   cf_wait(req2);
   sds_trace(EVENT_STOP);
   sds_trace(EVENT_START);
   cf_wait(req3);
   sds_trace(EVENT_STOP);
}
```

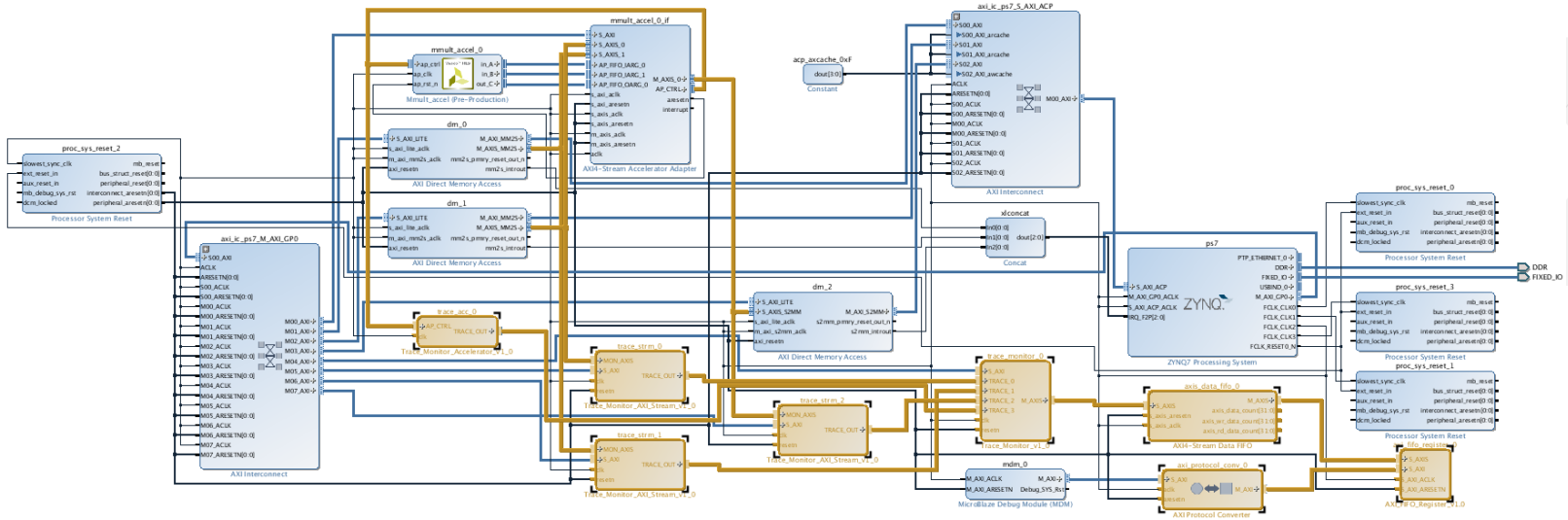

XILINX.

Tracing Capabilities (Hardware Tracing)

Without Trace

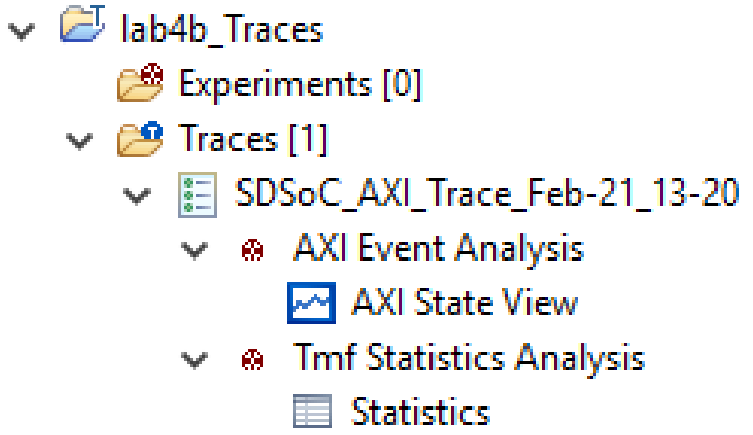


With Trace



Trace Flow

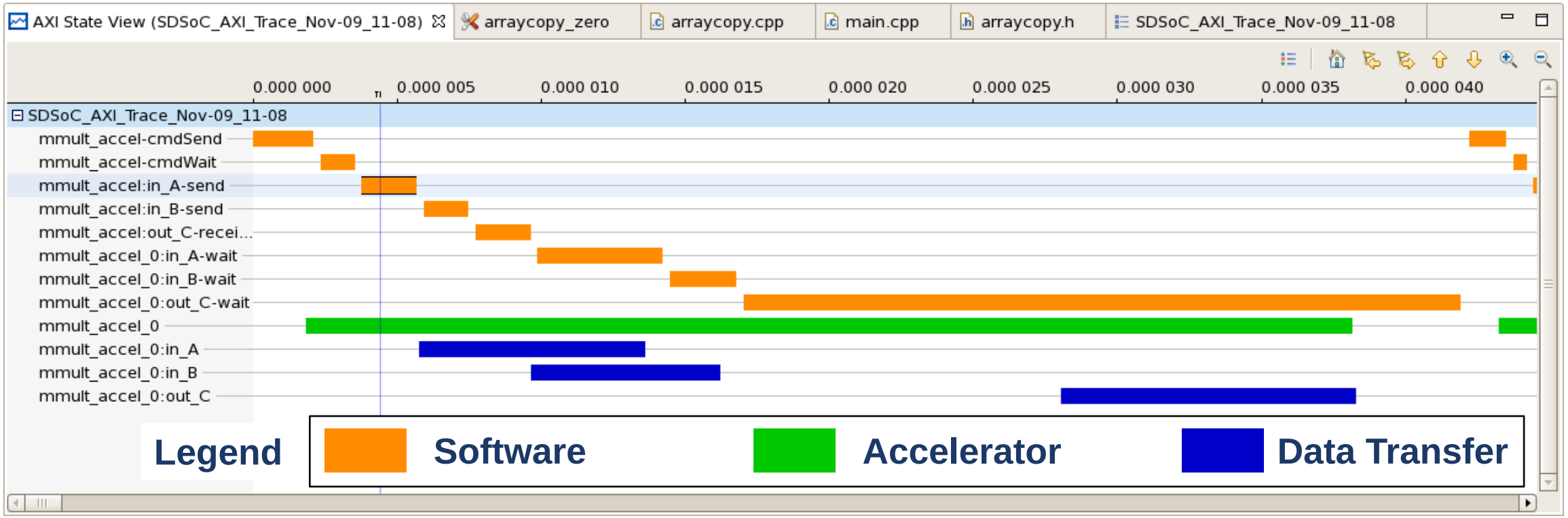
- **Enable Event tracing feature in SDx Project Settings**
- **Build the design**
- **Run the application**
  - `Run As > Trace Application (SDSoC Debugger)`
- **View the AXI State to analyze the application flow**



XILINX.

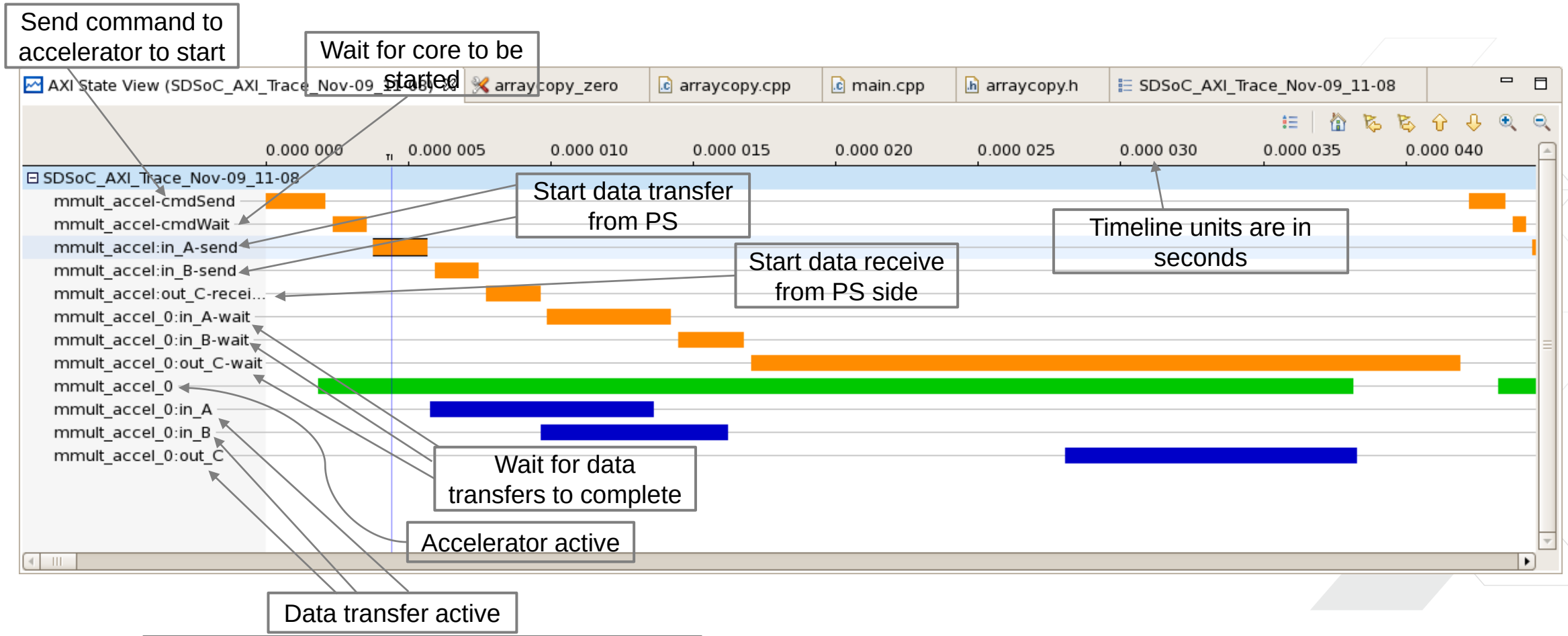
Trace Flow

- AXI State



© Copyright 2018 Xilinx

Trace Flow

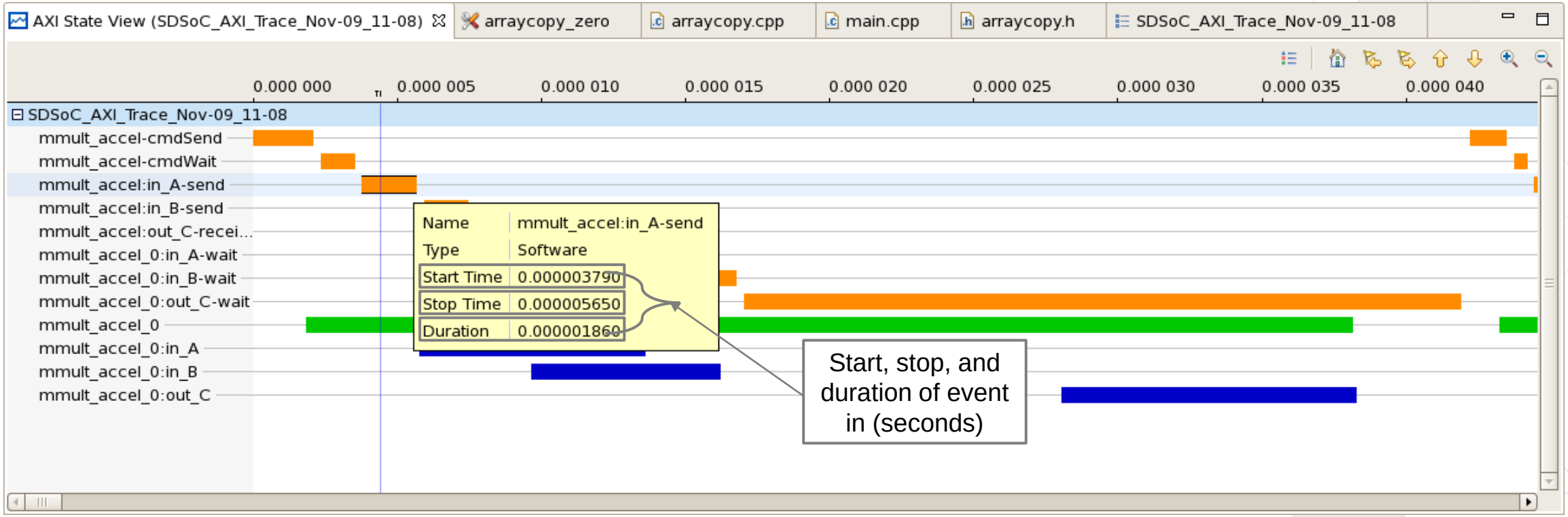


Legend SW Acc Tx

XILINX.

Trace Flow

> **Hover the mouse over each trace event to see more detailed information**



XILINX.

Summary

Summary

- **The performance estimation flow reports estimated performance improvement**
  - For *root* system
  - For individual hardware functions
- **SDSoC tool and Vivado HLS tool pragmas need to be used efficiently to improve accelerator performance**
- **SDSoC tool and HLS coding guidelines must be followed**
- **Performance estimation flow is not supported in the presence of asynchronous calls (using asynchronous pragmas, sds_wait, etc.)**
- **Profiling of an application which has accelerated functions requires full bitstream generation whereas the Estimation option in the Debug configuration uses HLS generated reports, reducing design exploration cycle time**
- **Event tracing provides insight into how various events are taking place and the relative time spent in data movement and data processing**

Lab4 Intro

Lab4 Intro

- **Introduction**
  - This lab guides you through the steps involved in estimating the expected performance of an application when functions are targeted in hardware, without going through the entire build cycle. You will further analyze how data movement is taking place by inserting events tracker
- **Objectives**
  - Use the SDSoC environment to obtain an estimate of the speedup that you can expect from your selection of functions to accelerate
  - Differentiate between the flows targeting Standalone OS and Linux OS
  - Track various events taking place with respect to hardware accelerators

Adaptable.
Intelligent.

XILINX