ACCELERATED KNN
ZYNQ-7000 SOC

E5ADD, Fall 2020

Janus Bo Andersen

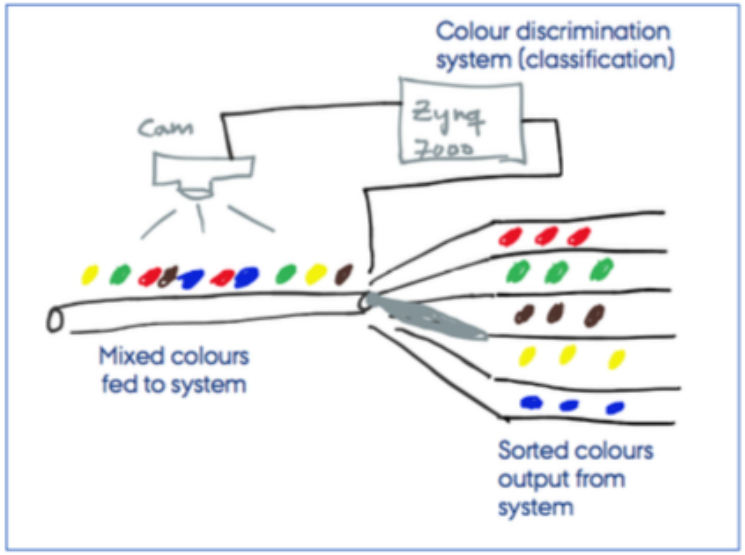
PROJECT

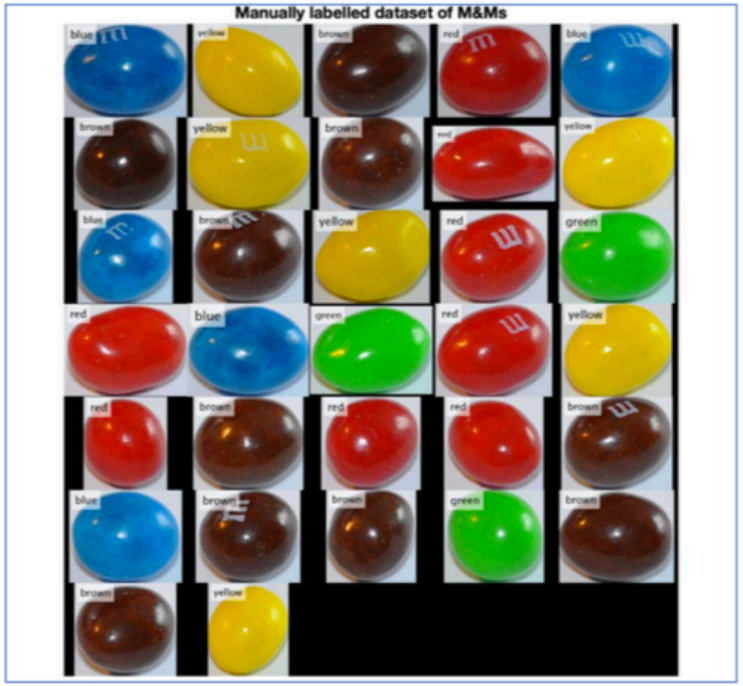
We implemented a **HW-accelerated ML algorithm** on the Zynq-7000 SoC:

- **Co-design**, partitioning:
  - Computationally intensive and parallelizable ML -> PL.
  - UI and less-intensive -> PS.
  - Fast data motion between PS <-> PL
- Using k-nearest neighbours (KNN)
  - Common, robust and generally performs well. No training, just set the hyper-param. *k*.
  - *Relatively* simple to code and analyse, and to experiment with available resources on MiniZed.

Problem is **colour classification** (recognition) of M&Ms

- For each classification: Sampling RGB-values from 21 pixels of an unknown M&M -> 63 features.
  - Opted to keep this feature dim. to
    - classifier with computational complexity, robust to noise. Other approaches also viable (dim. reduction, averaging, etc.).
    - algo that easily generalises to other classification problems.
- Problem size:
  - 100 unknown M&Ms (query vectors) using 1024 reference M&Ms (reference vectors).
  - Each vector has 63 features (query set is 100 x 63 matrix, reference set is 1024 x 63).





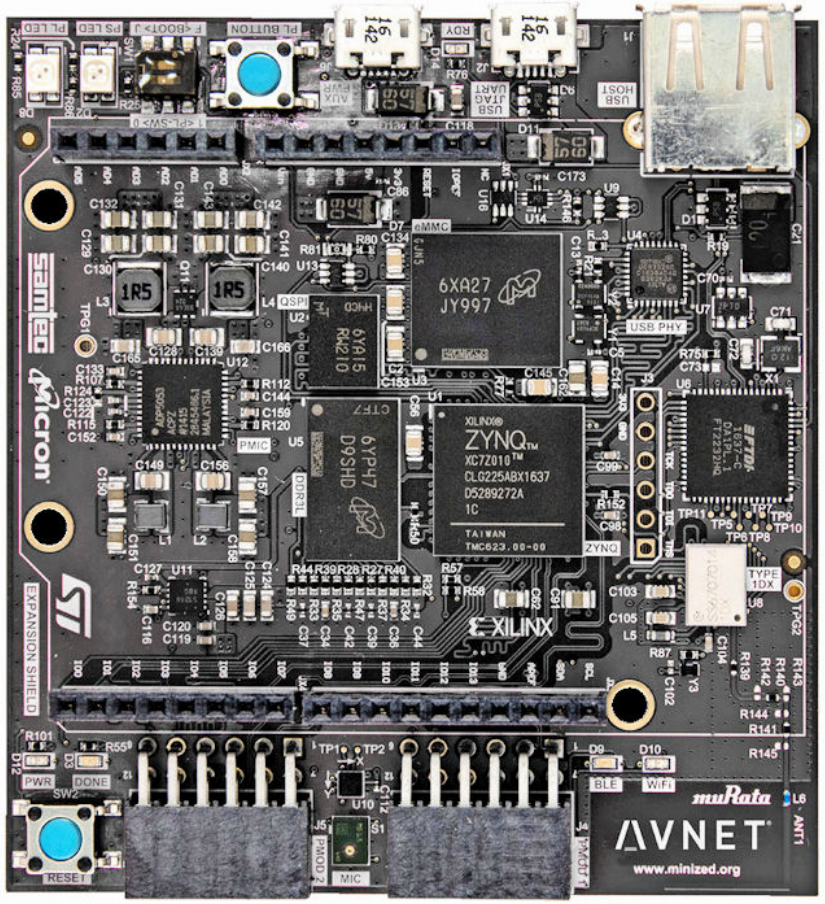
LEARNING: COVERING E5ADD CURRICULUM DURING PROJECT

HW theory:

- Case for SoC ✓
- Zynq processor ✓
- Peripherals ✓
- Merging PS and PL, AXI ✓
- Zynq DMA Controller ✓
- Creating custom IP ✓
- TCL ✓

SW theory:

- Zynq system architecture ✓
- SDK ✓
- BSP ✓
- Developing applications ✓
- FSBL ✓
- Flash-programming, boot ✓
- Interrupts



Design and development:

- System design on SoC / FPGA platform ✓
- Co-design ✓
- Xilinx tool-chain familiarity for development, debugging ✓

https://kursuskatalog.au.dk/da/course/101290/E5ADD-Avanceret-Digital-Design

GETTING STARTED

Big part of this project was to determine which Xilinx tools to use and what version.

The course introduced 2017.1/2019.1 with Vivado + SDK.

- SDx (SDSoC) was needed to implement accelerated designs.

Version 2019.2 introduced a replacement Vitis for SDK and SDx.

- Vitis combines the SDK and SDx to one package.
- Vitis AI

TWO MEMBERS. TWO EXPLORERS

To conquer both worlds(Vitis vs SDK) each member dived into different versions to explore.

The goal was to find a suitable solution for this project, which was achievable with the small to none knowledge known beforehand.

XILINX VITIS™

VS

KEY RESULTS

Performance

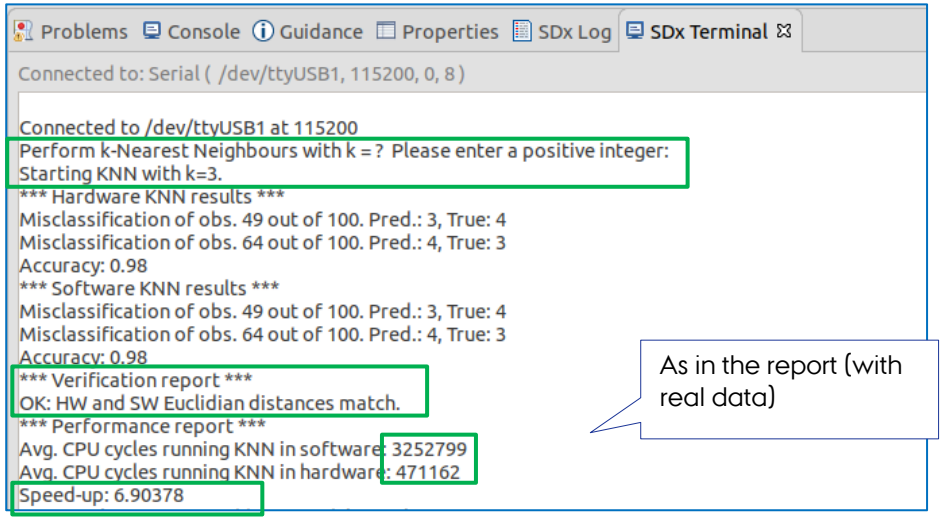
- HW-accelerated KNN is
  - ~6.9-7.4x faster than SW version (debuggable)
  - ~0.8x versus most optimized SW version (-O3)
- Significant speed-up considering:
  - Arm Cortex-A9 clocked at 667 MHz
    - A9 is capable of SIMD and DSP instructions
  - FPGA 'only' at 100 MHz (theoretically up to 250 MHz – hard to develop)

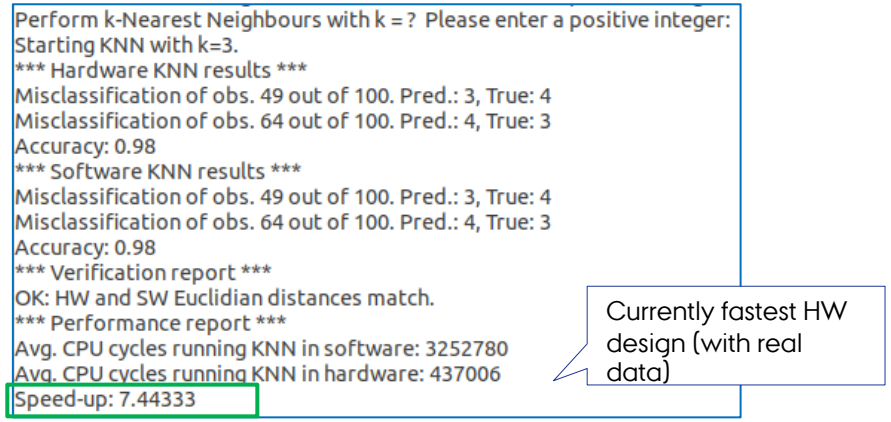
Correctness

- HW has same accuracy as SW-implem. & benchmark Matlab implementation.

Usability

- User can choose hyperparam. *K* (sort/search in PS)
- System deployed to device flash mem
  - -> PL programmed on device boot, device does not need to be tethered.





PRESENTATION GOALS (JANUS)

1. Summarise the project, key results and process -> joint presentation

2. **Elaborate** on the report:
   - Add more insight into the realised system from a HW point of view
   - Visualise the HW impact of some of the key design decisions

3. Open the discussion on **co-design** and development methods/tools:
   - Design-process and co-design in this project
   - DSE experience using the Xilinx toolchain
   - When to use SDS/HLS for rapid proto-typing

4. Summarise the **conclusion** and **key learning**
   - Ideas for improvement / **future work**

5. Question time

SYSTEM OVERVIEW (INTERFACES)

Zynq-7000 **PS**

- Runs UI and low-complexity sort/search
- Clocks FPGA (100 MHz)
- DDR3 controller
- Master for data -> PL
- Slave for data -> PS

KNN_HW:

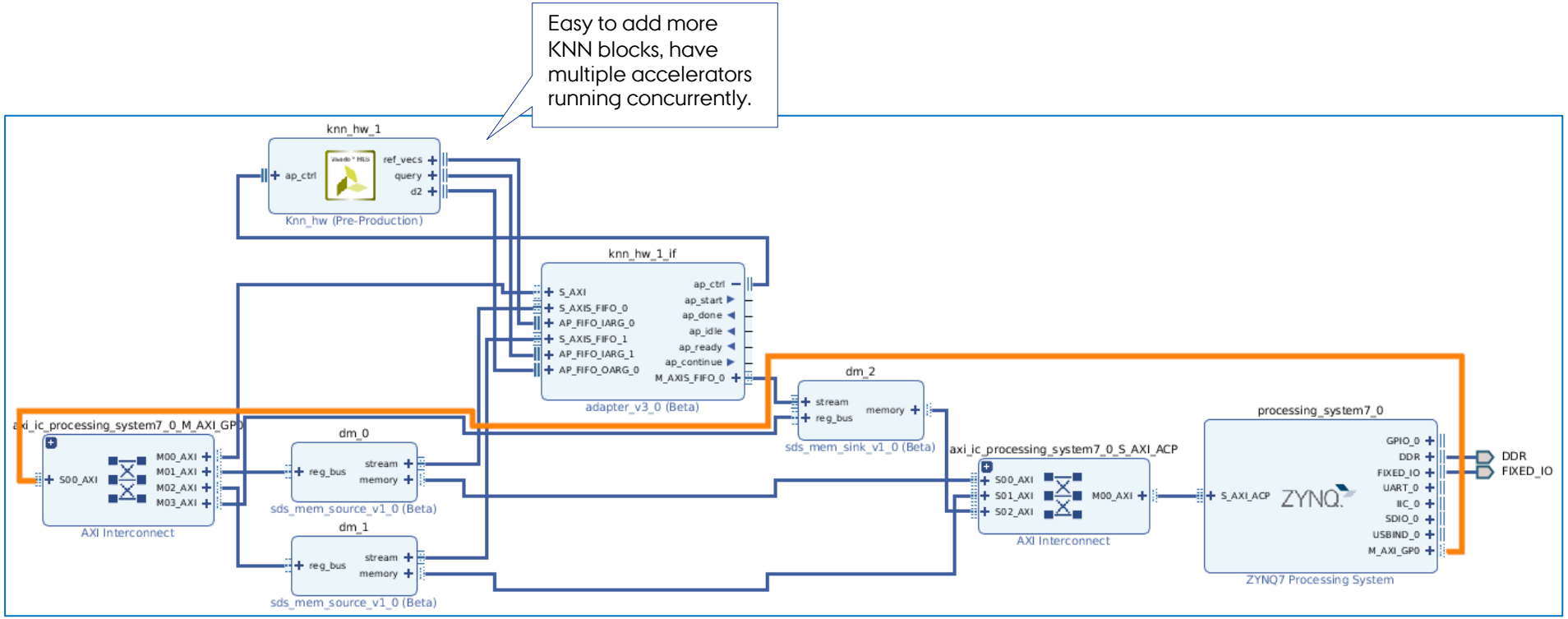
- HW accel. computation of squared distances

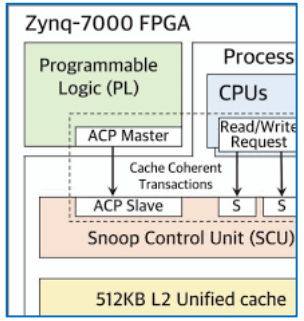
KNN_HW_IF:

- Interface and control
- Streams data in/out of acc.
- Handles stop/go

dm_x:

- MM2S / S2MM adapters





DATA MOVERS ARE MM2S ADAPTORS

Data motion is by **FastDMA***

- FastDMA is new in 2019.1 and is inferred by SDS automatically
  - Rather than AXIDMA_SIMPLE (contiguous) or AXIDMA_SG (non-contig.) which are IP-blocks that are implemented in PL and take up space.
- Hard to find documentation on it, but it is likely the actual Zynq-7000 DMA built-in in PS that gets PL access.

DMA Controller manages transfers

- **dm_x** blocks are essentially MM2S/S2MM adapters with AXI bus interfaces: Converts between AXI MM interface transaction and AXI data stream (FIFO) transaction
  - -> DMA can r/w to MM interf. in bursts, accelerator interf. can r/w streams sequentially or in bursts (fast!)
- **dm_0**: reference-vectors (1024x63)
- **dm_1**: query-vectors (100x63)
- **dm_2**: sq. distances vector, return value (1024x1)

Diagram | **Address Editor**

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 0x40000000 [1G])					
dm_0	reg_bus	reg0	0x44A0_0000	64K	0x44A0_FFFF
dm_1	reg_bus	reg0	0x44A1_0000	64K	0x44A1_FFFF
dm_2	reg_bus	reg0	0x44A2_0000	64K	0x44A2_FFFF
knn_hw_1_if	S_AXI	reg0	0x44A3_0000	64K	0x44A3_FFFF
dm_0					
memory (32 address bits : 4G)					
processing_system7_0	S_AXI_ACP	ACP_DDR_LOWOCM	0x0000_0000	512M	0x1FFF_FFFF
processing_system7_0	S_AXI_ACP	ACP_IOP	0xE000_0000	4M	0xE03F_FFFF
processing_system7_0	S_AXI_ACP	ACP_M_AXI_GP0	0x4000_0000	1G	0x7FFF_FFFF
processing_system7_0	S_AXI_ACP	ACP_QSPI_LINEAR	0xFC00_0000	16M	0xFCFF_FFFF
dm_1					
memory (32 address bits : 4G)					
processing_system7_0	S_AXI_ACP	ACP_DDR_LOWOCM	0x0000_0000	512M	0x1FFF_FFFF
processing_system7_0	S_AXI_ACP	ACP_IOP	0xE000_0000	4M	0xE03F_FFFF
processing_system7_0	S_AXI_ACP	ACP_M_AXI_GP0	0x4000_0000	1G	0x7FFF_FFFF
processing_system7_0	S_AXI_ACP	ACP_QSPI_LINEAR	0xFC00_0000	16M	0xFCFF_FFFF
dm_2					
memory (32 address bits : 4G)					
processing_system7_0	S_AXI_ACP	ACP_DDR_LOWOCM	0x0000_0000	512M	0x1FFF_FFFF
processing_system7_0	S_AXI_ACP	ACP_IOP	0xE000_0000	4M	0xE03F_FFFF
processing_system7_0	S_AXI_ACP	ACP_M_AXI_GP0	0x4000_0000	1G	0x7FFF_FFFF
processing_system7_0	S_AXI_ACP	ACP_QSPI_LINEAR	0xFC00_0000	16M	0xFCFF_FFFF

New Datamover: FASTDMA support

- Supports Zynq-7000 SoC and Zynq UltraScale+ MPSoC.
- Wider direct memory access (DMA)
- High-bandwidth bus support with auto-configuration to DDR width
- Reduced programmable logic resources
- Helps facilitate timing closure

ACCELERATOR IS SYNCHRONOUS

Accelerator is run **synchronously** in this design

- PS waits for PL to finish
- Calls and variables are mapped to RTL ports
- Stub-functions are inserted by SDS
  - Replaces C++ function name (knn_hw)
    - Rewrites at all call-sites
  - Sends commands to accelerator interface
  - Waits for completion of memory transfers
    - Only returns when accelerator has completed, only then can PS code execution resume.

Could also run accelerator **async. to PS**

- -> use IRQ to signal when done, or
- -> *manually* implement another waiting point in SW code (like joining threads)

Even if accelerator in PL was slower than PS, it could be used to free-up time in PS (like threads on multicore) -> realtime systems...

Interface

Summary

RTL Ports	Dir	Bits	Protocol	Source Object	C Type
ap_clk	in	1	ap_ctrl_hs	knn_hw	return value
ap_rst_n	in	1	ap_ctrl_hs	knn_hw	return value
ap_start	in	1	ap_ctrl_hs	knn_hw	return value
ap_done	out	1	ap_ctrl_hs	knn_hw	return value
ap_idle	out	1	ap_ctrl_hs	knn_hw	return value
ap_ready	out	1	ap_ctrl_hs	knn_hw	return value
ref_vecs_dout	in	8	ap_fifo	ref_vecs	pointer
ref_vecs_empty_n	in	1	ap_fifo	ref_vecs	pointer
ref_vecs_read	out	1	ap_fifo	ref_vecs	pointer
query_dout	in	8	ap_fifo	query	pointer
query_empty_n	in	1	ap_fifo	query	pointer
query_read	out	1	ap_fifo	query	pointer
d2_din	out	32	ap_fifo	d2	pointer
d2_full_n	in	1	ap_fifo	d2	pointer
d2_write	out	1	ap_fifo	d2	pointer

```
45 #include "cf_stub.h"
46 void p_0_knn_hw_1_noasync(uint8_t * ref_vecs, uint8_t * query, unsigned int * d2);
47 void p_0_knn_hw_1_noasync(uint8_t * ref_vecs, uint8_t * query, unsigned int * d2)
48 {
49   int start_seq[1];
50   start_seq[0] = 0;
51   cf_request_handle_t p_0_hwinst_knn_hw_1_cmd;
52   cf_send_i(&(p_0_hwinst_knn_hw_1.cmd_knn_hw), start_seq, 1 * sizeof(int), &p_0_hwinst_knn_hw_1_cmd);
53   cf_wait(p_0_hwinst_knn_hw_1_cmd);
54
55   cf_send_i(&(p_0_hwinst_knn_hw_1.ref_vecs), ref_vecs, (1024*63) * 1, &p_0_request_0);
56   cf_send_i(&(p_0_hwinst_knn_hw_1.query), query, (63) * 1, &p_0_request_1);
57
58   cf_receive_i(&(p_0_hwinst_knn_hw_1.d2), d2, (1024) * 4, &p_0_knn_hw_1_noasync_num_d2, &p_0_request_2);
59
60   cf_wait(p_0_request_0);
61   cf_wait(p_0_request_1);
62   cf_wait(p_0_request_2);
63 }
```

Return value is only transferred out when accelerator signals ready on output data

Stub-functions with *_i* are asynchronous -> wait must be called

KNN BLOCK USES 32 CASCADED DSP SLICES

KNN-block is data path and control path

Control path (FSM): Next-state and output logic

Data path, generally low-latency DSP48E1s cascaded several levels deep:

- Difference between query_i and reference_i is implemented using LUTs and carrylogic.
- 1st level DSP performs multiply
  - opmode b'0000101': output M
- Higher level DSPs perform multiply and accum.
  - opmode b'0010101': output M + PCIN
- FFs used for
  - Local input memory for query vector
  - Memory and carry logic for pipelined / unrolled loops

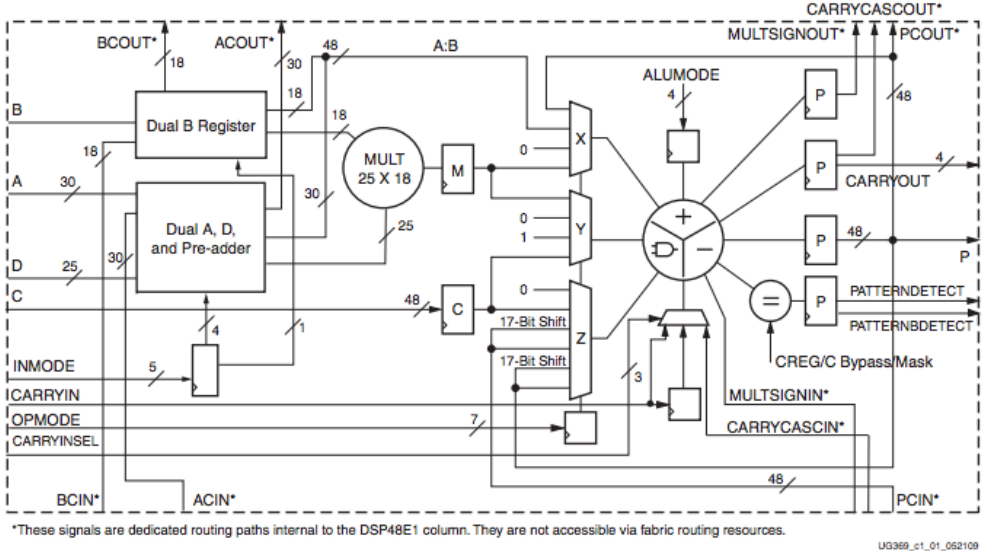
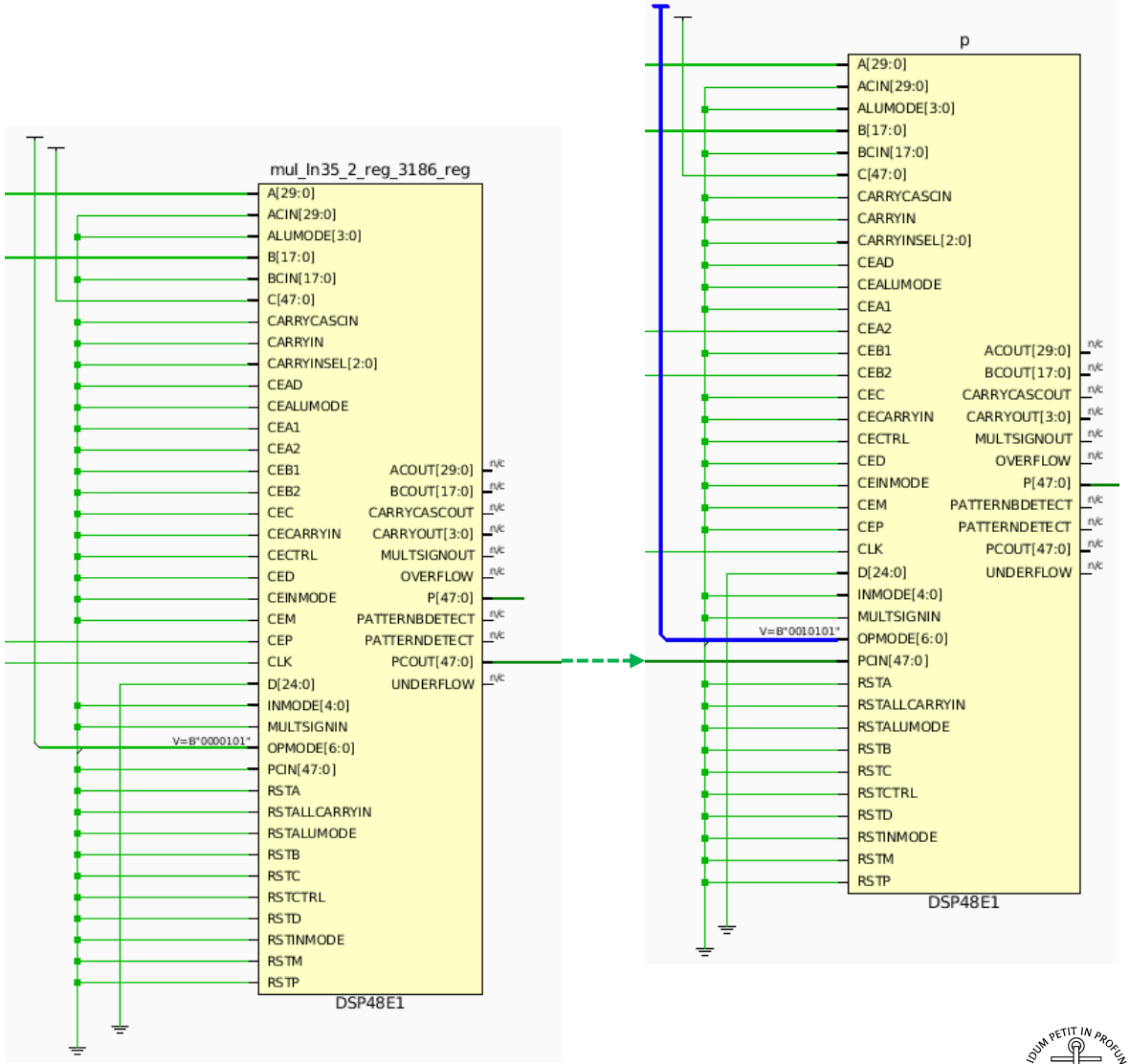


Figure 2-1: **7 Series FPGA DSP48E1 Slice**

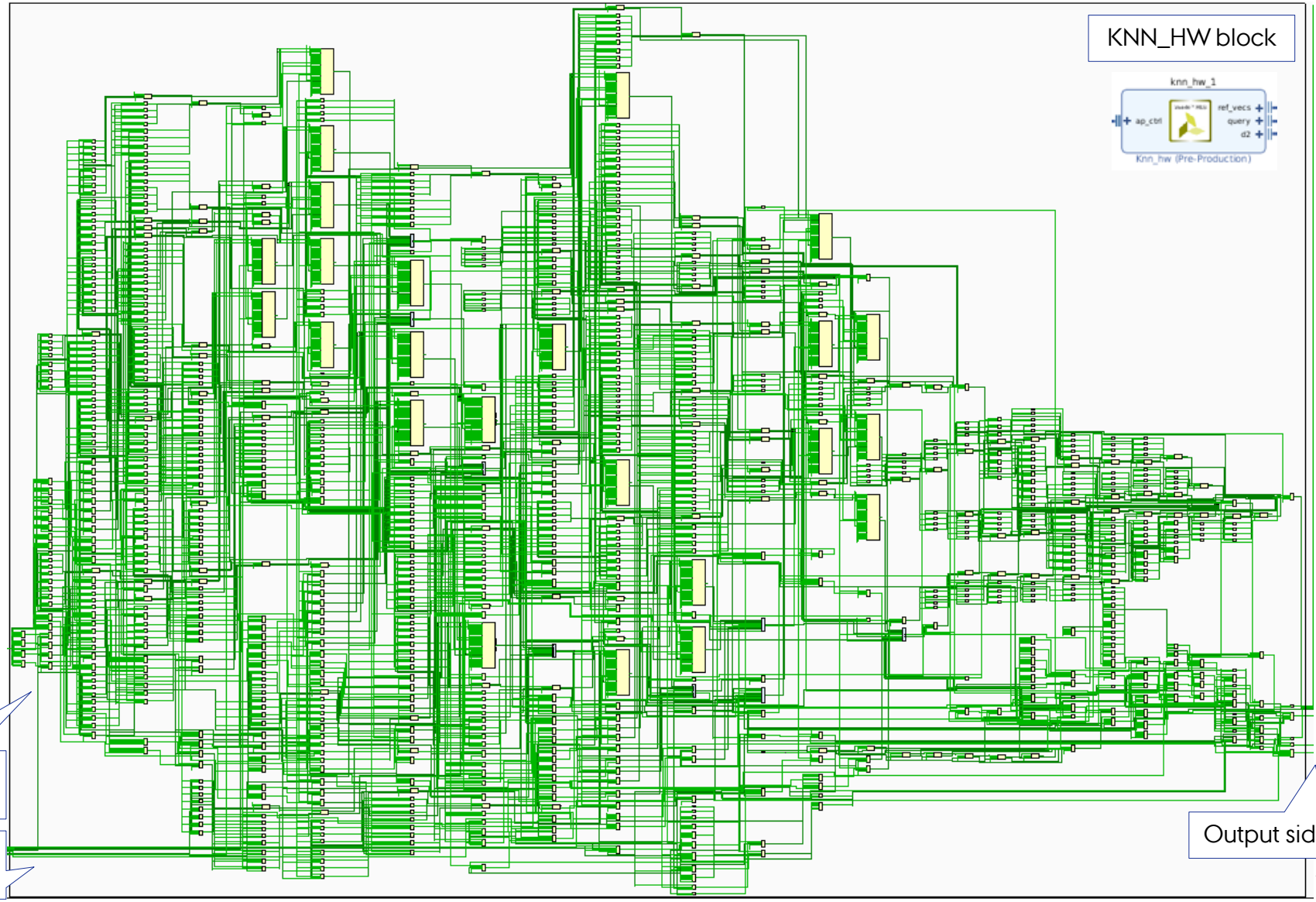


KNN ACCELERATOR REALISATION

Mile-high overview of RTL schematic:

- Large blocks are DSPs
  - Some are hidden inside small blocks because they are wrapped with additional control logic

- Circuitry **near query-vec** input are the FF registers to store partitioned array
- Circuitry to the **right of that** is pipelining (LUTs and carrylogic)
  - Due to loop-carried dependency from accumulated sum
- Circuitry **near ref.-vec input** handles the pipelined difference calculations and the control path (FSM)
- Circuitry **near output** side is the pipelined storage of results, comparators to check when results are written out (when full vector computed), and FSM logic to handle outputting to accel. I/F.



IMPORTANT DESIGN CHOICES

Design choices

Data width and signedness:

- Input data is 8-bit unsigned
- Output is 32-bit unsigned

Data motion network

- DMA in/out of PL

HLS optimizations

- Pipelining loops
- Cyclic array partitioning by factor 32
- Loop-unrolling by factor 32 (marginal effect)

Biggest optimisation challenge is to deal with loop-carried dependency (Accumulated MAC)

Implications

- Lower fabric requirement and faster DMA transfer than e.g. 32-bit.
- If floating point used, resources are used too quickly!
- *Supposedly*, unsigned operations are faster (in some cases)

- 3 data-movers, FIFO streams between accel. and accel. interf.
- Local mem. required inside accel for query-vector (multiple reads)

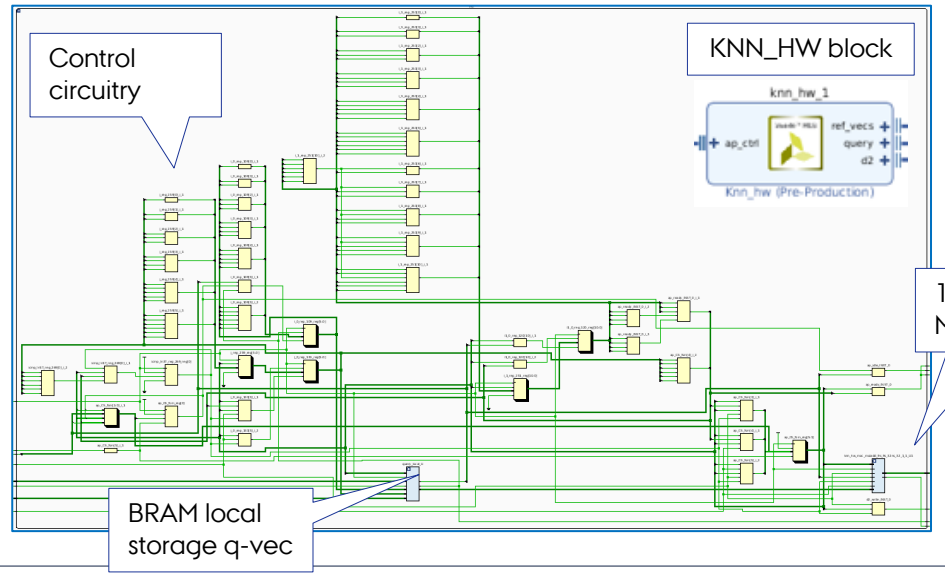
- More concurrency -> more DSPs
- Higher use of FFs, carrylogic and LUTs to handle pipelining (new data must be ready on every clock tick)
- SDS/HLS infers that FFs are faster than BRAM for storing partitioned array (BRAM only has 2 r/w ports)

EFFECT OF OPTIMISATIONS

No HLS optimisations:

- No pipelining
- No loop unrolling
- No array partitioning
- Keep FastDMA data transfer (SDS#)
- Much simpler implementation
  - -> 1 DSP, BRAM instead of FFs
  - ~2.5x faster than SW.
  - Primarily due to more efficient data access than processor (assumed, and not optimized).

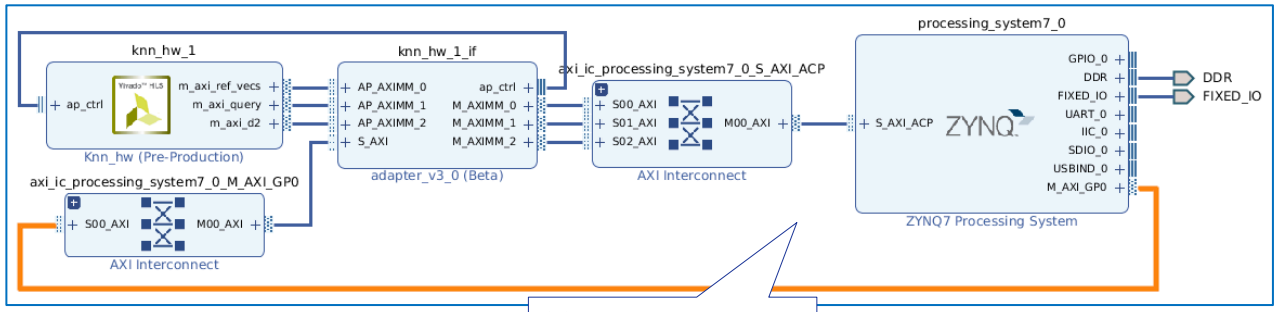
```
Perform k-Nearest Neighbours with k = ? Please enter a positive integer:
Starting KNN with k=3.
*** Hardware KNN results ***
Misclassification of obs. 49 out of 100. Pred.: 3, True: 4
Misclassification of obs. 64 out of 100. Pred.: 4, True: 3
Accuracy: 0.98
*** Software KNN results ***
Misclassification of obs. 49 out of 100. Pred.: 3, True: 4
Misclassification of obs. 64 out of 100. Pred.: 4, True: 3
Accuracy: 0.98
*** Verification report ***
OK: HW and SW Euclidian distances match.
*** Performance report ***
Avg. CPU cycles running KNN in software: 3252802
Avg. CPU cycles running KNN in hardware: 1310820
Speed-up: 2.4815
```



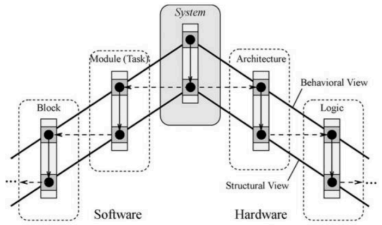
No HLS or SDS optimisations:

- As above, plus
- Direct MM interface to DDR3
- HW block implementation without the data movers
  - -> AXI MM interfaces
- SDS/HLS still infers some optimisations, e.g. burst-reads and burst-writes
  - -> Burst-size registers for input-data inside KNN-HW block

```
*** Verification report ***
OK: HW and SW Euclidian distances match.
*** Performance report ***
Avg. CPU cycles running KNN in software: 3252758
Avg. CPU cycles running KNN in hardware: 2315551
Speed-up: 1.40474
```



DEVELOPMENT METHOD



Co-design with DSE

- Software-first, then increase performance.
- Criteria:
  - Performance metric (profiling):
    - Clock cycles (abs. and rel.)
  - Usability concern:
    - Algo. flexibility for user
- Allocation
  - HW/SW connectivity: Memory interfaces, memory storage elements, data motion network, bus
- Binding
  - Partitioning: UI and flexible part of algo. to PS, number-crunching ops. to PL.
  - Bind MAC ops. to more concurrent DSP-slices
- Scheduling
  - FPGA clocking (faster makes synthesis much more difficult)
  - Force concurrency (pipelining, unrolling or code-change)
  - SDSoC SDS schedules system

All/most activities by changing algorithm, data storage and using #pragma directives for SDS and HLS.

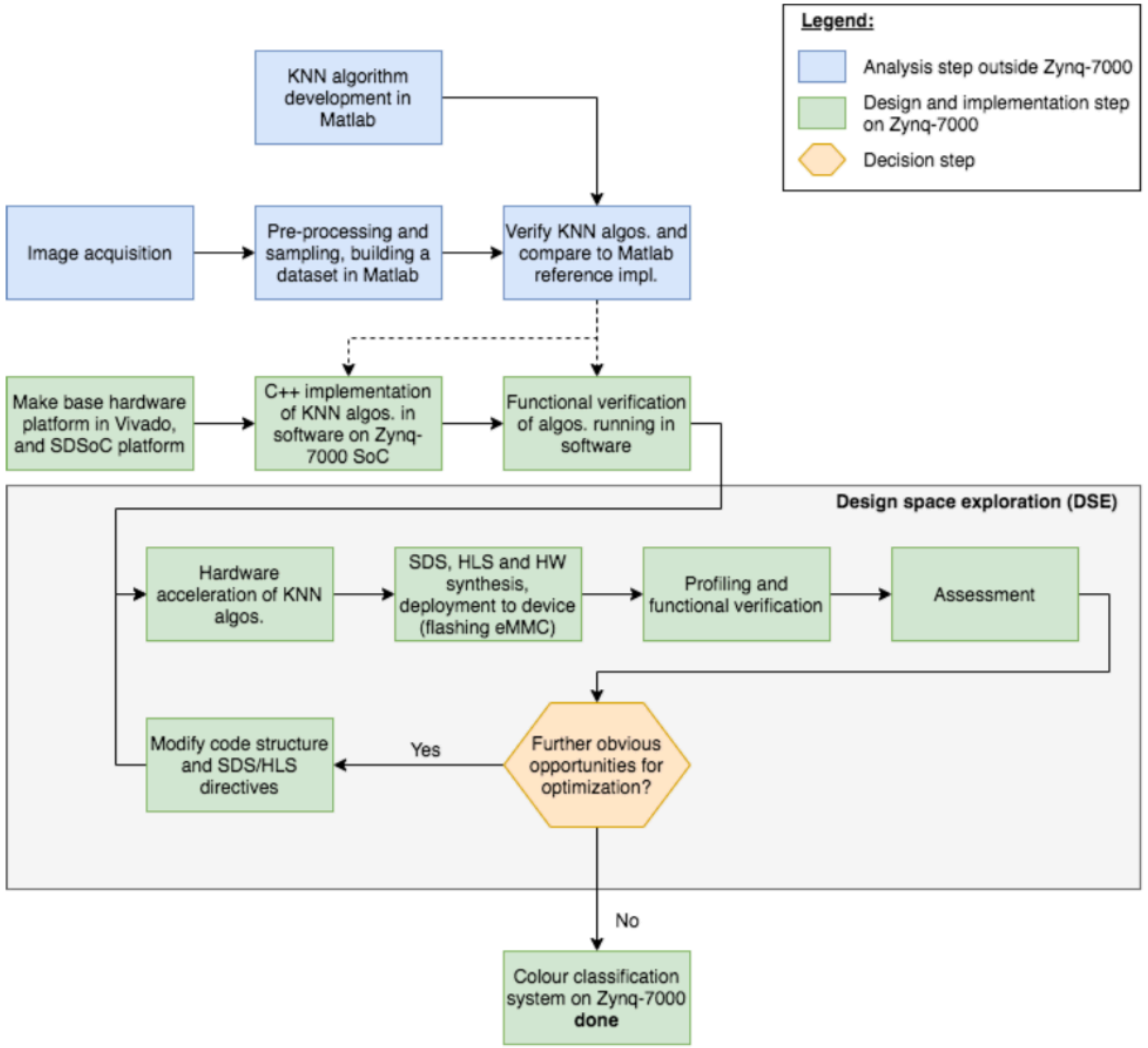
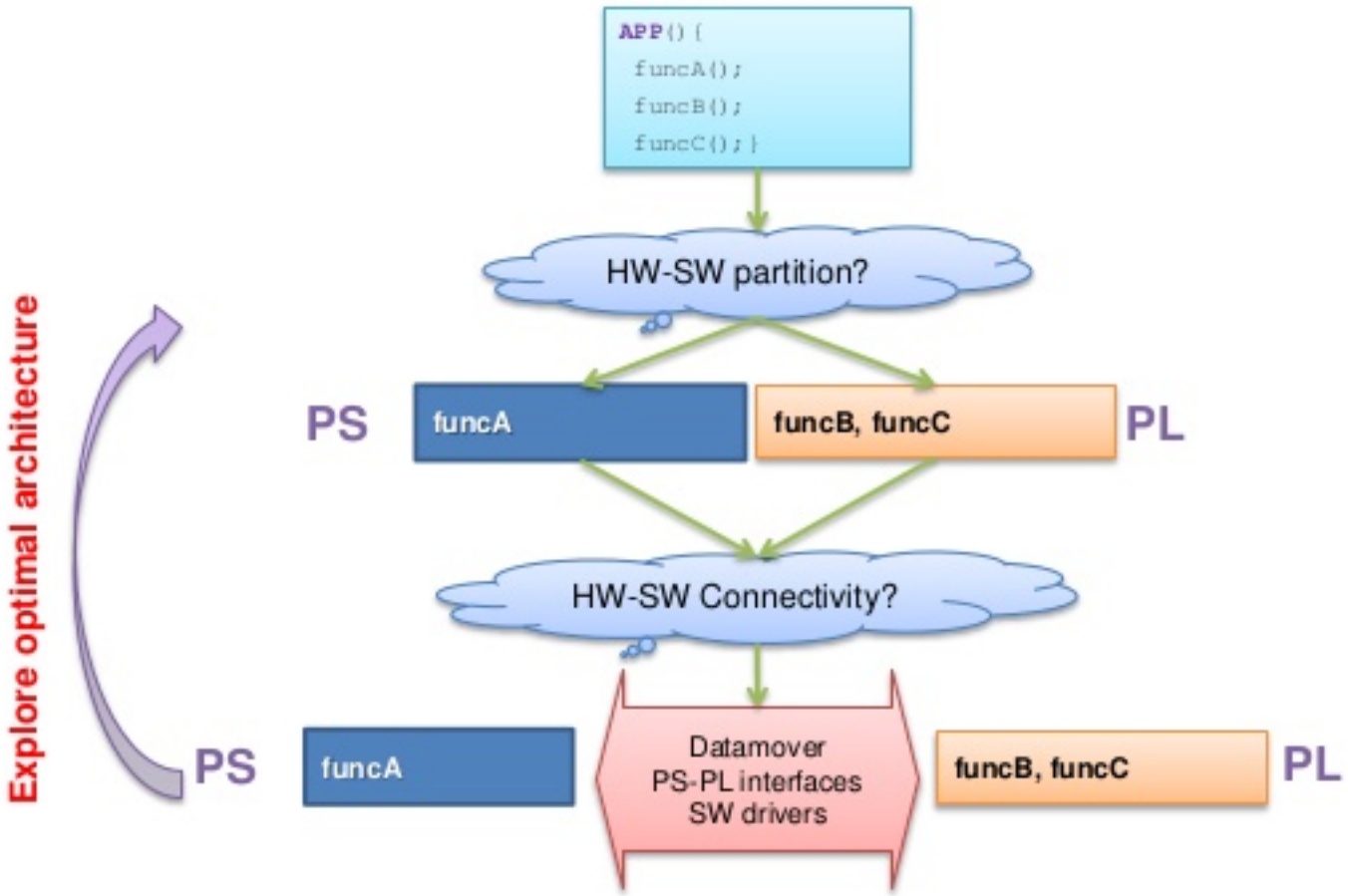


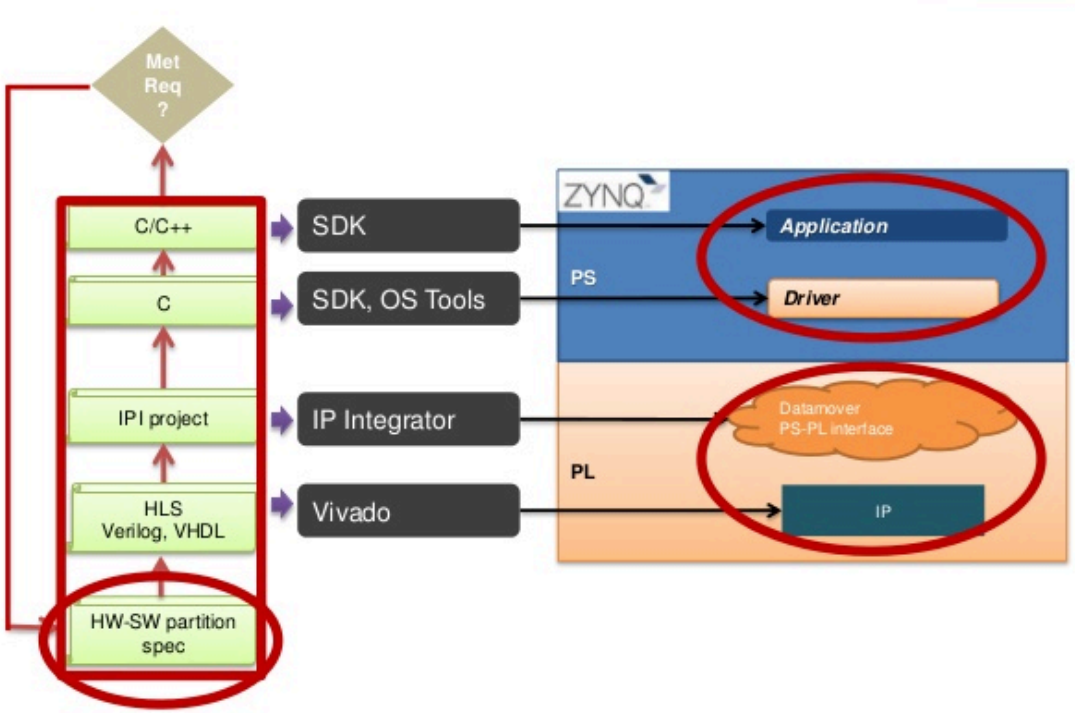
ILLUSTRATION OF CO-DESIGN ON ZYNQ SOC



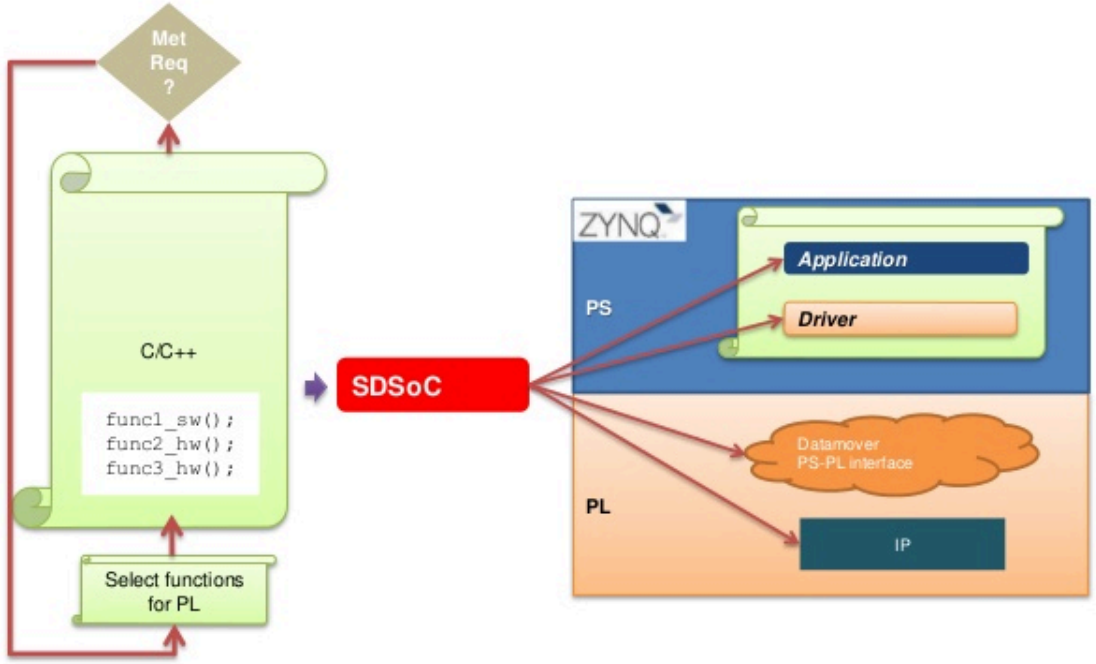
Source: https://www.slideshare.net/SundanceDotCom/emc2-xilinx-sdsoc-presentation

EASIER CO-DESIGN BY AUTOMATING THE XILINX TOOLCHAIN

Without SDSoC



With SDSoC (or Vitis...)



Source: https://www.slideshare.net/SundanceDotCom/emc2-xilinx-sdsoc-presentation

Based on project experience

WHEN TO USE SDS AND HLS

Indications (definitely use SDS/HLS when...)

Problem domain is

- Focused on **algorithm development**,
- **acceleration** of specific functions, or
- **DSE/rapid prototyping** of candidate functions to accelerate in a co-designed HW/SW-system.

System definition is

- Behavioural or **functional**, specifically given a
- **mathematical**/algorithmic problem formulation.

System constraints are

- **Flexible resource constraints**.
- Timing constraints are implicit (but can be added).

Contraindications (maybe use RTL instead, if...)

Problem-domain is

- Glue-logic, or
- interconnection of different existing IP blocks.

System definition is

- Hardware-focused, using existing peripherals/IP
- Low-level bit-handling and control paths
- Requiring specific interfaces and protocols (e.g. SPI, I^2C, etc.)

System constraints are

- Tight resource constraints, use of resources/footprint must be explicitly controlled. Timing requirements are explicit.

CONCLUSION AND KEY LEARNING

Implemented **ML-based classifier on Zynq-7000 SoC**

- according **to requirements from project proposal**
- using concepts and theory from E5ADD

Used co-design process (and Xilinx toolchain supporting this) to gradually improve system

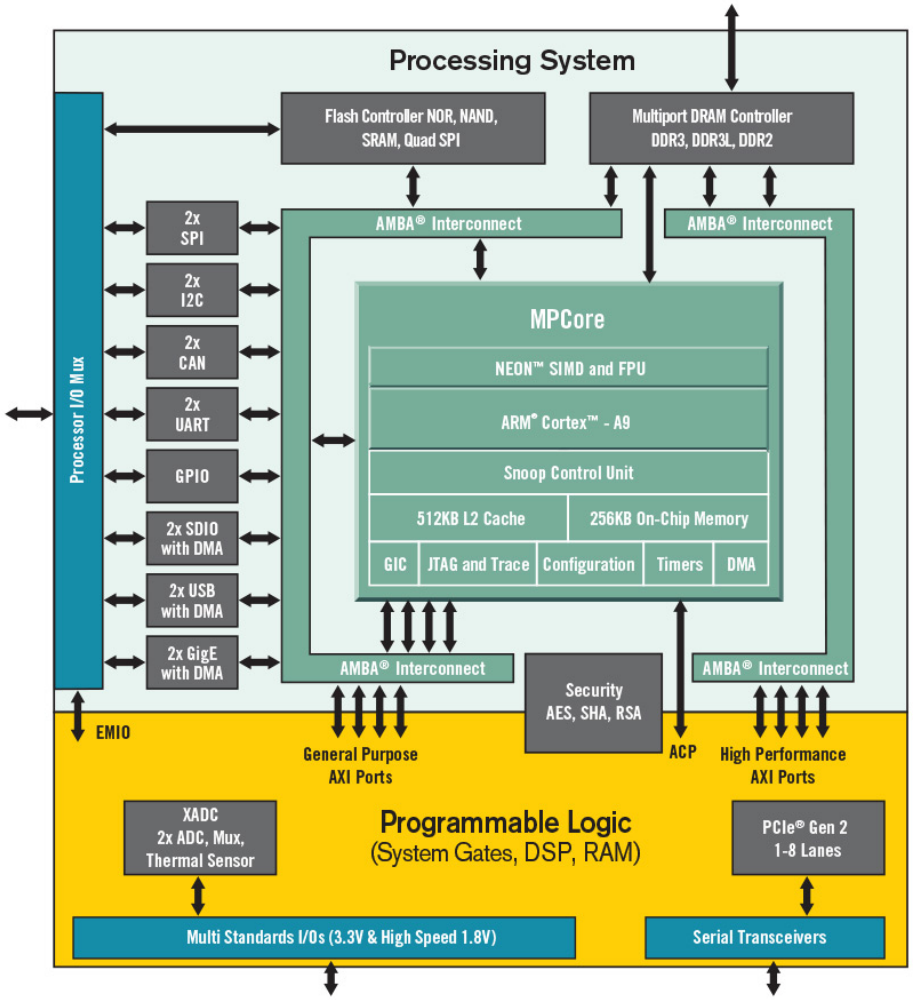
- **DSE for optimisation**, partitioning to balance usability (algo. flexibility and performance).

Attained ~7x speed-up using HW accelerator over SW algo.

- More DSE and optimisation remains for accelerator (and the SW solution)
- Several ideas for expanding the system
- Even with <1x speed-up, the case for using HW to off-load PS is still valid in realtime systems.

Learned and demonstrated how co-design with **SDS/HLS is very efficient for rapid prototyping**

- But it takes time to understand the tools and capabilities of the platform well!
- Even for simple system like this, DSE is LARGE! Interplay between design choices can be complex.



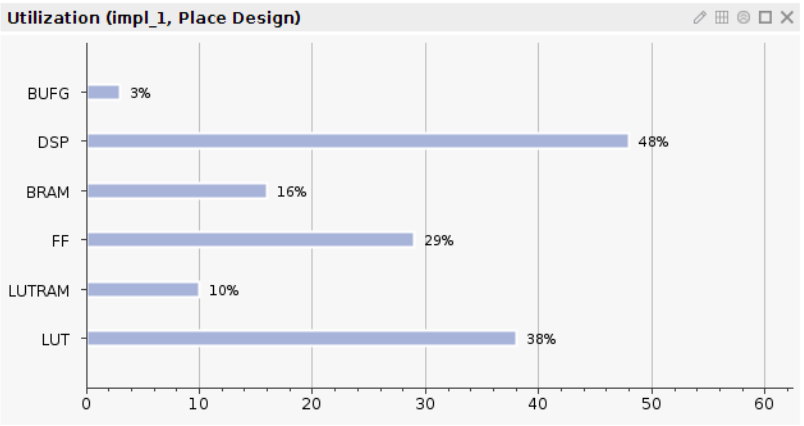
IDEAS FOR IMPROVEMENT, FUTURE WORK

Faster HW-acceleration:

1. Increase **FPGA clock frequency** and work on timing closure:
   - Currently: 100 MHz (10 ns clock period) and no timing violations.
   - Risk: See WNS, timing violations could occur at e.g. 5 ns clock period, so something must be done to ensure timing closure
     - Using a substantial part of the FPGA fabric, so there is risk that data not ready at clk if combinational delay > clock period)
2. Improve **algorithm to utilize more low-latency DSPs** without making critical path slower:
   - Increase concurrency e.g. by splitting into two problems of half size.
     - Problem: No longer sequential memory access, must use less efficient data acc.
3. Improve **data motion and algorithm together**:
   - Schedule movement of all query data in one-time burst (100 x 63).
   - Compute distances for all query vectors in ***one call*** of the accelerator, using very optimizable/parallelizable matrix multiplication (like on GPUs).
   - Return one result matrix of squared differences (1024 x 100), or potentially also implement the sort/search in PL and only return nearest neighbour classes (100 x 3).

Timing (impl_1, Route Design)

Report	WNS	WHS
impl_1, impl_1_route_report_timing_summary_0	0.995	0.012



Connect camera peripheral and implement image acquisition/pre-processing steps.

Connect actuator / other interaction with the physical world.

Fixed-point arithmetic for generalization of KNN to other classification problems (e.g. float -> Q15).

QUESTIONS?

BACKUP SLIDES

Build the hardware platform for acceleration

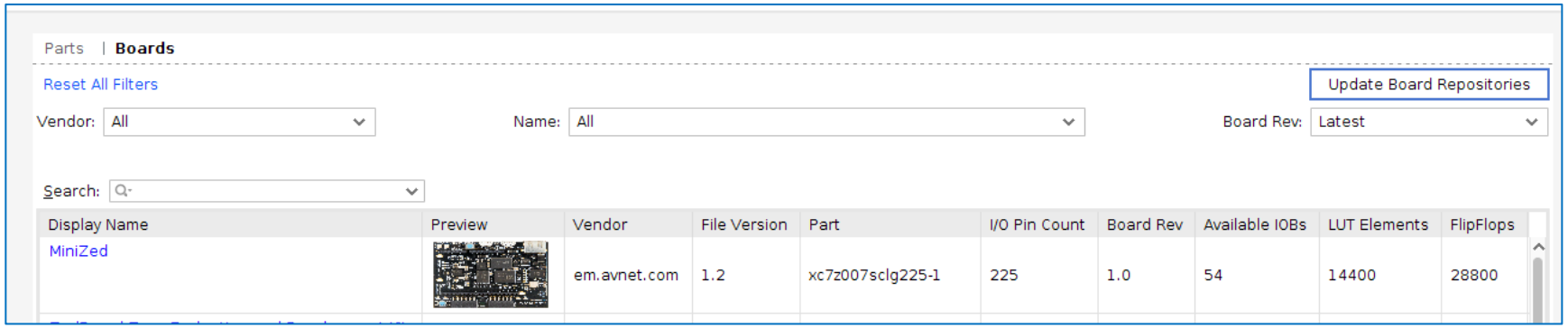
ENSURE MINIZED BOARD FILES IN VIVADO

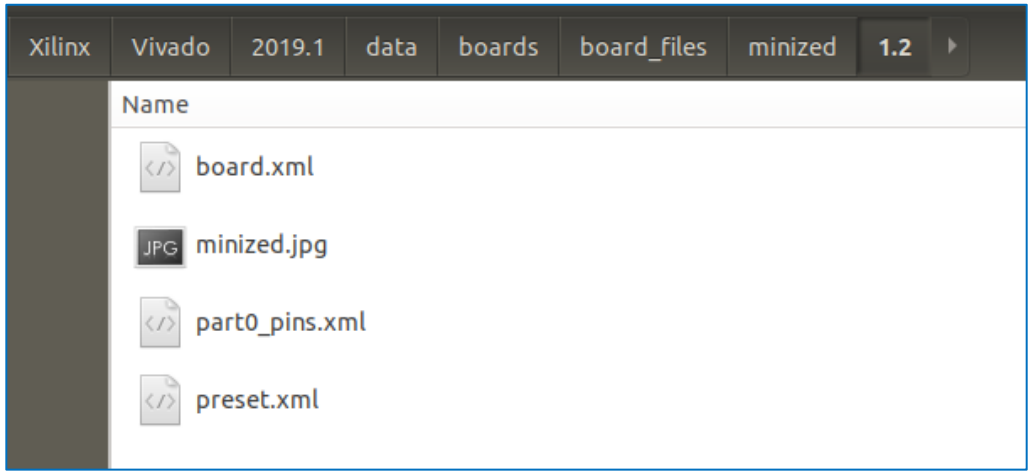
Confirm available or get from Github

- Place in

The part details must be available for Vivado to

- Generate custom hw platform in Vivado
- Perform SDSoC compilation/synthesis process later





USE VIVADO TO CREATE HARDWARE COMPONENT FOR THE SDSOC PLATFORM

Design with 4 fabric clocks

- 50, 100, 200, 250 MHz
- -> 4 PS resets instead of 2 in the original design

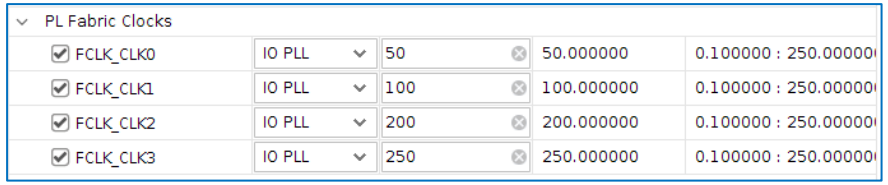
UART0 disabled

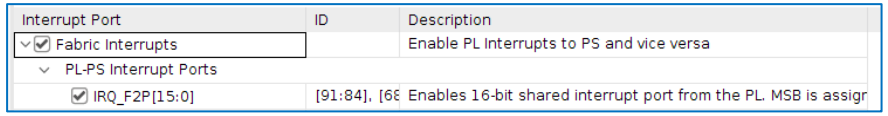
- Package limitations

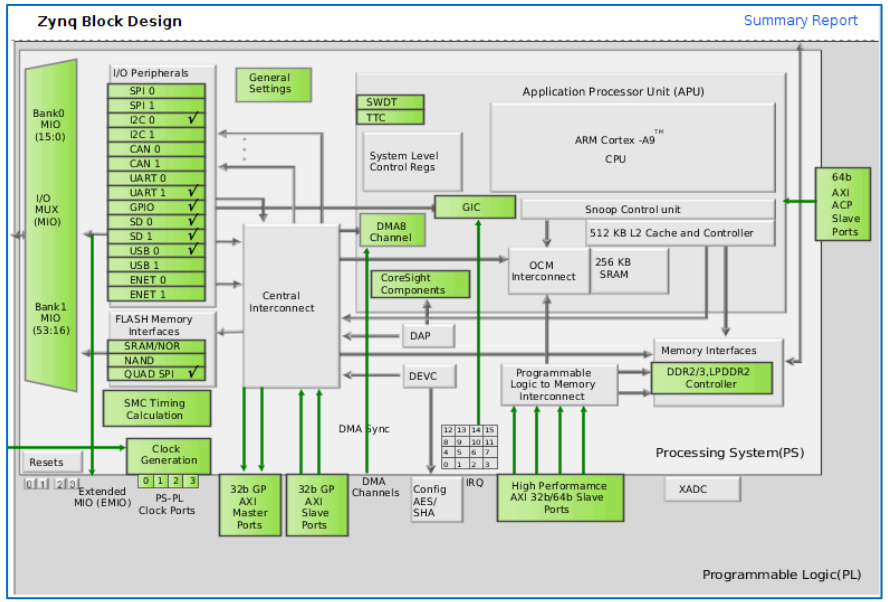
Set up PL-PS fabric interrupts and add concat block to connect these

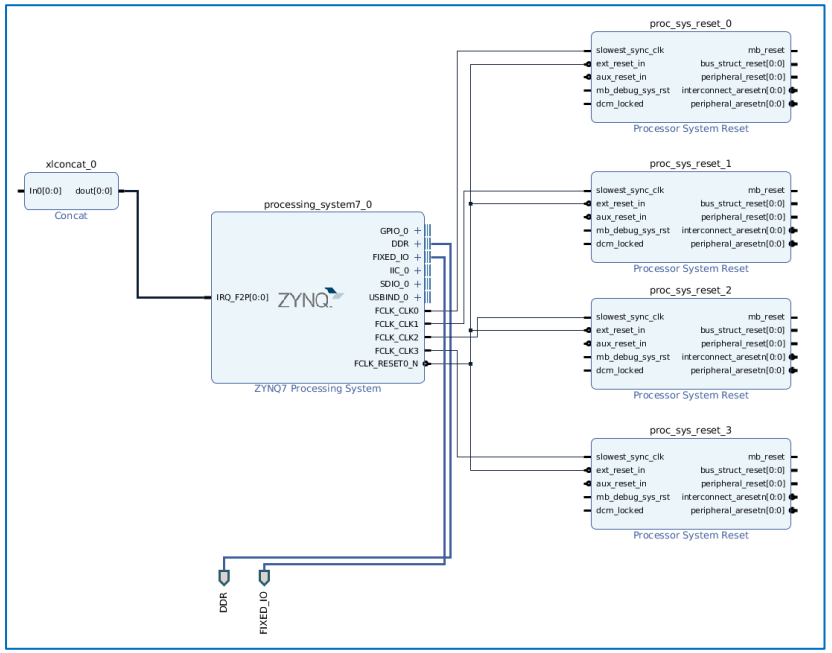
Create bitstream and export hardware to SDK

Follow UG1146 or SDSoC labs







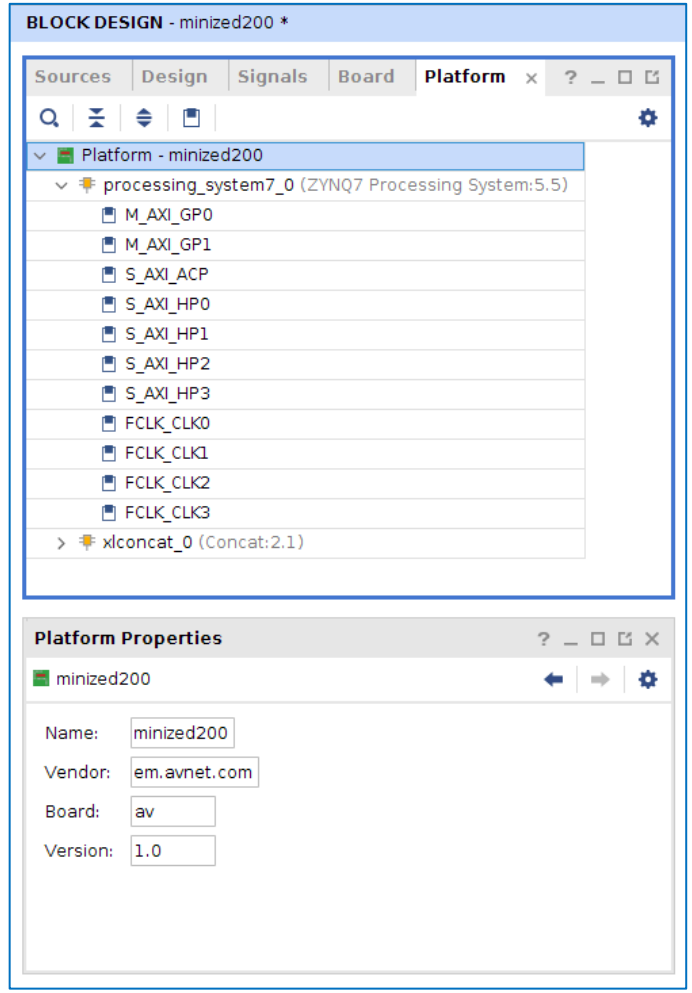


CREATE DSA (HARDWARE SPEC)

Run TCL commands:

- Declare platform name and version
- Declare clocks and resets
- Declare available AXI interfaces
- Declare 16 IRQ lines
- Validate design
- Write DSA
- Validate DSA

```
cd /home/janus/Xilinx-workfiles-E5ADD/_SDSoC/platforms/minized200
set_property PFM_NAME "em.avnet.com:av:minized200:1.0" [get_files ./minized200.srcs/sources_1/bd/minized200/minized200.bd]



set_property PFM.CLOCK {\
FCLK_CLK0 {id "0" is_default "true" proc_sys_reset "proc_sys_reset_0"}\
FCLK_CLK1 {id "1" is_default "false" proc_sys_reset "proc_sys_reset_1"}\
FCLK_CLK2 {id "2" is_default "false" proc_sys_reset "proc_sys_reset_2"}\
FCLK_CLK3 {id "3" is_default "false" proc_sys_reset "proc_sys_reset_3"}\
} [get_bd_cells /processing_system7_0]

set_property PFM.AXI_PORT { \
M_AXI_GP0 {memport "M_AXI_GP"} \
M_AXI_GP1 {memport "M_AXI_GP"} \
S_AXI_ACP {memport "S_AXI_ACP"} \
S_AXI_HP0 {memport "S_AXI_HP"} \
S_AXI_HP1 {memport "S_AXI_HP"} \
S_AXI_HP2 {memport "S_AXI_HP"} \
S_AXI_HP3 {memport "S_AXI_HP"} \
} [get_bd_cells /processing_system7_0]



set intVar []
for {set i 0} {$i < 16} {incr i} {
lappend intVar In$i {}
}
set_property PFM.IRQ $intVar [get_bd_cells /xlconcat_0]

validate_bd_design


write_dsa -force -unified minized200.dsa

validate_dsa minized200.dsa -verbose
```

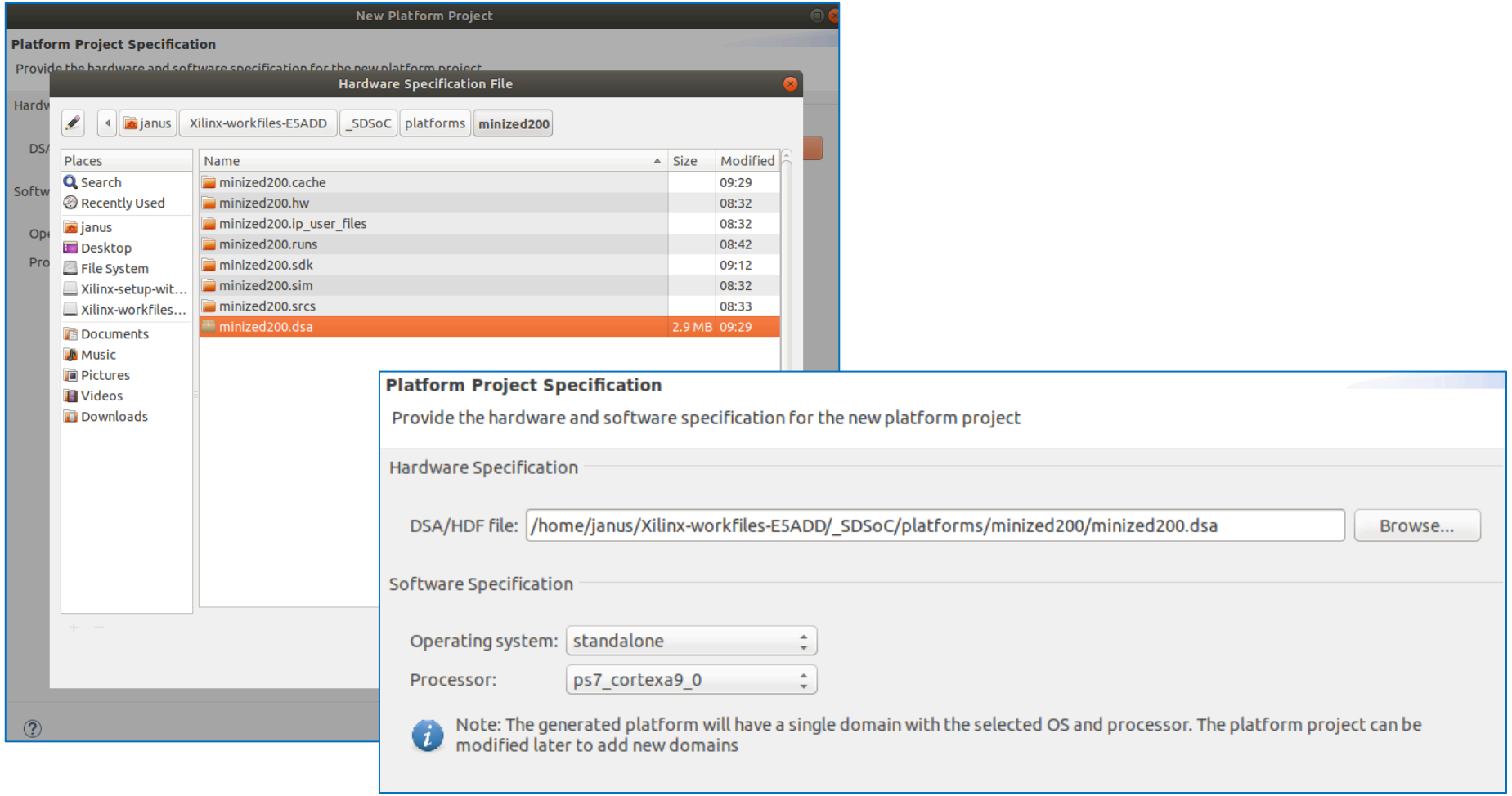


CREATE PLATFORM PROJECT IN SDSOC

New platform project -> from DSA

Purpose:

- Make a reusable custom platform
- Define further platform settings
- Generate FSBL, etc.



CONFIG AND EXPORT SDSOC PLATFORM

Define platform settings

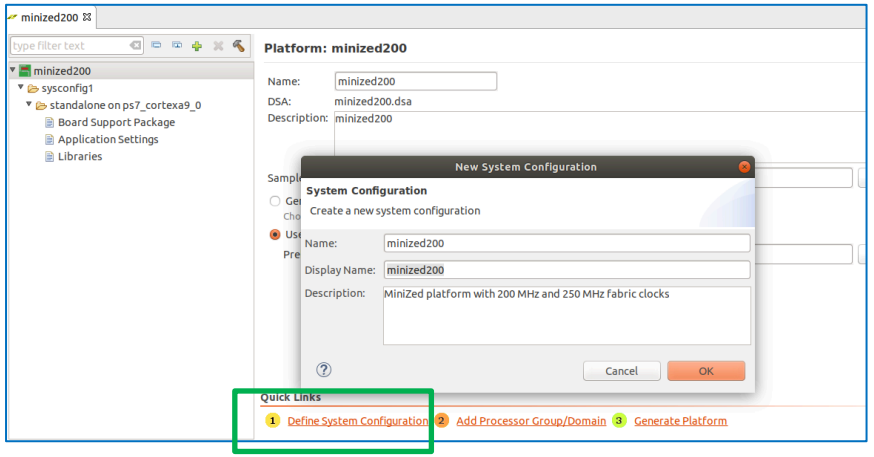
- Sys config
- Processor domain

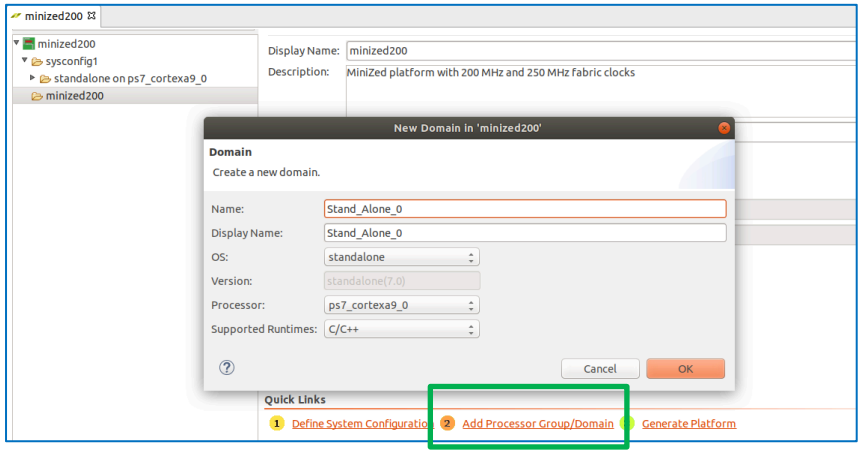
Generate platform

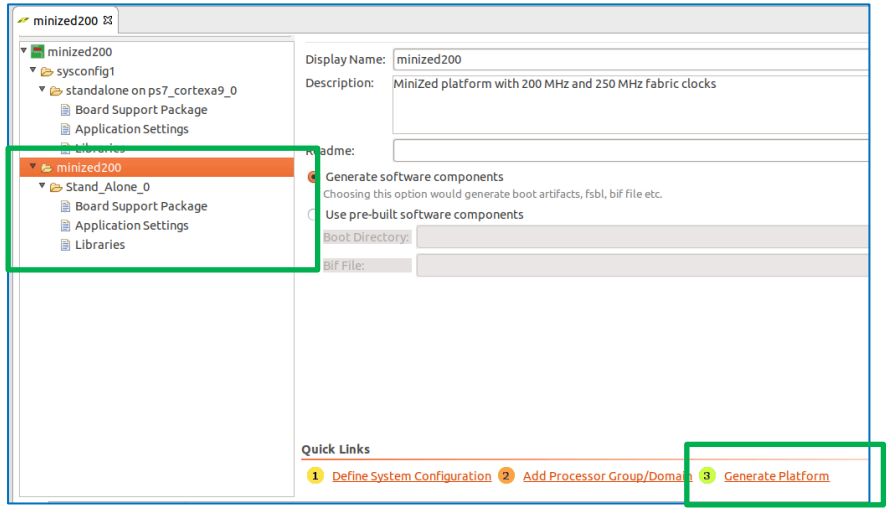
Move exported platform to location for custom platforms

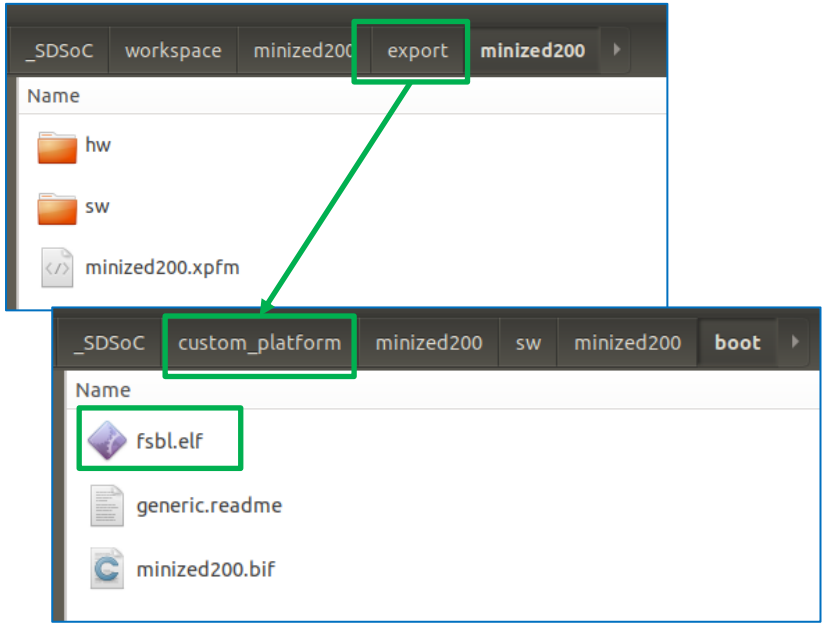
Verify that it has a new FSBL generated

- 559 kb
- So too big for OCM -> XIP execution









REPLACE LINKER SCRIPT

Use Xilinx Linker script (lscript.ld)

Replace current linker script with the Xilinx MiniZed specific one

- In the .../sw/minized200/Stand_Alone_0 folder

Original looks similar to the generated one – so perhaps they original one can be used...?

```
/*****************************************************************/
/*                                                               */
/* This file is automatically generated by linker script generator.*/
/*                                                               */
/* Version: 2019.1                                               */
/*                                                               */
/* Copyright (c) 2010-2016 Xilinx, Inc.  All rights reserved.    */
/*                                                               */
/* Description : Cortex-A9 Linker Script                         */
/*                                                               */
/*****************************************************************/

_STACK_SIZE = DEFINED(_STACK_SIZE) ? _STACK_SIZE : 0x994CD00;
_HEAP_SIZE = DEFINED(_HEAP_SIZE) ? _HEAP_SIZE : 0x994CD00;

_ABORT_STACK_SIZE = DEFINED(_ABORT_STACK_SIZE) ? _ABORT_STACK_SIZE : 1024;
_SUPERVISOR_STACK_SIZE = DEFINED(_SUPERVISOR_STACK_SIZE) ? _SUPERVISOR_STACK_SIZE : 2048;
_IRQ_STACK_SIZE = DEFINED(_IRQ_STACK_SIZE) ? _IRQ_STACK_SIZE : 1024;
_FIQ_STACK_SIZE = DEFINED(_FIQ_STACK_SIZE) ? _FIQ_STACK_SIZE : 1024;
_UNDEF_STACK_SIZE = DEFINED(_UNDEF_STACK_SIZE) ? _UNDEF_STACK_SIZE : 1024;

/* Define Memories in the system */

MEMORY
{
   ps7_ddr_0 : ORIGIN = 0x100000, LENGTH = 0x1FF00000
   ps7_qspi_linear_0 : ORIGIN = 0xFC000000, LENGTH = 0x1000000
   ps7_ram_0 : ORIGIN = 0x0, LENGTH = 0x30000
   ps7_ram_1 : ORIGIN = 0xFFFF0000, LENGTH = 0xFE00
}

/* Specify the default entry point to the program */

ENTRY(_vector_table)

/* Define the sections, and where they are mapped in memory */

SECTIONS
```

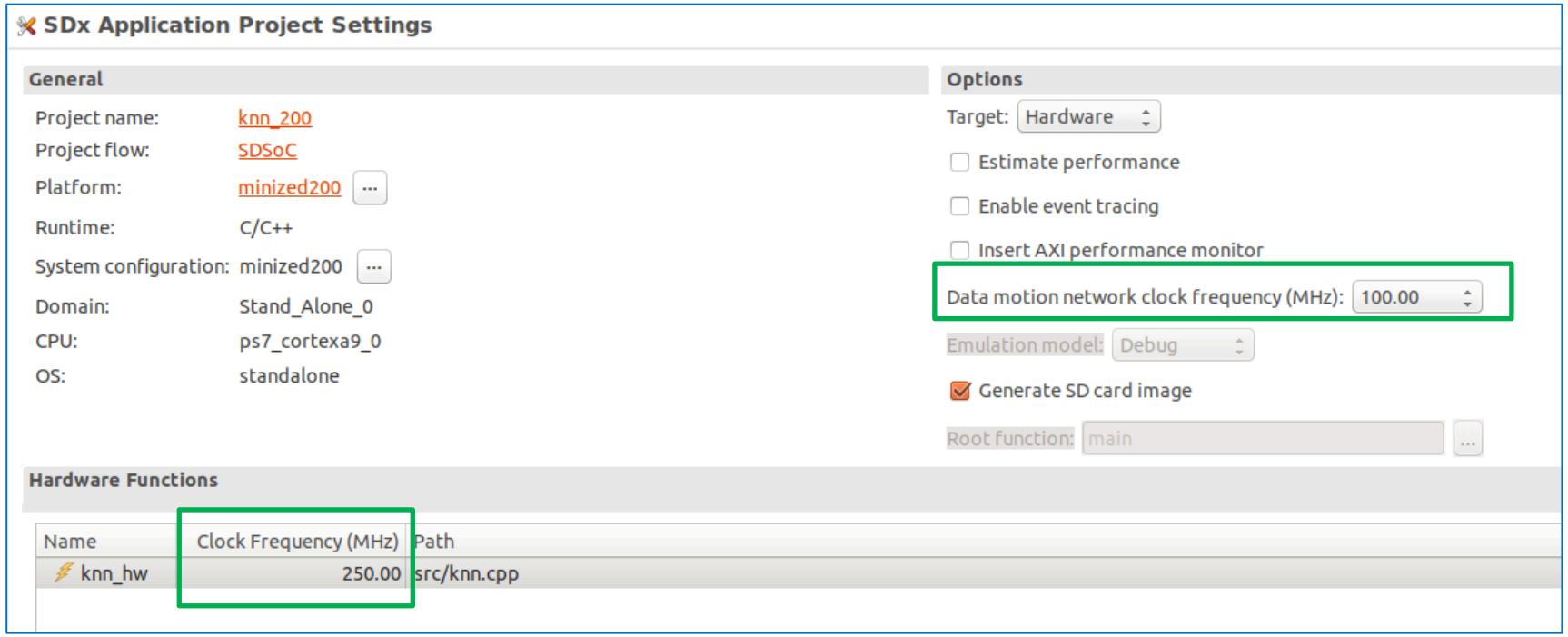
C ▾ Tab Width: 8 ▾ Ln 1, Col 1 ▾ INS

MAKE PROJECT WITH THE NEW CUSTOM SDSOC PLATFORM

Make application project

Use the minized200 system configuration (Stand_Alone_0 domain)

Accelerate a function with the new faster 200 MHz clock



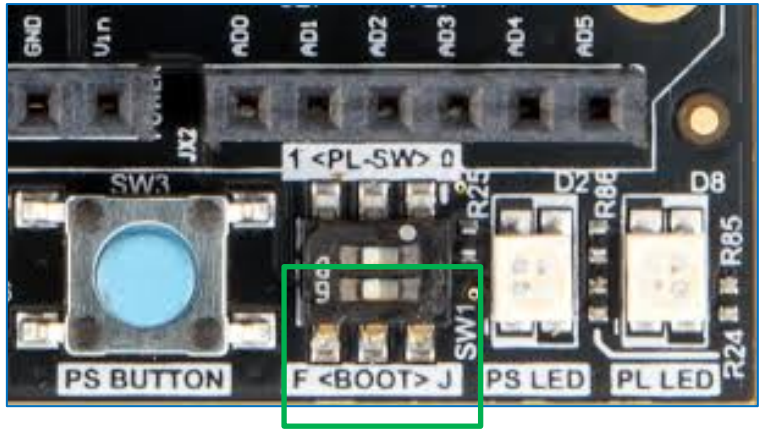
FLASH THE DEVICE

Puts FSBL and BOOT.bin into QSPI.

Boot device from Flash.

Startup order:

- FSBL boots device PS (clocks, etc.)
- BOOT.bin:
  - PS uses bitstream to program device PL
  - Application in ELF-file is executed on PS.



AARHUS
UNIVERSITET