Early Performance Verification of Embedded Inferencing Systems using open-source SystemC NVIDIA MatchLib

Herbert Taucher, Siemens CT

Russell Klein, Mentor

accellera SYSTEMS INITIATIVE

2019 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION EUROPE

Disclaimer

- Subject to changes and errors. The information given in this document only contains general descriptions and/or performance features which may not always specifically reflect those described, or which may undergo modification in the course of further development of the products. The requested performance features are binding only when they are expressly agreed upon in the concluded contract.

- All product designations, product names, etc. may contain trademarks or other rights of Siemens AG, its affiliated companies or third parties. Their unauthorized use may infringe the rights of the respective owner.

Code Samples Disclaimer

- Code samples in this presentation are for illustrative purposes only and are not to be considered complete and compilable. Simplifications have been made for the sake of clarity and brevity. For example, type casts, declarations, and irrelevant parameters have been removed to allow for a clearer presentation of the underlying concepts and algorithms. Not all code samples will be presented at DVCon Europe on October 29, 2019 due to time constraints. The complete code set is intended to be made available as an example with the “Catapult” high-level synthesis product at some time after the presentation at DVCon.

Agenda

- The need for early performance verification
- What is MatchLib?
- Communication channels in MatchLib
- Modeling AXI in MatchLib
- A real-world example, the “wake word”

Performance Analysis for HLS Design Flow

1. Motivation for performance analysis
    - Introduction to use case
    - Module/kernel level vs system level performance aspects
2. Module/kernel level performance analysis
3. System level performance analysis
4. Summary

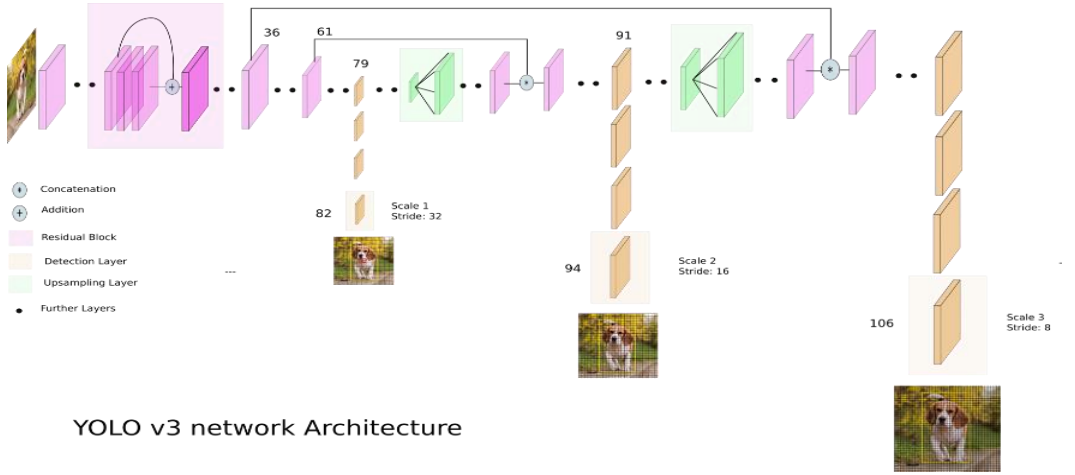
YOLO[1] v3 as a low-complexity use case for a High-level Synthesis flow to HW-accelerate AI workloads

HLS-ready C++ library for Mentor Catapult

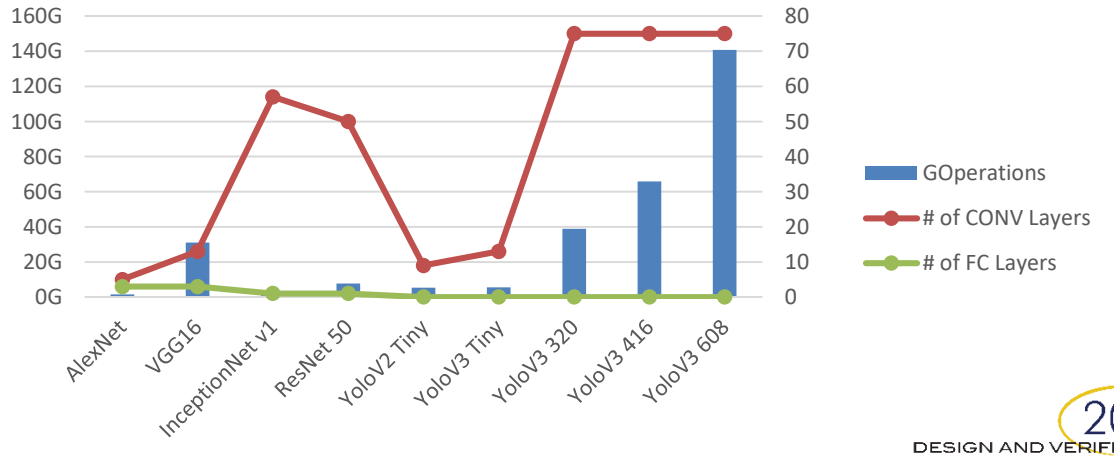
- Systolic array based NN inference accelerator
- Optimization for max-pooling
- Flexible reduced precision fixed-point data format
- Supporting FPGA and ASIC/SoC

AI optimization and exploration framework

- Graph optimization
- Retraining to recover reduced precision losses
- Exploration of optimized accelerator configuration for defined performance and resource constraints

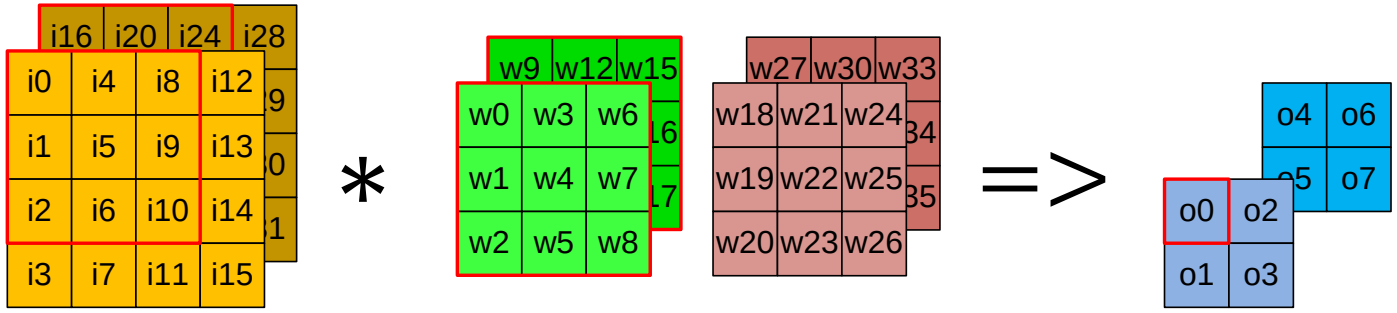


YOLO v3 network Architecture

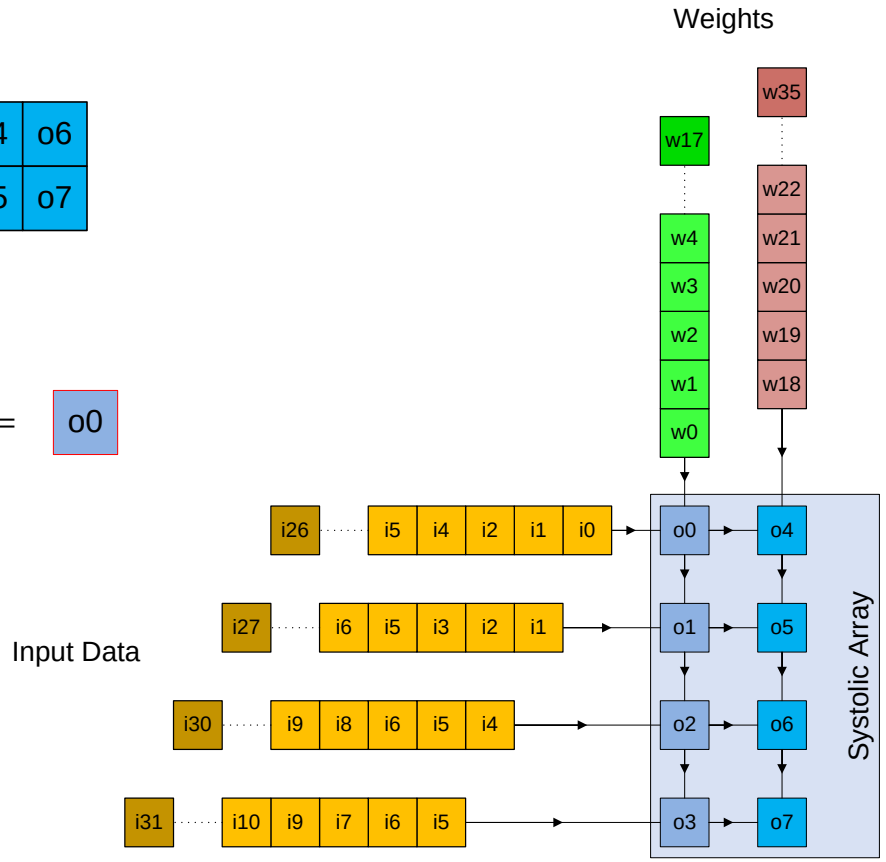


1- 'You only look once: Unified, real-time object detection', CVPR 2016, Redmon et al
http://pjreddie.com/darknet/yolo

Output-stationary Systolic Array micro-architecture for CNN acceleration



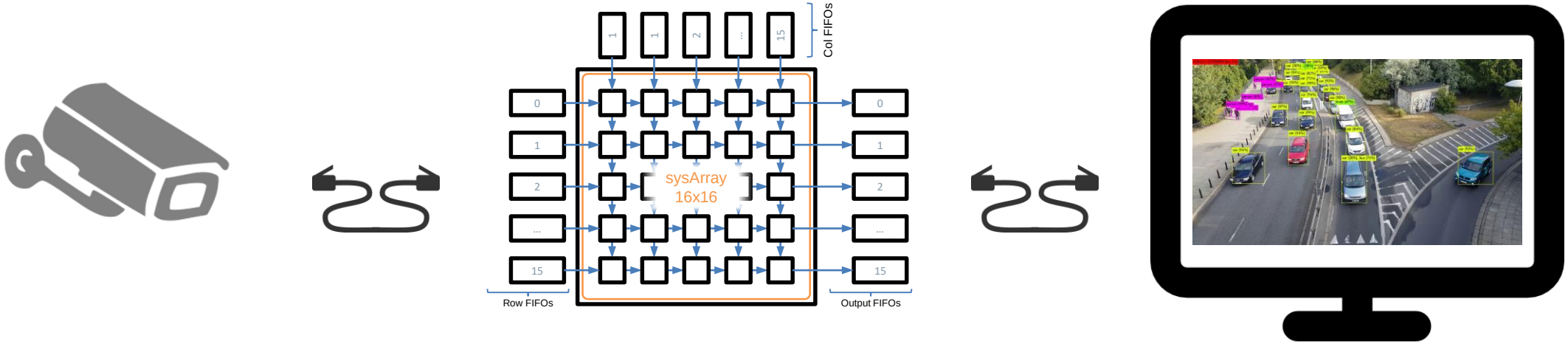
w0 x i0 + w1 x i1 + w2 x i2 + w3 x i4 + ... + w17 x i26 = o0



Systolic Array

- implements fine-grained mix of arithmetic, memory and logic resources
- keeps routing quite local
- is highly scalable

System performance depends on more than just the Systolic Array kernel performance



Pre-processing

- Interfacing with the camera and frame format
- Extracting frames to process with CNN
- Scaling resolution of camera to CNN

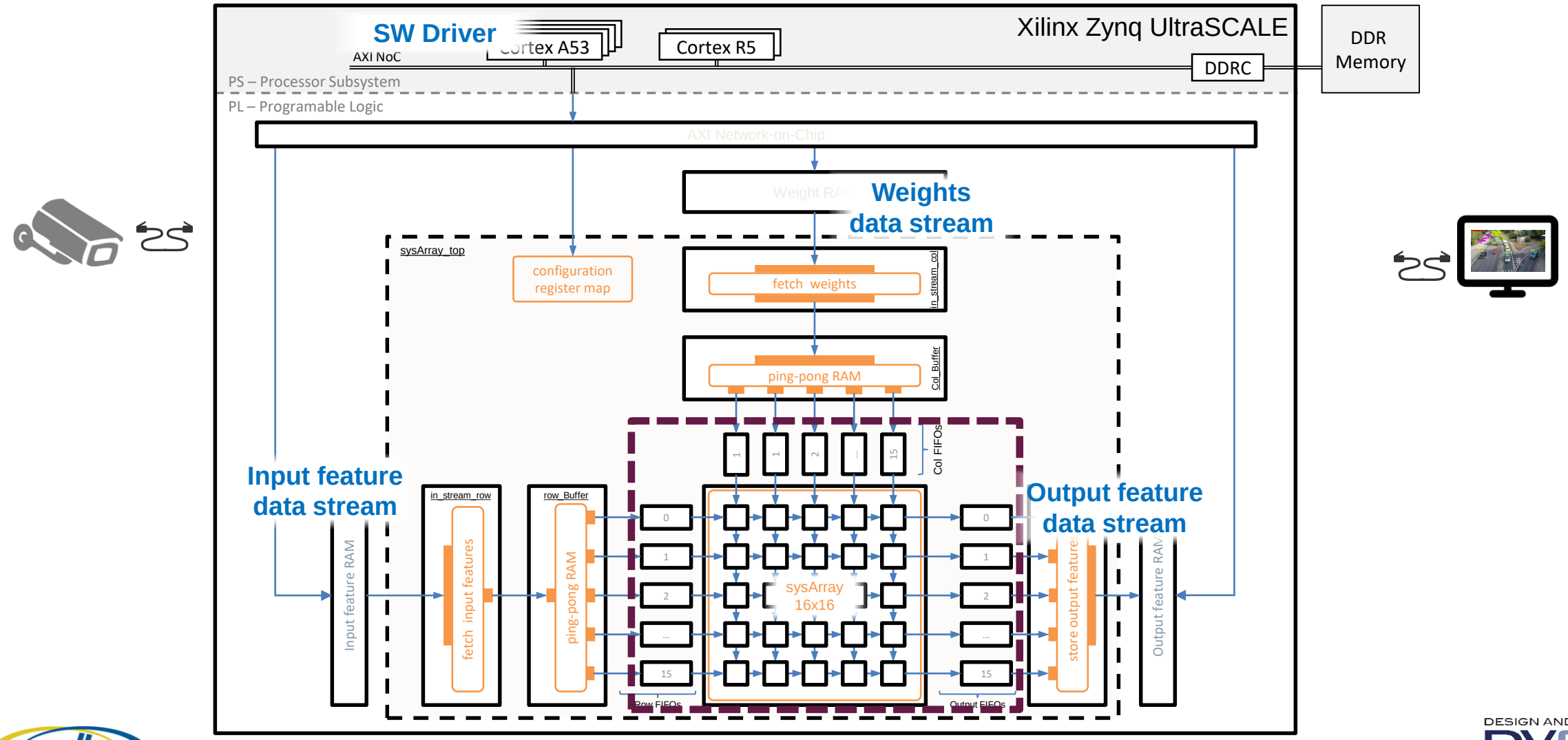
Processing the CNN

- Tiling the layers
- Managing the data to be transferred back and forth
- Non-convolution layers

Post-processing

- Map bounding-boxed into frames
- Scale resolution to be displayed

A complex architecture embeds the Systolic Array Kernel



Performance Analysis for HLS Design Flow

1 Motivation for performance analysis

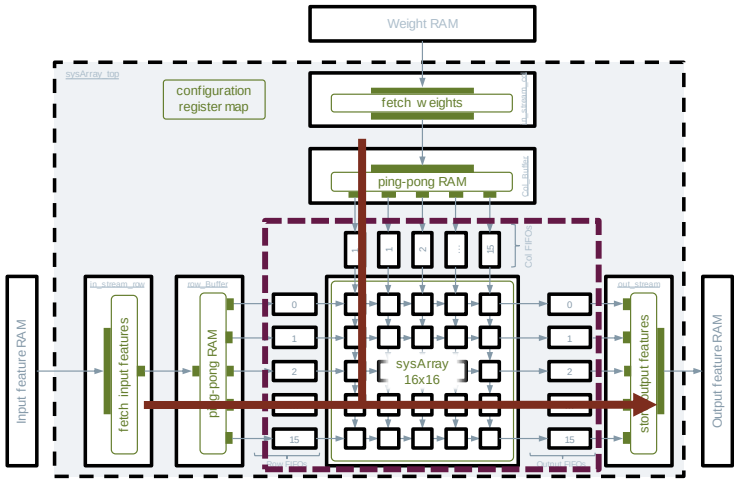
- Introduction to use case
- Module/kernel level vs system level performance aspects

2 Module/kernel level performance analysis

3 System level performance analysis

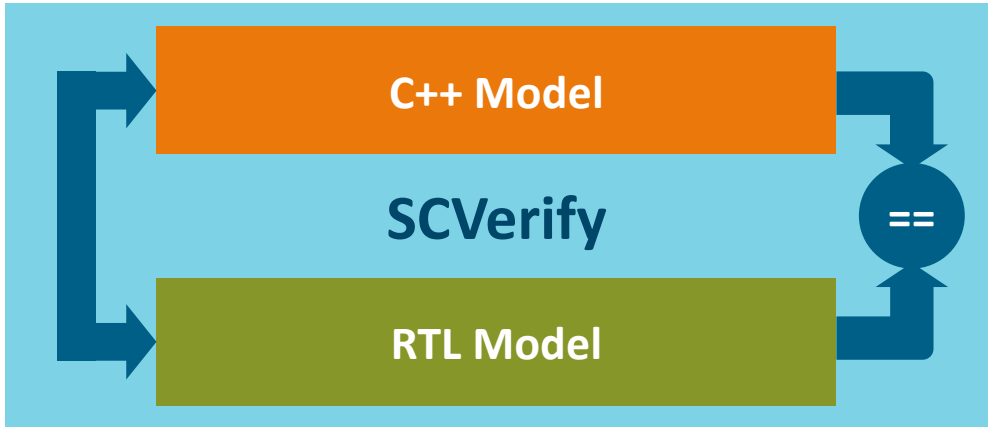
4 Summary

SCVerify automates verification of implementation and performance analysis



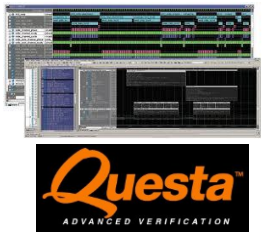
Functional Verification

- to verify functional correctness of accelerator kernel algorithm
- at high simulation performance

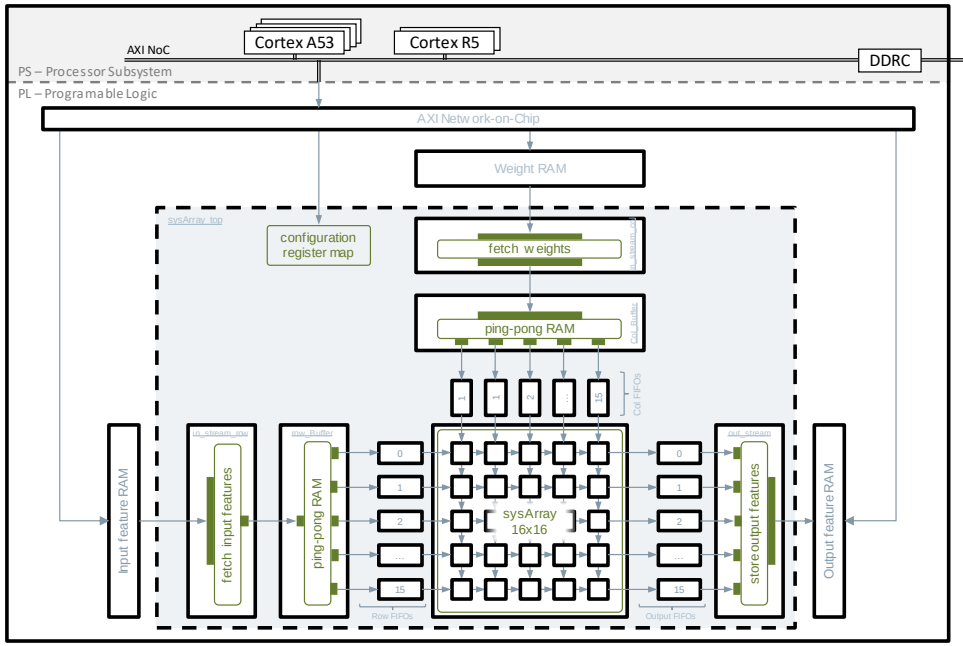


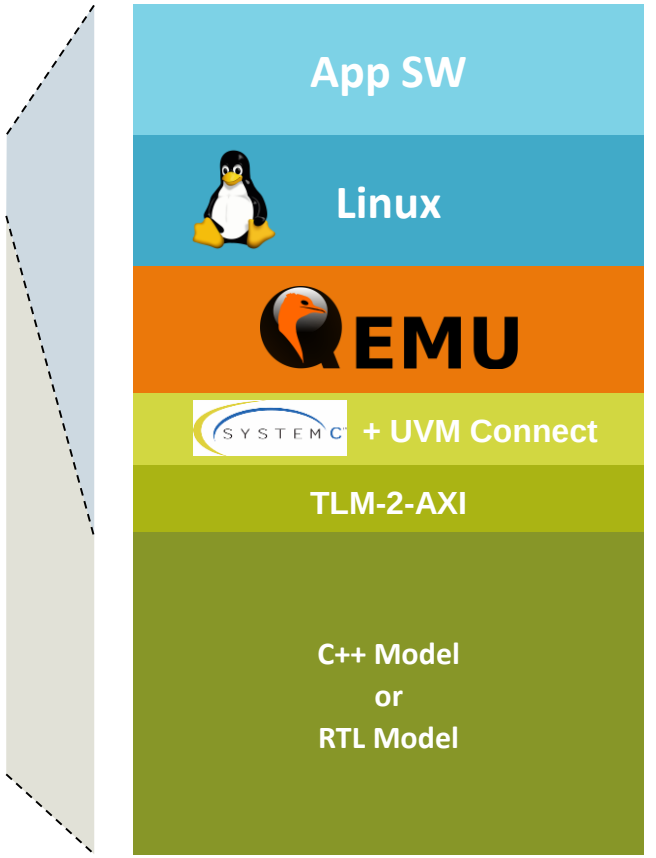
Verification of Implementation

- to verify non-functional correctness on top of functional correctness
- verification of “kernel-performance”



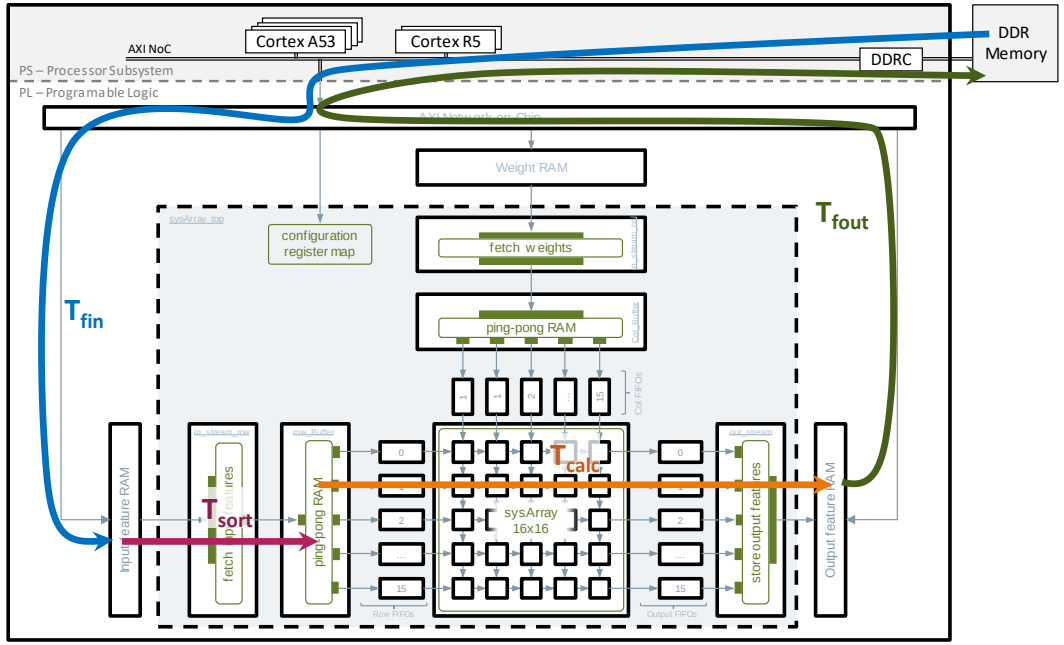
System-level models integrating the Systolic Array kernel uncover potential parallelism and bottlenecks

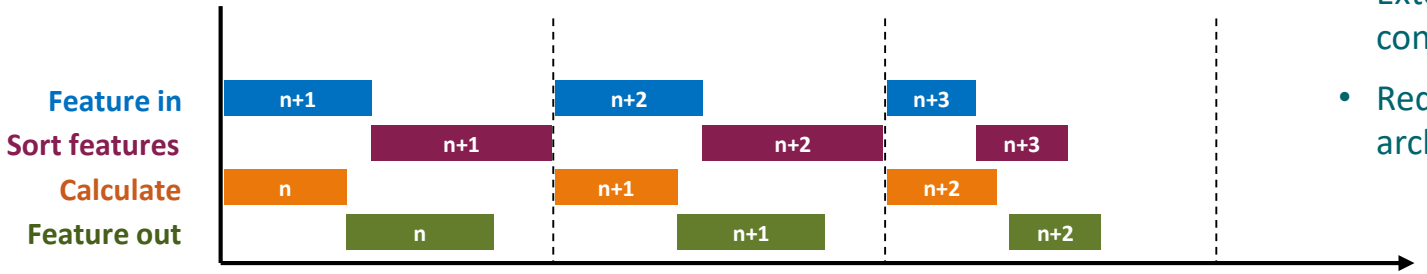




Questa™ ADVANCED VERIFICATION

Performance of individual paths vary to some extent depending on layer configuration of CNN





Detailed analysis of performance of individual paths

- Clock cycle accurate for RTL
- Full visibility into all details of “HLS-part” in RTL
- Final performance of implementation only available after Place&Route
- Approximate timing for compute subsystem modeled in QEMU

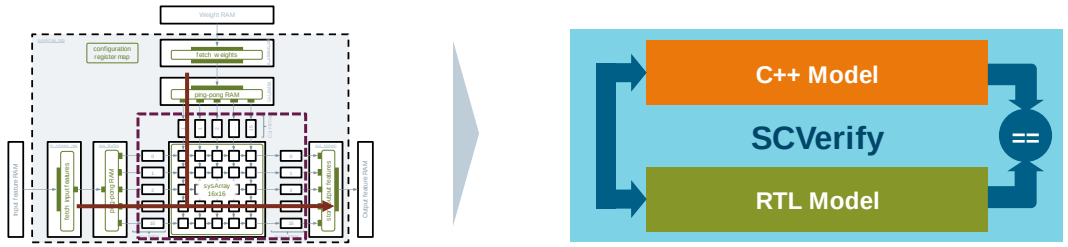
Identification and optimization of parallel paths

- Considering modeling uncertainties of high-level models
- Extending with layer-based scaling approach for all configurations
- Reduced order model as input for automated architecture exploration (design space exploration)

Some performance figures for simulation of a “2-tiles-sequence”

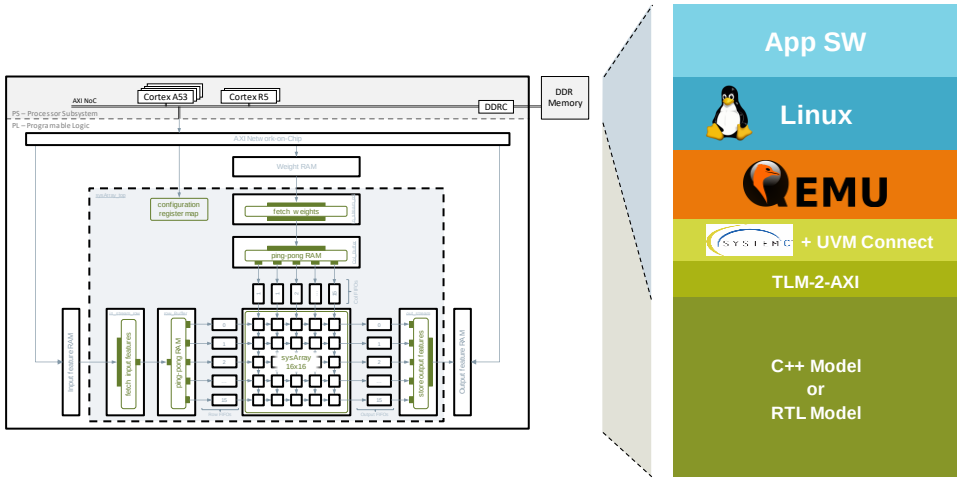
Module-level simulation performance is quite high

- 10min simulation for 4ms real-time
- ~150.000x slower than real-time



System-level simulation performance

- 1h48min for 119ms real-time
- ~6.500.000x slower than real-time
- ~40x slower than module-level simulation



Performance Analysis for HLS Design Flow

1 Motivation for performance analysis

- Introduction to use case
- Module/kernel level vs system level performance aspects

2 Module/kernel level performance analysis

3 System level performance analysis

4 Summary

Key takeaways

1

- System performance depends on module-level performance and system integration.
- **Multi-disciplinary** (HW, SW and application) **expertise** is required!
- Either **micro-architecture expertise** or optimized **reference designs/libraries** crucial for HLS implementation flows.

2

- Abstract system-level simulation models are key for analysis and optimization of system integration and parallelism in heterogeneous compute systems

accellera SYSTEMS INITIATIVE

2019 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION EUROPE

What is MatchLib?

- **M**odular **A**pproach **T**o **C**ircuits and **H**ardware **Lib**rary
- Developed by nVidia Labs while creating a machine learning accelerator
  - Needed a more abstract method for simulating system behavior
  - Needed to be able to closely (but not exactly) model performance
- Needed to evaluate many different architectures for performance and power
  - Could not afford to design them all in RTL
  - Could not afford to be significantly wrong

What is MatchLib?

- Library of reusable models and functions
  - Encapsulate verified functionality
  - Encapsulate QoR optimized implementation
  - Heavy use of templates and parameterization
- Common HW components modeled as
  - C++ functions: datapath description
  - C++ classes: state updating methods
  - SystemC modules: self contained modules
- Testbench components

MatchLib Addresses Complexity and Risk

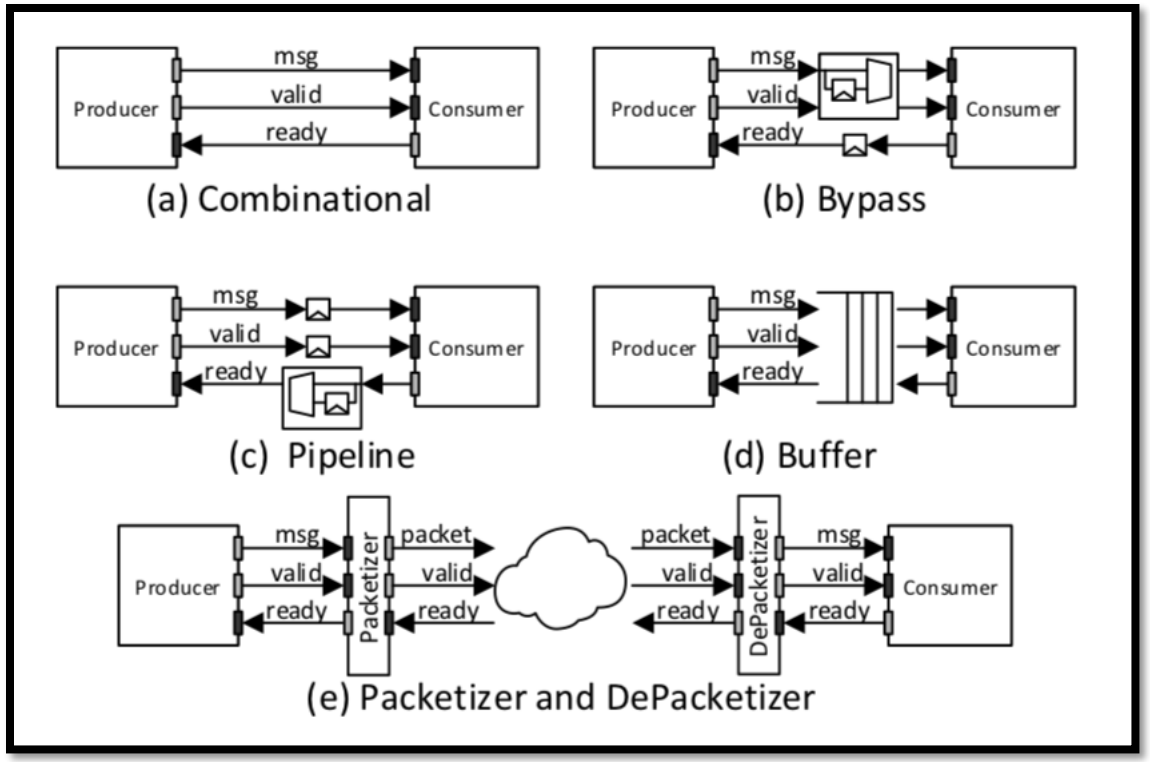
- The complexity/risk in many of today's advanced HW designs has shifted from the past.
- Today's HW designs often process huge sets of data, with large intermediate results.
  - Machine Learning
  - Computer Vision
  - 5G Wireless
- The design of the memory/interconnect architecture and the management of data movement in the system often has more impact on power/performance than the design of the computation units themselves.
- Evaluating and verifying memory/interconnect architecture at RTL level is not feasible:
  - Too late in design cycle
  - Too much work to evaluate multiple candidate architectures.
- The most difficult/costly HW (& HW/SW) problems are found during system integration.
  - If integration first occurs in RTL, it is very late and problems are very costly.
  - MatchLib lets integration occur early when fixing problems is much cheaper.

Key Parts of MatchLib

- "Connections"
  - Synthesizable Message Passing Framework
  - SystemC/C++ used to accurately model concurrent IO that synthesized HW will have
  - Automatic stall injection enables interconnect to be stress tested at C++ level
- Parameterized AXI4 Fabric Components
  - Router/Splitter
  - Arbiter
  - AXI4 <-> AXI4Lite
  - Automatic burst segmentation and last bit generation
- Parameterized Banked Memories, Crossbar, Reorder Buffer, Cache
- Parameterized NOC components

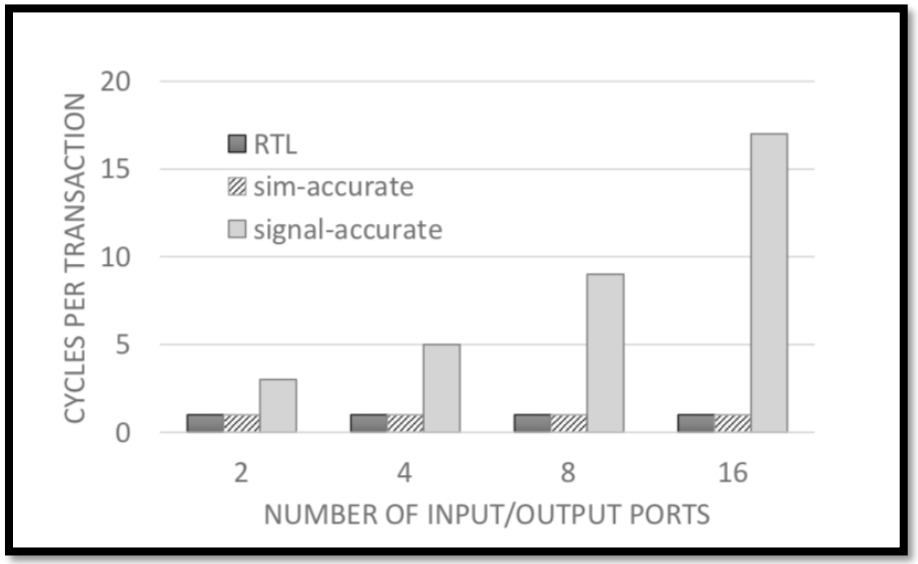
MatchLib Channels

- A set of classes for passing messages
- Channel types
  - Combinational
  - Bypass
  - Pipeline
  - Buffer
  - Network (NoC)
- Functions
  - Push(), PushNB()
  - Pop(), PopNB()



Timing Accuracy Across Abstractions

- One model for simulation, another for synthesis
  - To properly model timing, non-synthesizable constructs are needed
  - 2 implementations are used for the base classes
    - All protocols built on this will inherit these characteristics

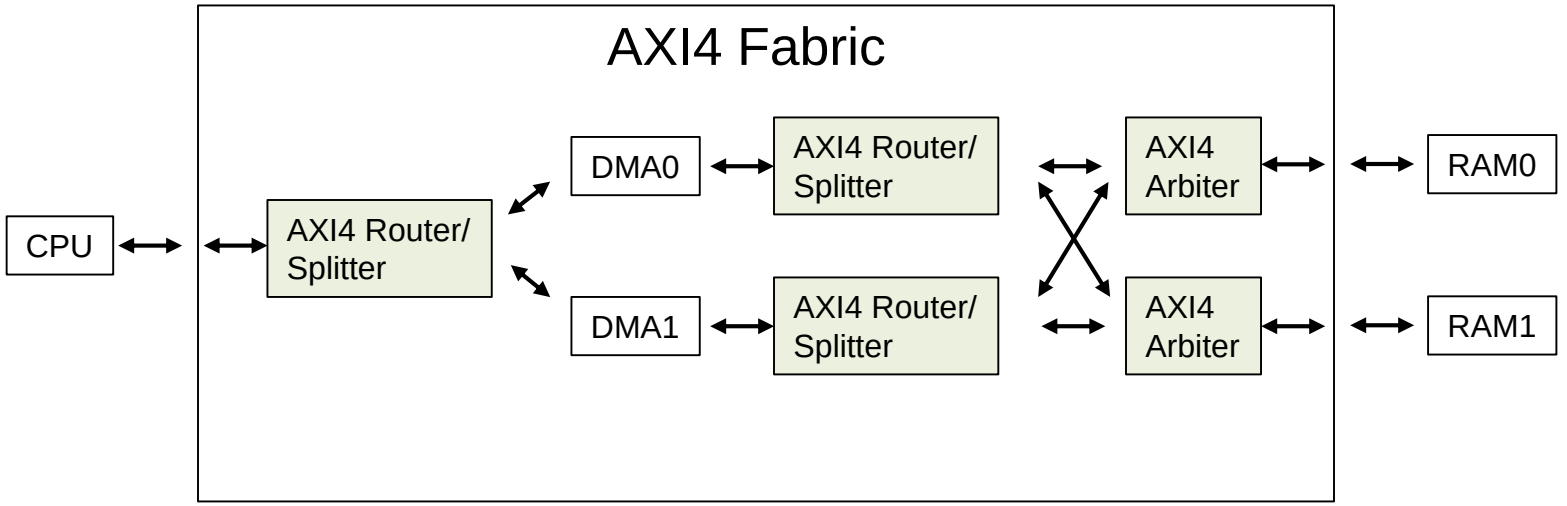


A Modular Digital VLSI Flow for High-Productivity SoC Design, DAC 2018, Khailany, et al

MatchLib AXI4

- Class that models the AXI-4 protocol using a combinatorial channel
- Configurable for
  - Width of address, data, ID, and user fields
  - Optional read response and “last” signal
- Access classes
  - axi::axi4<Cfg>::read::master and axi::axi4<Cfg>::read::slave
  - axi::axi4<Cfg>::write::master and axi::axi4<Cfg>::write::slave
- Current version only performs full bus-width accesses
  - We extended these class with read_xx and write_xx methods for partial bus width accesses

AXI4 Bus Fabric using Matchlib



Address Map

0x00000

0x7FFFF

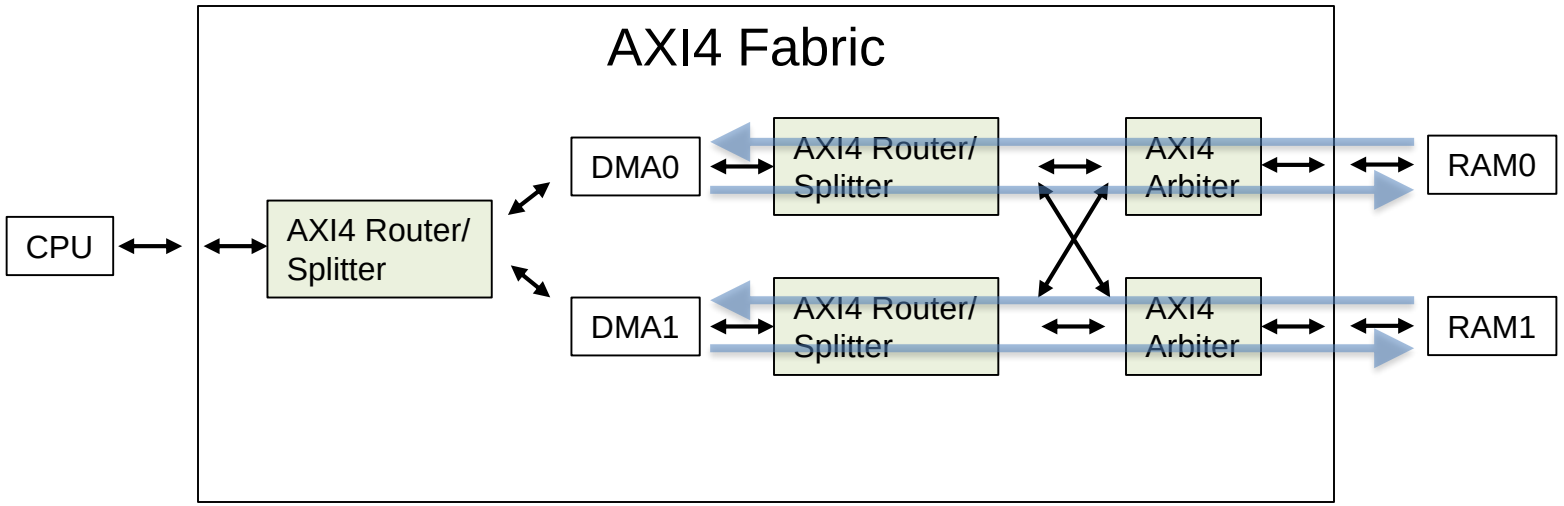
0x80000

0x8FFFF

```
/**
 *  * \brief fabric module
*/
#pragma hls_design top
class fabric : public sc_module, public local_axi {
public:
  sc_in<bool> INIT_S1(clk);
  sc_in<bool> INIT_S1(rst_bar);

  r_master INIT_S1(r_master0);
  w_master INIT_S1(w_master0);
  r_master INIT_S1(r_master1);
  w_master INIT_S1(w_master1);
  r_slave  INIT_S1(r_slave0);
  w_slave  INIT_S1(w_slave0);
  Connections::Out<bool> INIT_S1(dma0_done);
  Connections::Out<bool> INIT_S1(dma1_done);
```

Blue boxes are Matchlib Components

AXI4 Bus Fabric using Matchlib – Test #0



Test #0: Concurrently,
DMA0 reads/writes to RAM0
DMA1 reads/writes to RAM1

AXI4 Bus Fabric Test #0 Logs

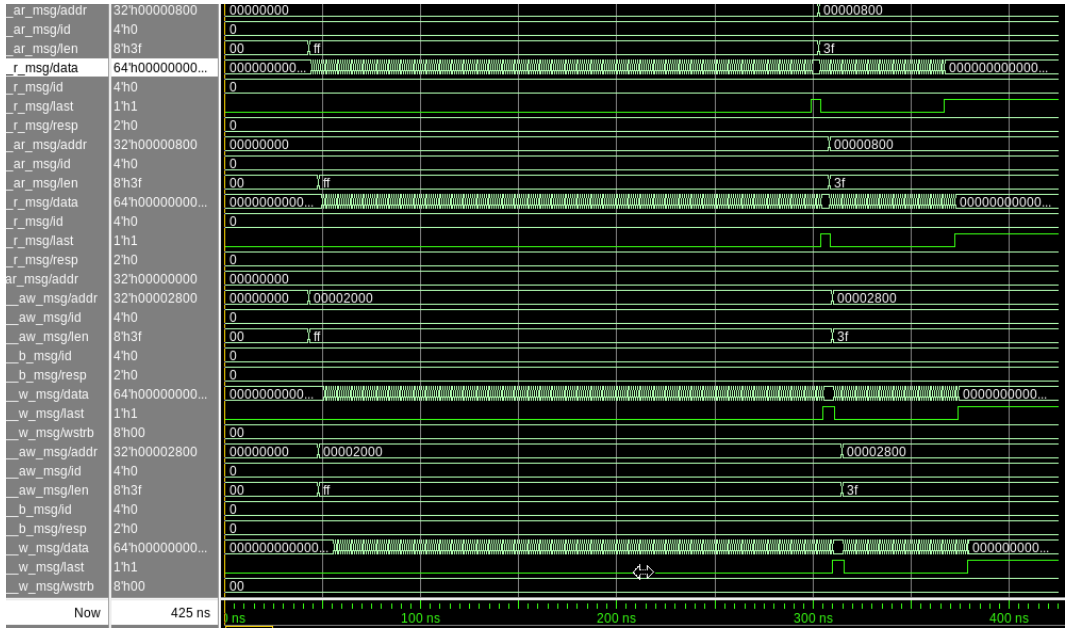
AS SystemC

```
0 s top Stimulus started
6 ns top Running FABRIC_TEST # : 0
44 ns top.ram0 ram read  addr: 000000000 len: 0ff
44 ns top.ram0 ram write addr: 000002000 len: 0ff
49 ns top.ram1 ram write addr: 000002000 len: 0ff
49 ns top.ram1 ram read  addr: 000000000 len: 0ff
304 ns top.ram0 ram read  addr: 000000800 len: 03f
309 ns top.ram1 ram read  addr: 000000800 len: 03f
311 ns top.ram0 ram write addr: 000002800 len: 03f
316 ns top.ram1 ram write addr: 000002800 len: 03f
385 ns top dma_done detected. 1 1
385 ns top start_time: 46 ns end_time: 385 ns
385 ns top axi beats (dec): 320
385 ns top elapsed time: 339 ns
385 ns top beat rate: 1059 ps
385 ns top clock period: 1 ns
425 ns top finished checking memory contents
```

AS RTL

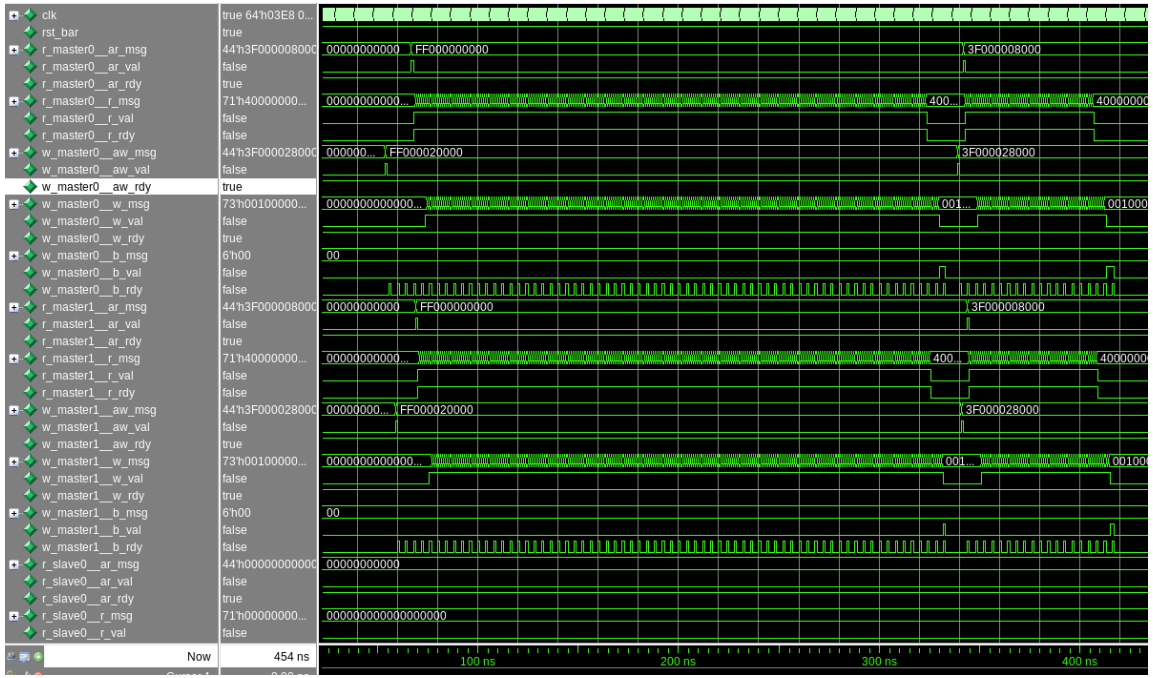
```
# 0 s top Stimulus started
# 6 ns top Running FABRIC_TEST # : 0
# 55 ns top/ram0 ram write addr: 000002000 len: 0ff
# 60 ns top/ram1 ram write addr: 000002000 len: 0ff
# 68 ns top/ram0 ram read  addr: 000000000 len: 0ff
# 70 ns top/ram1 ram read  addr: 000000000 len: 0ff
# 340 ns top/ram0 ram write addr: 000002800 len: 03f
# 342 ns top/ram1 ram write addr: 000002800 len: 03f
# 343 ns top/ram0 ram read  addr: 000000800 len: 03f
# 345 ns top/ram1 ram read  addr: 000000800 len: 03f
# 414 ns top dma_done detected. 1 1
# 414 ns top start_time: 55 ns end_time: 414 ns
# 414 ns top axi beats (dec): 320
# 414 ns top elapsed time: 359 ns
# 414 ns top beat rate: 1122 ps
# 414 ns top clock period: 1 ns
# 454 ns top finished checking memory contents
```

AXI4 Fabric Waveforms

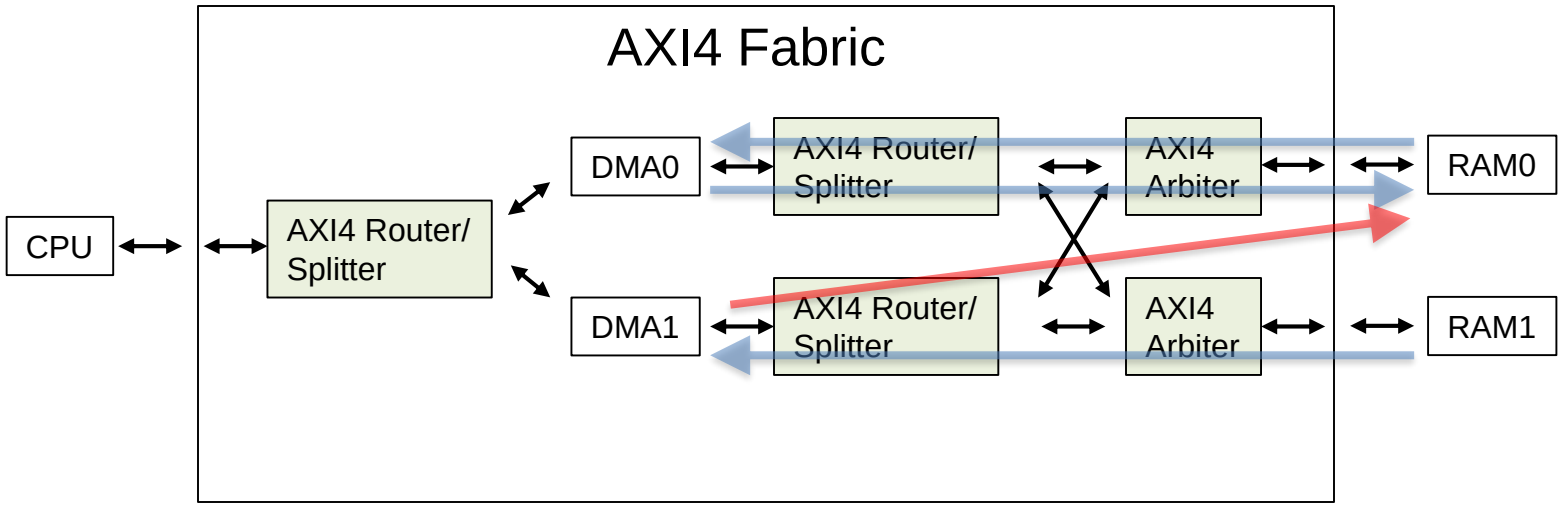


SystemC

RTL



AXI4 Bus Fabric using Matchlib – Test #1



Test #1: Concurrently,
DMA0 reads/writes to RAM0
DMA1 reads from RAM1 and writes to RAM0
Note contention on RAM0 writes

AXI4 Bus Fabric Test #1 Logs

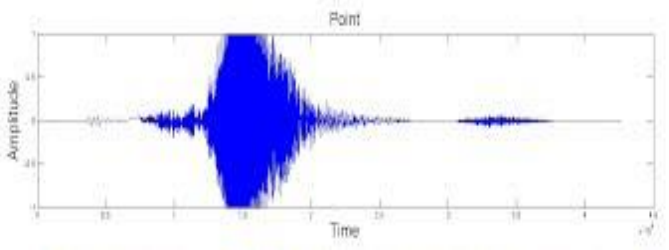
```
As SystemC
0 s top Stimulus started
6 ns top Running FABRIC_TEST # : 1
44 ns top.ram0 ram read  addr: 000000000 len: 0ff
44 ns top.ram0 ram write addr: 000002000 len: 0ff
49 ns top.ram1 ram read  addr: 000000000 len: 0ff
304 ns top.ram0 ram read  addr: 000000800 len: 03f
308 ns top.ram0 ram write addr: 000006000 len: 0ff
560 ns top.ram1 ram read  addr: 000000800 len: 03f
566 ns top.ram0 ram write addr: 000002800 len: 03f
632 ns top.ram0 ram write addr: 000006800 len: 03f
701 ns top dma_done detected. 1 1
701 ns top start_time: 46 ns end_time: 701 ns
701 ns top axi beats (dec): 320
701 ns top elapsed time: 655 ns
701 ns top beat rate: 2047 ps
701 ns top clock period: 1 ns
741 ns top finished checking memory contents
```

```
As RTL
# 0 s top Stimulus started
# 6 ns top Running FABRIC_TEST # : 1
# 55 ns top/ram0 ram write addr: 000002000 len: 0ff
# 68 ns top/ram0 ram read  addr: 000000000 len: 0ff
# 70 ns top/ram1 ram read  addr: 000000000 len: 0ff
# 335 ns top/ram0 ram write addr: 000006000 len: 0ff
# 343 ns top/ram0 ram read  addr: 000000800 len: 03f
# 598 ns top/ram1 ram read  addr: 000000800 len: 03f
# 598 ns top/ram0 ram write addr: 000002800 len: 03f
# 670 ns top/ram0 ram write addr: 000006800 len: 03f
# 736 ns top dma_done detected. 1 1
# 736 ns top start_time: 55 ns end_time: 736 ns
# 736 ns top axi beats (dec): 320
# 736 ns top elapsed time: 681 ns
# 736 ns top beat rate: 2128 ps
# 736 ns top clock period: 1 ns
# 776 ns top finished checking memory contents
```

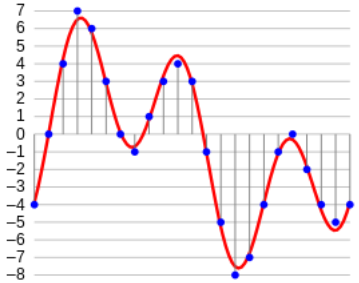
Wake Word

- Microphone ”listens” for set of phrases that will turn on complete system for user interaction
  - E.g. “Hey Google!” or “Alexa”
- Needs to be very low power for battery powered systems, as it runs continuously
- Example system:
  - Processes 1 second of audio data
    - 16,000 samples per second
    - Processes a rolling sample every 20 milliseconds
  - Performs an MFCC to get a spectral signature of the audio sample
    - **M**el-**F**requency **C**epstrum **C**oefficients, energy levels of human audible frequencies
  - Uses machine learning techniques to match sample against set of 10 known keywords

Wake Word Audio Pre-processing



Audio input

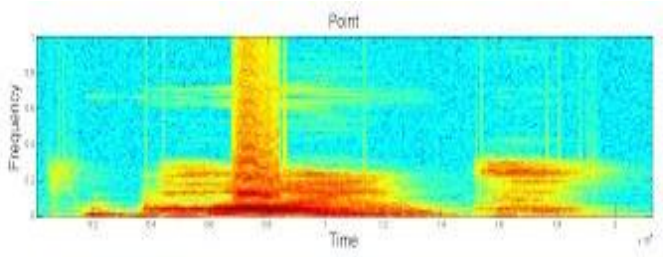


Quantization

-4	0	4	7	

Integer array
(16k x 16 bits)

MFCC()



Spectral Array

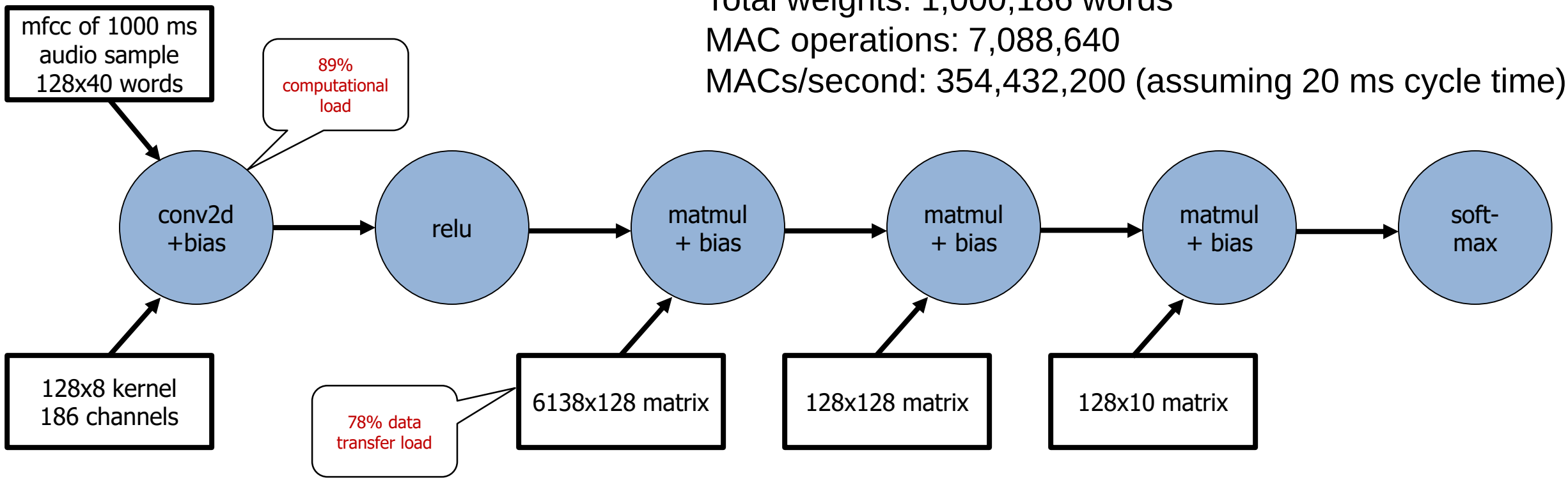
0.123	0.456	-0.872	0.567	
0.324	0.547	0.376	-0.231	
0.846	0.183	0.834	0.937	
0.625	0.737	0.746	0.827	

Float array
(128 x 40 x 32-bits)

To Neural Network
as feature map for
training and inferencing

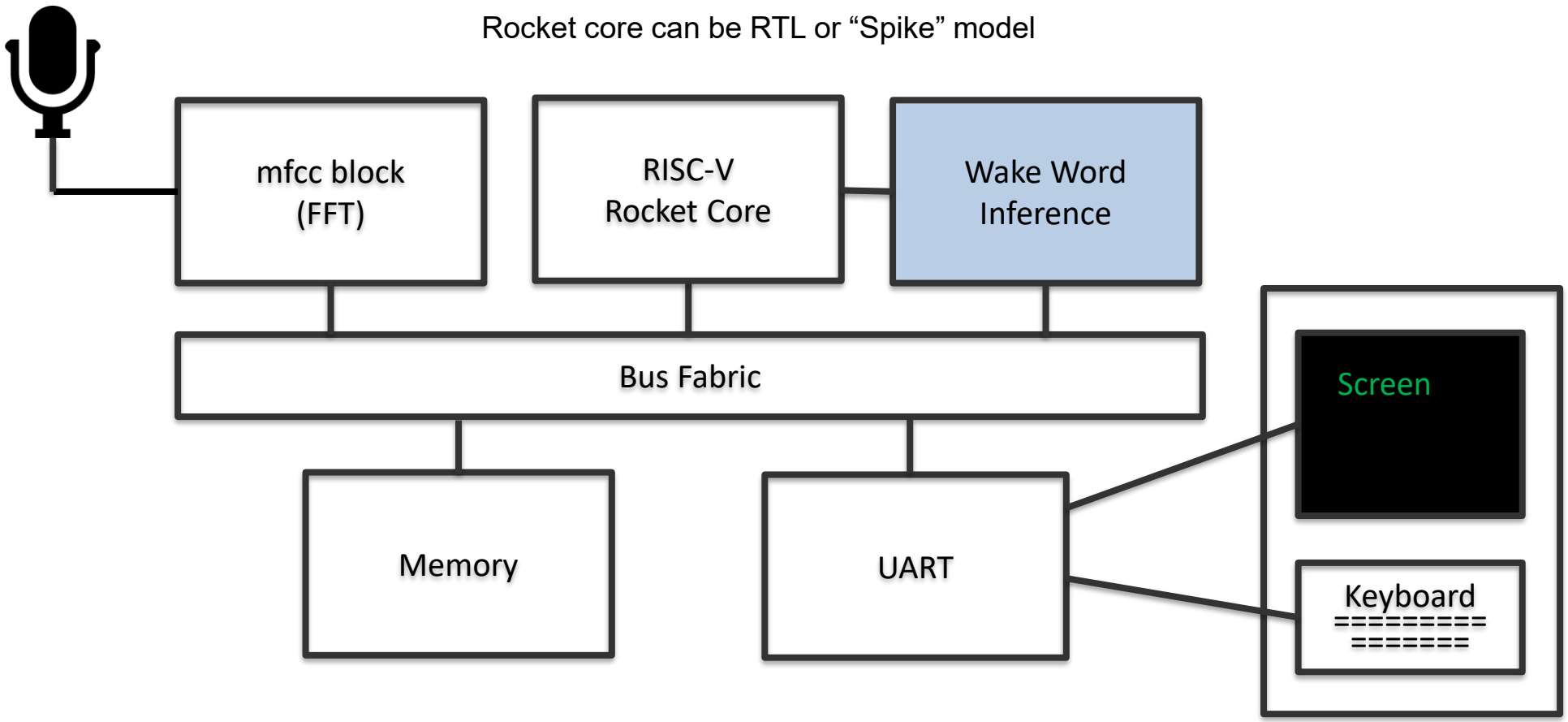
Wake Word Neural Network

Total weights: 1,000,186 words
MAC operations: 7,088,640
MACs/second: 354,432,200 (assuming 20 ms cycle time)



'cnn-one-fstride4' from 'Convolutional Neural Networks for Small-footprint Keyword Spotting':
http://www.isca-speech.org/archive/interspeech_2015/papers/i15_1478.pdf

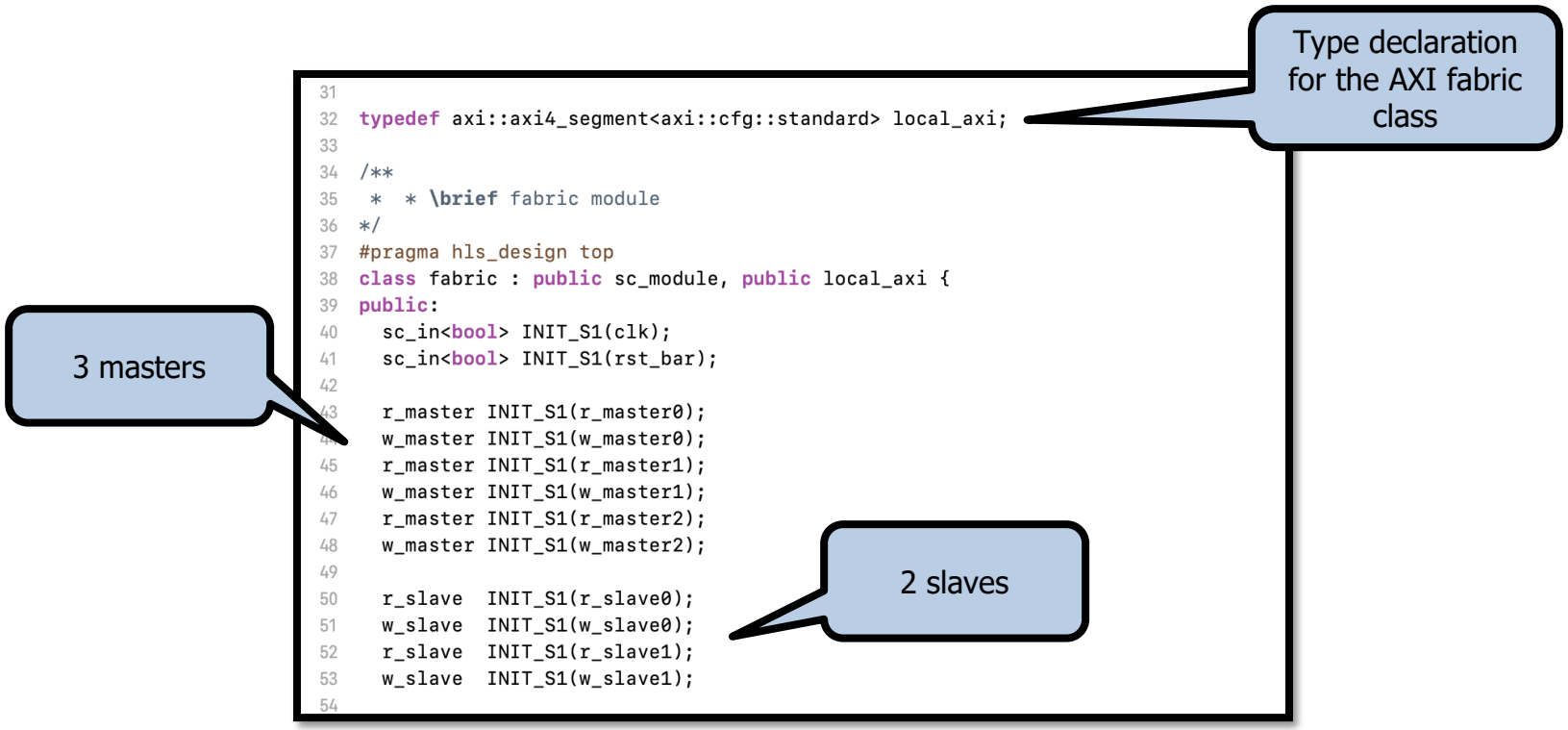
Wake Word Design



System Modeling

- Stimulus
  - Pre-sampled waveform of 2 minutes (6000 inferences)
- Not modeling mfcc(), other than calling a C function
  - A complete analysis of mfcc warrants a complete tutorial on it's own
  - Assumed to be instantaneous and zero power ☺
- System C models for
  - Bus Fabric – using MatchLib connections for AXI4
  - Accelerator
  - Memory
- C models for
  - Rocket core (SPIKE)
  - UART
  - MFCC and audio input (with System C interface to put spectral data into system memory)

Bus Fabric Declaration



Top Level Design

```
1
2  #include <systemc.h>
3  #include <mc_scverify.h>
4
5  #include "fabric.h"
6  #include "ram.h"
7  #include "uart.h"
8  #include "spectral.h"
9  #include "inference.h"
10
11 #define NVHLS_VERIFY_BLOCKS (fabric)
12 #include <nvhls_verify.h>
13 #include <mc_trace.h>
14 using namespace::std;
15 #include <testbench/Pacer.h>
16
17 typedef axi::axi4_segment<axi::cfg::standard> local_axi;
18
19 class Top : public sc_module, public local_axi {
20 public:
21
22   // slaves
23
24   ram                  INIT_S1(memory);
25   uart                 INIT_S1(terminal);
26
27   NVHLS_DESIGN(fabric) INIT_S1(fabric1);
28
29   sc_clock clk;
30   sc_event frame_event;
31   sc_event data_ready;
```

Masters

Slaves

Fabric Instantiation

```
102
103     SC_CTHREAD(reset, clk);
104
105     SC_THREAD(mfcc);
106     sensitive << clk.posedge_event();
107     async_reset_signal_is(rst_bar, false);
108
109     SC_THREAD(neural_network);
110     sensitive << clk.posedge_event();
111     async_reset_signal_is(rst_bar, false);
112
113     SC_THREAD(rocket_core);
114     sensitive << clk.posedge_event();
115     async_reset_signal_is(rst_bar, false);
116
117     SC_THREAD(frame_timer);
118     sensitive << clk.posedge_event();
119
120     sc_object_tracer<sc_clock> trace_clk(clk);
121   }
122
```

Neural Network

```
378
379   void neural_network() {
380
381     feature_type probabilities[WORDS];
382     tb_w_master1.reset();
383     tb_r_master1.reset();
384
385     wait();
386
387     while (1) {
388         wait(data_ready);
389
390         LOG("inference started");
391         compute_inference(feature_map, weight_data, probabilities);
392         for (int i=0; i<WORDS; i++) {
393            printf("prob[%d]: %f \n", i, probabilities[i] * 100.0);
394         }
395         LOG("inference completed");
396     }
397   }
398
```

Compute Inference

```
310
311   void compute_inference(feature_type *feature_map, weight_type *weights, feature_type *probabilities)
312   {
313     weight_memory   *wm = (weight_memory *)  FEATURE_MAP_BASE + sizeof(feature_memory);
314     scratch_memory  *sm = (scratch_memory *) weight_memory + sizeof(weight_memory);
315
316     conv2d(feature_map, wm->convolution_filter, sm->conv_out);
317
318     bias_add(sm->conv_out, wm->bias_0, sm->bias0_out);
319     relu((sm->bias0_out, sm->relu_out);
320
321     matmul(sm->relu_out, wm->factor_1, sm->matmul1_out);
322     bias_add(sm->matmul1_out, wm->bias_1, sm->bias1_out);
323
324     matmul(sm->bias1_out, wm->factor_2, sm->matmul2_out);
325     bias_add(sm->matmul2_out, wm->bias_2, sm->bias2_out);
326
327     matmul(sm->bias2_out, wm->factor_3, sm->matmul3_out);
328     bias_add(sm->matmul3_out, wm->bias_3, sm->bias3_out);
329
330     softmax(sm->bias3_out, probabilities);
331   }
332
```

Weights, feature map and intermediate results all in one memory instance

Memory Map Struct Overlays

```
62
63 //------------------------------------------------------------//
64 //  Memory layouts
65 //------------------------------------------------------------//
66
67
68 typedef struct weight_struct {
69     weight_type     convolution_filter [FILTER_CHANS][FILTER_ROWS][FILTER_COLS];
70     weight_type     factor_1          [F1_ROWS][F1_COLS];
71     weight_type     factor_2          [F2_ROWS][F2_COLS];
72     weight_type     factor_3          [F3_ROWS][F3_COLS];
73     weight_type     bias_0            [B0_ROWS][B0_COLS];
74     weight_type     bias_1            [B1_ROWS][B0_COLS];
75     weight_type     bias_2            [B2_ROWS][B0_COLS];
76     weight_type     bias_3            [B3_ROWS][B0_COLS];
77 } weight_memory;
78
79 typedef struct feature_struct {
80     feature_type    features          [CHANNELS][SPECTRA][SAMPLES];
81 } feature_memory;
82
83 typedef struct scratch_struct {
84     feature_type    conv_out          [CONV_OUT_CHANS][CONV_OUT_ROWS][CONV_OUT_COLS];
85     feature_type    relu_out          [R1_ROWS][R1_COLS];
86     feature_type    bias0_out         [B0_ROWS][B0_COLS];
87     feature_type    bias1_out         [B1_ROWS][B1_COLS];
88     feature_type    bias2_out         [B2_ROWS][B2_COLS];
89     feature_type    bias3_out         [B3_ROWS][B3_COLS];
90     feature_type    matmul1_out       [P1_ROWS][P1_COLS];
91     feature_type    matmul2_out       [P2_ROWS][P2_COLS];
92     feature_type    matmul3_out       [P3_ROWS][P3_COLS];
93 } scratch_memory;
94
95
96 //------------------------------------------------------------//
```

Memory Accesses

Memory access routines, drives AXI bus cycles

```
122
123   weight_type read_weight(weight_type *addr)
124   {
125       r_payload r;
126
127       r = tb_r_master1.read_32((char *) addr);
128       return r.data.to_int();
129   }
130
131   feature_type read_feature(feature_type *addr)
132   {
133       r_payload r;
134
135       r = tb_r_master1.read_32((char *) addr);
136       return r.data.to_int();
137   }
138
139   void write_feature(feature_type *addr, feature_type val)
140   {
141       tb_w_master1.write_32((char *) addr, val);
142   }
143
144
```

Function that accesses AXI memory

```
236
237   void bias_add(
238       feature_type    *a,
239       weight_type     *b,
240       feature_type    *sum,
241       int             size
242      )
243   {
244       int n;
245
246       for (n=0; n<size; n++) {
247           //sum[n] = a[n] + b[n];
248           write_feature(sum+n, read_feature(a+n) + read_weight(b+n));
249       }
250   }
251
```

Naïve Implementation

- Take a software implementation and directly convert it to SystemC
  - No accommodation for data flows or caching
  - No pipelining or explicit parallelism
- Each inference runs 2.5 million AXI transactions
  - Not burst transactions, one word access per bus cycle
  - Does not use full width of data bus
- Requires 2.9 GHz to do real-time inferencing
- Solution:
  - Get 16 adjacent data elements at a time and cache locally for computations

Cached Operation

```
236
237  void bias_add(
238      feature_type    *a,
239      weight_type     *b,
240      feature_type    *sum,
241      int             size
242     )
243  {
244      int n;
245
246      for (n=0; n<size; n++) {
247          //sum[n] = a[n] + b[n];
248          write_feature(sum+n, read_feature(a+n) + read_weight(b+n));
249      }
250  }
251
```

Original code

Change needs to be done for all operations, convolution, matrix multiply, etc.

```
236
237  #define CACHE_SIZE 16
238
239    void bias_add(
240        feature_type    *a,
241        weight_type     *b,
242        feature_type    *sum,
243        int             size
244       )
245    {
246        int n, m;
247        feature_type a_cache[CACHE_SIZE];
248        weight_type  b_cache[CACHE_SIZE];
249        feature_type sum_cache[CACHE_SIZE];
250
251        for (n=0; n<size; n+=CACHE_SIZE) {
252            load_cache(a+n, a_cache, CACHE_SIZE);
253            load_cache(b+n, b_cache, CACHE_SIZE);
254  #pragma hls loop_unroll
255            for (m=0; m<CACHE_SIZE; m++) {
256               sum_cache[m] = a_cache[m] + b_cache[m];
257            }
258            store_cache(sum_cache, sum+n, CACHE_size);
259        }
260    }
261
```

No change to function signature

Local buffers

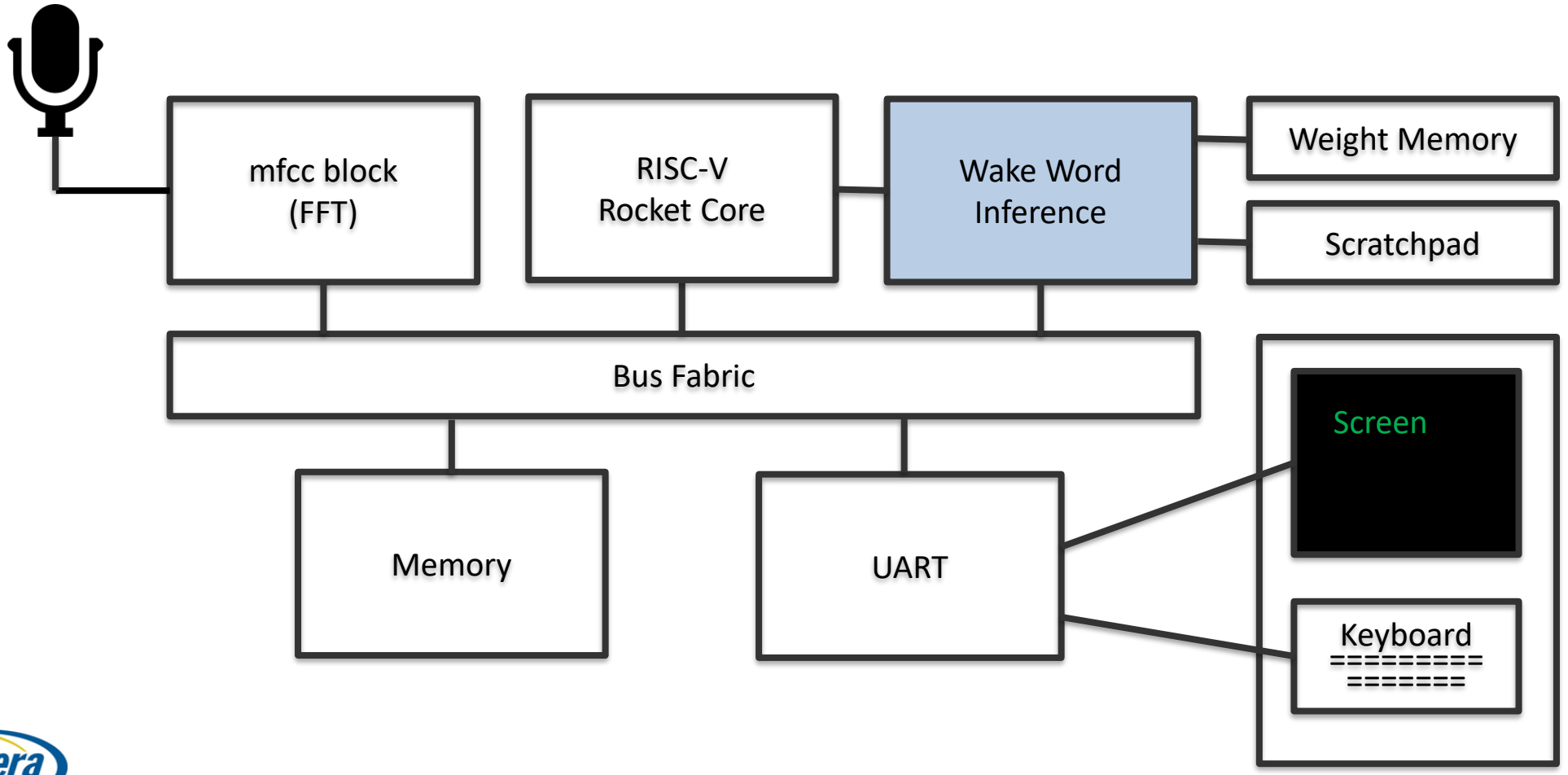
No data or data-path dependency

Cached version

Cached Operation

- Instead of moving each data word as it is needed, a burst of 16 words is performed and it is cached locally
- Improves performance significantly
  - Approximately 19X faster, from improved bus utilization
- Memory is the bottleneck
  - Features, weights, and intermediate results are all stored in AXI memory
  - 64 bits per clock is maximum data movement
- Solution:
  - Move intermediate results and weight memory off AXI bus to local connection to accelerator
  - With no arbitration, expanding memory width is not too expensive

Wake Word Design



Instantiate Additional Memories

```
16
17 typedef axi::axi4_segment<axi::cfg::standard> local_axi;
18
19 class Top : public sc_module, public local_axi {
20 public:
21
22   // slaves
23
24   ram                  INIT_S1(memory);
25   ram                  INIT_S1(weight_memory);
26   ram                  INIT_S1(scratch_pad);
27   uart                 INIT_S1(terminal);
28
29   NVHLS_DESIGN(fabric) INIT_S1(fabric1);
30
31   sc_clock clk;
32   sc_event frame_event;
33   sc_event data_ready;
34   SC_SIG(bool, rst_bar);
```

Weight memory and scratchpad Not connected to AXI bus

Define Memory Regions

```
310
311   void compute_inference(feature_type *feature_map, weight_type *weights, feature_type *
312   {
313     weight_memory   *wm = (weight_memory *)  FEATURE_MAP_BASE + sizeof(feature_memory);
314     scratch_memory  *sm = (scratch_memory *) weight_memory + sizeof(weight_memory);
315
316     conv2d(feature_map, wm->convolution_filter, sm->conv_out);
317
318     bias_add(sm->conv_out, wm->bias_0, sm->bias0_out);
319     relu((sm->bias0_out, sm->relu_out);
320
321     matmul(sm->relu_out, wm->factor_1, sm->matmul1_out);
322     bias_add(sm->matmul1_out, wm->bias_1, sm->bias1_out);
323
324     matmul(sm->bias1_out, wm->factor_2, sm->matmul2_out);
325     bias_add(sm->matmul2_out, wm->bias_2, sm->bias2_out);
326
327     matmul(sm->bias2_out, wm->factor_3, sm->matmul3_out);
328     bias_add(sm->matmul3_out, wm->bias_3, sm->bias3_out);
329
330     softmax(sm->bias3_out, probabilities);
331   }
332
```

Original code

Memory regions now independent

```
322
323   void compute_inference(feature_type *feature_map, weight_type *weights, fe
324   {
325     weight_memory   *wm = (weight_memory *)  WEIGHT_MEM_BASE;
326     scratch_memory  *sm = (scratch_memory *) SCRATCH_PAD_BASE;
327
328     conv2d(feature_map, wm->convolution_filter, sm->conv_out);
329
330     bias_add(sm->conv_out, wm->bias_0, sm->bias0_out);
331     relu((sm->bias0_out, sm->relu_out);
332
333     matmul(sm->relu_out, wm->factor_1, sm->matmul1_out);
334     bias_add(sm->matmul1_out, wm->bias_1, sm->bias1_out);
335
336     matmul(sm->bias1_out, wm->factor_2, sm->matmul2_out);
337     bias_add(sm->matmul2_out, wm->bias_2, sm->bias2_out);
338
339     matmul(sm->bias2_out, wm->factor_3, sm->matmul3_out);
340     bias_add(sm->matmul3_out, wm->bias_3, sm->bias3_out);
341
342     softmax(sm->bias3_out, probabilities);
343   }
344
```

Local memories

Memory Access Routines

```
124
125   weight_load_cache(weight_type *src, weight_type *dst, int size)
126   {
127       ar_payload ar;
128       r_payload  r;
129       int i;
130
131       aw.len = size / (bytesPerBeat / sizeof(weight_type));
132       aw.addr = src;
133       weight_chan.Push(ar);
134       r = weight_chan.Pop();
135       for (i=0; i<size; i++) dst[i] = r.data[i];
136   }
137
```

Memory specific access function

Burst accesses

```
262
263 #define CACHE_SIZE 16
264
265   void bias_add(
266       feature_type    *a,
267       weight_type     *b,
268       feature_type    *sum,
269       int             size
270      )
271   {
272       int n, m;
273       feature_type a_cache[CACHE_SIZE];
274       weight_type  b_cache[CACHE_SIZE];
275       feature_type sum_cache[CACHE_SIZE];
276
277       for (n=0; n<size; n+=CACHE_SIZE) {
278          scratch_load_cache(a+n, a_cache, CACHE_SIZE);
279          weight_load_cache(b+n, b_cache, CACHE_SIZE);
280 #pragma hls loop_unroll
281          for (m=0; m<CACHE_SIZE; m++) {
282             sum_cache[m] = a_cache[m] + b_cache[m];
283          }
284          scratch_store_cache(sum_cache, sum+n, CACHE_size);
285       }
286   }
287
```

Reads now happen in parallel

Banked Memories

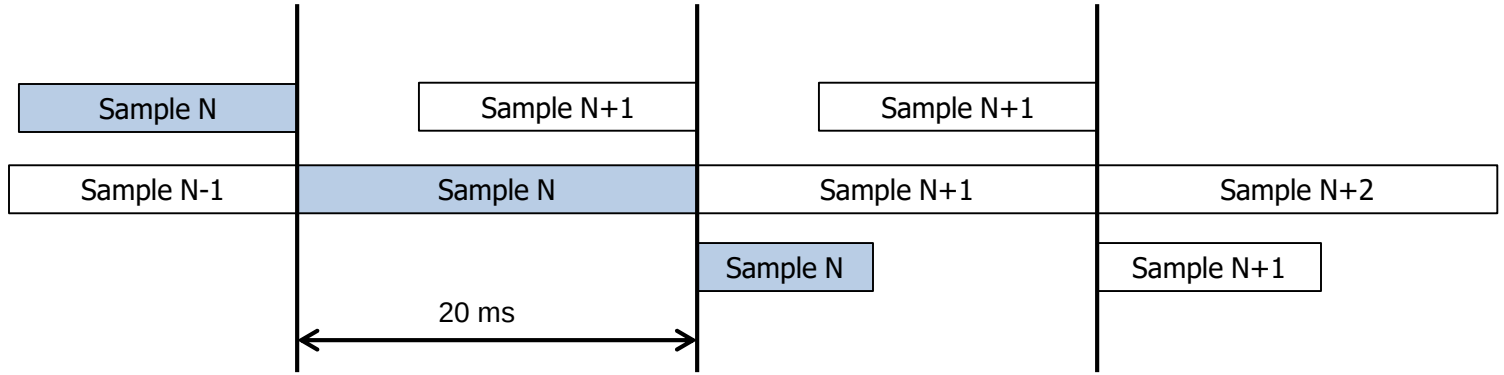
- Data for weights and intermediate results are accessed over wider buses, and in separate memories, not arbitrated
  - Performance ~6X faster
- Different widths of memories and size of caches can be explored for area/performance tradeoffs

- Further optimization can be achieved by structural pipelining of the algorithm

Create Separate Threads

- Convolution consumes 75% of the time for an inference
  - Can be pipelined with remaining calculations
- Move convolution to a separate thread
  - Separate threads enables HLS to synthesize pipelined implementation
  - Break weight and scratchpad memories into 2 separate regions to avoid contention
  - Create “ping-pong” buffers between convolution and matrix multiplies
  - Add sync signals between threads

mfcc()
compute convolution()
compute_matmul()



Pipelined Inferencing

```
346
347   SC_THREAD(compute_convolution);
348   sensitive << clk.posedge_event();
349   async_reset_signal_is(rst_bar, false);
350
351   SC_THREAD(compute_matmul);
352   sensitive << clk.posedge_event();
353   async_reset_signal_is(rst_bar, false);
354
```

Separate threads

Separate weight and scratch memories

```
355
356   void compute_convolution(feature_type *feature_map, feature_type *relu_out)
357   {
358     weight_memory   *wm = (weight_memory *)  WEIGHT_MEM_BASE_CONV;
359     scratch_memory  *sm = (scratch_memory *) SCRATCH_PAD_BASE_CONV;
360
361     while(1) {
362       wait(data_ready);
363
364       conv2d(feature_map, wm->convolution_filter, sm->conv_out);
365
366       bias_add(sm->conv_out, wm->bias_0, sm->bias0_out);
367       relu((sm->bias0_out, relu_out);
368
369       conv_complete.notify();
370     }
371   }
```

Synchronization events

```
372
373   void compute_matmul(feature_type *relu_in, feature_type *probabilities)
374   {
375     weight_memory   *wm = (weight_memory *)  WEIGHT_MEM_BASE_MATMUL;
376     scratch_memory  *sm = (scratch_memory *) SCRATCH_PAD_BASE_MATMUL;
377
378     while(1) {
379       wait(conv_complete);
380
381       matmul(relu_in, wm->factor_1, sm->matmul1_out);
382       bias_add(sm->matmul1_out, wm->bias_1, sm->bias1_out);
383
384       matmul(sm->bias1_out, wm->factor_2, sm->matmul2_out);
385       bias_add(sm->matmul2_out, wm->bias_2, sm->bias2_out);
386
387       matmul(sm->bias2_out, wm->factor_3, sm->matmul3_out);
388       bias_add(sm->matmul3_out, wm->bias_3, sm->bias3_out);
389
390       softmax(sm->bias3_out, probabilities);
391
392       inference_done.notify();
393   }
```

Summary of Results

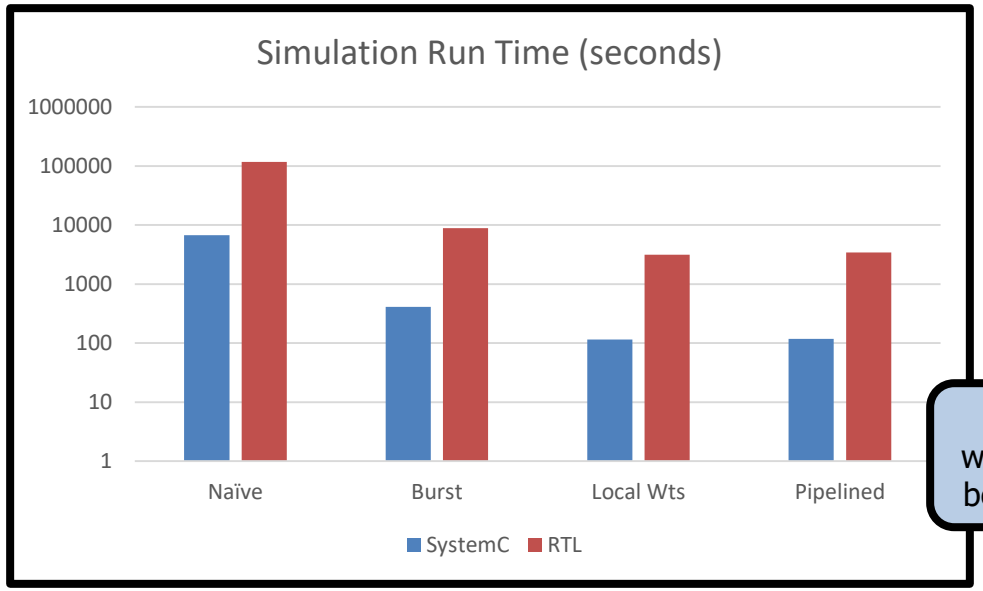
- All weights in main memory, naïve algorithm, no caching
  - ~2.9 GHz needed to perform real-time
- 16 word bursts, local caching of data
  - ~157 MHz
- Move coefficient data to local memory (local to inference block)
  - ~33 MHz
- Pipeline convolution with matrix multiplies
  - ~26 MHz
  - Latency increases from 20 ms to 27 ms

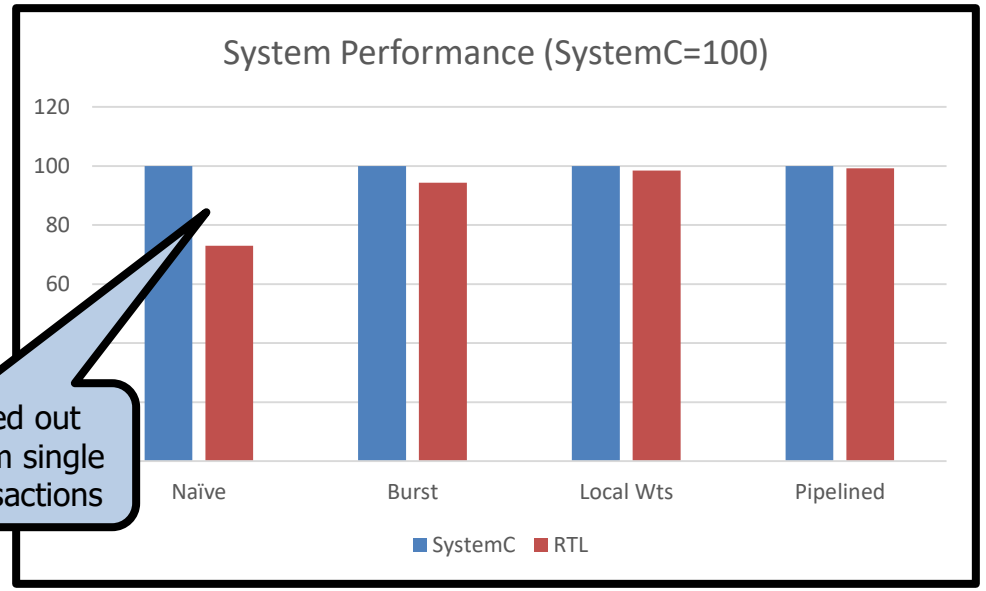
SystemC to RTL Synthesis

- High level synthesis converts abstract C or SystemC to synthesizable RTL
- MatchLib connection and AXI components are synthesizable through Catapult HLS compiler
  - Algorithmic code describing data transformations can be synthesized too
- Resulting RTL will closely match performance of the SystemC
  - MatchLib communications are clock cycle accurate
  - System is assumed to be I/O bound
- nVidia saw SystemC performance at +/- 3% compare to RTL
  - While simulation run-times were 30 times faster[1]

1 - A Modular Digital VLSI Flow for High-Productivity SoC Design, DAC 2018, Khailany, et al

Simulation Performance vs. RTL





HLS optimized out wait state from single beat AXI transactions

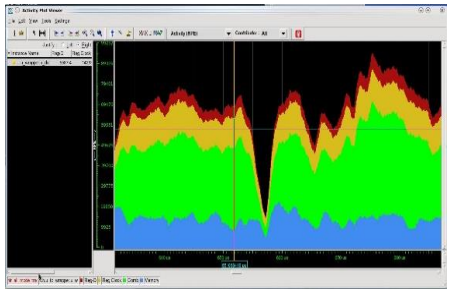
SystemC is 23.9 times faster, on average (untimed SystemC was less than 1 second for all cases)

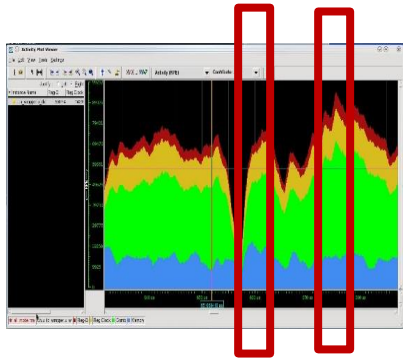
Performance predicted by SystemC is averages +/- 11% of RTL results (2.8% discounting naïve)

Power Consumption

- In a MatchLib flow HSL synthesis can be used to create an equivalent design at any time.
  - This can be run through RTL synthesis and place and route to perform power analysis using traditional EDA tools
- Once a candidate architecture is created, power estimates can be derived without an expensive, time consuming RTL design cycle

Peak Power Analysis





Design Statistics	
Number of Flops	815
User Enabled Flops	296 (36.32 %)
Clock Gating Efficiency	21.45 %
Number of Memories	5
Memory Efficiency	60.38 %

Total Switching Power	4254.63 uW
Total Internal Power	11968.8 uW
Total Design Power	17298.5 uW

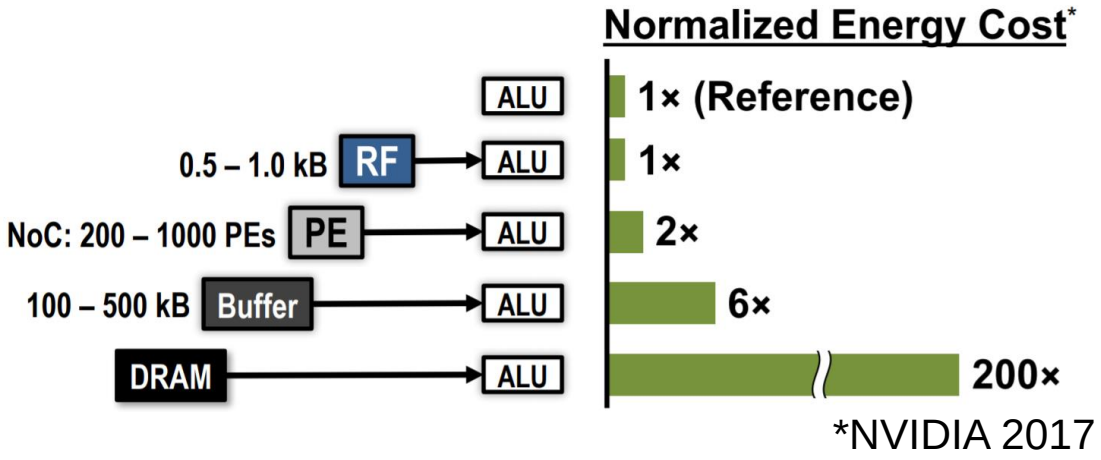
```
always@(posedge clk)
  q<=d;

always@(posedge clk)
  if(en)
  q<=d;
```

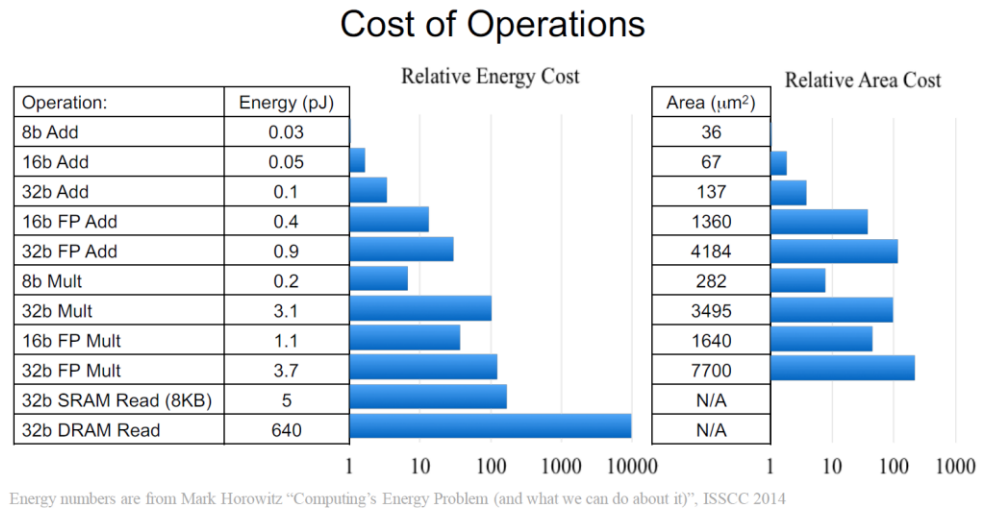
View Activity → **Collect more data at hotspots** → **Power Analysis** → **Power Optimization**

Emulation System

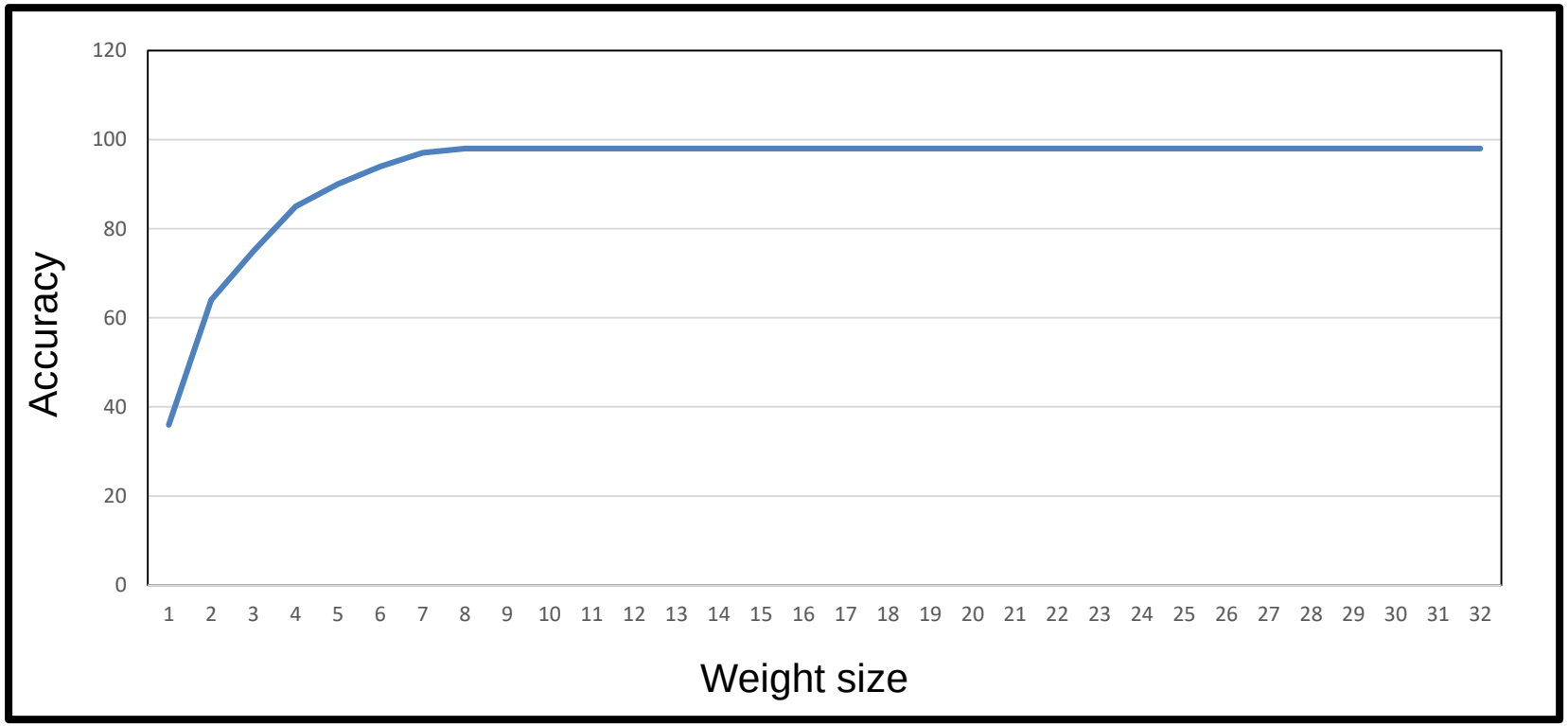
Reduced Precision

- Reduces amount of data that needs to be moved
  - For wake word algorithm 3 MB for 32 bit weights becomes 0.75 MB for 8 bit weights
- Reduces size of operations (multipliers)
  - 8 bit multipliers are about 1/16th the area of 32 bit multipliers, and consume 1/20th the power
- There is a corresponding reduction in power
  - For both data movement and calculations





Accuracy vs. Bit Width for CNN



- For ResNET
  - 32-bit weights improves accuracy by less than 0.1% over 8-bit weights

Power Optimization Options

- Move from floating point to fixed point math operations
- Reduce bit representation of weights and features
- Reduce number of samples in spectral data
- Reduce number of frequencies computed in spectral data
- Reduce number of inference per second

This work will show up in a future tutorial

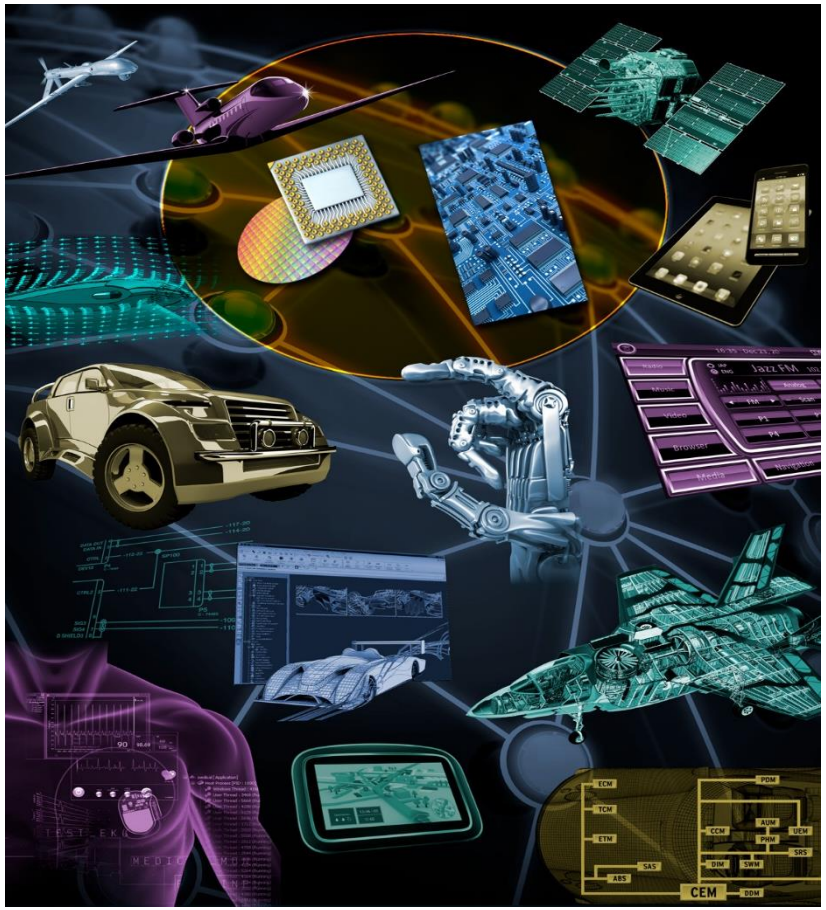
MatchLib

- Based on a powerful message passing framework
- Allows meaningful performance measurement of a system early in the design cycle
- With abstract models for computational elements, delivers fast simulation performance
- With HLS enables an automated path to RTL implementation
  - Ensuring consistency between high level simulations and RTL
  - Facilitating power analysis and optimization
- Open source
- Proven
  - Used by nVidia during the development of AI hardware accelerators

A Siemens Business

www.mentor.com

Contact page



Russell Klein
HLS Platform Director
Mentor

8005 SW Boeckman Rd
Wilsonville, OR, 97070
U.S.A.

Phone: +1 503-685-1416
Mobile: +1 971-832-4155

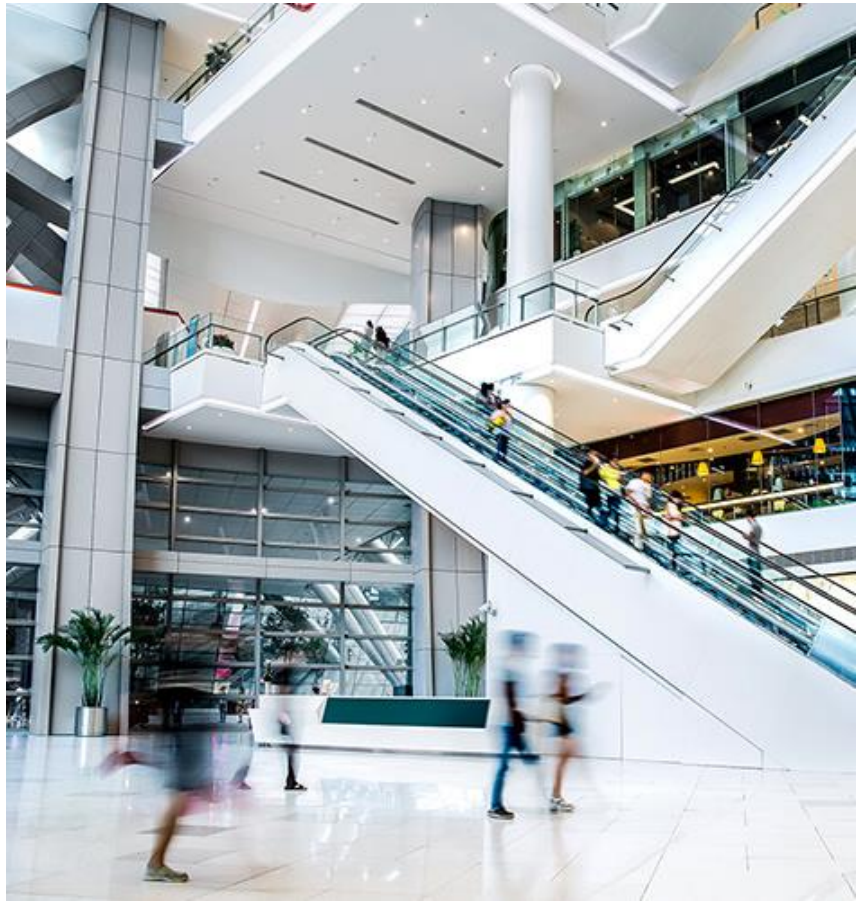
E-mail:
Russell_Klein@mentor.com

Mentor.com/catapult

accellera SYSTEMS INITIATIVE

2019 DESIGN AND VERIFICATION DVCON CONFERENCE AND EXHIBITION EUROPE

Contact page



Published by Siemens

Herbert Taucher
Head of Research Group Electronic Design

Corporate Technology

Siemensstraße 90
1210 Vienna
Austria

Phone: +43 51707 37626
Mobile: +43 664 80117 37626

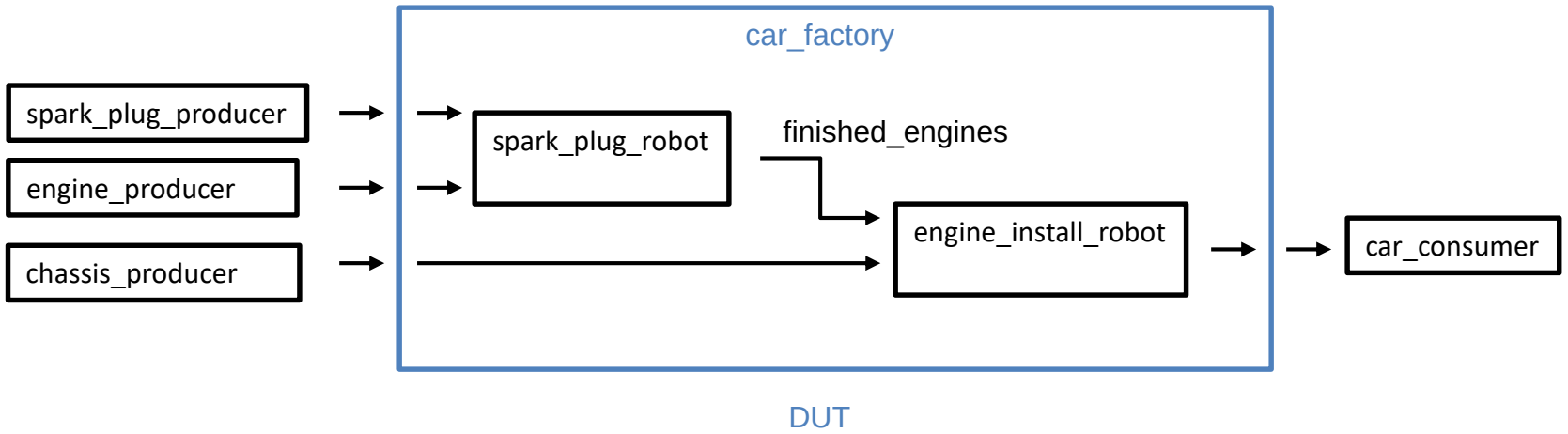
E-mail:
herbert.taucher@siemens.com

siemens.com

Bonus Material

MATCHLIB ARCHITECTURAL EXAMPLE

Simple Example of HW Architectural Model using SC + Matchlib



spark_plug_producer

```
15 class spark_plug_producer : public sc_module {
16 public:
17   sc_in<bool>                    INIT_S1(clk);
18   Connections::Out<spark_plug_t> INIT_S1(spark_plugs);
19
20   SC_CTOR(spark_plug_producer) {
21     SC_THREAD(main);
22     sensitive << clk.pos();
23   }
24
25   void main() {
26     int count=0;
27
28     while (1) {
29       spark_plug_t spark_plug;
30       spark_plug.spark_plug = count++;
31       spark_plugs.Push(spark_plug);
32       wait(3);
33       if (rand() & 1)
34         wait(3);
35     }
36   }
37 };
```

Source: Mentor Graphics, 2019

produces new spark_plug every 3-6 seconds

engine_producer

```
39 class engine_producer : public sc_module {
40 public:
41   sc_in<bool>                INIT_S1(clk);
42   Connections::Out<engine_t> INIT_S1(engines);
43
44   SC_CTOR(engine_producer) {
45     SC_THREAD(main);
46     sensitive << clk.pos();
47   }
48
49   void main() {
50     int count=0;
51
52     while (1) {
53       engine_t engine;
54       engine.engine = count++;
55       engines.Push(engine);
56       wait(20);
57     }
58   }
59 };
```

Source: Mentor Graphics, 2019

produces new engine every 20 seconds

chassis_producer

```
61 class chassis_producer : public sc_module {
62 public:
63   sc_in<bool>                   INIT_S1(clk);
64   Connections::Out<chassis_t>   INIT_S1(chassis_out);
65
66   SC_CTOR(chassis_producer) {
67     SC_THREAD(main);
68     sensitive << clk.pos();
69   }
70
71   void main() {
72     int count=0;
73
74     while (1) {
75       chassis_t chassis;
76       chassis.chassis = count++;
77       chassis_out.Push(chassis);
78       wait(25);
79     }
80   }
81 };
```

Source: Mentor Graphics, 2019

produces new chassis every 25 seconds

car_consumer

```
 83 class car_consumer : public sc_module {
 84 public:
 85   sc_in<bool>             INIT_S1(clk);
 86   Connections::In<car_t>  INIT_S1(cars);
 87
 88   SC_CTOR(car_consumer) {
 89     SC_THREAD(main);
 90     sensitive << clk.pos();
 91   }
 92
 93   void main() {
 94     int count = 0;
 95     while (1) {
 96       cars.Pop();
 97       ++count;
 98       LOG("got car # " << count);
 99       if (count == 10)
100       {
101         LOG("total cars produced: " << count);
102         LOG("time per car: " << sc_time_stamp() / count);
103         sc_stop();
104       }
105     }
106   }
107 };
```

Source: Mentor Graphics, 2019

consumes cars as quickly as possible

Simple car_factory

```
139 #if defined(SIMPLE)
140 class car_factory : public sc_module {
141 public:
142   sc_in<bool>                   INIT_S1(clk);
143   Connections::In<spark_plug_t> INIT_S1(spark_plugs);
144   Connections::In<engine_t>     INIT_S1(engines);
145   Connections::In<chassis_t>    INIT_S1(chassis);
146   Connections::Out<car_t>       INIT_S1(cars);
147
148   Connections::Combinational<engine_t> INIT_S1(finished_engines);
149
150   spark_plug_robot      INIT_S1(spark_plug_robot1);
151   engine_install_robot  INIT_S1(engine_install_robot1);
152
153   SC_CTOR(car_factory)
154   {
155     spark_plug_robot1.clk(clk);
156     spark_plug_robot1.spark_plugs(spark_plugs);
157     spark_plug_robot1.engines_in(engines);
158     spark_plug_robot1.engines_out(finished_engines);
159
160     engine_install_robot1.clk(clk);
161     engine_install_robot1.chassis(chassis);
162     engine_install_robot1.engines(finished_engines);
163     engine_install_robot1.cars(cars);
164   }
165 };
```

Source: Mentor Graphics, 2019

spark_plug_robot

```
71 class spark_plug_robot : public sc_module {
72 public:
73   sc_in<bool>                     INIT_S1(clk);
74   Connections::In<spark_plug_t>   INIT_S1(spark_plugs);
75   Connections::In<engine_t>       INIT_S1(engines_in);
76   Connections::Out<engine_t>      INIT_S1(engines_out);
77   SC_SIG(bool, busy);
78   SC_SIG(bool, maintenance);
79
80   SC_CTOR(spark_plug_robot)
81   {
82     SC_THREAD(main);
83     sensitive << clk.pos();
84   }
85
86   void main() {
87     int count = 0;
88     while (1) {
89        engine_t engine_in = engines_in.Pop();
90        for (int i=0; i < engine_t::plugs; i++)
91           engine_in.spark_plugs[i] = spark_plugs.Pop();
92        busy = 1;
93        wait(60);
94        busy = 0;
95        engines_out.Push(engine_in);
96        if ((count++ & 1) && (rand() & 3))
97        {
98          maintenance = 1;
99          wait(60);
100         maintenance = 0;
101       }
102    }
103  }
104 };
```

Source: Mentor Graphics, 2019

Consumes 4 spark_plugs and 1 unfinished engine
Produces finished_engine after 60 seconds
After every other engine, 75% of time
needs 60 seconds of maintenance (ie idle time)

engine_install_robot

```
106 class engine_install_robot : public sc_module {
107 public:
108   sc_in<bool>                  INIT_S1(clk);
109   Connections::In<chassis_t>   INIT_S1(chassis);
110   Connections::In<engine_t>    INIT_S1(engines);
111   Connections::Out<car_t>      INIT_S1(cars);
112   SC_SIG(bool, busy);
113
114   SC_CTOR(engine_install_robot)
115   {
116     SC_THREAD(main);
117     sensitive << clk.pos();
118   }
119
120   void main() {
121     while (1) {
122       car_t car;
123       car.chassis = chassis.Pop();
124       car.engine = engines.Pop();
125       busy = 1;
126       wait(30);
127       busy = 0;
128       cars.Push(car);
129     }
130   }
131 };
```

Source: Mentor Graphics, 2019

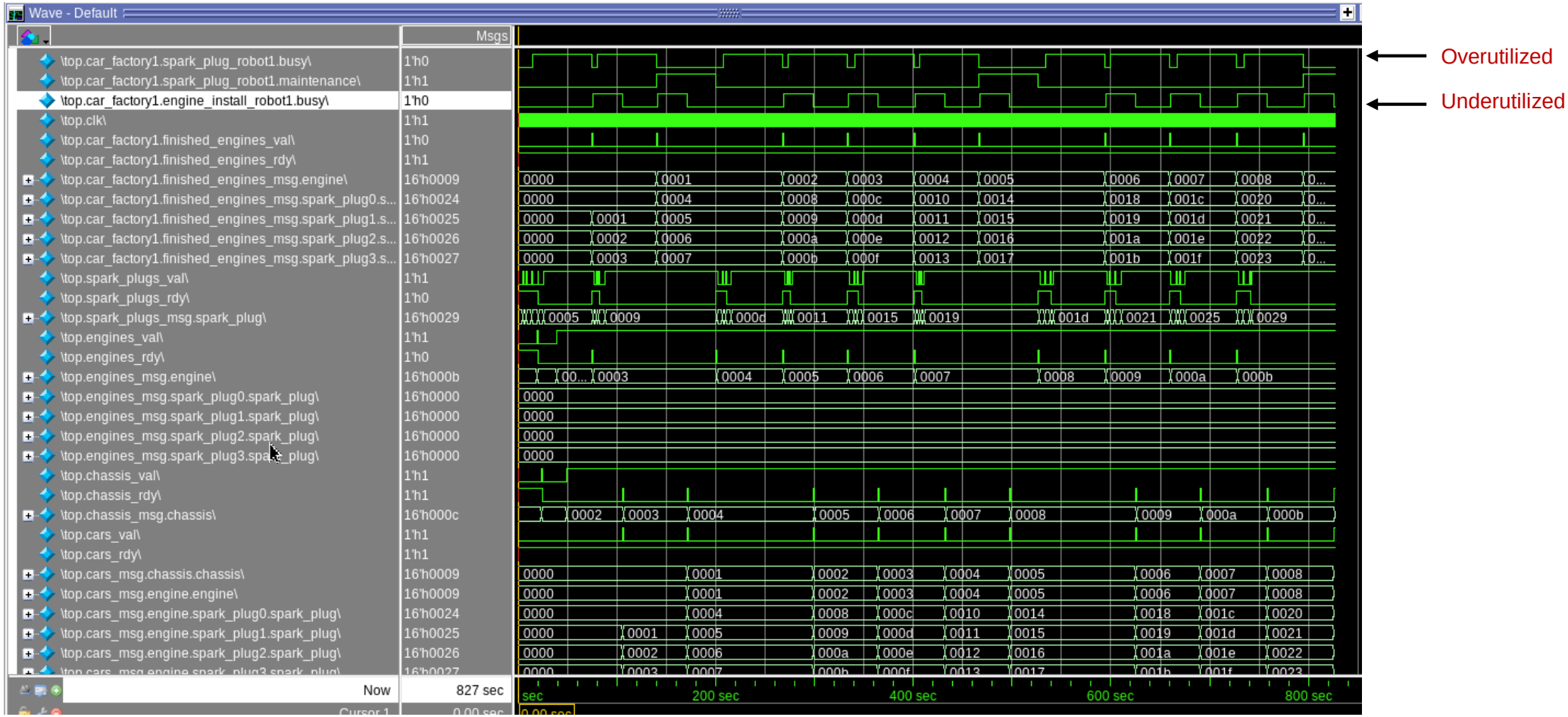
Consumes 1 chassis and 1 finished_engine
Produces car after 30 seconds

Running simple car_factory

```
107 s top.car_consumer1 got car # 1
172 s top.car_consumer1 got car # 2
300 s top.car_consumer1 got car # 3
365 s top.car_consumer1 got car # 4
433 s top.car_consumer1 got car # 5
498 s top.car_consumer1 got car # 6
626 s top.car_consumer1 got car # 7
691 s top.car_consumer1 got car # 8
759 s top.car_consumer1 got car # 9
827 s top.car_consumer1 got car # 10
827 s top.car_consumer1 total cars produced: 10
827 s top.car_consumer1 time per car: 82700 ms

Info: /OSCI/SystemC: Simulation stopped by user.
```

Goal is to produce each car in smallest amount of time

Running simple car_factory



Sequential car_factory

```
#elif defined(SEQUENTIAL)
class car_factory : public sc_module {
public:
  sc_in<bool>                   INIT_S1(clk);
  Connections::In<spark_plug_t> INIT_S1(spark_plugs);
  Connections::In<engine_t>     INIT_S1(engines);
  Connections::In<chassis_t>    INIT_S1(chassis);
  Connections::Out<car_t>       INIT_S1(cars);

  SC_CTOR(car_factory)
  {
    SC_THREAD(main);
    sensitive << clk.pos();
  }

  SC_SIG(bool, spark_plug_robot_busy);
  SC_SIG(bool, spark_plug_robot_maintenance);
  SC_SIG(bool, engine_install_robot_busy);

  void main() {
    spark_plug_t plugs[engine_t::plugs];
    int plug_count = 0;

    engine_t unfinished_engine;
    int unfinished_engine_count = 0;
    engine_t finished_engine;
    int finished_engine_count = 0;
    chassis_t chassis_inst;
    int chassis_count = 0;
    int spark_plug_robot_count = 0;

    while (1) {
      if (plug_count < engine_t::plugs)
        if (spark_plugs.PopNB(plugs[plug_count]))
          ++plug_count;

      if (unfinished_engine_count == 0)
        if (engines.PopNB(unfinished_engine))
          ++unfinished_engine_count;

      if (chassis_count == 0)
        if (chassis.PopNB(chassis_inst))
          ++chassis_count;

      if ((unfinished_engine_count == 1) && (plug_count == engine_t::plugs)
         && (finished_engine_count == 0))
      {
       finished_engine = unfinished_engine;
       for (int i=0; i < engine_t::plugs; i++)
          finished_engine.spark_plugs[i] = plugs[i];
       spark_plug_robot_busy = 1;
       wait(60);
       spark_plug_robot_busy = 0;
       if ((spark_plug_robot_count++ & 1) && (rand() & 3))
       {
         spark_plug_robot_maintenance = 1;
         wait(60);
         spark_plug_robot_maintenance = 0;
       }
       finished_engine_count = 1;
       plug_count = 0;
       unfinished_engine_count = 0;
      }

      if ((finished_engine_count == 1) && (chassis_count == 1))
      {
        car_t car;
        car.chassis = chassis_inst;
        car.engine = finished_engine;
        engine_install_robot_busy = 1;
        wait(30);
        engine_install_robot_busy = 0;
        cars.Push(car);
        finished_engine_count = 0;
        chassis_count = 0;
      }

      wait();
    }
  }
};
```

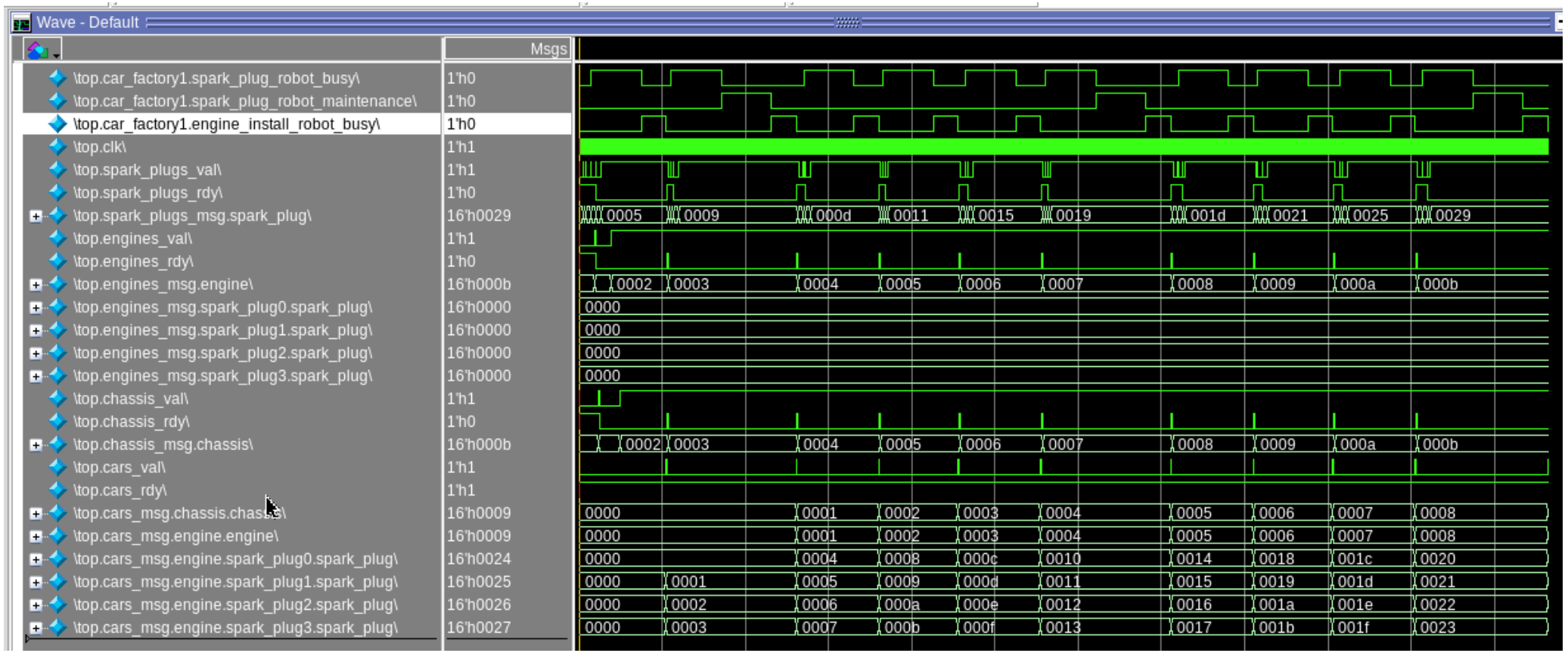
Source: Mentor Graphics, 2019

Running sequential car_factory

```
106 s top.car_consumer1 got car # 1
262 s top.car_consumer1 got car # 2
361 s top.car_consumer1 got car # 3
457 s top.car_consumer1 got car # 4
556 s top.car_consumer1 got car # 5
712 s top.car_consumer1 got car # 6
811 s top.car_consumer1 got car # 7
907 s top.car_consumer1 got car # 8
1006 s top.car_consumer1 got car # 9
1165 s top.car_consumer1 got car # 10
1165 s top.car_consumer1 total cars produced: 10
1165 s top.car_consumer1 time per car: 116500 ms

Info: /OSCI/SystemC: Simulation stopped by user.
```

Car production time got worse!

Running sequential car_factory



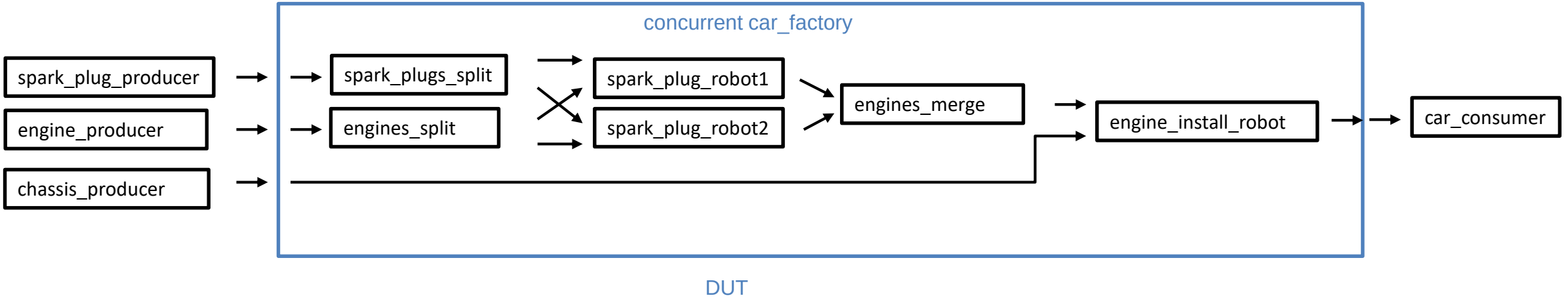
← Worse!

← Worse!

How do we fix the car_factory architecture?

- Primary problem in “simple” car_factory is overutilization of spark_plug_robot
- Obvious solution: add another spark_plug_robot

concurrent car_factory



spark_plugs_split and engines_split

```
class spark_plugs_split : public sc_module {
public:
  sc_in<bool>                       INIT_S1(clk);
  Connections::In<spark_plug_t>     INIT_S1(spark_plugs_in);
  Connections::Out<spark_plug_t>    INIT_S1(spark_plugs_out1);
  Connections::Out<spark_plug_t>    INIT_S1(spark_plugs_out2);

  SC_CTOR(spark_plugs_split)
  {
    SC_THREAD(main);
    sensitive << clk.pos();
  }

  void main() {
    while (1) {
      spark_plug_t spark_plug = spark_plugs_in.Pop();
      while (1) {
        if (spark_plugs_out1.PushNB(spark_plug))
          break;
        if (spark_plugs_out2.PushNB(spark_plug))
          break;
        wait();
      }
    }
  }
};
```

```
class engines_split : public sc_module {
public:
  sc_in<bool>                   INIT_S1(clk);
  Connections::In<engine_t>     INIT_S1(engines_in);
  Connections::Out<engine_t>    INIT_S1(engines_out1);
  Connections::Out<engine_t>    INIT_S1(engines_out2);

  SC_CTOR(engines_split)
  {
    SC_THREAD(main);
    sensitive << clk.pos();
  }

  void main() {
    while (1) {
      engine_t engine = engines_in.Pop();
      while (1) {
        if (engines_out1.PushNB(engine))
          break;
        if (engines_out2.PushNB(engine))
          break;
        wait();
      }
    }
  }
};
```

Source: Mentor Graphics, 2019

engines_merge

```
class engines_merge : public sc_module {
public:
  sc_in<bool>                    INIT_S1(clk);
  Connections::In<engine_t>      INIT_S1(engines_in1);
  Connections::In<engine_t>      INIT_S1(engines_in2);
  Connections::Out<engine_t>     INIT_S1(engines_out);

  SC_CTOR(engines_merge)
  {
    SC_THREAD(main);
    sensitive << clk.pos();
  }

  void main() {
    while (1) {
      engine_t engine;
      while (1) {
        if (engines_in1.PopNB(engine))
          break;
        if (engines_in2.PopNB(engine))
          break;
        wait();
      }
      engines_out.Push(engine);
    }
  }
};
```

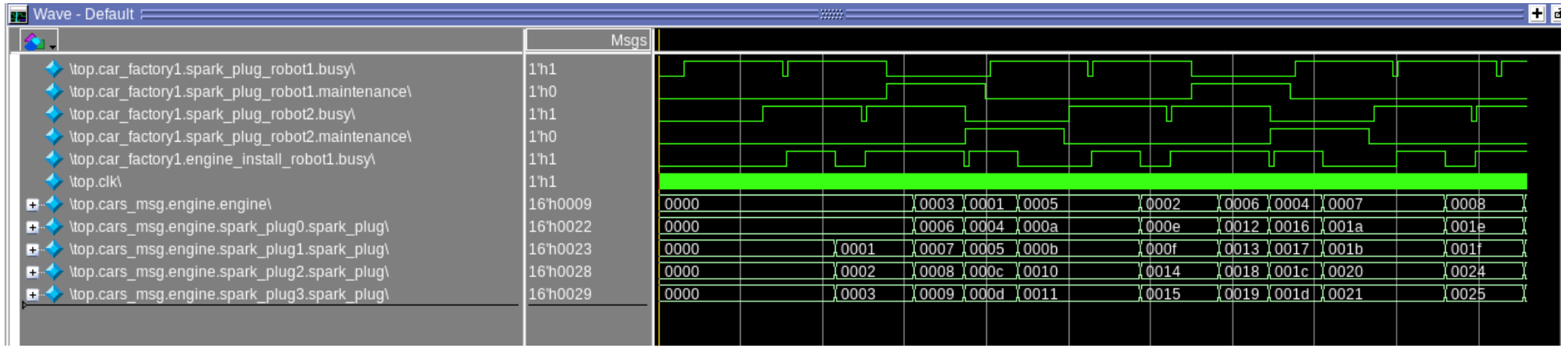
Source: Mentor Graphics, 2019

Running concurrent car_factory

```
109 s top.car_consumer1 got car # 1
157 s top.car_consumer1 got car # 2
187 s top.car_consumer1 got car # 3
220 s top.car_consumer1 got car # 4
295 s top.car_consumer1 got car # 5
343 s top.car_consumer1 got car # 6
373 s top.car_consumer1 got car # 7
406 s top.car_consumer1 got car # 8
481 s top.car_consumer1 got car # 9
529 s top.car_consumer1 got car # 10
529 s top.car_consumer1 total cars produced: 10
529 s top.car_consumer1 time per car: 52900 ms

Info: /OSCI/SystemC: Simulation stopped by user.
```

Big improvement in car production time!

Running concurrent car_factory



Both spark_plug_robots busy at same time

Better utilization

Output order of engines and plugs no longer matches input order