Early Performance Analysis and Architectural Optimization of a Machine Learning Accelerator Using MatchLib

Michael Fingeroff | High-Level Synthesis Technologist
Siemens EDA, a part of Siemens Digital Industries Software

SIEMENS

Agenda

AI/ML Application Challenges

Algorithmic Complexity

- Growing faster than the ability of RTL designers to code and verify

Memory Architecture Complexity

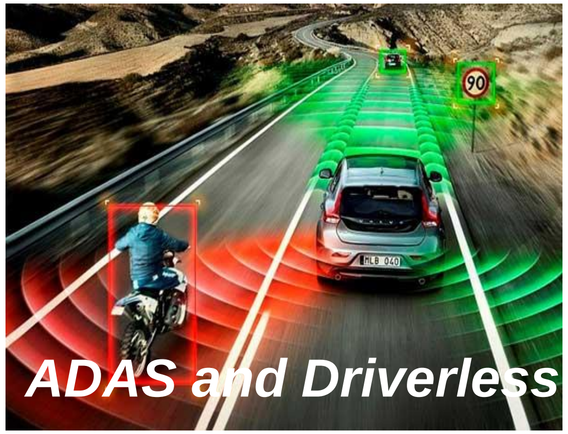
- Efficient data movement is key for power, performance and area

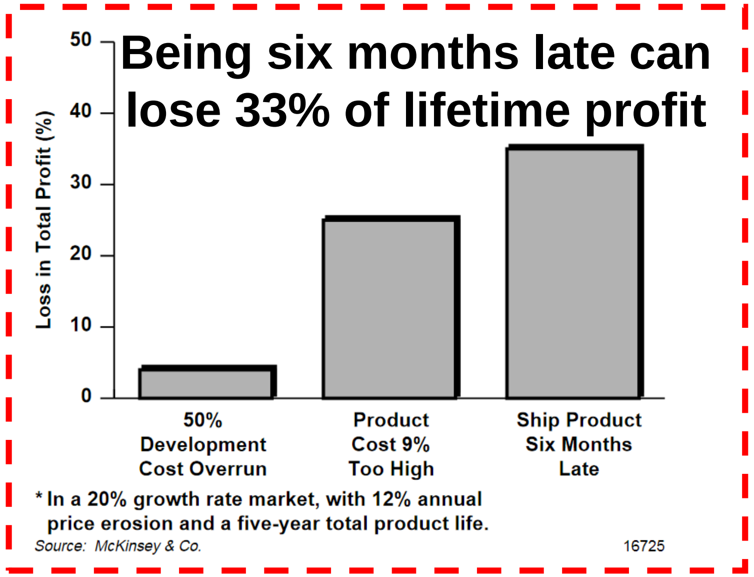
RTL Verification Costs Increasing

- Increased design complexity increases bugs introduced during hand-coding of RTL
- RTL regressions involve server farms, electricity cost, licenses and time

Slips in Design Schedule Kills Total Profit

- Finding bugs during system integration is too late





* In a 20% growth rate market, with 12% annual price erosion and a five-year total product life.
Source: McKinsey & Co.

SIEMENS

Convolutional Neural Network Overview

SIEMENS

Convolutional Neural Network Overview

Used in object detection and classification

Mostly Convolutional layers

- Majority of computation done here (over 99%)
- Majority of memory traffic
- Bias and activation functions

Pooling layers

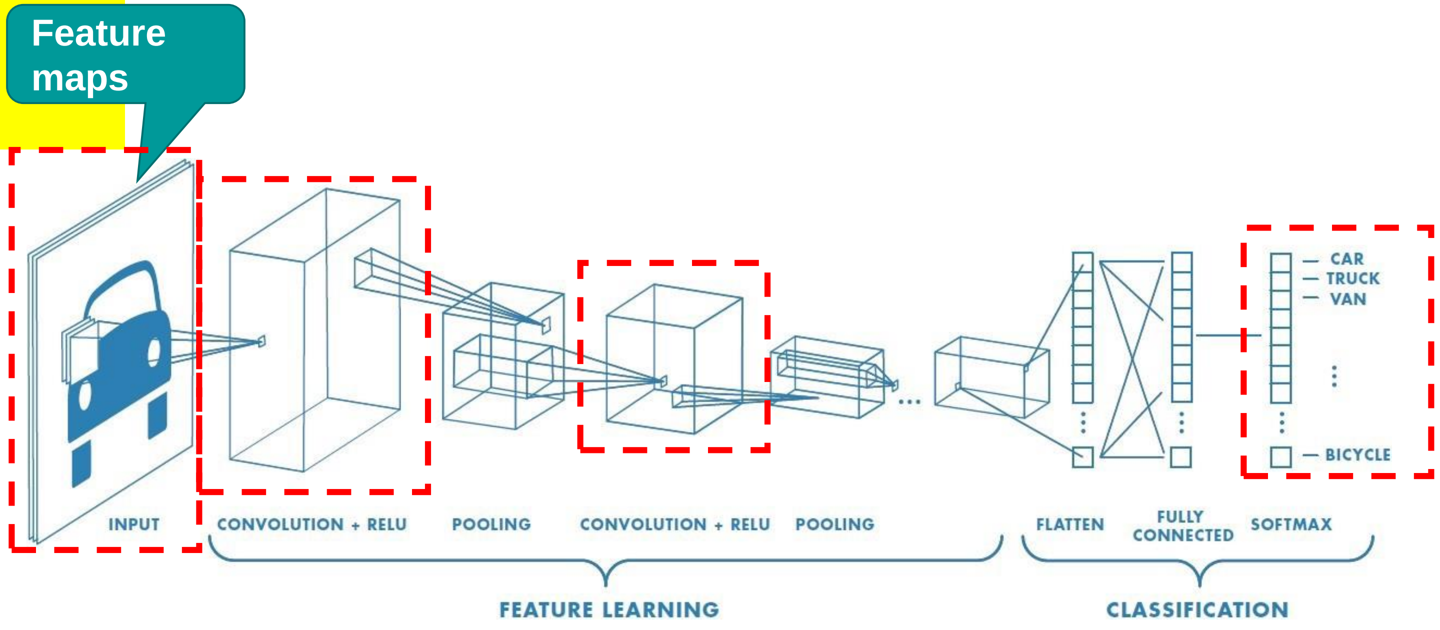
- Reduce feature map size

Fully connected

- Classification

Softmax

- normalize class probabilities

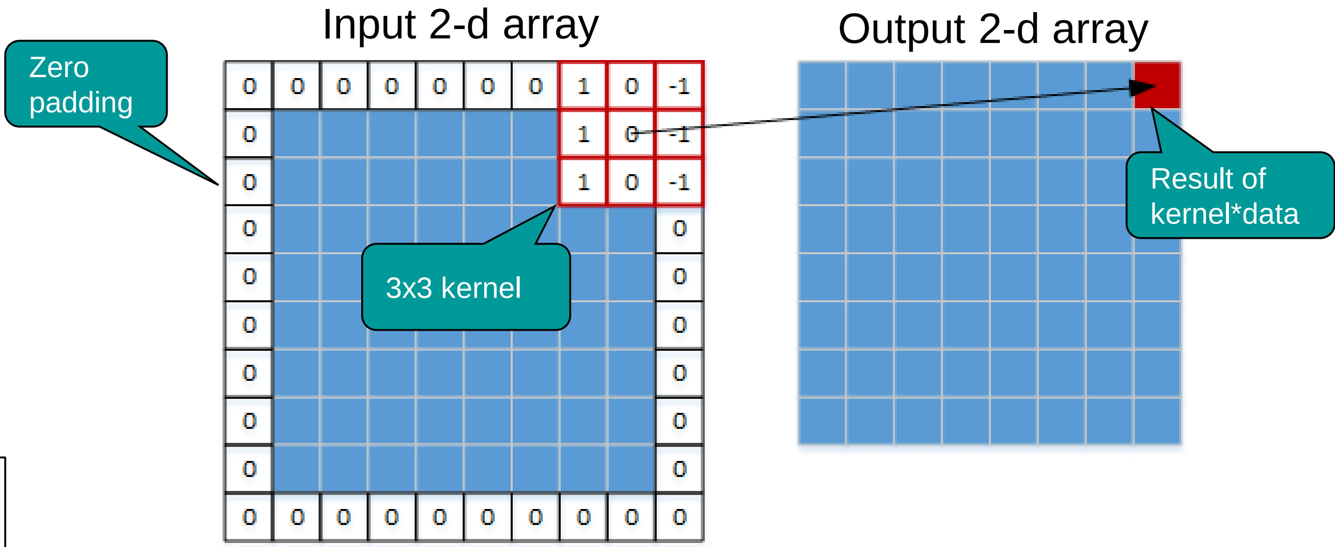


2-d 3x3 Convolution Algorithm

- 2-d kernel
- 2-d input array
- Padding left, right, and top and bottom
- Stride = number of elements kernel jumps by

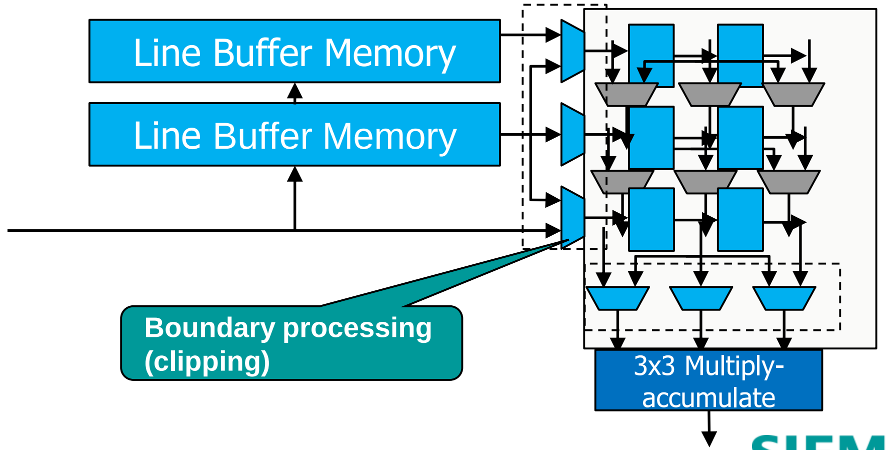
2-d Convolution Algorithm – stride 1, zero pad

```
HEIGHT:for(int r=0;r<IN_HEIGHT;r++){
  WIDTH:for(int c=0;c<IN_WIDTH;c++){
    KERNEL_R:for(int i=0;i<3;i++){
      KERNEL_C:for(int j=0;j<3;j++){
        if(r+i-1 < 0 || r+i-1 >= IN_HEIGHT)
          data = 0;//zero pad
        else if(c+j-1 < 0 || c+j-1 >= IN_WIDTH)
          data = 0;//zero pad
      else
        data = data_in[r+i-1][c+j-1];
      acc += data * kernel[i][j];
  }
   result[c] = acc;
}
```



!=

Efficient Convolution Hardware Architecture



SIEMENS

CNN Convolution – conv2d

- CNN convolutional layers have multiple input and output feature maps
- Each output feature map is a sum of separate convolutions across all input feature maps

Output feature map

Input feature map

2-d convolution

Lots of data movement for feature maps and weights

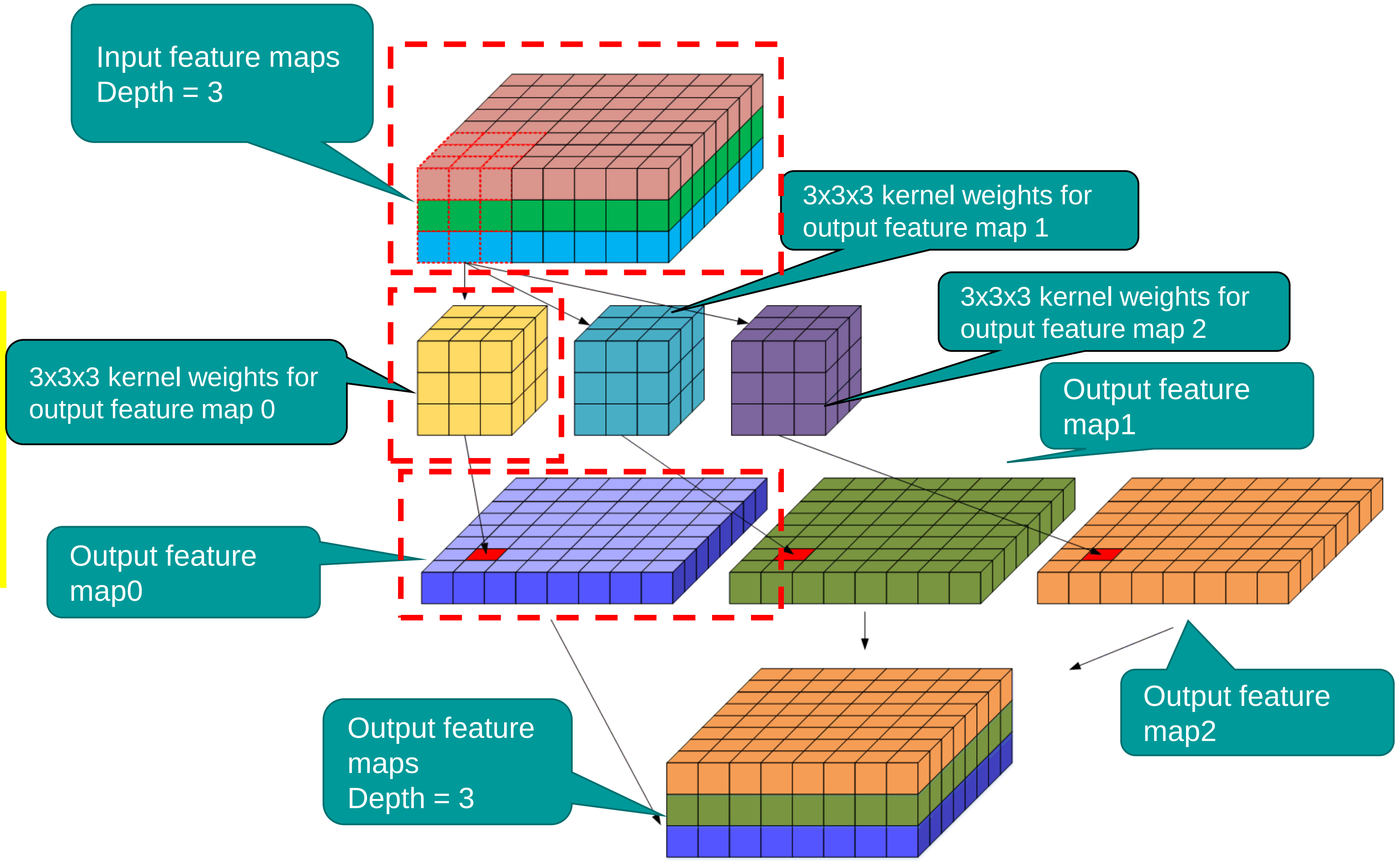
```
OUT_FMAP:for(int oc=0;oc<OUT_FMAPS;oc++){
  IN_FMAP:for(int ic=0;ic<IN_FMAPS;ic++){
    FMAP_HEIGHT:for(int r=0;r<IN_HEIGHT;r++){
      FMAP_WIDTH:for(int c=0;c<IN_WIDTH;c++){
        KERNEL_Y:for(int i=0;i<3;i++){
          KERNEL_X:for(int j=0;j<3;j++){
            acc[r][c] += fmap[ic][r-i/2][c-j/2] * kernel[ic][oc][i][j];
          }
        }
        if(<last input fmap>)
          fmap_out[oc][r][c] = acc[r][c];
      }
    }
  }
}
```

SIEMENS

CNN Convolution – conv2d

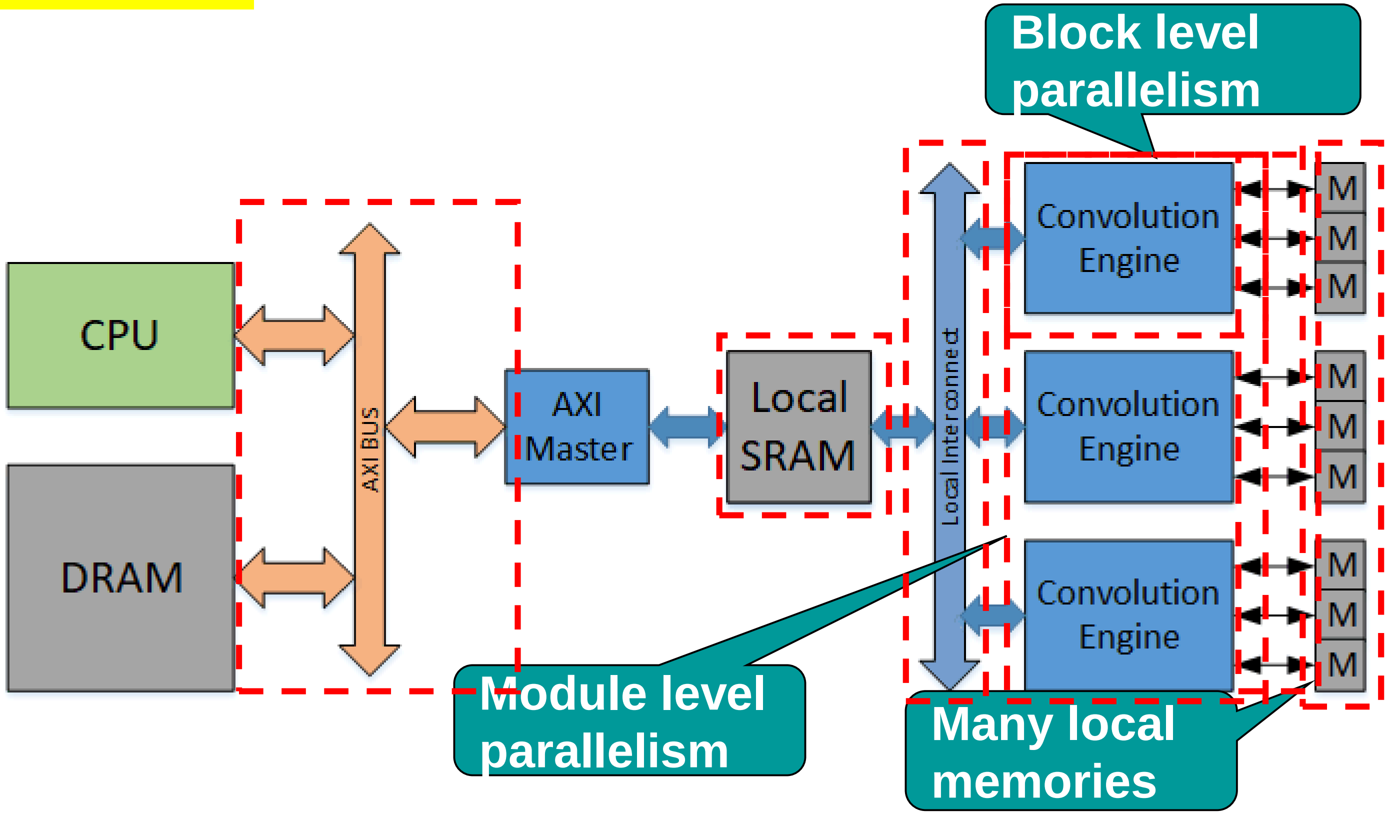
- Every output fmap is 3-d convolution across the input fmaps

- Real CNNs have 100’s or 1000’s of fmaps
  - Lots of convolutions/computations
  - Lots of data movement


Input feature maps
Depth = 3
3x3x3 kernel weights for output feature map 1
3x3x3 kernel weights for output feature map 2
3x3x3 kernel weights for output feature map 0
Output feature map1
Output feature map0
Output feature map2
Output feature maps
Depth = 3

CNN Architectural Challenges

- Memory architectures need to leverage data reuse and parallelism
- May have multiple engines or processing elements
  - Block level parallelism
  - Module level parallelism
- Many local memories
- Complex interconnect
  - AXI4, local interconnect, etc.



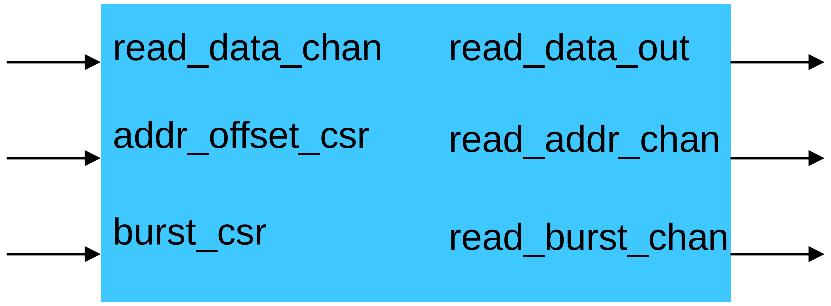
SIEMENS

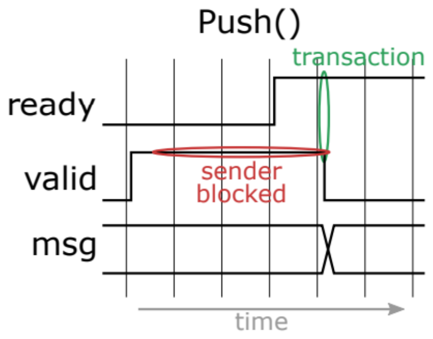
Coding Designs with MatchLib

Modelling Bus I/F With MatchLib

Users can easily model bus interfaces using MatchLib

- This example models a simple read bus I/F with burst





```
class dut : public sc_module {
 public:
   …
   sc_in<uint32>                addr_offset_csr;
   sc_in<uint8>                 burst_csr;
   Connections::In <uint32>     read_data_chan;
   Connections::Out <uint32>    read_addr_chan;
   Connections::Out<uint32>     read_burst_chan;
  …
  void main() {
    wait();
    while (1) {
      uint32 addr = addr_offset_csr.read();
      uint8 burst_size = burst_csr.read();
      read_addr_chan.Push(addr);
      read_burst_chan.Push(burst_size);
      do {
        uint32 data = read_data_chan.Pop();
        read_data_out.Push(data);
      } while (--burst_size !=0);
}
```

Initiate a burst by sending address and burst size

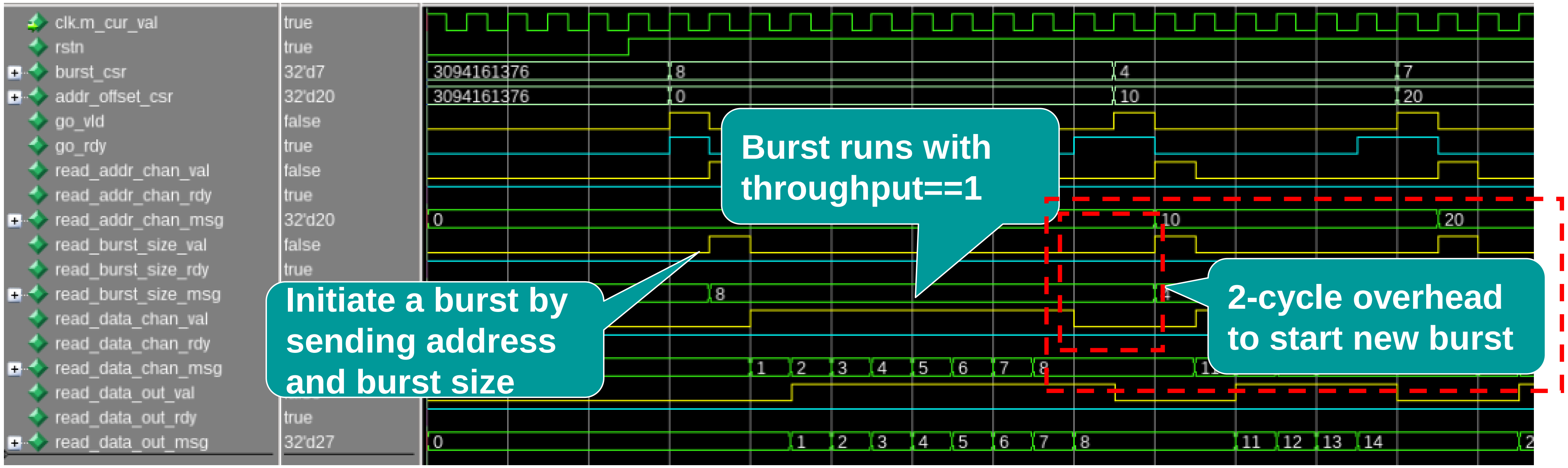
Read “burst_size” data from bus I/F

SIEMENS

Modelling Bus I/F With MatchLib

Small burst size and/or non-consecutive addresses will hurt performance by injecting dead cycles

Pre-HLS Simulation



clk.m_cur_val	true
rstn	true
burst_csr	32'd7
addr_offset_csr	32'd20
go_vld	false
go_rdy	true
read_addr_chan_val	false
read_addr_chan_rdy	true
read_addr_chan_msg	32'd20
read_burst_size_val	false
read_burst_size_rdy	true
read_burst_size_msg	
read_data_chan_val	
read_data_chan_rdy	
read_data_chan_msg	
read_data_out_val	
read_data_out_rdy	true
read_data_out_msg	32'd27

Burst runs with throughput==1

Initiate a burst by sending address and burst size

2-cycle overhead to start new burst

SIEMENS

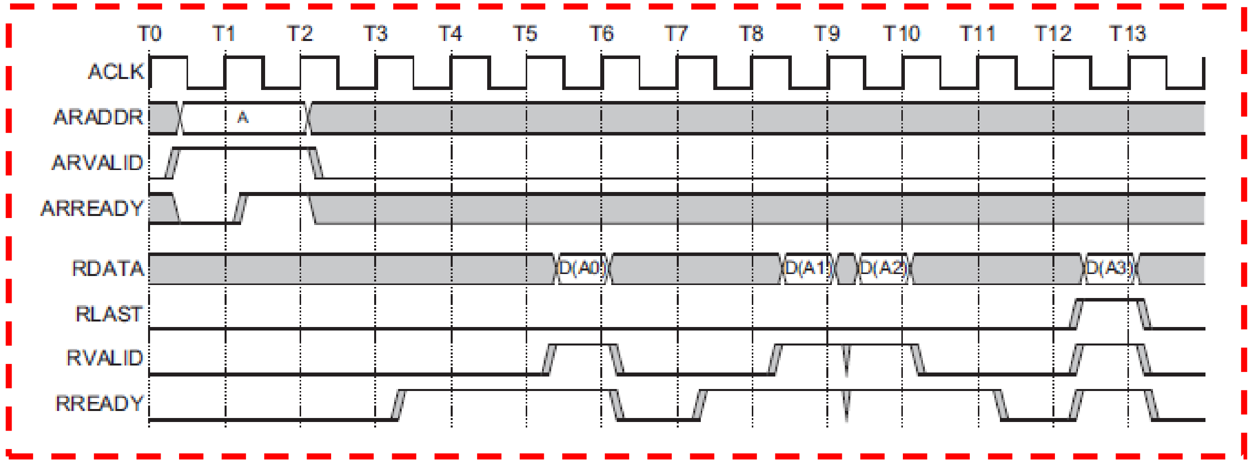
Complex Bus Protocols are Easily Modelled with MatchLib

- AXI4 Master and Slaves are part of the MatchLib library
  - AXI4 segmenter transparently manages burst size, 4k boundaries, etc
- AXI4 Read Master I/F with built-in segmentation
  - r_master<>

  AXI4 Write Master I/F with built-in segmentation
  - w_master<>

  AXI4 Slave I/F with built-in segmentation
  - slave<>
- Interface reads/writes use connections Push/Pop methods
  - Generates AXI4 transactions

```
class bus_if: public sc_module
…
 r_master<>  r_master0;
…
while (1) {
   …
   r_payload r = r_master0.r.Pop();
   …
 }
…
```



SIEMENS

Early Performance Analysis of CNN Convolution

Design Goals

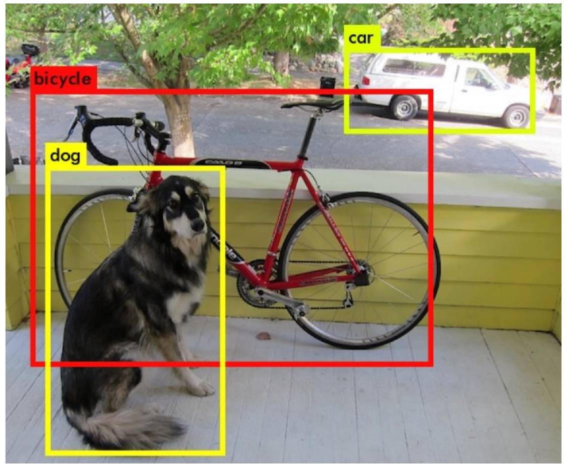
Implement a CNN for object detection and classification

- 9 layers
- Mostly 3x3 convolution (9 multiply-acc)
- 3.5 billion macs/inference

Low power/performance Ring-doorbell type application

- 1 inference/sec

500MHz clock frequency



SIEMENS

Step 1: Original Algorithmic Model of conv2d

- Direct conversion of algorithm to HLS synthesizable bit-accurate model
- Generic bus interfaces with burst
  - Read burst size limited to one due to non-sequential addressing
  - Writes of feature maps can sustain large burst size
- No opportunity for parallelism

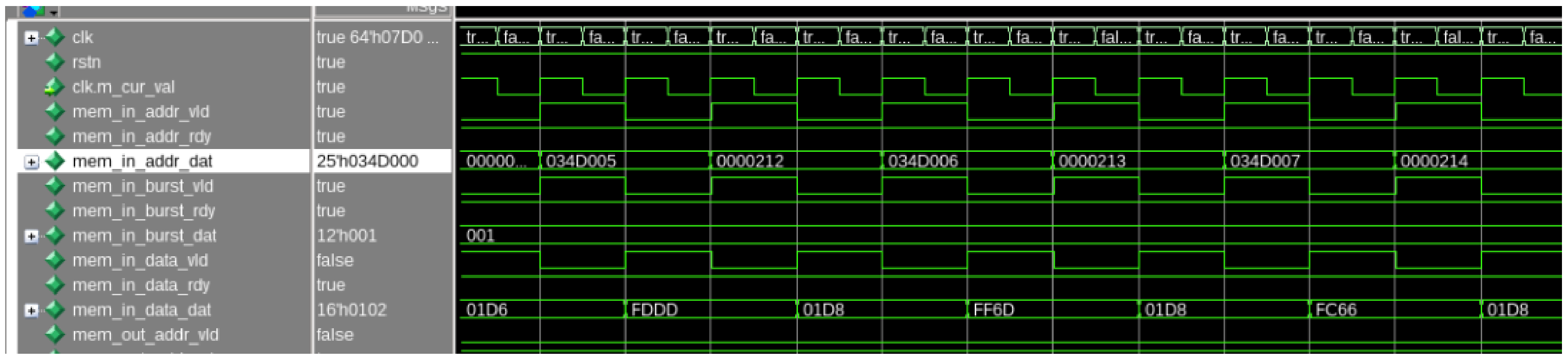
```
OFM:for(int ofm=0; ofm<OUT_FMAP; ofm++) {
  IFM:for (int ifm=0; ifm<IN_FMAP; ifm++) {
    ROW:for (int r=0; r<MAX_HEIGHT; r++) {
      COL:for (int c=0; c<MAX_WIDTH; c++) {
        K_X:for (int kr=0; kr<KSIZE; kr++) {
          K_Y:for (int kc=0; kc<KSIZE; kc++) {
            int ridx = r + kr - KSIZE/2;
            int cidx = c + kc - KSIZE/2;
            <zero pad>
            data_idx=rdoffset+ifm*ht*wt+ridx*wt+cidx;
            mem_in_addr.Push(data_idx);
            mem_in_burst.Push(1);
            data = mem_in_data.Pop();
            <weight read bus transaction>
            acc += data*mem_in_data.Pop();
          }
        }
        acc_buf[r][c] += acc; ...
<Copy feature maps to system memory>
```

SIEMENS

Step 1: Algorithmic Model Pre-HLS Simulation Results

- SystemC simulation "wall-clock time" took very long (~ 2 hours)
  - Context switching due to non-sequential memory accesses
  - Redundant memory accesses
- Simulation time was ~14 seconds to simulate 1 inference
- Poor design
  - No need to go any further

Pre-HLS Simulation



	Msgs
clk	true 64'h07D0 ...
rstn	true
clk.m_cur_val	true
mem_in_addr_vld	true
mem_in_addr_rdy	true
mem_in_addr_dat	25'h034D000
mem_in_burst_vld	true
mem_in_burst_rdy	true
mem_in_burst_dat	12'h001
mem_in_data_vld	false
mem_in_data_rdy	true
mem_in_data_dat	16'h0102
mem_out_addr_vld	false

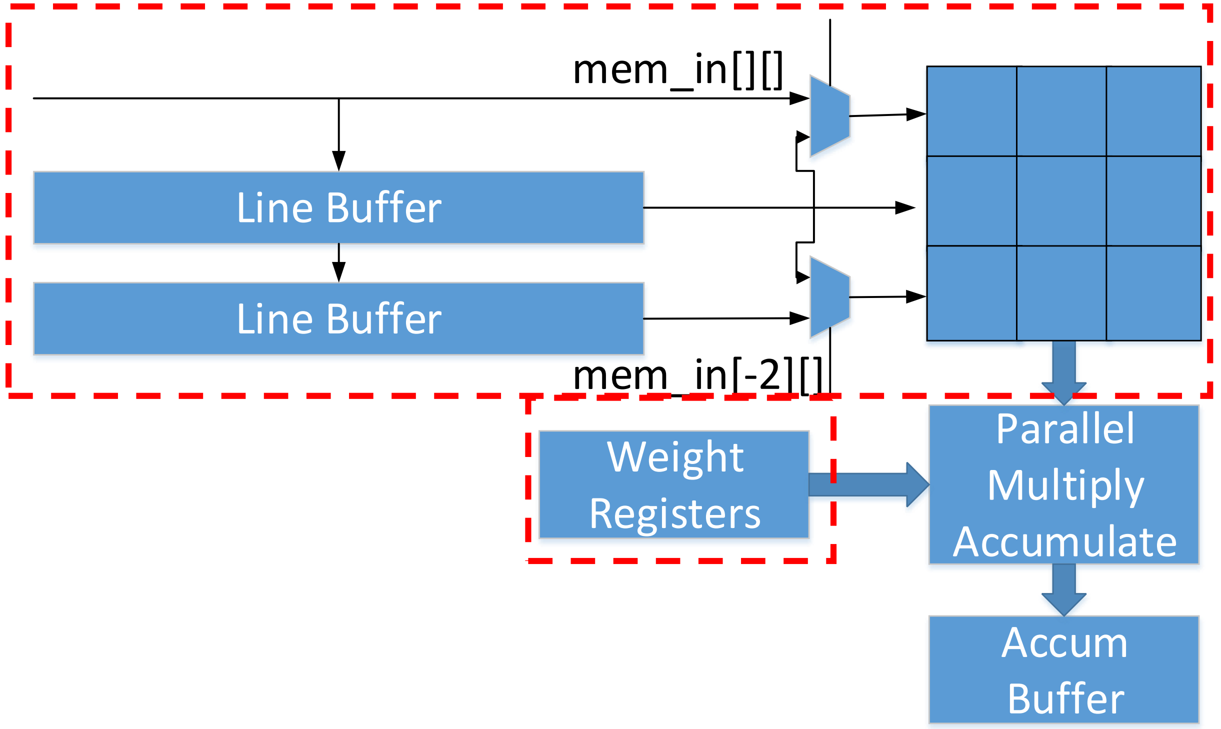
tr... fa... tr... fa... tr... fa... tr... fa... tr... fa... tr... fa... tr... fa... tr... fal... tr... fa... tr... fa... tr... fa... tr... fal... tr... fa...
00000... 034D005 0000212 034D006 0000213 034D007 0000214
001
01D6 FDDD 01D8 FF6D 01D8 FC66 01D8

SIEMENS

Architectural Refinement

Step 2: On-chip Buffering and Windowing

- SystemC HLS designs must be architected for efficient data movement and reuse
  - Improved simulation performance
  - Will allow HLS to extract parallelism
- Sliding-window architecture allows feature map data reuse
- Weight register cache read once for each input/output feature map computation



Step 2: On-chip Buffering and Windowing

- 9 weight bursts
  - Stored in register cache

- Feature maps burst a row at a time
  - Could also burst entire feature map

- Sliding window architecture allows K_X and K_Y to execute in parallel

```
OFM:for(int ofm=0;ofm<OUT_FMAP;ofm++){
  IFM:for(int ifm=0;ifm<IN_FMAP;ifm++){
    mem_in_addr.Push(weight_idx);
    mem_in_burst.Push(9);
    <cache weights>
    ROW:for(int r=0;r<MAX_HEIGHT+1;r++){
      data_idx=read_offset+ifm*height*width+r*width;
      if(r != height){
        mem_in_addr.Push(data_idx);
        mem_in_burst.Push(width);
      }
      COL:for(int c=0;c<MAX_WIDTH+1;c++){
        if(r != height && c != width)
          data[0] = mem_in_data.Pop();
          <sliding window architecture>
          K_X:for(int kr=0;kr<KSIZE;kr++){
            K_Y:for(int kc=0;kc<KSIZE;kc++){
              acc += window[kr][kc]*weights[kr][kc];
            }
          }
```

"window" and "weights" can be read in parallel

Step 2: On-chip Buffering and Windowing Results

- Simulation results
  - Design goal met with simulation time of 0.864 secs
  - Pre-hls simulation wall-clock time 34 minutes for 1 inference

- All other operations run in software
  - Bias, RELU, max pooling, etc.
  - SystemC testbench runs in zero time

- What can MatchLib and SystemC tell us about the system-level performance?

CPU Software Function Calls

```
preprocessing()
setup layer parameters()
start_conv2d()
<HW executing>
wait_for_done()
bias_add();
leakyRelu()
max_pooling()
post_processing()
```

SIEMENS

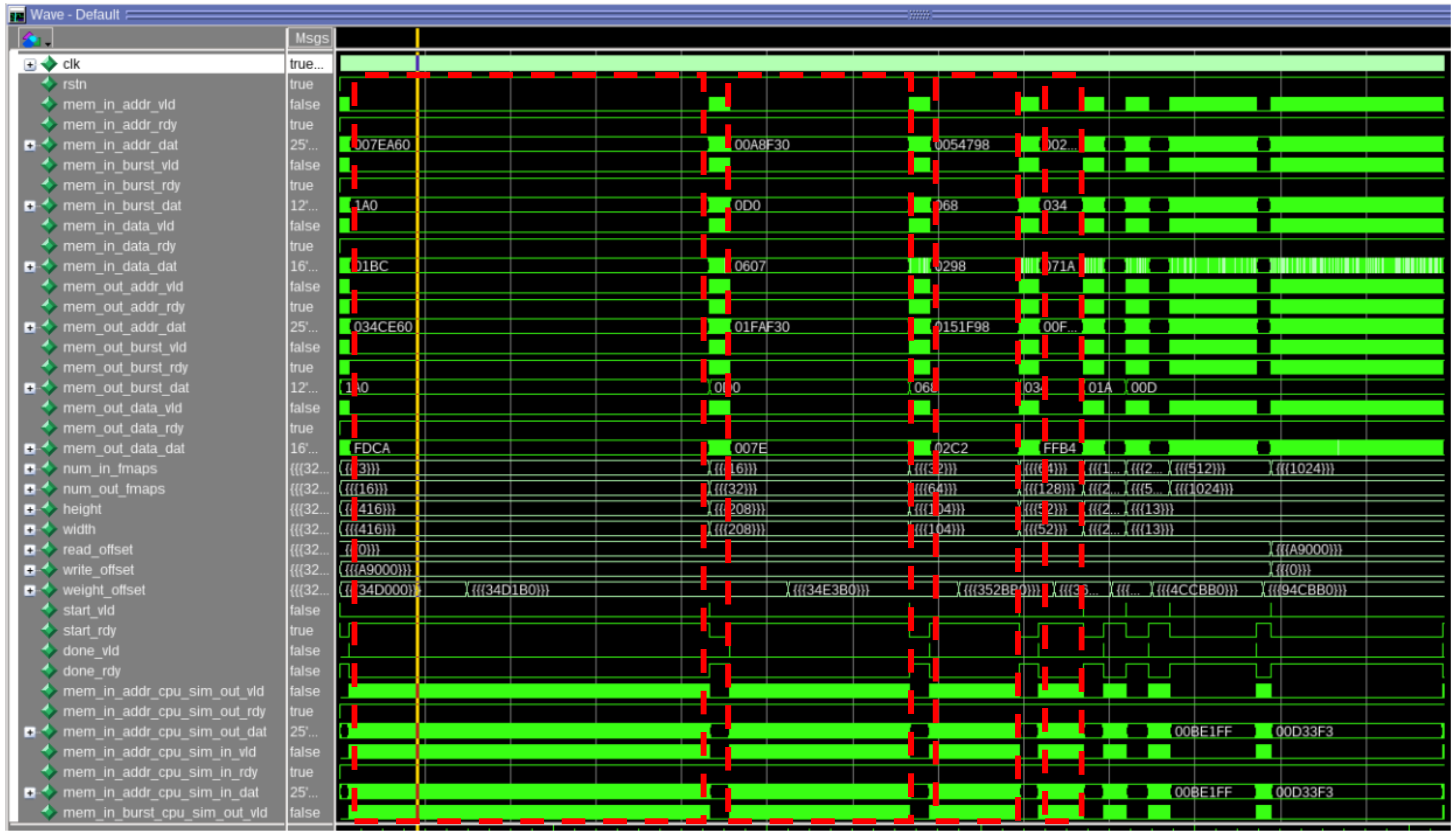
Step 3: Interaction with the CPU

- Target hardware platform
  - System memory is shared between the CPU and the conv2d accelerator
  - There is no CPU cache

- SystemC testbench models arbitrated memory between CPU and ML accelerator
  - Approximated CPU instruction execution and memory access time

- The performance of the accelerator is throttled by the CPU
  - Simulation time 2.6 secs
  - Simulation wall-clock time 63 minutes
  - Time spent converting from fixed-point to float

SystemC Simulation of One Inference



SIEMENS

Step 4: Fusing Computational Layers

- Move Bias, ReLU, and max pooling into the accelerator
  - Cost little more in hardware area

- Can be coded into the design where feature map data is copied back to system memory

- Design simulation time 0.9 secs for one inference

- Pre-HLS simulation wall-clock time 30 minutes

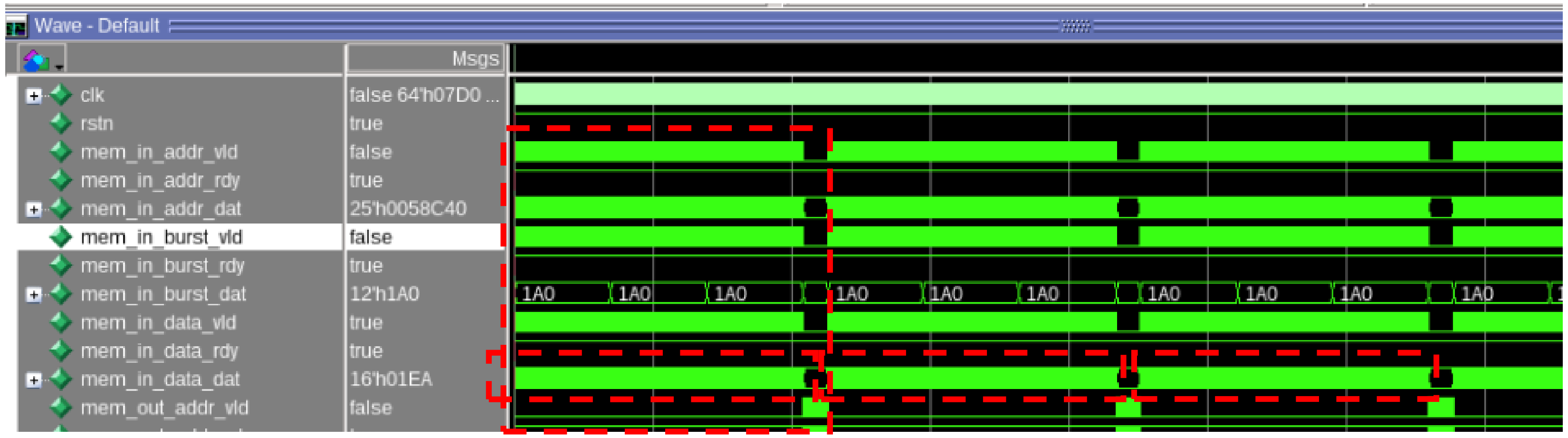
```
 <Get bias from system memory>
 ROW_CPY:for (int r=0; r<MAX_HEIGHT+1; r++) {
   <setup burst size>
   mem_out_addr.Push(out_idx);
   mem_out_burst.Push(burst_size);
   COL_CPY:for (int c=0; c<MAX_WIDTH+1; c++) {
     add_bias = acc_buf[r][c] + bias;
     if (relu)
       if (add_bias < 0)
         add_bias = add_bias * SAT_TYPE(0.1);
     …
     if(pool){
       <max_pooling>
       mem_out_data.Push(max);
     }else
       mem_out_data.Push(add_bias);
}
```

SIEMENS

Step4 – Bus Utilization

- Memory bus is ~100% utilized by the ML accelerator
- Input feature maps are re-read from system memory for each output feature map computation
  - System memory accesses are an order of magnitude larger for power consumption compared to on-chip SRAM

Pre-HLS Simulation



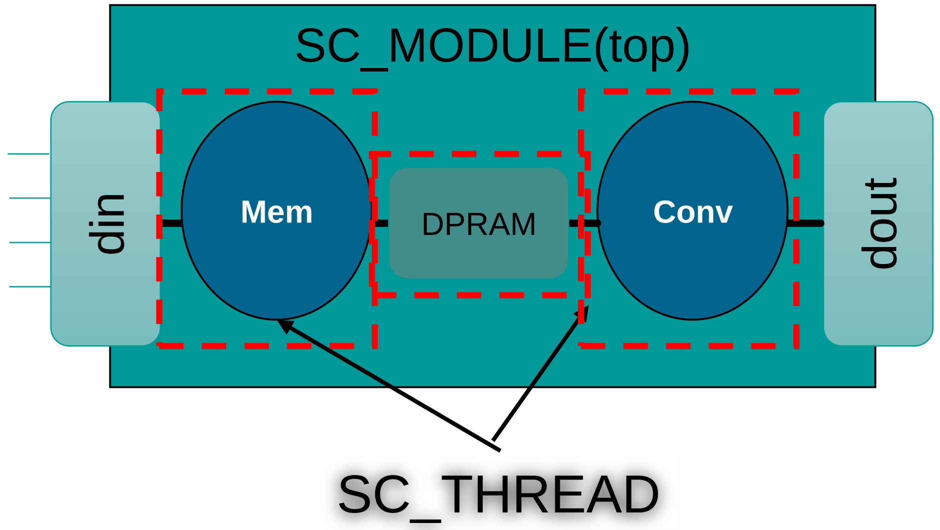
	Msgs
clk	false 64'h07D0 ...
rstn	true
mem_in_addr_vld	false
mem_in_addr_rdy	true
mem_in_addr_dat	25'h0058C40
mem_in_burst_vld	false
mem_in_burst_rdy	true
mem_in_burst_dat	12'h1A0
mem_in_data_vld	true
mem_in_data_rdy	true
mem_in_data_dat	16'h01EA
mem_out_addr_vld	false

SIEMENS

Step 5: Adding On-chip Buffering

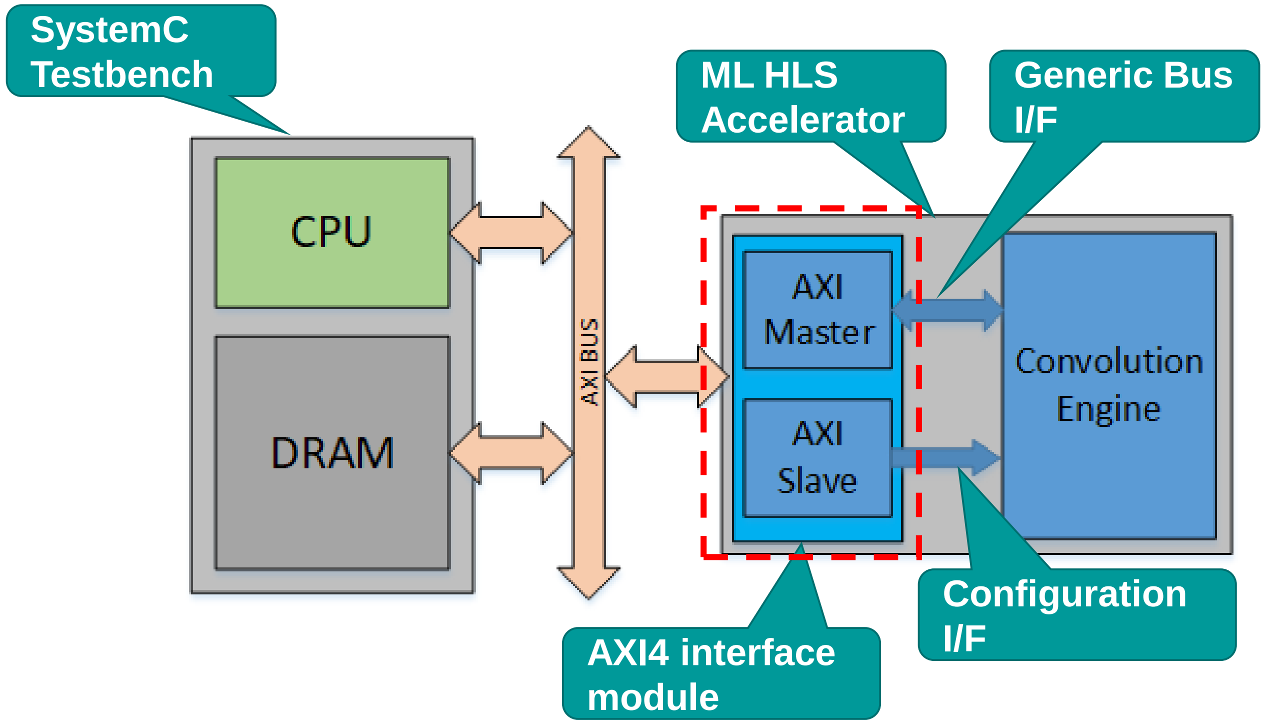
- Buffer feature maps on-chip
  - ~800 KB for full buffering
  - SystemC memory generated my Catapult memory generator

- Split design into multiple processes
  - Memory read process to access system memory
  - Convolution, bias, ReLu, and max pooling process
  - Shared instantiated SystemC feature map memory between processes



Step 5: Adding AXI Master and Slave interfaces

- Existing design with generic bus I/F can be easily “bolted” to an AXI4 master and slave I/F module
- Generic I/F configures the AXI4 master I/F
  - Push/pop



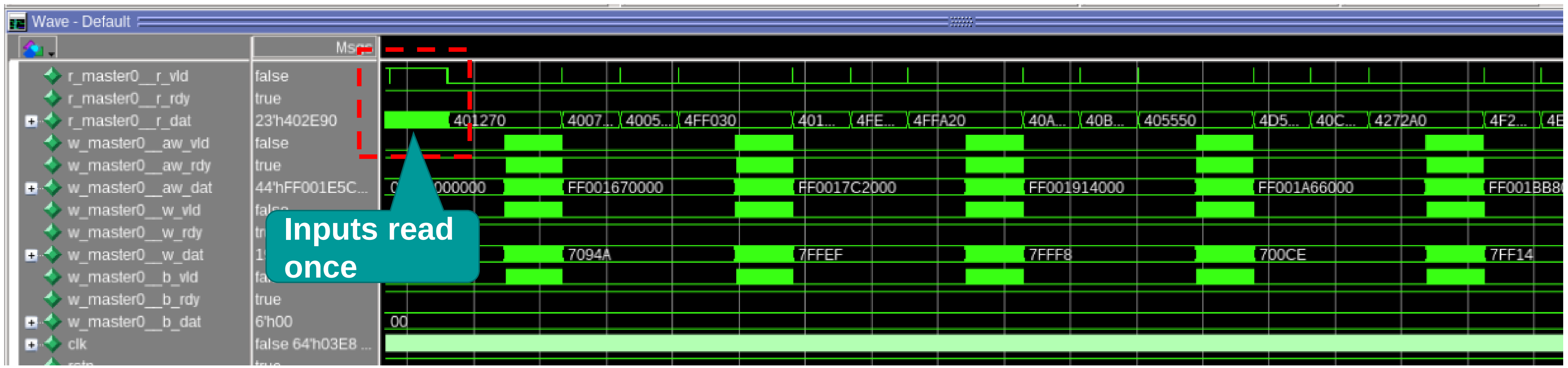
```
class bus_interface : public sc_module, public local_axi {
 public:
  sc_in<bool> clk);
  sc_in<bool> rstn);

  //AXI4 Bus I/Fs
  r_master<> r_master0;
  w_master<> w_master0;
  typename local_axi4_lite::write::template slave<> w_slave0;

  //User I/F to DUT
  //Read I/F
  Connections::In<MEM_ADDR_TYPE>  mem_in_addr;
  Connections::In<BURST_TYPE>     mem_in_burst;
  Connections::Out<DTYPE>         mem_in_data;
…
 void read_master_process() {
…
   ar.ex_len = mem_in_burst.Pop();
   ar.addr = uint32(mem_in_addr.Pop())<<1;
   r_segment0_ex_ar_chan.Push(ar);
   while (1) {
     r_payload r = r_master0.r.Pop();
     data. = r.data;
     mem_in_data.Push(data);
     if (ar.ex_len-- == 0) { break; }
   }
 }
```

Step 5: Adding AXI4 I/Fs and On-chip Buffering

- Bus traffic dramatically reduced to reading input feature maps once

- Simulation time was 0.93 secs for one inference
- Simulation wall-clock time was 20 minutes

Pre-HLS Simulation



SIEMENS

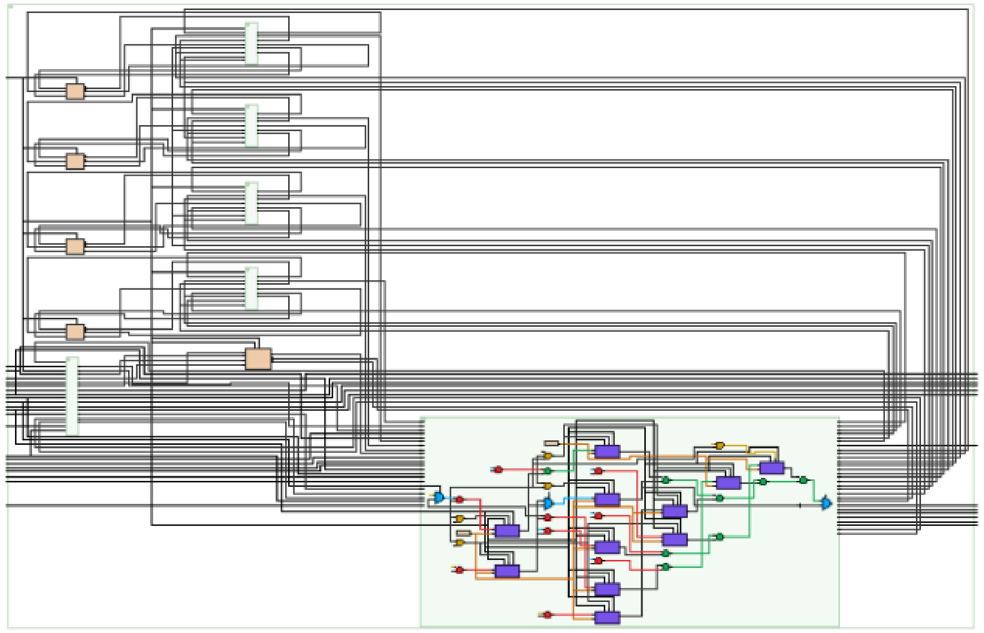
Synthesis and RTL Verification

Post-HLS Synthesis RTL Simulation Results

Post-HLS simulation results were very close to the pre-HLS simulation

Post-HLS simulation runtime was over 30x longer than the pre-HLS simulation

- Not practical for simulating multiple frames of video



Simulation Type	Simulation Time (secs)	Simulation Wall-clock time(mins)
Pre-HLS	0.93	20
Post-HLS	0.97	630

SIEMENS

Source Code Steps

Source code examples and other tutorials can be found at:

- https://hlslibs.org/
- https://github.com/hlslibs

SIEMENS

Summary

- Increasing AI/ML algorithm complexity is making RTL verification more difficult
- MatchLib and SystemC allows designers to model and verify the true hardware performance, catching bugs early that would normally be exposed during system integration when it’s too late
- MatchLib models can be directly synthesized to RTL and performance of the pre-hls and post-hls results are near identical
- Customers are using MatchLib today to solve the design challenges associated with building AI/ML hardware

SIEMENS

Thank you!

SIEMENS