Chapter 8 (B)

SystemC Machine Learning Automotive Application

B

LSTM

Zainalabedin Navabi

Slides prepared by Nooshin Nosrati

SystemC Machine Learning Automotive Application

- Introduction
- + DiBA, an RTL Hardware Architecture for LSTM (Top-Down)
- + SystemC Implementation of DiBA (Bottom-Up)

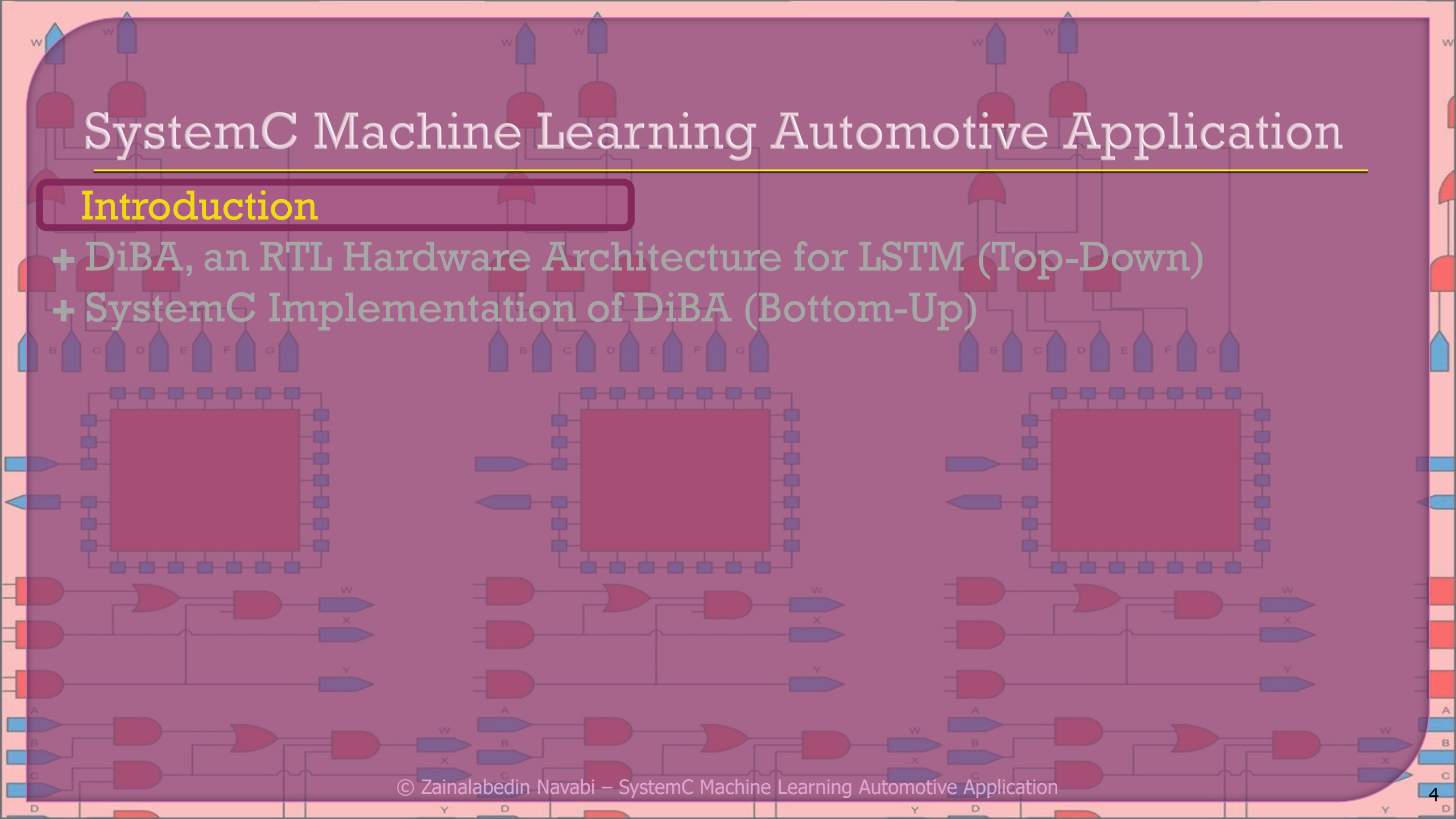
SystemC Machine Learning Automotive Application

Introduction

- DiBA, an RTL Hardware Architecture for LSTM (Top-Down)
  - Data Arrangement
  - Matrix-multiplication and Accumulation Plates (MAPs)
    - Multiply and Accumulate (MAC)
    - Convergence Adder Plate (CAP)
    - Pipeline
  - Activation and Accumulation Plate (AAP)
- SystemC Implementation of DiBA (Bottom-Up)
  - Multiply and Accumulate (MAC)
  - Convergence Adder Plate (CAP)
  - Matrix-multiplication and Accumulation Plates (MAPs)
  - Activation and Accumulation Plate (AAP)
  - nDimensional Bitslice Architecture (DiBA)
  - Data Arrangement (Testbench)

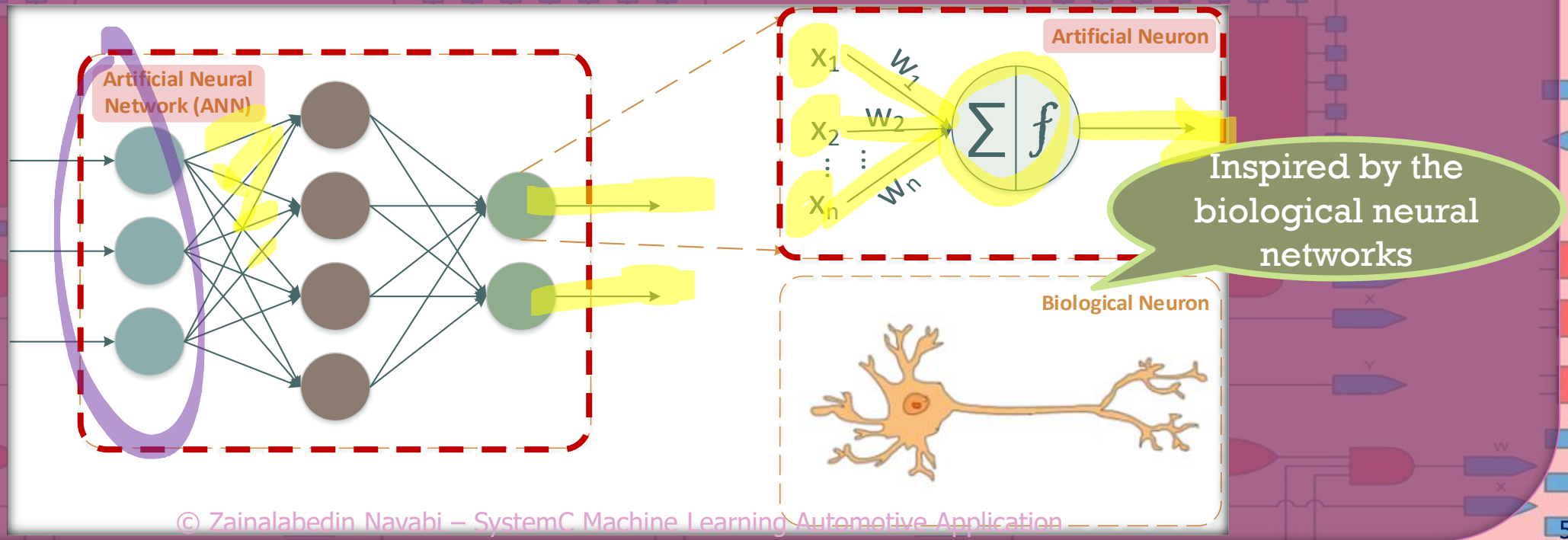
SystemC Machine Learning Automotive Application

- Introduction
- + DiBA, an RTL Hardware Architecture for LSTM (Top-Down)
- + SystemC Implementation of DiBA (Bottom-Up)



Artificial Neural Network

- Is an information processing paradigm
- Allows modeling of nonlinear processes
- Consists of a large number of simple units (neurons) which are interconnected together in layers

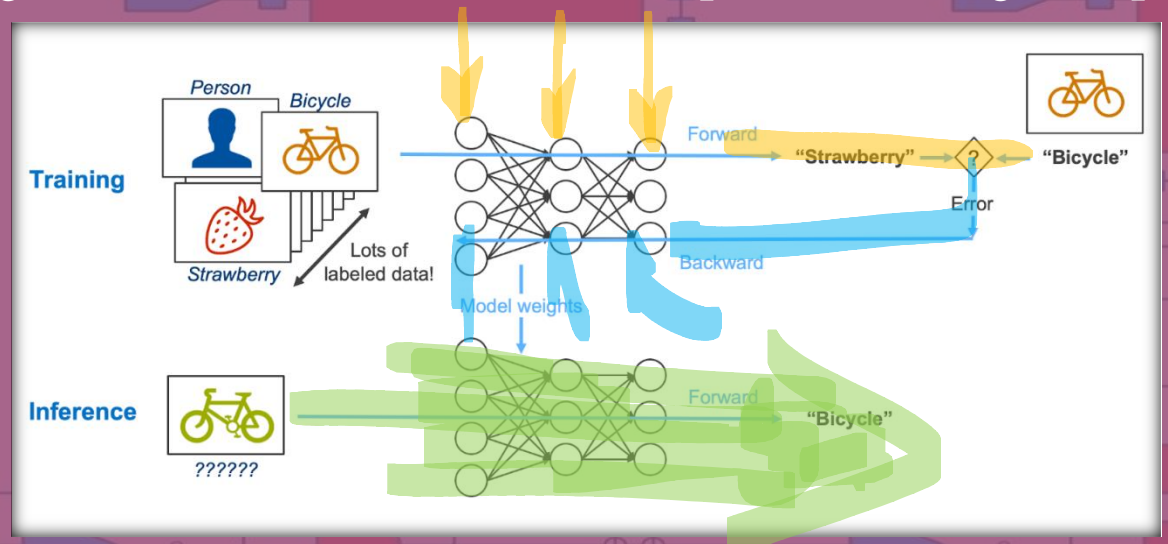


Artificial Neural Network

- **Training Phase: Adjusting the weights**
  - Known data is fed to the DNN, and the DNN makes a prediction about what the data represents. Any error in the prediction is used to update the strength of the connections between the artificial neurons. As the training process continues, the connections are further adjusted until the DNN is making predictions with sufficient accuracy
- **Inference Phase: Using the deep learning model**
  - The process of using a trained DNN model to make predictions against previously unseen data

A compute-intensive process often run in a data center

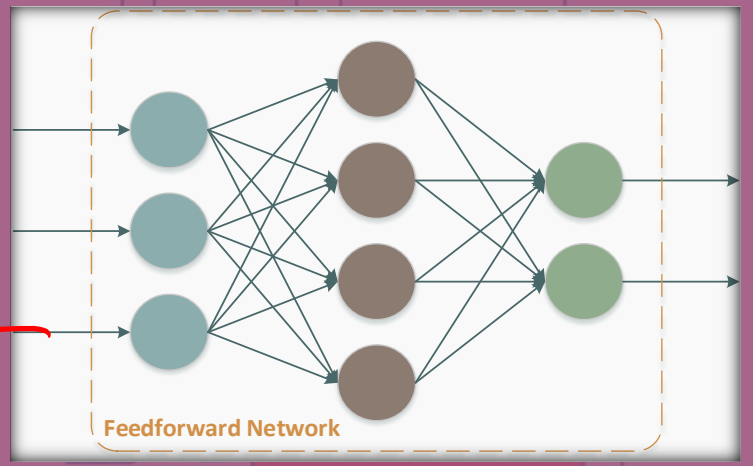
Efficient processing to meet real-world power and performance requirements

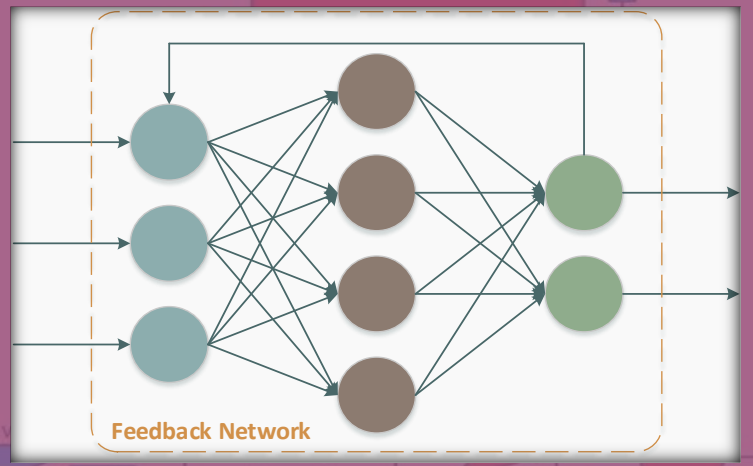


https://www.intel.com/content/www/us/en/artificial-intelligence/posts/deep-learning-training-and-inference.html

Artificial Neural Network

- An ANN can be characterized by
  - Network Topology
    - Feedforward Network
    - Feedback Network
  - Learning (Adjustments of weights)
    - Supervised Learning
    - Unsupervised Learning
  - Activation Functions
    - Linear
    - Unit step function
    - Sigmoid
    - ArcTan
    - Rectified Linear Unit (ReLU)
    - …





ANN Applications

- Classification
- Clustering
- Regression
- Pattern recognition
- Dimension reduction
- Structured prediction
- Machine translation
- Anomaly detection
- Decision making
- Visualization
- Computer vision
- ….

Some types of ANN

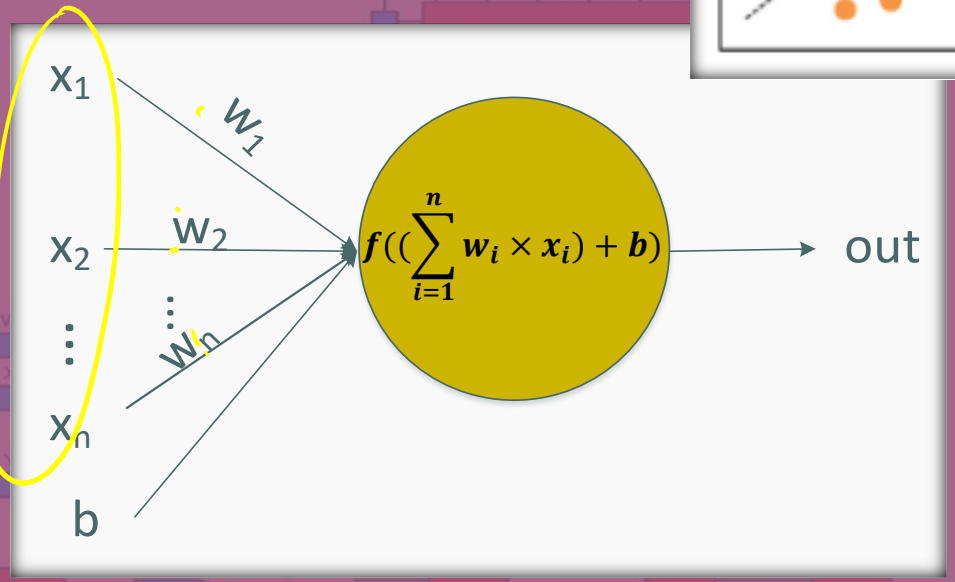
- Multi-Layer Perceptron (MLP)
- Convolutional Neural Network (CNN)
- Recurrent neural network (RNN)
- Long short-term memory (LSTM)
- Gated Recurrent Unit (GRU)
- Support Vector Machine (SVM)
- ….

Perceptron

- Is an artificial neuron using the unit step function as the activation function
- Is an algorithm for supervised learning of binary classifiers
- $Out = \begin{cases} 1, if\ (\mathbf{W}.\mathbf{X} + b) > 0 \\ 0,\ otherwise \end{cases}$

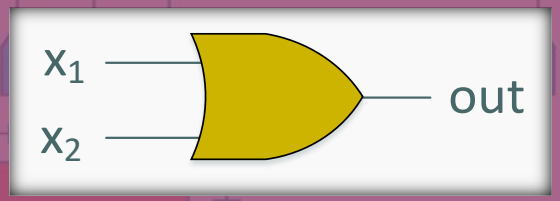
$$\mathbf{W}.\mathbf{X} = \sum_{i=1}^{n} w_i \times x_i$$

- Learning algorithm
  0. Initialize weight values and bias
  1. Calculate the output
  2. Check the error
     - $Error = Out_j - Target_j$
  3. Update weights and bias
     - $w_{ij}(new) = w_{ij}(old) + \gamma x_{ij} Target_j$, γ: learning rate
     - $b_j(new) = b_j(old) + \gamma\, Target_j$
  - Repeat for all training examples in a database
- Is also termed the single-layer perceptron



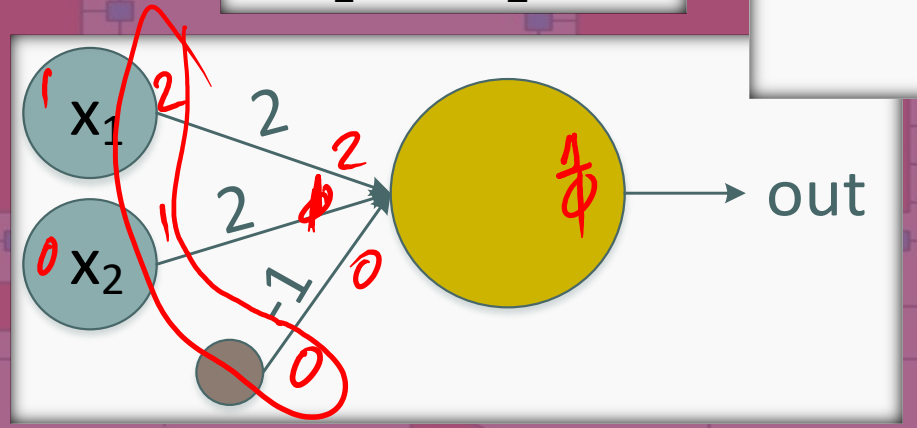
Perceptron for Logic Gate Modeling

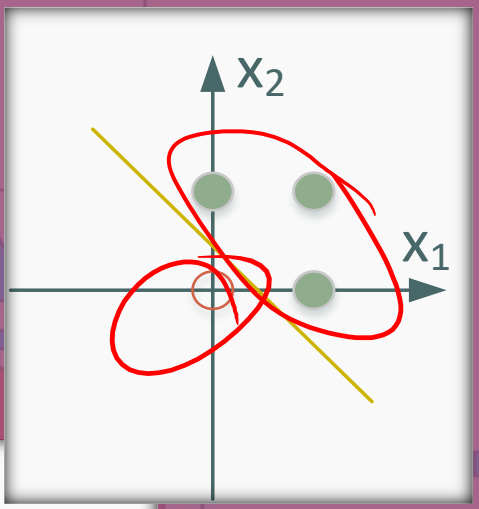
- OR Gate



x1	x2	out
0	0	0
0	1	1
1	0	1
1	1	1

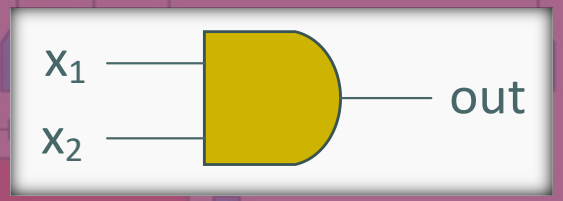
$2x_1 + 2x_2 - 1$



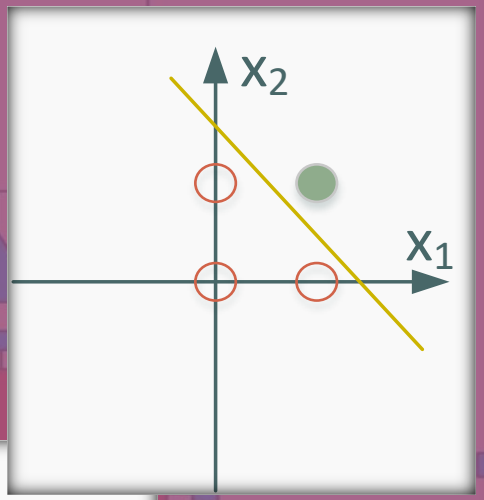


Perceptron for Logic Gate Modeling

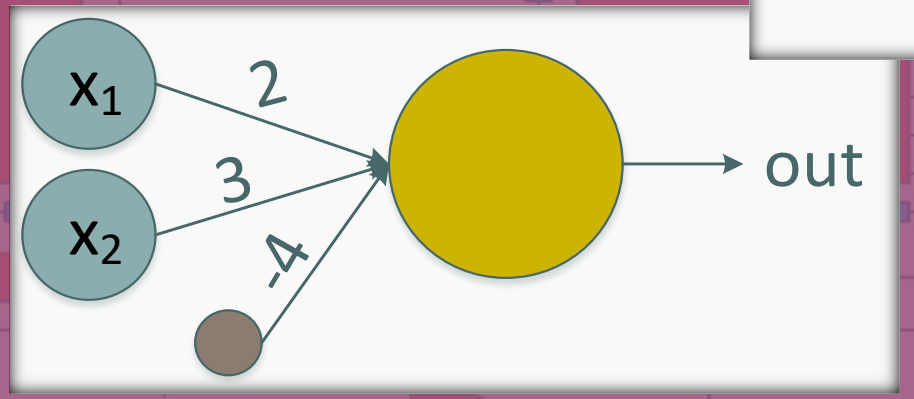
- AND Gate



x1	x2	out
0	0	0
0	1	0
1	0	0
1	1	1

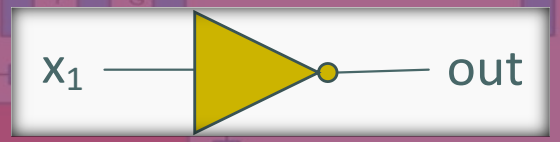


$$2x_1 + 3x_2 - 4$$



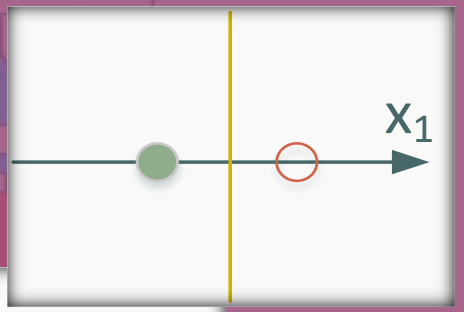
Perceptron for Logic Gate Modeling

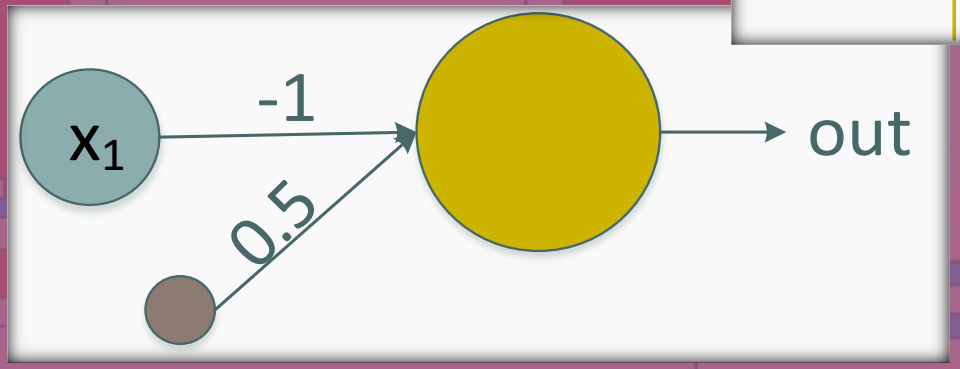
- NOT Gate



x1	out
0	1
1	0

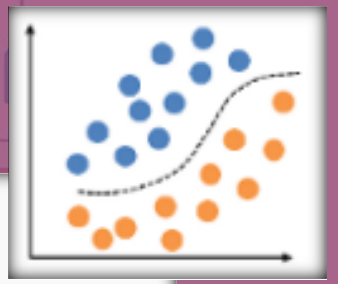
$-x_1 + 0.5$

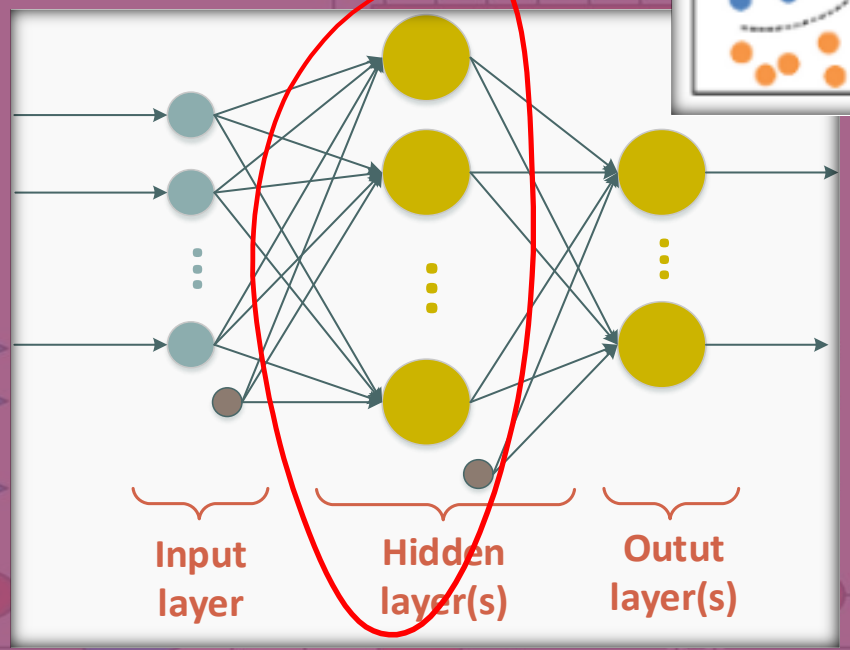




Multi-layer Perceptron (MLP)

- Is a class of feedforward artificial neural network
- Consists of at least three layers of nodes: an input layer, a hidden layer and an output layer
- Except for the input nodes, each node is a neuron that uses a nonlinear activation function
- Utilizes a supervised learning technique called backpropagation for training
- Can distinguish data that is not linearly separable
- Applications
  - Is useful in research for their ability to solve problems stochastically, which often allows approximate solutions for extremely complex problems like fitness approximation
  - Can be used to create mathematical models by regression analysis
  - Was a popular machine learning solution in the 1980s, finding applications in diverse fields such as speech recognition, image recognition, and machine translation



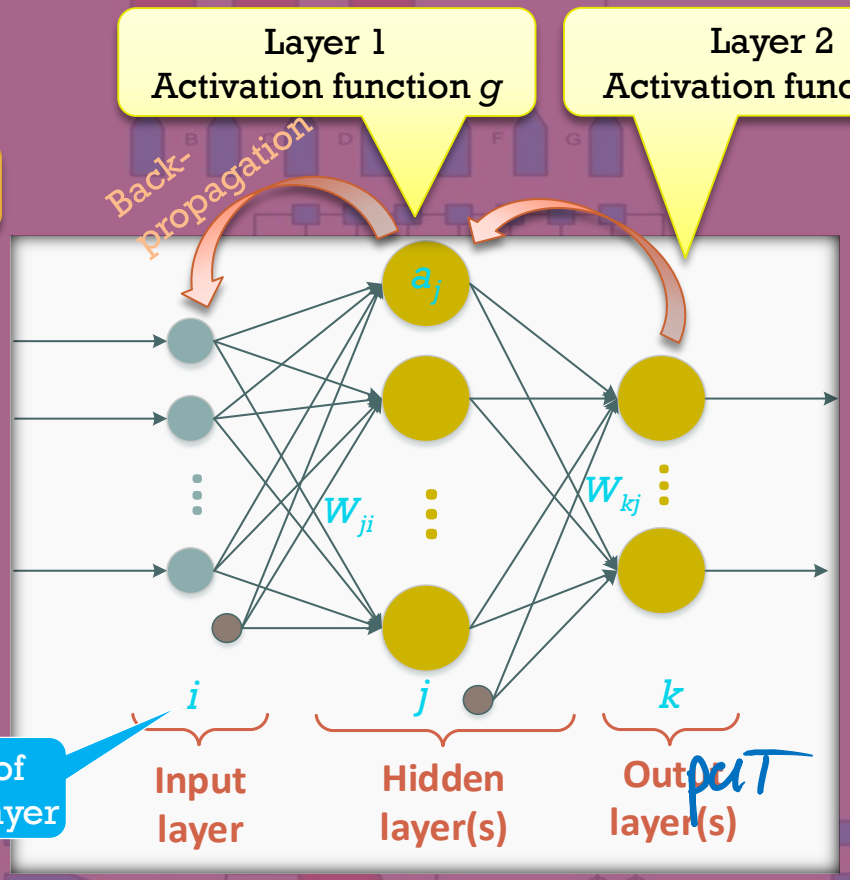


Multi-layer Perceptron (MLP)(cont.)

- Learning algorithm
  0. Initialize weight values and biases
  1. Forward-propagate and calculate the output
  2. Calculate the error (loss function)
     - $Error_j = \frac{1}{2}\sum_j \|Out_k - Target_k\|^2$

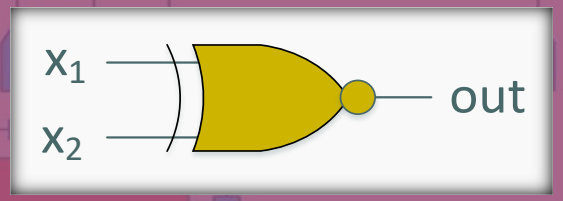
     Least Mean Square (LMS) Error Cost Function
  3. Back-propagate and adjust weights and biases
     - Layer2:
       - $w_{kj}(new) = w_{kj}(old) - \gamma \cdot (Out_k - Target_k) \cdot f' \cdot a_j \quad ; \quad a_j = \sum_i (x_i \cdot w_{ji} + b)$
       - $b_k(new) = b_k(old) - \gamma \cdot (Out_k - Target_k) \cdot f'$

       γ: learning rate
     - Layer1:
       - $w_{ji}(new) = w_{ji}(old) - \gamma \cdot (\sum_k (Out_k - Target_k) \cdot f' \cdot w_{kj}) \cdot g' \cdot x_i$
       - $b_j(new) = b_j(old) - \gamma \cdot (\sum_k (Out_k - Target_k) \cdot f' \cdot w_{kj}) \cdot g'$
  - Repeat for all training examples in a database



MLP for Logic Gate Modeling

- XNOR Gate



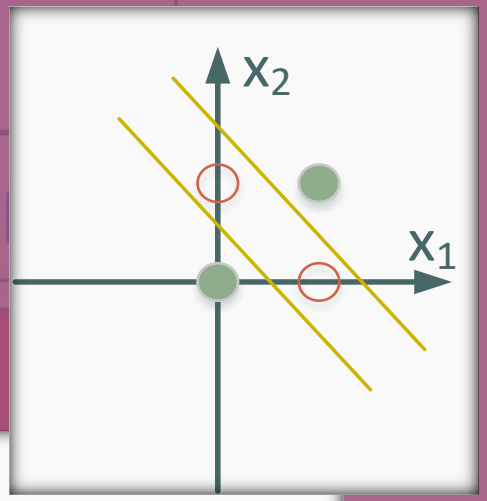
x1	x2	out
0	0	1
0	1	0
1	0	0
1	1	1

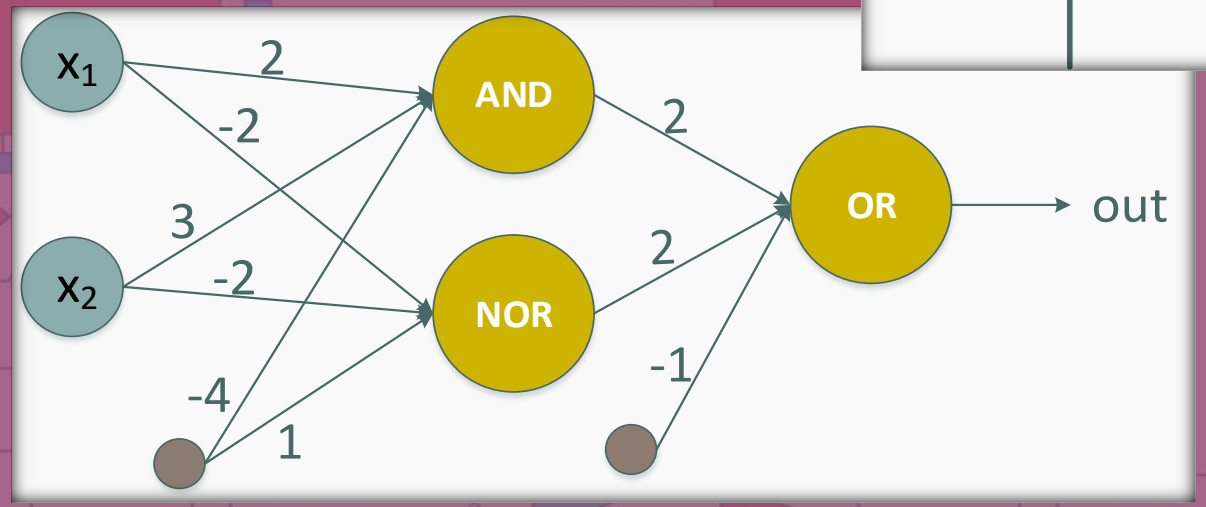
$$\overline{x_1 \oplus x_2} = x_1 x_2 + \overline{x_1}.\overline{x_2}$$

$$AND(2x_1 + 3x_2 - 4)$$

$$NOR(-2x_1 - 2x_2 + 1)$$

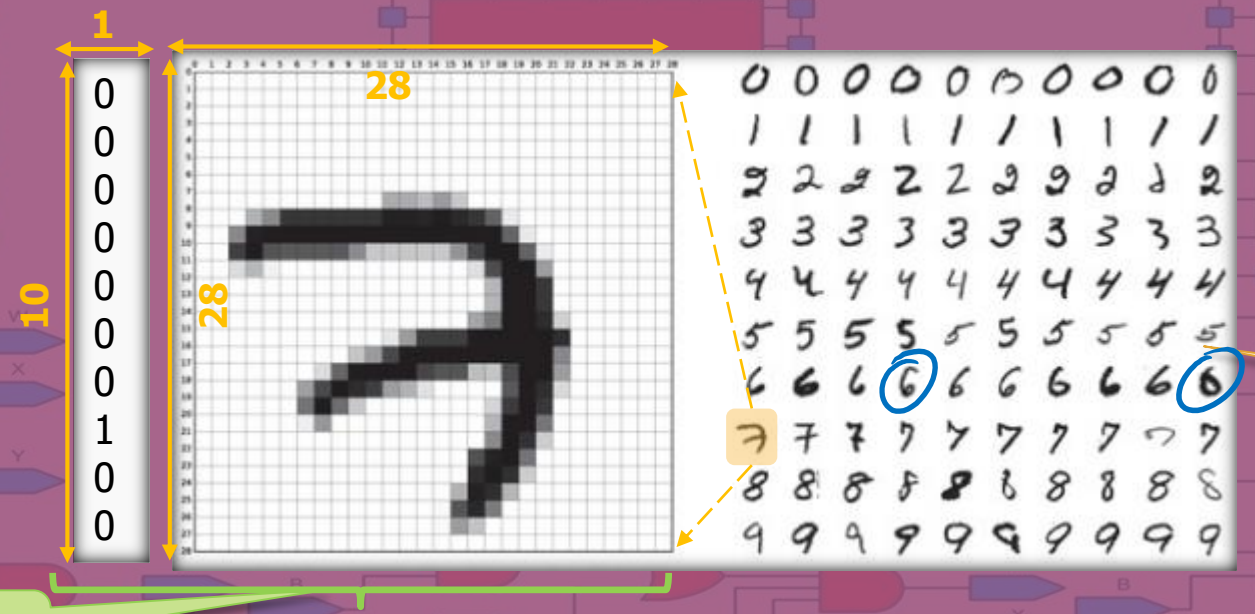
$$OR(2x_1 + 2x_2 - 1)$$





MLP for Image Classification

- MNIST Database
  - Is a well-known dataset for handwritten digit classification task
  - Contains images with 28 × 28 pixels in gray scale, each coped with a label from 0 to 9, indicating the corresponding digit
  - There are 60000 training images and 10000 test images

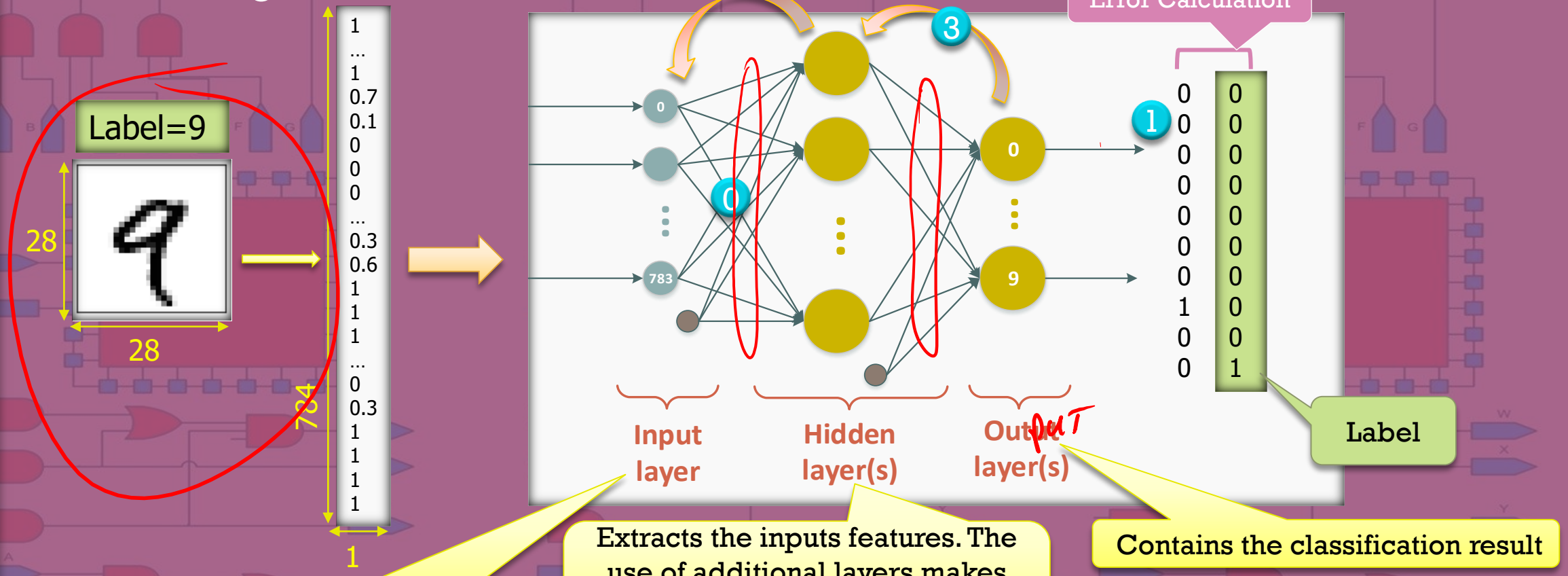


100 samples from MNIST training set

Digit 7 and its corresponding label

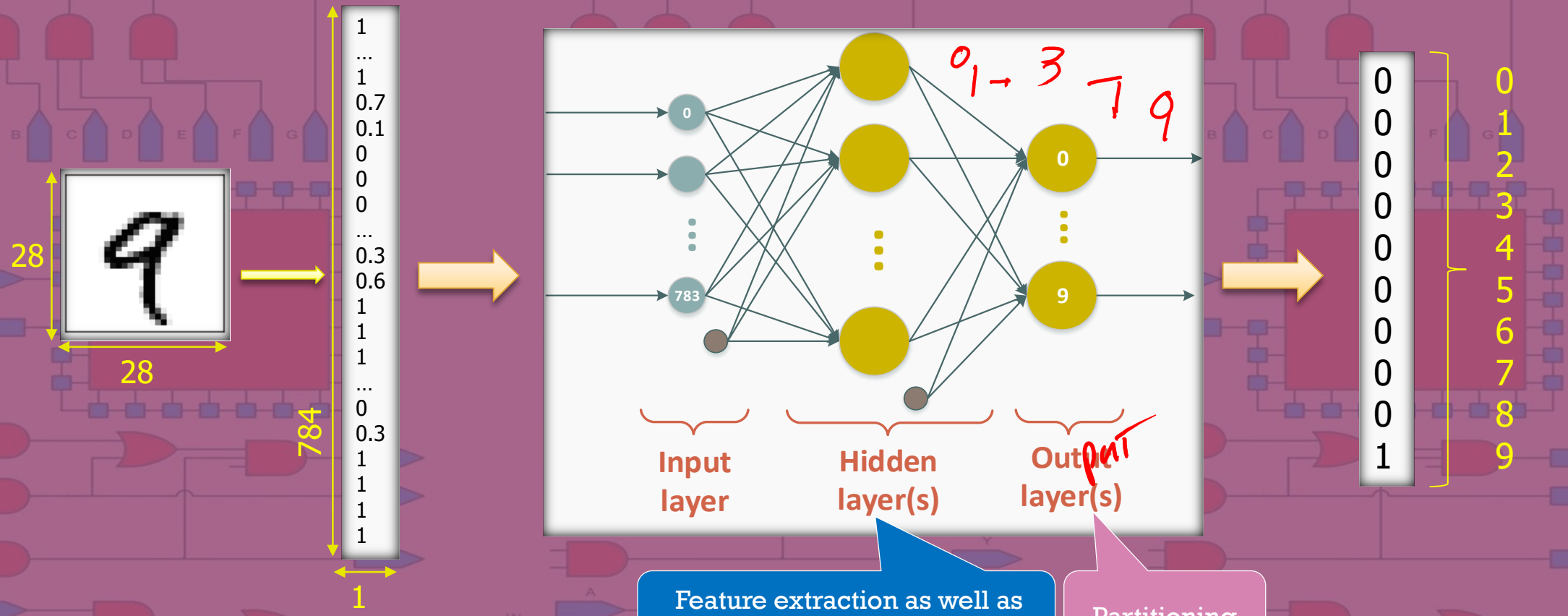
MLP for Image Classification

- Training Phase



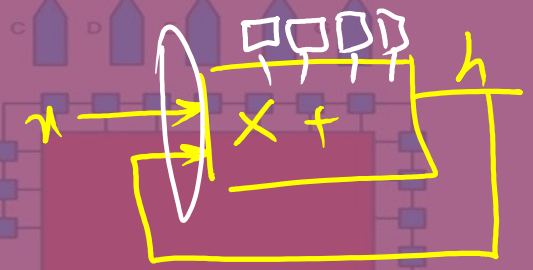
MLP for Image Classification

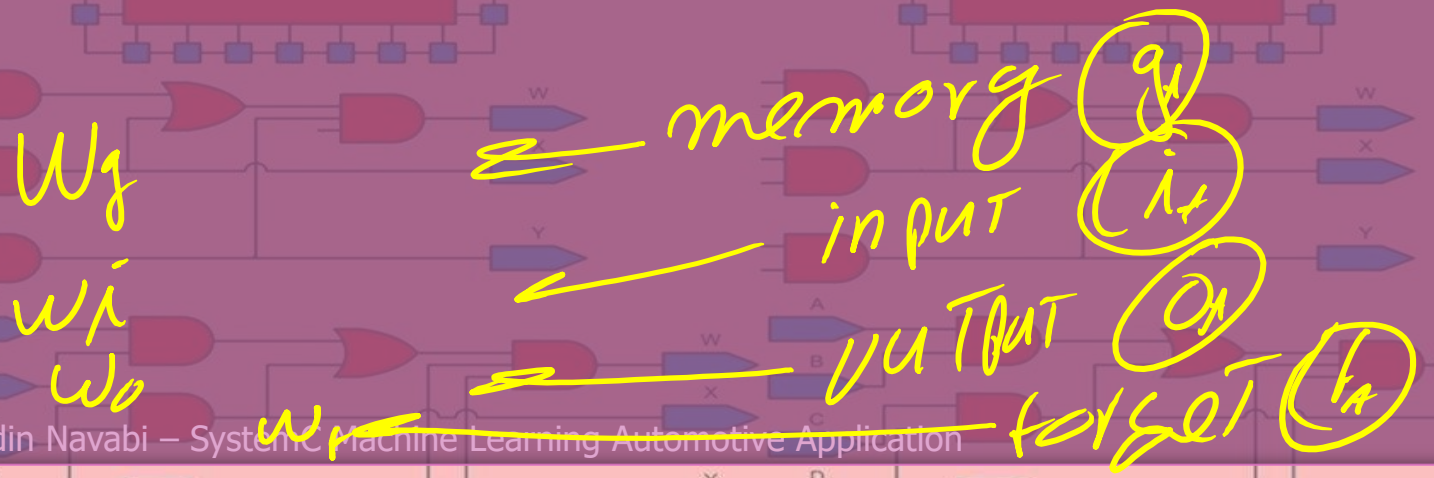
- Inference Phase



LSTM Neural Network

- Is generally used for recognition of dynamic behavior of a problem based on a sequence of inputs
- Uses a recurrence of internal states or outputs as part of the input addition to data input
- An LSTM unit is composed of four main elements
  - A memory cell (g_t) for holding data
  - Three gates for regulating information flow, which are input gate (i_t)
  - Output gate (o_t)
  - Forget gate (f_t)

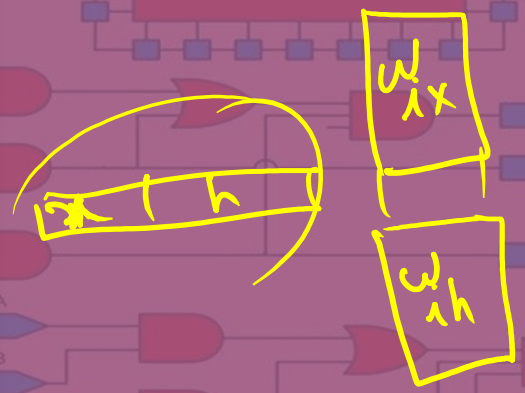




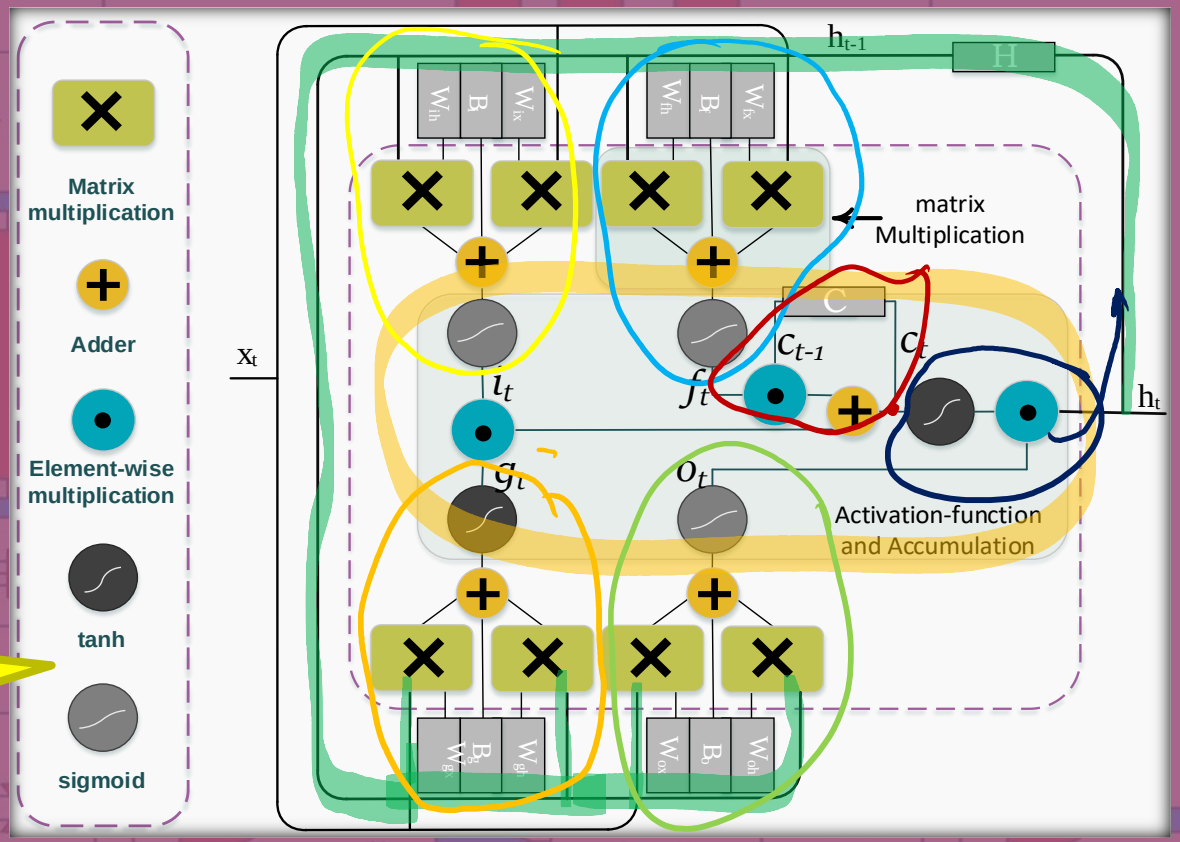
LSTM Neural Network (cont.)

- The LSTM neuron can be characterized by the following set of equation

$$g_t = tanh(w_{gx}x_t + w_{gh}h_{t-1} + B_g)$$
$$i_t = sig(w_{ix}x_t + w_{ih}h_{t-1} + B_i)$$
$$o_t = sig(w_{ox}x_t + w_{oh}h_{t-1} + B_o)$$
$$f_t = sig(w_{fx}x_t + w_{fh}h_{t-1} + B_f)$$
$$c_t = f_t \odot c_{t-1} + i_t \odot g_t$$
$$h_t = o_t \odot tanh(c_t)$$

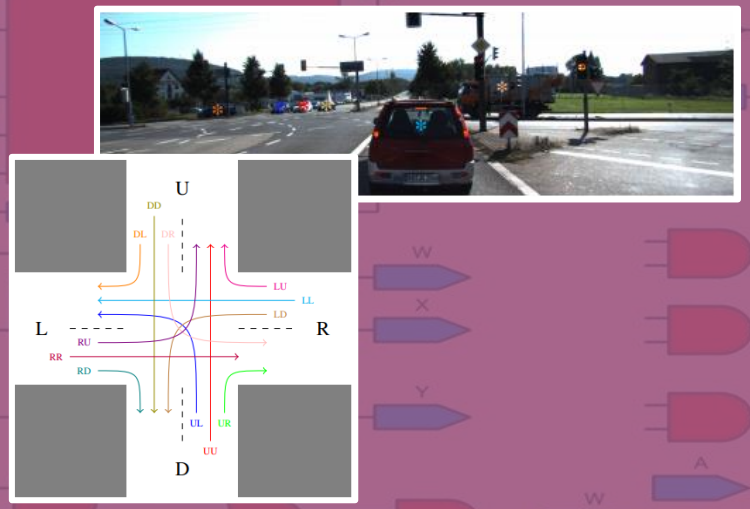


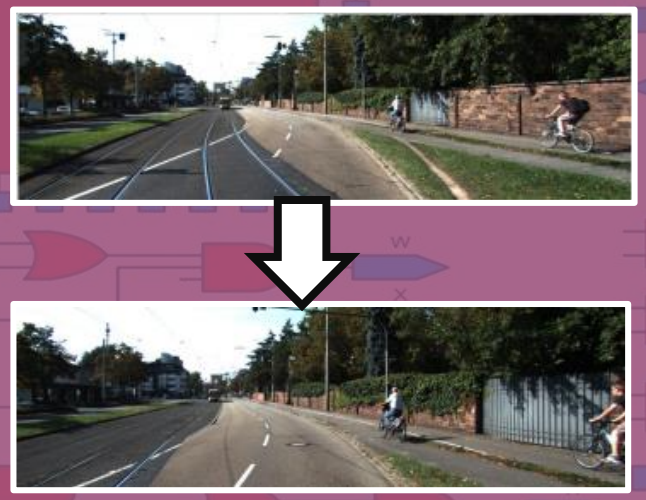
Receives the x_t input and the output of the previous time step h_{t-1} to calculate h_t

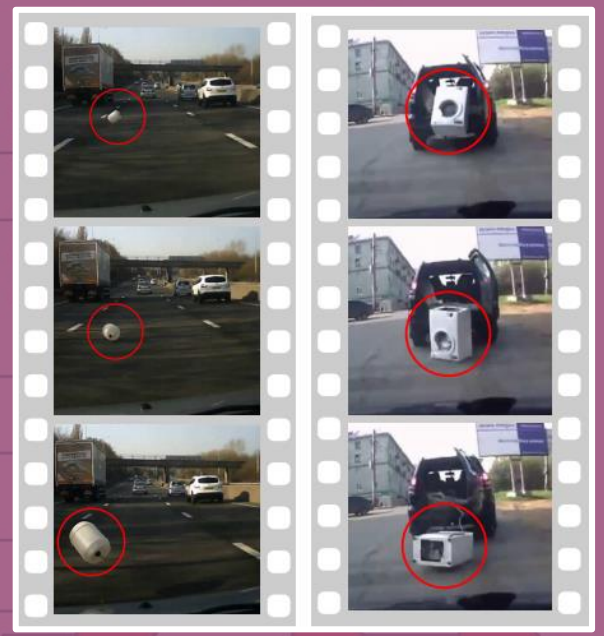


LSTM in Automotive Application

- The property of a dynamic object based on its motion pattern
- Video frame prediction
- Surround Vehicles Trajectory Analysis
- ....





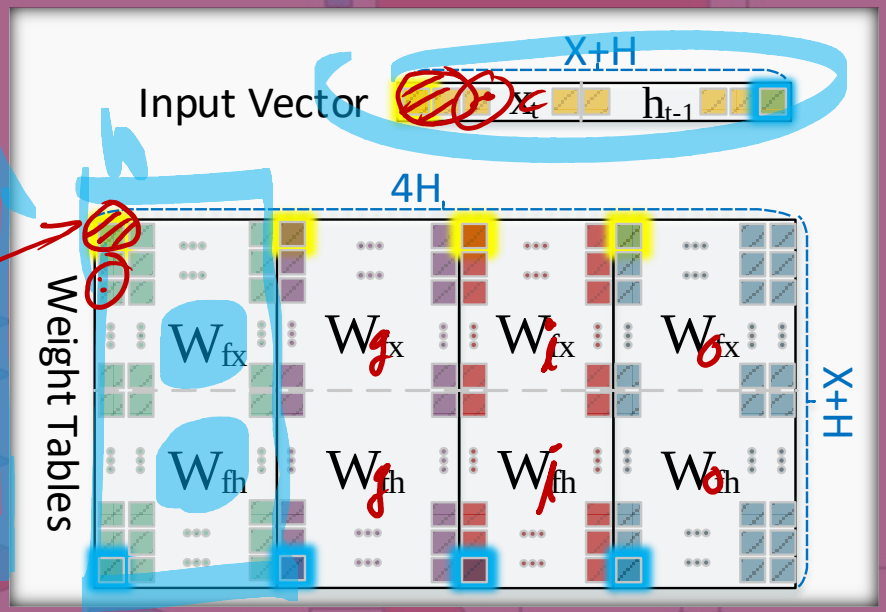


SystemC Machine Learning Automotive Application

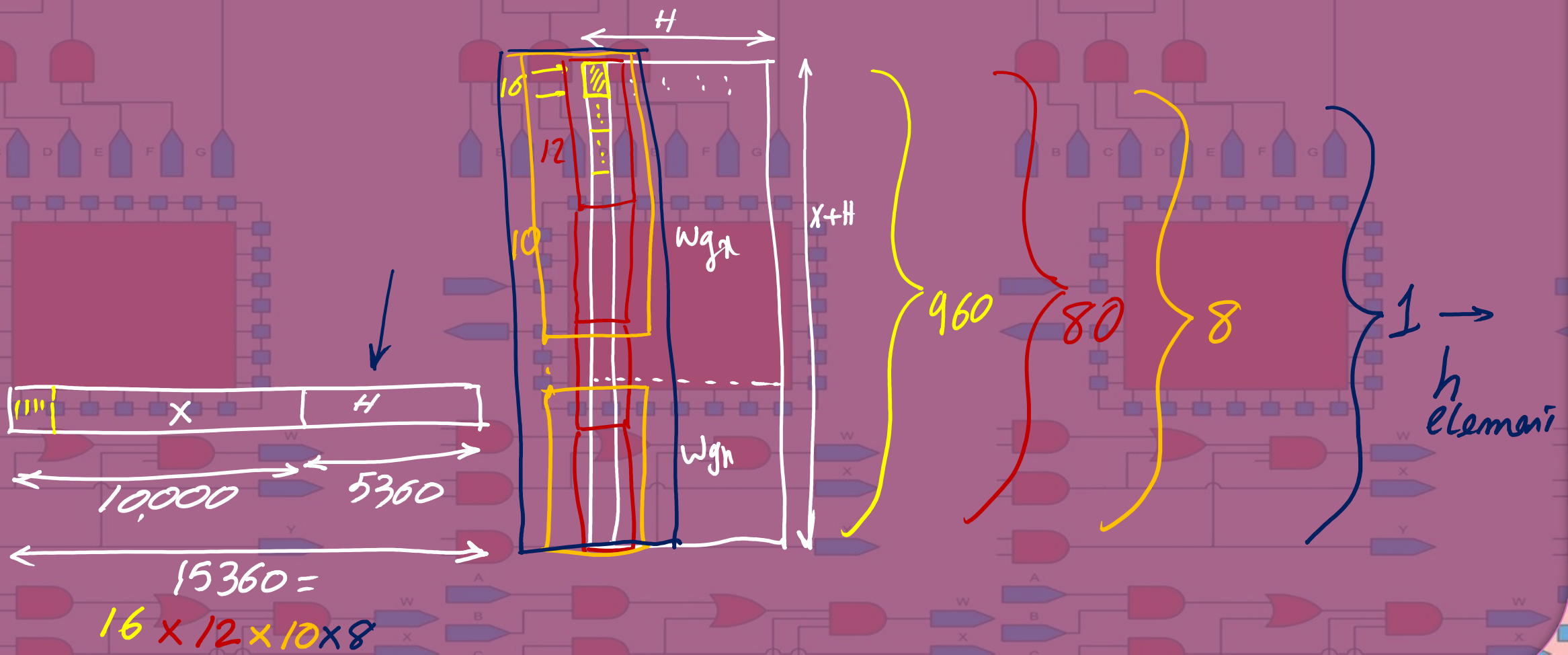
- Introduction
- DiBA, an RTL Hardware Architecture for LSTM (Top-Down)
  - Data Arrangement
  - Matrix-multiplication and Accumulation Plates (MAPs)
    - Multiply and Accumulate (MAC)
    - Convergence Adder Plate (CAP)
    - Pipeline
  - Activation and Accumulation Plate (AAP)
- SystemC Implementation of DiBA (Bottom-Up)

Data Arrangement

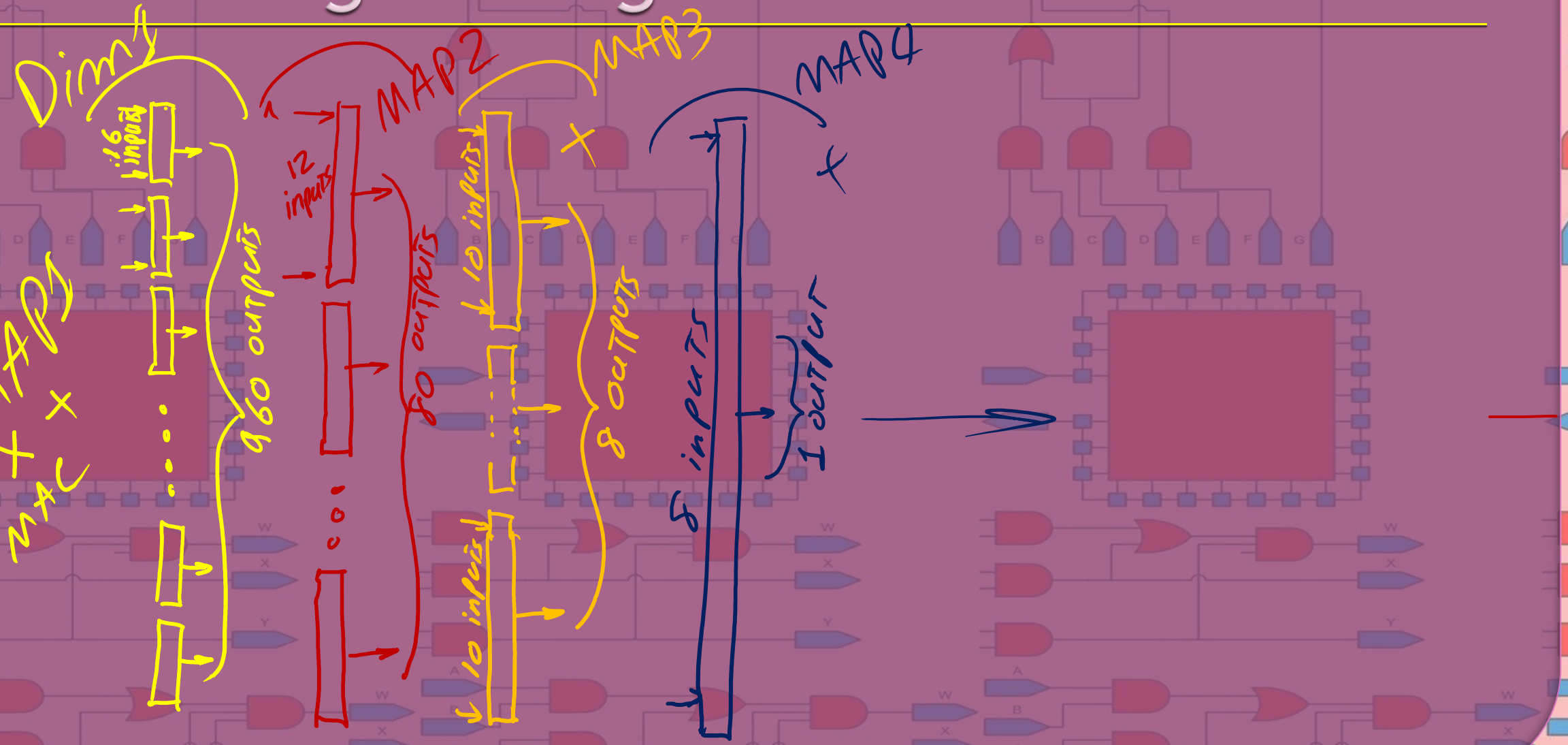
- To calculate a h_t element of the LSTM module, we require four Matrix Multiplications
- To arrange input elements of Matrix Multiplications, i.e., input vector and weight
  - x_t and h_{t-1} with sizes X and H respectively are concatenate to form the P input vector
  - P also indicates the problem size that is $X+H$
  - Row-wise concatenation of the x and h weights are performed to form four weight tables with H columns and $X+H$ rows



Data Arrangement



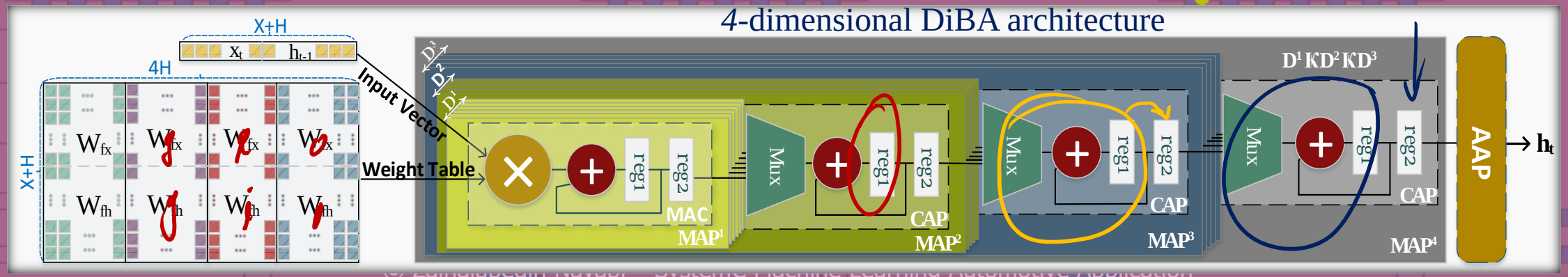
Processing Arrangement



DiBA Architecture – Overall View

- Consists of two parts:
  - MAPs hardware structure
    - Stands for Matrix-multiplication and Accumulation Plate
    - Are responsible for multiplication and accumulations
    - Performs a complete MAC operation on all elements of its two input vectors
  - AAP hardware structure
    - Stands for Activation and Accumulation Plate
    - Is responsible for activation function and accumulations
    - Takes the MAP outputs and applies the activation function

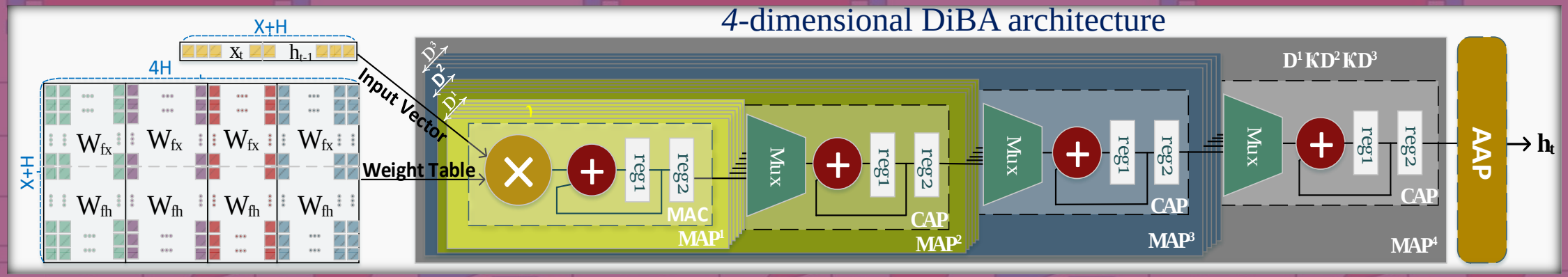
DiBA: n-Dimensional Bitslice Architecture



DiBA Architecture – MAPs: MAC

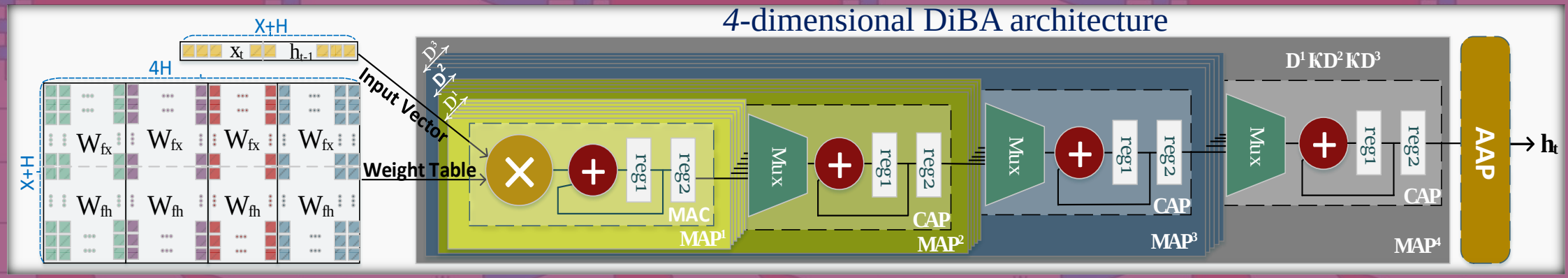
16 12 10 8

- Consists of MAP^1, MAP^2, MAP^3, and MAP^4 structures in this example
- A MAP^1 structure that is only responsible for MAC operation of a segment of the problem size, sequentially multiplies and adds the elements of its two operands
  - In each clock a two-operand multiply-and-add operation is performed in MAP^1s
  - There are enough number of MAP^1 structures operating concurrently
  - In addition to the datapath shown below, a MAP also has a controller that is responsible for continuing the necessary number of loops, and when completed, triggering the next level hardware (perhaps a CAP or an AAP). A simple counter is all that is needed for this controller



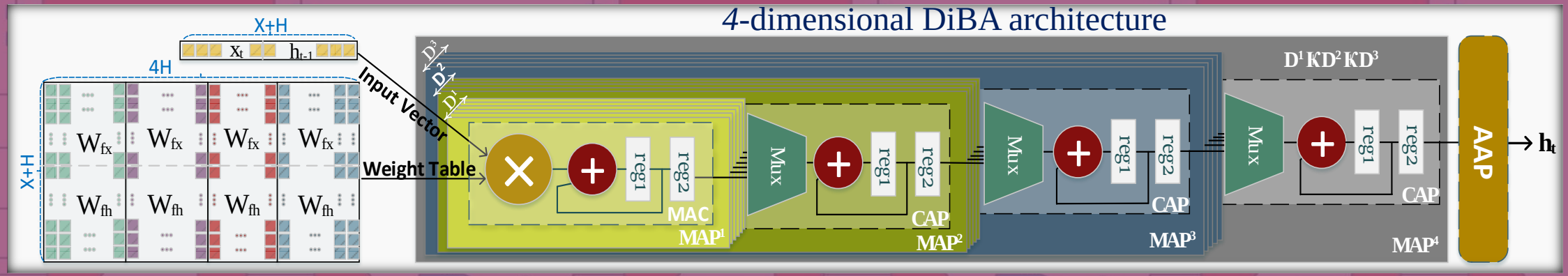
DiBA Architecture – MAPs: CAP

- The *n1* group of MAP[1] structures are put into each MAP[2] where there are *n2* MAP[2]. In a MAP[2], the outputs of *n1* MAP[1] are sequentially added by a structure that we refer to as a CAP
  - CAP stands for Convergence Adder Plate
- Formation of upper-dimension MAP structures follow the same procedure in an upper-level of hierarchy until we reach MAP[4] (the last MAP) where the *n3* MAP[3] converge to a CAP
- The CAP controller is programmed for a specific number of connected MAP units. As with a MAP controller, a CAP controller is basically a counter that selects the inputs



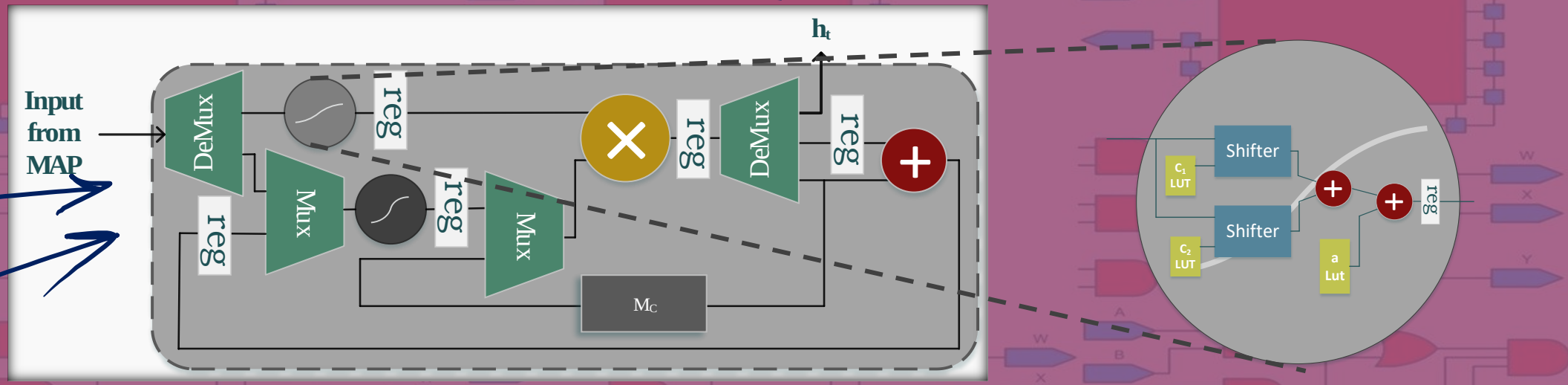
DiBA Architecture – Pipeline

- Allows a 4-stage pipeline for all required MAC operations
- When MAP^1 structures complete their work on a weight table column, MAP^2 structures engage, and MAP^1s are free to take on the same column of the next weight table
- When DiBA moves into Dimension 3 for the first table column, MAP^2s take on the same column of the second table, and MAP^1s perform MAC tasks on the first column of the third table
- When MAP^1s complete a column of the fourth table, pipelining moves into the next column of the first table. This concurrency makes DiBA a 4-stage pipeline structure



DiBA Architecture – AAP

- Receives four elements resulting from multiplications of x_t and h_{t-1} by the four weight matrices
- Calculations of the two activation functions (*sigmoid()* and *tanh()*) are implemented by just two shifters and two adders, instead of using large exponential and floating-point hardware or large LUTs
- The activation functions are calculated by piecewise linear approximation with split logarithmic coefficients, as in $f = (c_1 + c_2) \cdot x + a$



SystemC Machine Learning Automotive Application

Introduction

\+ DiBA, an RTL Hardware Architecture for LSTM (Top-Down)

– SystemC Implementation of DiBA (Bottom-Up)

- Multiply and Accumulate (MAC)
- Convergence Adder Plate (CAP)
- Matrix-multiplication and Accumulation Plate (MAP)
- Activation and Accumulation Plate (AAP)
- n-Dimensional Bitslice Architecture (DiBA)
- Data Arrangement (Testbench)

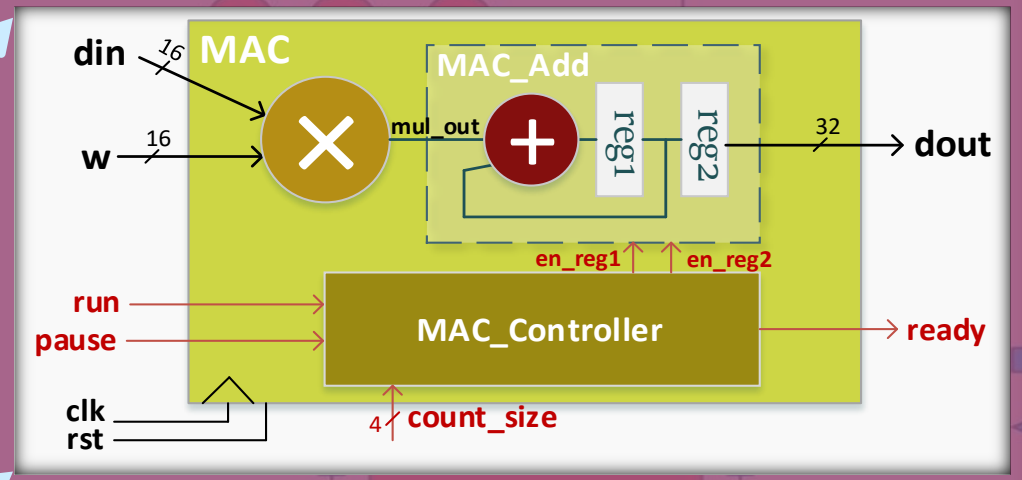
SystemC Machine Learning Automotive Application

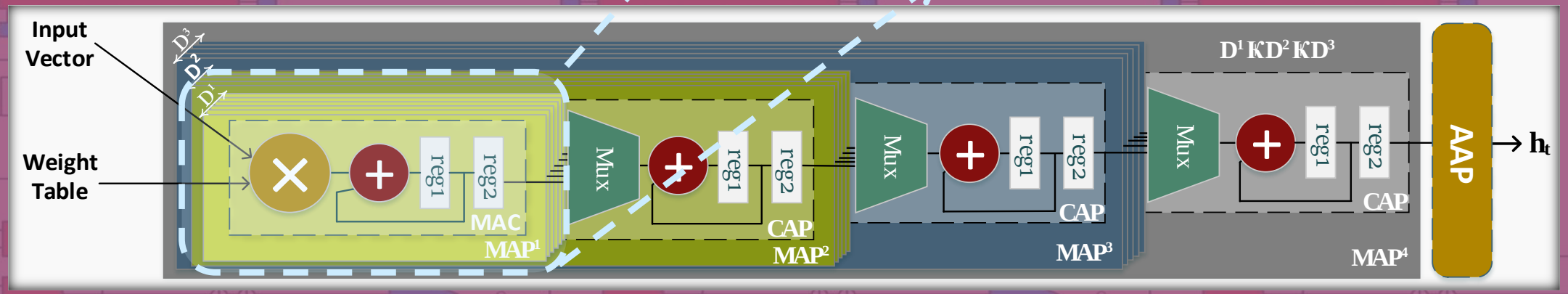
Introduction

\+ DiBA, an RTL Hardware Architecture for LSTM (Top-Down)

– SystemC Implementation of DiBA (Bottom-Up)

- Multiply and Accumulate (MAC)
- Convergence Adder Plate (CAP)
- Matrix-multiplication and Accumulation Plate (MAP)
- Activation and Accumulation Plate (AAP)
- n-Dimensional Bitslice Architecture (DiBA)
- Data Arrangement (Testbench)



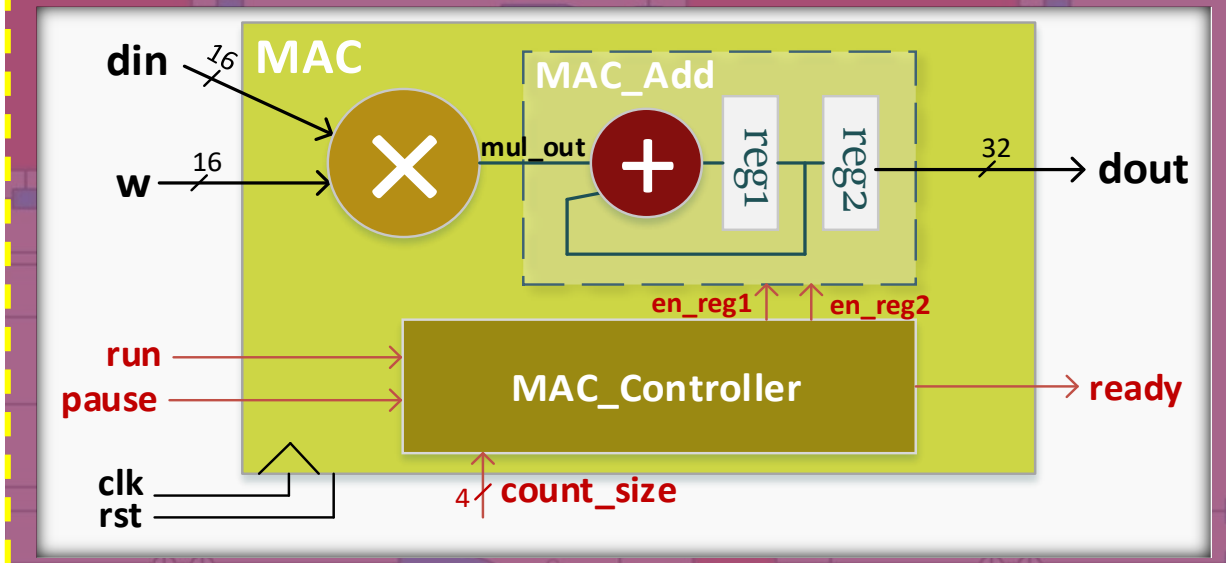


Multiply and Accumulate (MAC)

MAC.h

```
#include <systemc.h>
#include "MAC_Add.h"
#include "MAC_Controller.h"
using namespace std;

template <int n> SC_MODULE(MAC) {
    sc_in<sc_logic> clk, rst, run, pause;
    sc_out<sc_logic> ready;
    sc_in<sc_lv<16>> din;
    sc_in<sc_lv<16>> w;
    sc_out<sc_lv<32>> dout;

    sc_signal<sc_lv<32>> mul_out;
    sc_signal<sc_logic> en_reg1, en_reg2;
    sc_signal<sc_lv<4>> count_size;

    MAC_Add<32>* add;
    MAC_Controller* MAC_Cntrlr;

    SC_CTOR(MAC) {
        int count_size_int = static_cast<int>(n);
        count_size.write(count_size_int);

        add = new MAC_Add<32>("MAC_Add_module");
        (*add)(clk, rst, en_reg1, en_reg2, mul_out, dout);

        MAC_Cntrlr = new MAC_Controller("MAC_Controller_module");
        (*MAC_Cntrlr)(clk, rst, run, pause, count_size, en_reg1, en_reg2);

        SC_METHOD(process_others);
        sensitive << din << w << en_reg2;
    }
    void process_others();
};
```

```
template <int n> void MAC<n>::process_others(){
    if (din.read().range(0, 0) == "Z" || din.read().range(0, 0) == "X") mul_out = (din.read(),w.read());
    else if (w.read().range(0, 0) == "Z" || w.read().range(0, 0) == "X") mul_out = (din.read(),w.read());
    else mul_out = din.read().to_int() * w.read().to_int();
    ready = en_reg2;
}
```



Multiply and Accumulate (MAC)

- Multiply and Accumulate (MAC): MAC_Add

MAC_Add.h

MAC.h | MAC_Controller.h

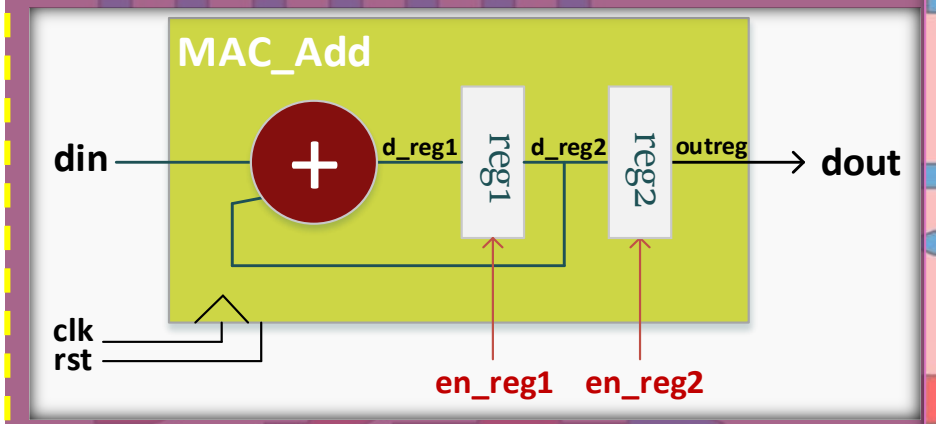
MAC_Add<T>

```
1   #pragma once
2   #include <systemc.h>
3   using namespace std;
4
5   template <int T> SC_MODULE(MAC_Add) {
6       sc_in<sc_logic> clk, rst, en_reg1, en_reg2;
7       sc_in<sc_lv<T>> din;
8       sc_out<sc_lv<T>> dout;
9
10      sc_lv<T> all_0;
11      sc_signal<sc_lv<T>>  outreg, d_reg1, d_reg2;
12
13      SC_CTOR(MAC_Add) {
14          for (int i = 0; i <T; i++)
15              all_0[i] = sc_logic(0);
16          SC_METHOD(seq);
17          sensitive << clk;
18          SC_METHOD(process_others);
19          sensitive << din << d_reg2 << outreg;
20      }
21      void seq();
22      void process_others();
23  };
```

MAC.h | MAC_Controller.h | MAC_Add.h

(Global Scope)

```
25  template <int T> void MAC_Add<T>::seq() {
26      if ((clk->event()) && (clk == '1')) {
27          if (rst == '1')
28              d_reg2 = all_0;
29          else if(en_reg1 == '1')
30              d_reg2 = d_reg1;
31          else if (en_reg2 == '1'){
32              outreg = d_reg2;
33              d_reg2 = all_0;
34          }
35          else{
36              d_reg2 = d_reg2;
37              outreg = outreg;
38          }
39      }
40  }
```



MAC.h | MAC_Controller.h | MAC_Add.h

(Global Scope)

```
42  template <int T> void MAC_Add<T>::process_others(){
43      if (din.read().range(0, 0) == "Z" || din.read().range(0, 0) == "X") d_reg1 = din;
44      else if (d_reg2.read().range(0, 0) == "Z" || d_reg2.read().range(0, 0) == "X") d_reg1 = d_reg2;
45      else d_reg1 = din.read().to_uint() + d_reg2.read().to_uint();
46      dout = outreg;
47  }
```

Multiply and Accumulate (MAC)

- Multiply and Accumulate (MAC): MAC_Controller

MAC_Controller.h

MAC.h | MAC_Controller.h

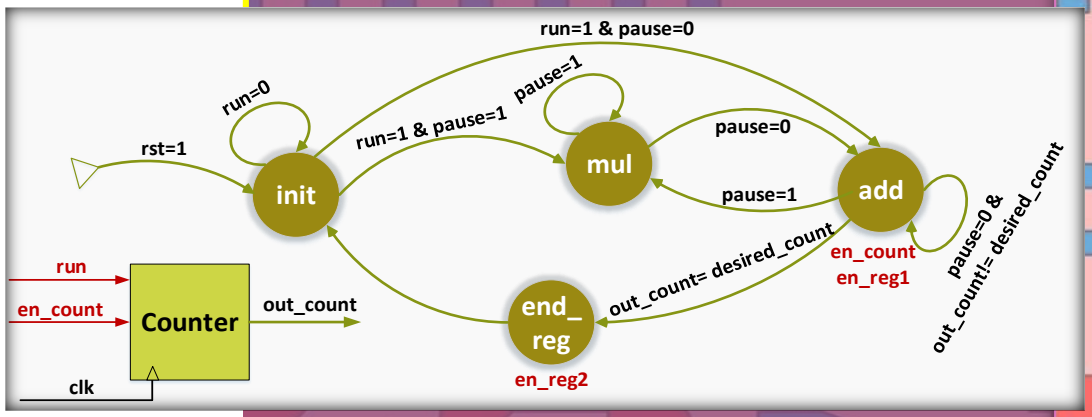
MAC_Controller

```
1   #pragma once
2   #include <systemc.h>
3   using namespace std;
4
5   SC_MODULE(MAC_Controller) {
6       sc_in<sc_logic> clk, rst, run, pause;
7       sc_in<sc_lv<4>> count;
8       sc_out<sc_logic> en_reg1, en_reg2;
9
10      enum mini_state { init, mull, add, end_reg };
11      sc_signal<mini_state> ps, ns;
12      sc_signal<sc_logic> en_count;
13      sc_signal<sc_lv<4>> out_count, desired_count;
14      SC_CTOR(MAC_Controller) {
15          SC_METHOD(controller_combS);
16          sensitive << ps<< run<< pause<< out_count;
17          SC_METHOD(controller_combO);
18          sensitive << ps;
19          SC_METHOD(controller_seq);
20          sensitive << clk;
21          SC_METHOD(counting);
22          sensitive << clk;
23          //trace();
24      }
25      void controller_seq();
26      void controller_combS();
27      void controller_combO();
28      void counting();
29      void trace();
30  };
```

MAC.h | MAC_Controller.h | MAC_Add.h

MAC_Controller

```
45  void MAC_Controller::controller_combS() {
46      switch (ps.read()){
47      case init :
48          if (run == '1'){
49              desired_count = count;
50              if (pause == '0') ns = add;
51              else ns = mull;
52          }
53          else ns = init;
54          break;
55      case mull :
56          if(pause == '1') ns = mull;
57          else ns = add;
58          break;
59      case add :
60          if(pause == '1') ns = mull;
61          else if(out_count.read() == desired_count.read())
62              ns = end_reg;
63          else ns = add;
64          break;
65      case end_reg :
66          ns = init;
67          break;
68      default:
69          ns = init;
70          break;
71      }
72  }
```



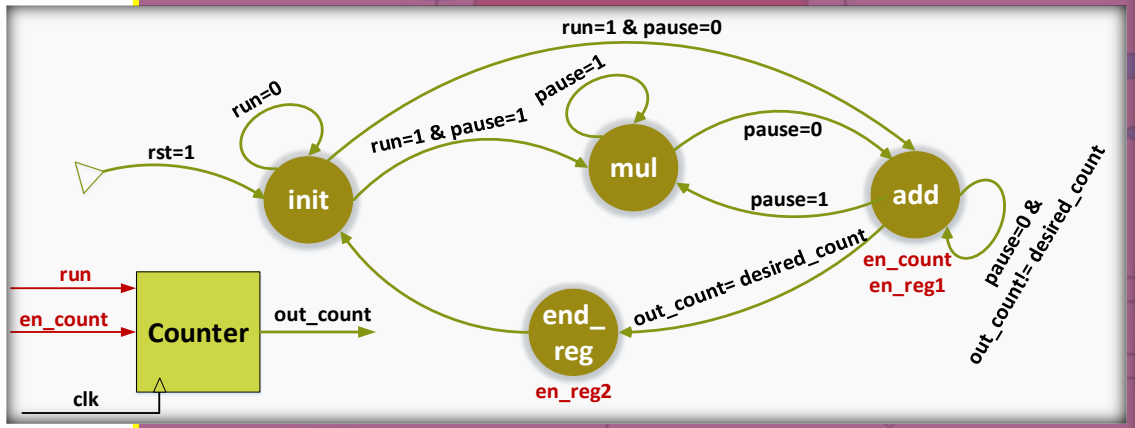
Multiply and Accumulate (MAC)

- Multiply and Accumulate (MAC): MAC_Controller (cont.)

MAC_Controller.h

```
MAC.h | MAC_Controller.h
MAC_Controller
 1  #pragma once
 2  #include <systemc.h>
 3  using namespace std;
 4
 5  SC_MODULE(MAC_Controller) {
 6      sc_in<sc_logic> clk, rst, run, pause;
 7      sc_in<sc_lv<4>> count;
 8      sc_out<sc_logic> en_reg1, en_reg2;
 9
10      enum mini_state { init, mull, add, end_reg };
11      sc_signal<mini_state> ps, ns;
12      sc_signal<sc_logic> en_count;
13      sc_signal<sc_lv<4>> out_count, desired_count;
14      SC_CTOR(MAC_Controller) {
15          SC_METHOD(controller_comb5);
16          sensitive << ps<< run<< pause<< out_count;
17          SC_METHOD(controller_comb0);
18          sensitive << ps;
19          SC_METHOD(controller_seq);
20          sensitive << clk;
21          SC_METHOD(counting);
22          sensitive << clk;
23          //trace();
24      }
25      void controller_seq();
26      void controller_comb5();
27      void controller_comb0();
28      void counting();
29      void trace();
30  };
```

```
MAC.h | MAC_Controller.h | MAC_Add.h
MAC_Controller
74  void MAC_Controller::controller_comb0() {
75      en_reg1 = SC_LOGIC_0;
76      en_reg2 = SC_LOGIC_0;
77      en_count = SC_LOGIC_0;
78      switch (ps.read()){
79      case add:
80          en_count = SC_LOGIC_1;
81          en_reg1 = SC_LOGIC_1;
82          break;
83      case end_reg:
84          en_reg2 = SC_LOGIC_1;
85          break;
86      default:
87          en_count = SC_LOGIC_0;
88          en_reg1 = SC_LOGIC_0;
89          en_reg2 = SC_LOGIC_0;
90          break;
91      }
92  }
```



Multiply and Accumulate (MAC)

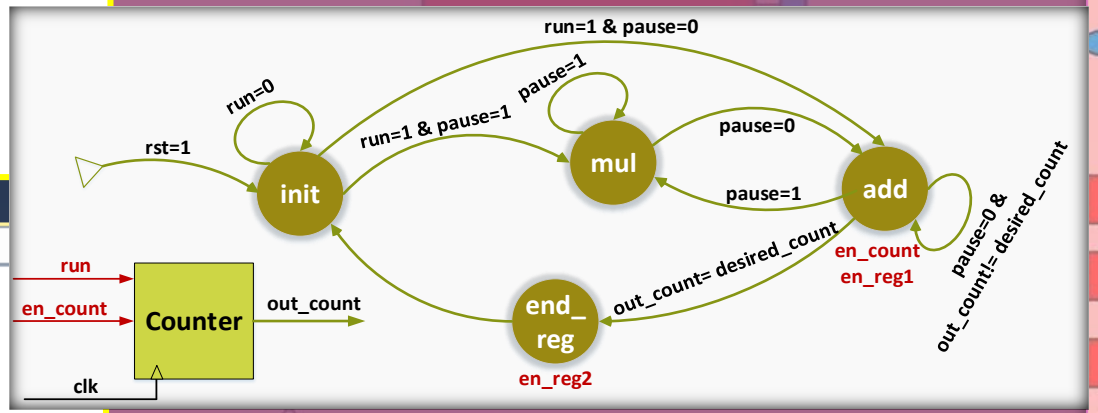
- Multiply and Accumulate (MAC): MAC_Controller (cont.)

MAC_Controller.h

```
MAC.h | MAC_Controller.h
MAC_Controller
 1  #pragma once
 2  #include <systemc.h>
 3  using namespace std;
 4
 5  SC_MODULE(MAC_Controller) {
 6      sc_in<sc_logic> clk, rst, run, pause;
 7      sc_in<sc_lv<4>> count;
 8      sc_out<sc_logic> en_reg1, en_reg2;
 9
10      enum mini_state { init, mull, add, end_reg };
11      sc_signal<mini_state> ps, ns;
12      sc_signal<sc_logic> en_count;
13      sc_signal<sc_lv<4>> out_count, desired_count;
14      SC_CTOR(MAC_Controller) {
15          SC_METHOD(controller_combS);
16          sensitive << ps<< run<< pause<< out_count;
17          SC_METHOD(controller_combO);
18          sensitive << ps;
19          SC_METHOD(controller_seq);
20          sensitive << clk;
21          SC_METHOD(counting);
22          sensitive << clk;
23          //trace();
24      }
25      void controller_seq();
26      void controller_combS();
27      void controller_combO();
28      void counting();
29      void trace();
30  };
```

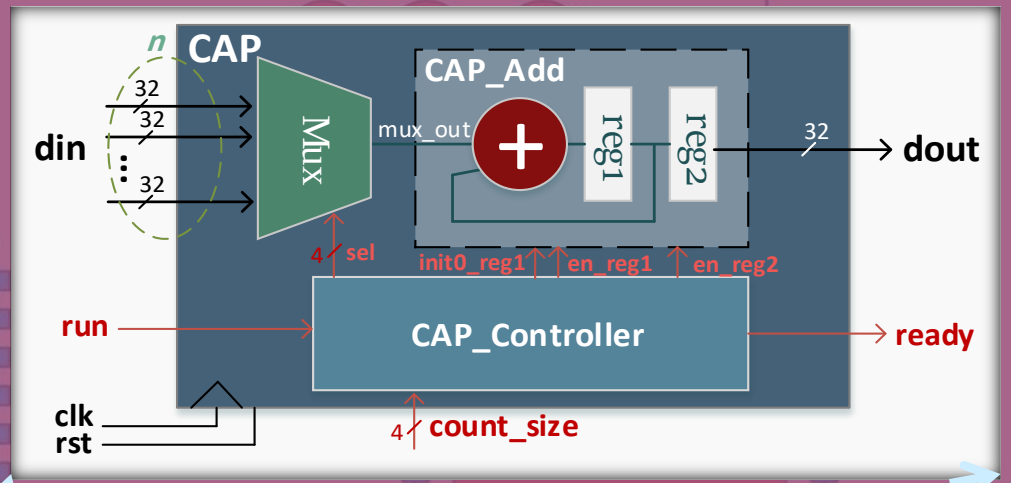
```
MAC.h | MAC_Controller.h | MAC_Add.h
MAC_Controller
94  void MAC_Controller::controller_seq(){
95      if (clk->event() && clk == SC_LOGIC_1){
96          if (rst == SC_LOGIC_1)
97              ps = init;
98          else
99              ps = ns;
100     }
101 }
```

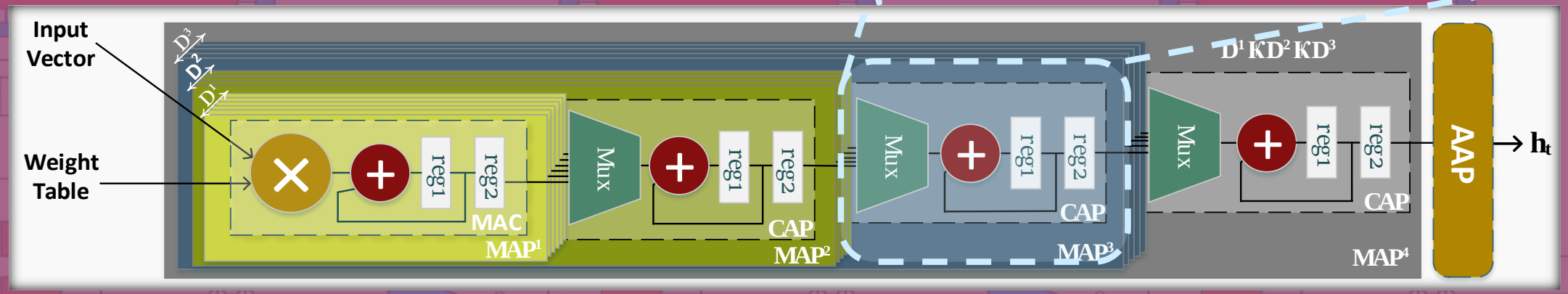
```
MAC.h | MAC_Controller.h | MAC_Add.h
MAC_Controller
103 void MAC_Controller::counting(){
104     if (clk->event() && clk == SC_LOGIC_1){
105         if (run == SC_LOGIC_1)
106             out_count = "0001";
107         else if (en_count == '1')
108             out_count = (out_count.read().to_uint()) + 1;
109     }
110 }
```



SystemC Machine Learning Automotive Application

- Introduction
+ DiBA, an RTL Hardware Architecture for LSTM (Top-Down)
– SystemC Implementation of DiBA (Bottom-Up)
  - Multiply and Accumulate (MAC)
  - Convergence Adder Plate (CAP)
  - Matrix-multiplication and Accumulation Plate (MAP)
  - Activation and Accumulation Plate (AAP)
  - n-Dimensional Bitslice Architecture (DiBA)
  - Data Arrangement (Testbench)





Convergence Adder Plate (CAP)

CAP.h

CAP.h | CAP_Controller.h | CAP_Add.h | CAP_Mux.h

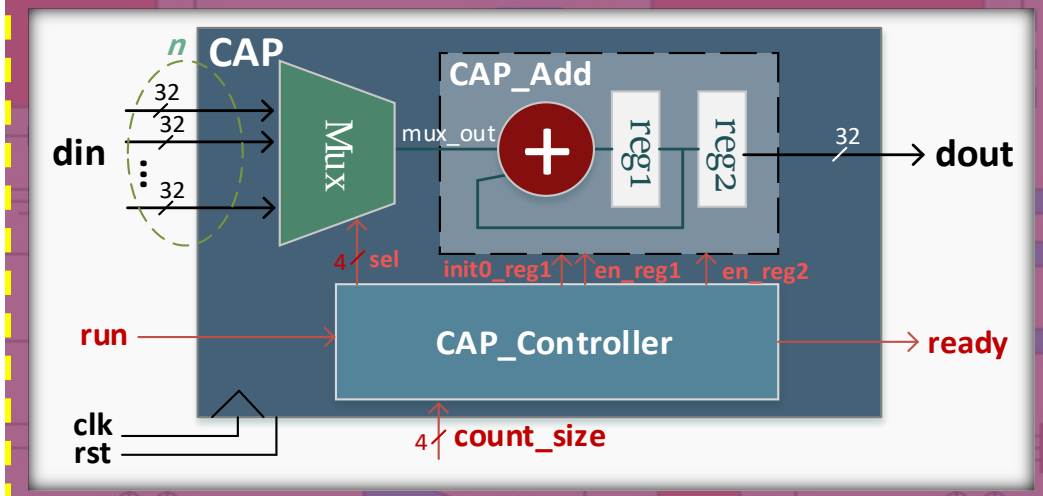
CAP<n>

```
1   #pragma once
2   #include <systemc.h>
3   #include "CAP_Controller.h"
4   #include "CAP_Add.h"
5   #include "CAP_Mux.h"
6   template <int n> SC_MODULE(CAP) {
7       sc_in<sc_logic> clk, rst, run;
8       sc_out<sc_logic> ready;
9       sc_in<sc_lv<32*n>> din;
10      sc_out<sc_lv<32>> dout;
11      sc_signal<sc_lv<32>> mux_out;
12      sc_signal<sc_logic> en_reg1, init0_reg1, en_reg2;
13      sc_signal<sc_lv<4>> sel, count_size;
14      sc_signal<sc_lv<32*16>> mux_in;
15      sc_lv<512 - 32 * n> all_0;
16      CAP_Add<32>* Adder;
17      CAP_Mux<512,32,4>* mux;
18      CAP_Controller* CAP_Cntrlr;
19      SC_CTOR(CAP) {
20          for (int i = 0; i < 512 - 32 * n; i++) all_0[i] = sc_logic(0);
21          int count_size_int = static_cast<int>(n);
22          count_size.write(count_size_int);
23          CAP_Cntrlr = new CAP_Controller("CAP_Controller_module");
24          (*CAP_Cntrlr)(clk, rst, run, count_size, sel, init0_reg1, en_reg1, en_reg2);
25          Adder = new CAP_Add<32>("add_module");
26          (*Adder)(clk, init0_reg1, en_reg1, en_reg2, mux_out, dout);
27          mux = new CAP_Mux<512, 32, 4>("Mux_cascading_module");
28          (*mux)(sel, mux_in, mux_out);
29          SC_METHOD(process_din);
30          sensitive << din;
31          SC_METHOD(process_others);
32          sensitive << en_reg2;
33          //trace();
34      }
35      void process_others();
36      void process_din();
37      void trace();
38  };
```

CAP.h | CAP_Controller.h | CAP_Add.h | CAP_Mux.h

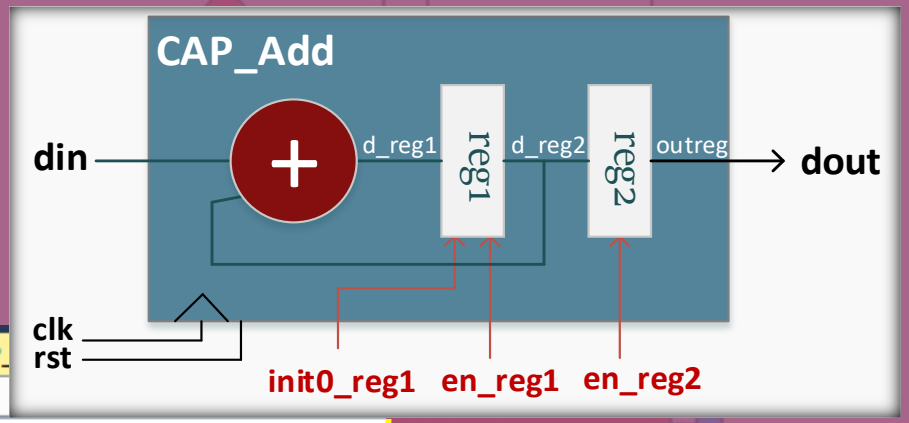
CAP<n>

```
73  template <int n> void CAP<n>::process_others(){
74      ready = en_reg2;
75  }
76
77  template <int n> void CAP<n>::process_din(){
78      if (n < 16){
79          mux_in.write((all_0, din.read().range(32 * n - 1, 0)));
80      }
81      else
82          mux_in = din.read().range(511, 0);
83  }
```



Convergence Adder Plate (CAP)

- Convergence Adder Plate (CAP): CAP_Add



CAP_Add.h

```
CAP.h  CAP_Controller.h  CAP_Add.h
CAP_Add<T>
 1  #pragma once
 2  #include <systemc.h>
 3  using namespace std;
 4
 5  template <int T> SC_MODULE(CAP_Add) {
 6      sc_in<sc_logic> clk, rst, init0_reg1, en_reg1, en_reg2;
 7      sc_in<sc_lv<T>> din;
 8      sc_out<sc_lv<T>> dout;
 9
10      sc_lv<T> all_0;
11      sc_signal<sc_lv<T>>  d_reg1, d_reg2, outreg;
12
13      SC_CTOR(CAP_Add) {
14          for (int i = 0; i <T; i++)
15              all_0[i] = sc_logic(0);
16          SC_METHOD(seq);
17          sensitive << clk;
18          SC_METHOD(process_others);
19          sensitive << din << d_reg2 << outreg;
20      }
21      void seq();
22      void process_others();
23  };
```

```
CAP.h  CAP_Controller.h  CAP_
CAP_Add<T>
25  template <int T> void CAP_Add<T>::seq() {
26      if ((clk->event()) && (clk == '1')) {
27          if (rst == '1')
28              d_reg2 = all_0;
29          else if (en_reg1 == '1') d_reg2 = d_reg1;
30          else if (init0_reg1 == '1') d_reg2 = all_0;
31          else if (en_reg2 == '1') d_reg2 = all_0;
32          if (en_reg2 == '1') outreg = d_reg2;
33      }
34  }
```

```
CAP.h  CAP_Controller.h  CAP_Add.h  CAP_Mux.h
(Global Scope)
34  template <int T> void CAP_Add<T>::process_others(){
35      if (din.read().range(0, 0) == "Z" || din.read().range(0, 0) == "X") d_reg1 = din;
36      else if (d_reg2.read().range(0, 0) == "Z" || d_reg2.read().range(0, 0) == "X") d_reg1 = d_reg2;
37      else d_reg1 = din.read().to_uint() + d_reg2.read().to_uint();
38      dout = outreg;
39  }
```

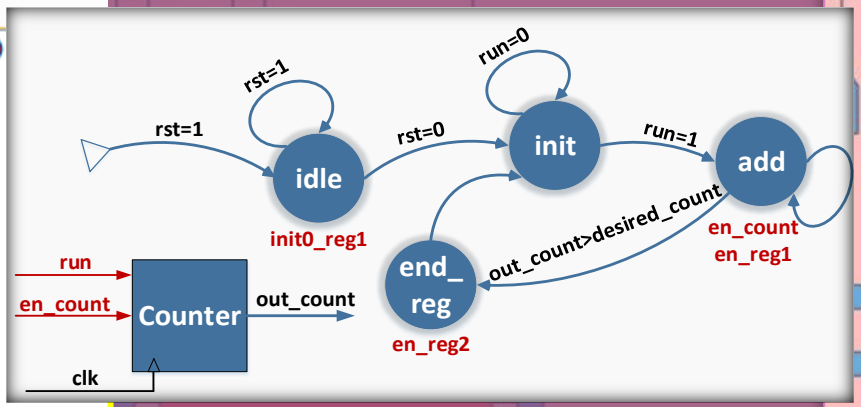
Convergence Adder Plate (CAP)

- Convergence Adder Plate (CAP): CAP_Controller

CAP_Controller.h

```
34  SC_MODULE(CAP_Controller) {
35      sc_in<sc_logic> clk, rst, run;
36      sc_in<sc_lv<4>> count;
37      sc_out<sc_lv<4>> sel;
38      sc_out<sc_logic> init0_reg1, en_reg1, en_reg2;
39
40      sc_signal<sc_lv<4>> out_count, desired_count;
41      enum cascade_state { idel, init, add, end_reg };
42      sc_signal<cascade_state> ps, ns;
43      sc_signal<sc_logic> en_count;
44      CAP_Counter<4>* count_i;
45
46      SC_CTOR(CAP_Controller) {
47          count_i = new CAP_Counter<4>("CAP_Counter_module");
48          (*count_i)(clk, run, en_count, out_count);
49          SC_METHOD(controller_combS);
50          sensitive << ps<< rst<< run<< out_count<< desired_count;
51          SC_METHOD(controller_combO);
52          sensitive << ps;
53          SC_METHOD(controller_seq);
54          sensitive << clk;
55          SC_METHOD(process_others);
56          sensitive << out_count << ps << ns << count;
57      }
58      void controller_seq();
59      void controller_combS();
60      void controller_combO();
61      void process_others();
62  };
```

```
64  void CAP_Controller::controller_combS()
65      ns = idel;
66      switch (ps.read()){
67      case idel:
68          if(rst == '0') ns = init;
69          else ns = idel;
70          break;
71      case init:
72          if(run == '1') ns = add;
73          else ns = init;
74          break;
75      case add:
76          if (out_count.read().to_uint() >
77              desired_count.read().to_uint())
78              ns = end_reg;
79          else ns = add;
80          break;
81      case end_reg:
82          ns = init;
83          break;
84      default:
85          ns = idel;
86          break;
87      }
88  }
```



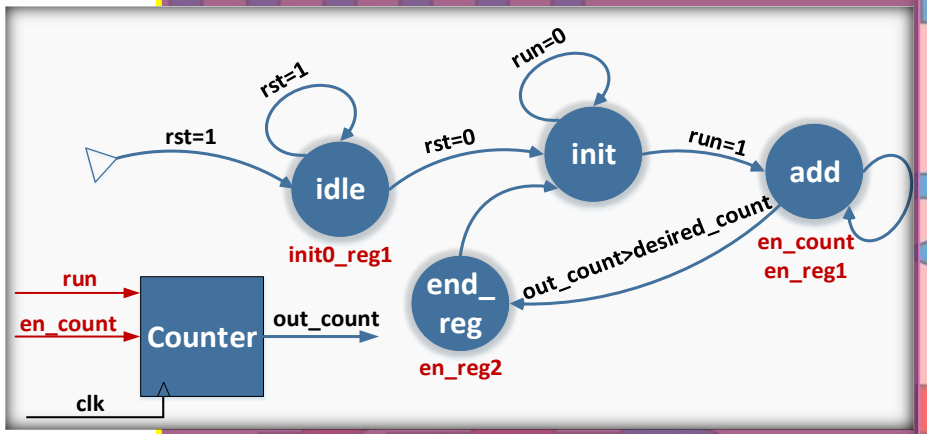
Convergence Adder Plate (CAP)

- Convergence Adder Plate (CAP): CAP_Controller

CAP_Controller.h

```
CAP.h | CAP_Controller.h | CAP_Add.h
CAP_Controller
34  SC_MODULE(CAP_Controller) {
35      sc_in<sc_logic> clk, rst, run;
36      sc_in<sc_lv<4>> count;
37      sc_out<sc_lv<4>> sel;
38      sc_out<sc_logic> init0_reg1, en_reg1, en_reg2;
39
40      sc_signal<sc_lv<4>> out_count, desired_count;
41      enum cascade_state { idel, init, add, end_reg };
42      sc_signal<cascade_state> ps, ns;
43      sc_signal<sc_logic> en_count;
44      CAP_Counter<4>* count_i;
45
46      SC_CTOR(CAP_Controller) {
47          count_i = new CAP_Counter<4>("CAP_Counter_module");
48          (*count_i)(clk, run, en_count, out_count);
49          SC_METHOD(controller_comb5);
50          sensitive << ps<< rst<< run<< out_count<< desired_count;
51          SC_METHOD(controller_comb0);
52          sensitive << ps;
53          SC_METHOD(controller_seq);
54          sensitive << clk;
55          SC_METHOD(process_others);
56          sensitive << out_count << ps << ns << count;
57      }
58      void controller_seq();
59      void controller_comb5();
60      void controller_comb0();
61      void process_others();
62  };
```

```
AP.h | CAP_Controller.h | CAP_Add.h | CAP_Mux
CAP_Controller
90  void CAP_Controller::controller_comb0() {
91      en_count = SC_LOGIC_0;
92      en_reg1 = SC_LOGIC_0;
93      en_reg2 = SC_LOGIC_0;
94      init0_reg1 = SC_LOGIC_0;
95      switch (ps.read()){
96      case init:
97          break;
98      case idel:
99          init0_reg1 = SC_LOGIC_1;
100         break;
101     case add:
102         en_reg1 = SC_LOGIC_1;
103         en_count = SC_LOGIC_1;
104         break;
105     case end_reg:
106         en_reg2 = SC_LOGIC_1;
107         break;
108     default:
109         en_count = SC_LOGIC_0;
110         en_reg1 = SC_LOGIC_0;
111         en_reg2 = SC_LOGIC_0;
112         init0_reg1 = SC_LOGIC_0;
113         break;
114     }
115 }
```



Convergence Adder Plate (CAP)

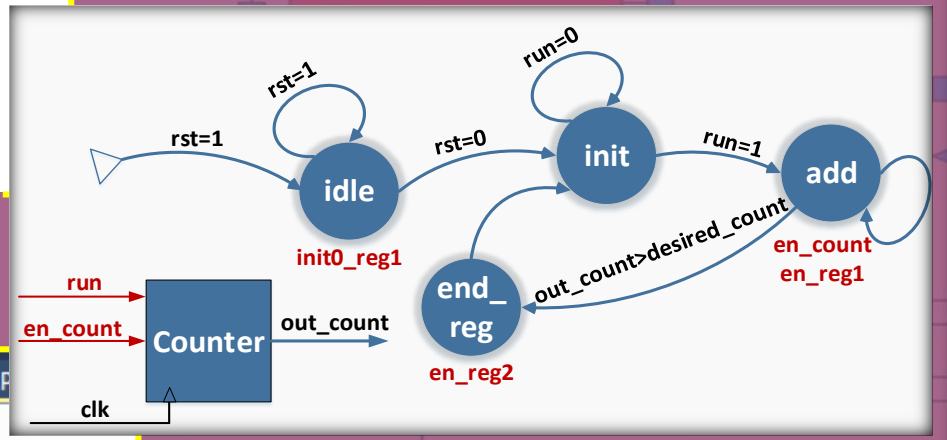
- Convergence Adder Plate (CAP): CAP_Controller (cont.)

CAP_Controller.h

```
CAP.h | CAP_Controller.h | CAP_Add.h
CAP_Controller
34  SC_MODULE(CAP_Controller) {
35      sc_in<sc_logic> clk, rst, run;
36      sc_in<sc_lv<4>> count;
37      sc_out<sc_lv<4>> sel;
38      sc_out<sc_logic> init0_reg1, en_reg1, en_reg2;
39
40      sc_signal<sc_lv<4>> out_count, desired_count;
41      enum cascade_state { idel, init, add, end_reg };
42      sc_signal<cascade_state> ps, ns;
43      sc_signal<sc_logic> en_count;
44      CAP_Counter<4>* count_i;
45
46      SC_CTOR(CAP_Controller) {
47          count_i = new CAP_Counter<4>("CAP_Counter_module");
48          (*count_i)(clk, run, en_count, out_count);
49          SC_METHOD(controller_combS);
50          sensitive << ps<< rst<< run<< out_count<< desired_count;
51          SC_METHOD(controller_combO);
52          sensitive << ps;
53          SC_METHOD(controller_seq);
54          sensitive << clk;
55          SC_METHOD(process_others);
56          sensitive << out_count << ps << ns << count;
57      }
58      void controller_seq();
59      void controller_combS();
60      void controller_combO();
61      void process_others();
62  };
```

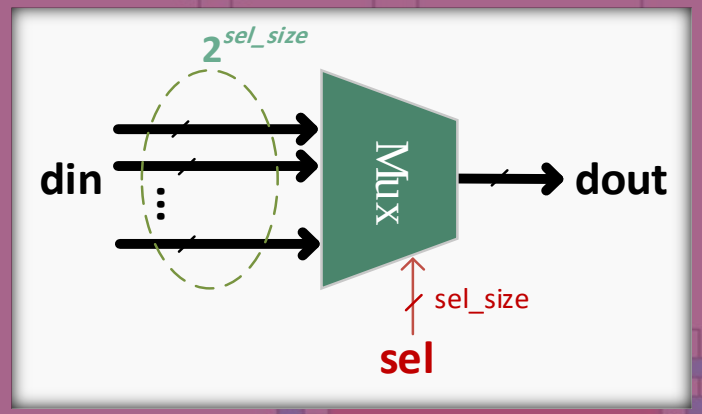
```
CAP.h | CAP_Controller.h | CAP_Add.h | CAP_Mux.h
CAP_Controller
117 void CAP_Controller::controller_seq(){
118     if (clk->event() && clk == SC_LOGIC_1){
119         if (rst == SC_LOGIC_1)
120             ps = idel;
121         else
122             ps = ns;
123     }
124 }
```

```
CAP.h | CAP_Controller.h | CAP_Add.h | CAP
CAP_Controller
126 void CAP_Controller::process_others(){
127     sel = out_count;
128     desired_count = count.read().to_uint() - 2;
129 }
```



Convergence Adder Plate (CAP)

- Convergence Adder Plate (CAP): CAP_Mux



CAP_Mux.h

```
CAP.h | CAP_Mux.h
CAP_Mux<in_size, out_size, sel_size>
 1  #pragma once
 2  #include <systemc.h>
 3  using namespace std;
 4
 5  template <int in_size, int out_size, int sel_size>
 6  SC_MODULE(CAP_Mux){
 7      sc_in<sc_lv<sel_size>> sel;
 8      sc_in<sc_lv<in_size>> din;
 9      sc_out<sc_lv<out_size>> dout;
10
11      sc_lv<out_size> all_0;
12      SC_CTOR(CAP_Mux) {
13          for (int i = 0; i <out_size; i++)
14              all_0[i] = sc_logic(0);
15          SC_METHOD(muxing);
16          sensitive << din << sel;
17      }
18      void muxing();
19  };
```

```
CAP.h | CAP_Mux.h
(Global Scope)                                   SC_CTOR(C
21  template <int in_size, int out_size, int sel_size>
22  void CAP_Mux<in_size, out_size, sel_size>::muxing() {
23      bool flag_else = 0;
24      for (int i = 0; i < sel_size; i++){
25          if (sel->read().range(i, i)== "Z") flag_else = 1;
26          if (sel->read().range(i, i)== "X") flag_else = 1;
27      }
28
29      if (flag_else) dout = all_0;
30      else{
31          unsigned int sel_us = sel->read().to_uint();
32          dout = din->read().range((sel_us + 1)*out_size - 1, sel_us*out_size);
33      }
34  }
```

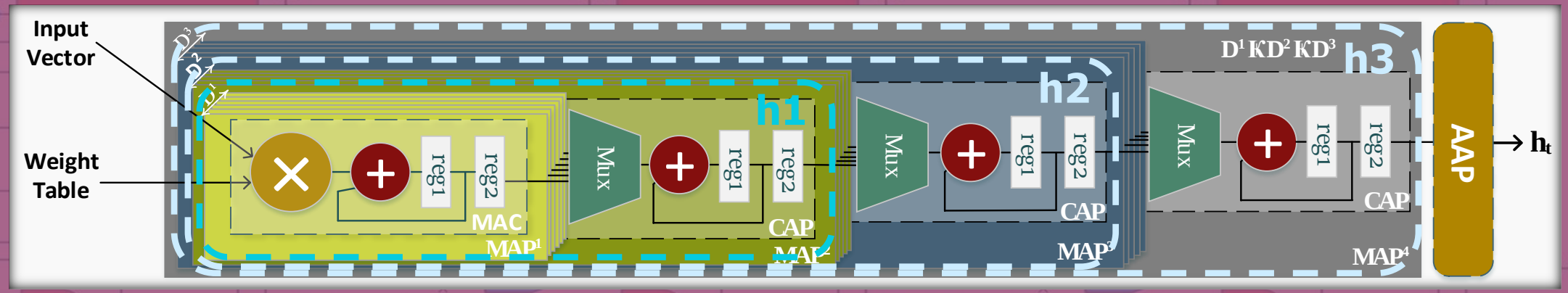
SystemC Machine Learning Automotive Application

Introduction

\+ DiBA, an RTL Hardware Architecture for LSTM (Top-Down)

– SystemC Implementation of DiBA (Bottom-Up)

- Multiply and Accumulate (MAC)
- Convergence Adder Plate (CAP)
- Matrix-multiplication and Accumulation Plate (MAP)
- Activation and Accumulation Plate (AAP)
- n-Dimensional Bitslice Architecture (DiBA)
- Data Arrangement (Testbench)



C++ Recursion

C++ Review

- The process in which a function calls itself directly or indirectly is called recursion and the corresponding function is called as recursive function
- In the recursive program, the solution to the base case is provided and the solution of the bigger problem is expressed in terms of smaller problems
- To prevent infinite recursion, if...else statement can be used where one branch makes the recursive call and the other doesn't
- An example:

```
int factorial(int n) {
    if (n > 1) {
        return n * factorial(n - 1);
    } else {
        return 1;
    }
}
```

If...else statement

Base case

The recursion continues until some condition is met

```
void recurse() {
    ... .. ...
    recurse();
    ... .. ...
}

int main() {
    ... .. ...
    recurse();
    ... .. ...
}
```

recursive call

function call

C++ Template Recursion

C++ Review

- One more template definition is needed as base case to terminate the recursion achieved by template specialization
- This declares a template with the same name but is specialized to the integer value of 1 for the given template parameter
- An example:

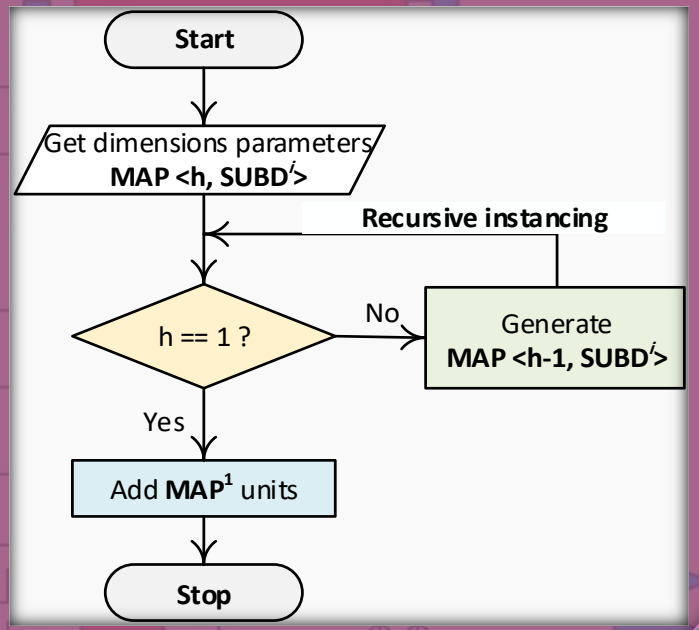
```
template <unsigned int n>
class Factorial {
    enum { Value = n * Factorial<n-1>::Value };
};

template <>
class Factorial<1> {
    enum { Value = 1 };
};
```

Template specialization

One more template definition as base case

Recursive Model for MAPs

- Recursive Modeling for the DiBA architecture
  - The hierarchical DiBA hardware for each involutive dimension presents a symbolic parameter of MAP $<h, SUBD^{n-1}, SUBD^{n-2}, \ldots SUBD^{1}, k>$
  - h shows the current dimension, $SUBD^{i}$ shows the number of inner MAPs for the *ith* sub-dimension from top to the bottom, and k indicates the number of clock cycles in MAC structures
- The architecture body of this module includes two sc_module with the same name for recursive modeling. One is for the base case and the other one is for the recursive MAPs. The recursion exit condition is h==1



Recursive Model for MAPs

- Matrix-multiplication and Accumulation Plate (MAP), h>1

MAP.h

```
1   #pragma once
2   #include <systemc.h>
3   #include "MAC.h"
4   #include "CAP.h"
5   using namespace std;
6
7   template <int h, int a, int  b, int c, int d, int e, int f, int k>
8   class MAP : public sc_module{
9   public:
10      sc_in<sc_logic> clk, rst, run, pause;
11      sc_out<sc_logic>ready;
12      sc_in<sc_lv<16 * a*b*c*d*e*f>> din;
13      sc_in<sc_lv<16 * a*b*c*d*e*f>> w;
14      sc_out< sc_lv<32>> dout;
15
16      sc_signal <sc_lv<32*a>> outMAP_cur;
17      sc_signal<sc_lv<32>> outMAP_pre[a];
18      sc_signal<sc_lv<16 * b*c*d*e*f>> dinMAP_pre[a], wMAP_pre[a];
19      sc_signal<sc_logic> runCAP_pre, runCAP_cur;
20      sc_signal<sc_logic> readyMAP_pre[a], readyCAP_cur;
21      string module_name;
22      char* name;
23
24      CAP<a>* cap;
25      MAP<h-1, b, c, d, e, f, 1, k>* map[a];
```

```
26  public:
27      SC_CTOR(MAP){
28          cout << "creating MAP id:"<< id << " h:" << h << " a:" << a << " b:" << b
29          << " c:" << c << " d:" << d << " e:" << e << " f:" << f << " k:" << k << endl;
30          id++;
31          readyCAP_cur = sc_logic(0);
32          runCAP_pre = sc_logic(0);
33          for (int i = 1; i <= a; i++){
34              module_name = "MAP_" + std::to_string(i);
35              name = &module_name[0];
36              map[i - 1] = new MAP<h-1, b, c, d, e, f, 1, k>(name);
37              map[i - 1]->clk(clk);
38              map[i - 1]->rst(rst);
39              map[i - 1]->run(runCAP_pre);
40              map[i - 1]->pause(pause);
41              map[i - 1]->ready(readyMAP_pre[i-1]);
42              map[i - 1]->din(dinMAP_pre[i - 1]);
43              map[i - 1]->w(wMAP_pre[i - 1]);
44              map[i - 1]->dout(outMAP_pre[i - 1]);
45          }
46          cap = new CAP<a>("CAP");
47          (*cap)(clk, rst, runCAP_cur, readyCAP_cur, outMAP_cur, dout);
48
49          SC_METHOD(process_others);
50          sensitive << run << readyCAP_cur << readyMAP_pre[0];
51          SC_METHOD(process_in);
52          sensitive << din << w ;
53          SC_METHOD(process_out);
54          sensitive << outMAP_pre[0];
55  //      trace();
56      }
57      void process_others();
58      void process_in();
59      void process_out();
60      void trace();
```

Recursive Model for MAPs

- Matrix-multiplication and Accumulation Plate (MAP), h>1 (cont.)

MAP.h

```
MAP.h | CAP.h | MAC.h
MAP<h, a, b, c, d, e, f, k>
78   template <int h, int a, int  b, int c, int d, int e, int f, int k>
79   void MAP<h, a, b, c, d, e, f, k>::process_out(){
80       if (a > 1){
81           sc_lv<32 * (a - 1)> all_0;
82           sc_lv<32 * a> temp, tempout;
83           for (int j = 0; j < 32 * (a - 1); j++) all_0[j] = sc_logic(0);
84           for (int j = 0; j < 32 * a; j++) tempout[j] = sc_logic(0);
85           for (int i = 0; i < a; i++){
86               temp = (all_0, outMAP_pre[a - i-1].read());
87               tempout = tempout << 32;
88               tempout = tempout | temp;
89               outMAP_cur = tempout;
90           }
91       }
92       else
93           outMAP_cur = outMAP_pre[0].read();
94   }
```

```
MAP.h | CAP.h | MAC.h
MAP<h, a, b, c, d, e, f, k>
70   template <int h, int a, int  b, int c, int d, int e, int f, int k>
71   void MAP<h, a, b, c, d, e, f, k>::process_in(){
72       for (int i = 1; i <= a; i++){
73           dinMAP_pre[i - 1] = din.read().range(16 * b*c*d*e*f*(i)-1, 16 * b*c*d*e*f*(i - 1));
74           wMAP_pre[i - 1] = w.read().range(16 * b*c*d*e*f*(i)-1, 16 * b*c*d*e*f*(i - 1));
75       }
76   }
```

```
MAP.h | CAP.h | MAC.h
MAP<h, a, b, c, d, e, f, k>
63   template <int h, int a, int  b, int c, int d, int e, int f, int k>
64   void MAP<h, a, b, c, d, e, f, k>::process_others(){
65       runCAP_cur = readyMAP_pre[0];
66       runCAP_pre = run;
67       ready = readyCAP_cur;
68   }
```

Recursive Model for MAPs

- Matrix-multiplication and Accumulation Plate (MAP), h=1

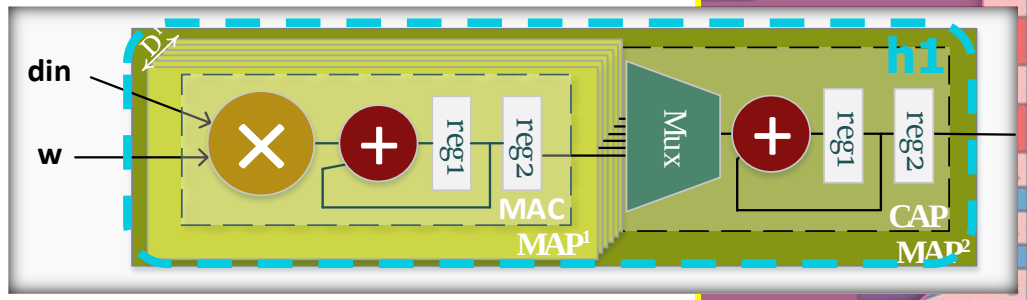
MAP.h

MAP.h | CAP.h | MAC.h

MAP<h, a, b, c, d, e, f, k>

```
131  template < int a, int  b, int c, int d, int e, int f, int k>
132  class MAP<1, a, b, c, d, e, f, k> : public sc_module{
133  public:
134      sc_in<sc_logic> clk, rst, run, pause;
135      sc_out<sc_logic>ready;
136      sc_in<sc_lv<16 * a*b*c*d*e*f>> din;
137      sc_in<sc_lv<16 * a*b*c*d*e*f>> w;
138      sc_out< sc_lv<32>> dout;
139
140      sc_signal <sc_lv<32 * a>> outMAP_cur;
141      sc_signal<sc_lv<32>> outMAP_pre[a];
142      sc_signal<sc_lv<16>> dinMAP_pre[a], wMAP_pre[a];
143      sc_signal<sc_logic> runCAP_pre, runCAP_cur;
144      sc_signal<sc_logic> readyMAP_pre[a], readyCAP_cur;
145      string module_name;
146      char* name;
147      int h = 1;
148
149      MAC<k>* MAP1[a];
150      CAP<a>* cap;
```

```
151  public:
152      SC_CTOR(MAP){
153          cout << "creating MAP id:" << id << " h:" << h << " a:" << a << " b:" << b <<
154          " c:" << c << " d:" << d << " e:" << e << " f:" << f << " k:" << k <<endl;
155          id++;
156          readyCAP_cur = sc_logic(0);
157          runCAP_pre = sc_logic(0);
158          for (int i = 1; i <= a; i++){
159              module_name = "MAP1_" + std::to_string(i);
160              name = &module_name[0];
161              MAP1[i - 1] = new MAC<k>(name);
162              MAP1[i - 1]->clk(clk);
163              MAP1[i - 1]->rst(rst);
164              MAP1[i - 1]->run(run);
165              MAP1[i - 1]->pause(pause);
166              MAP1[i - 1]->ready(readyMAP_pre[i-1]);
167              MAP1[i - 1]->din(dinMAP_pre[i - 1]);
168              MAP1[i - 1]->w(wMAP_pre[i - 1]);
169              MAP1[i - 1]->dout(outMAP_pre[i - 1]);
170          }
171          cap = new CAP<a>("CAP");
172          (*cap)(clk, rst, runCAP_cur, readyCAP_cur, outMAP_cur, dout);
173
174          SC_METHOD(process_others);
175          sensitive << run << readyCAP_cur << readyMAP_pre[0];
176          SC_METHOD(process_in);
177          sensitive << din << w ;
178          SC_METHOD(process_out);
179          sensitive << runCAP_cur;
180  //      trace();
181      }
182      void process_others();
183      void process_in();
184      void process_out();
185      void trace();
186  };
```



Recursive Model for MAPs

- Matrix-multiplication and Accumulation Plate (MAP), h=1 (cont.)

MAP.h

MAP.h | CAP.h | MAC.h

MAP<h, a, b, c, d, e, f, k>

```
203   template <int a, int  b, int c, int d, int e, int f, int k>
204  void MAP<1, a, b, c, d, e, f, k>::process_out(){
205       if (a > 1){
206           sc_lv<32 * (a - 1)> all_0;
207           sc_lv<32 * a> temp, tempout;
208           for (int j = 0; j < 32 * (a - 1); j++) all_0[j] = sc_logic(0);
209           for (int j = 0; j < 32 * a; j++) tempout[j] = sc_logic(0);
210           for (int i = 0; i < a; i++){
211               temp = (all_0, outMAP_pre[a - i-1].read());
212               tempout = tempout << 32;
213               tempout = tempout | temp;
214               outMAP_cur = tempout;
215           }
216       }
217       else
218           outMAP_cur = outMAP_pre[0].read();
219
220   }
```

MAP.h | CAP.h | MAC.h

MAP<h, a, b, c, d, e, f, k>

```
195   template < int a, int  b, int c, int d, int e, int f, int k>
196  void MAP<1, a, b, c, d, e, f, k>::process_in(){
197       for (int i = 1; i <= a; i++){
198           dinMAP_pre[i - 1] = din.read().range((i * 16) - 1, (i - 1) * 16);
199           wMAP_pre[i - 1] = w.read().range((i * 16) - 1, (i - 1) * 16);
200       }
201   }
```

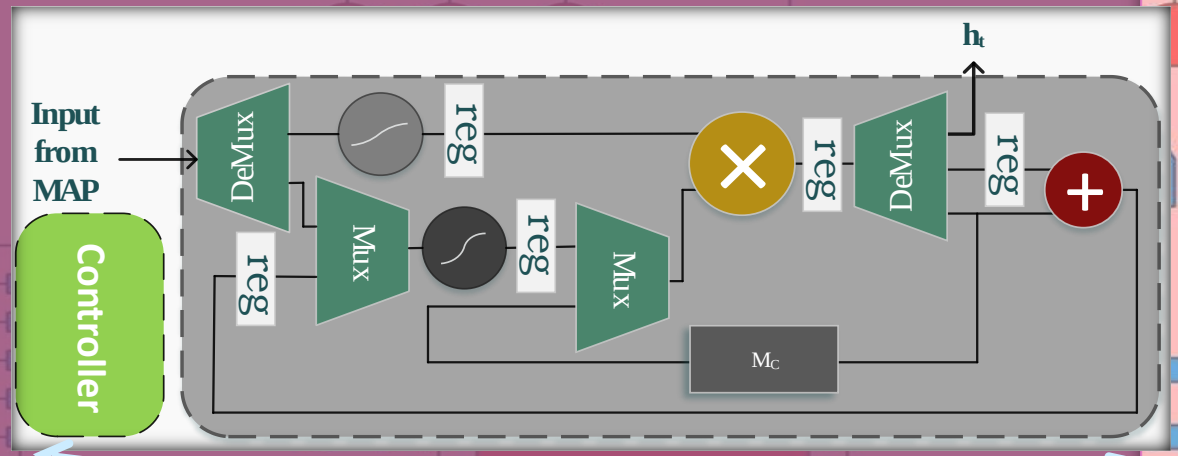
MAP.h | CAP.h | MAC.h

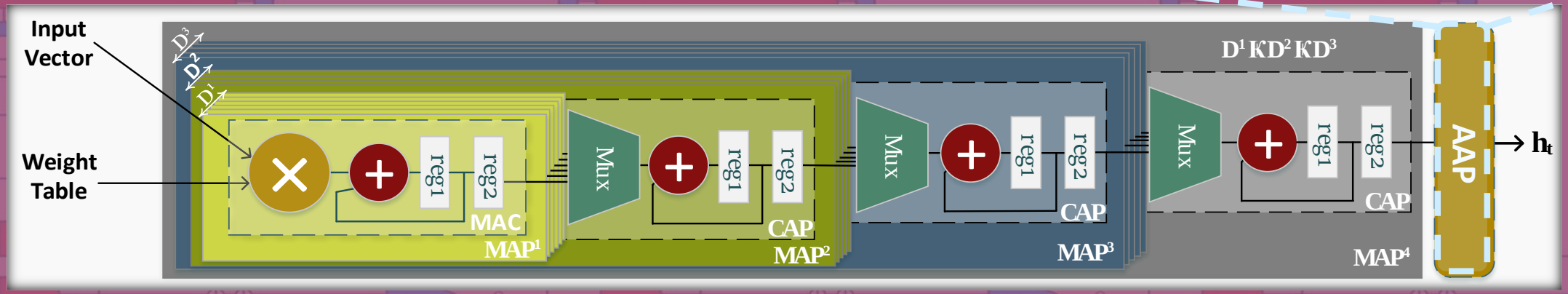
MAP<h, a, b, c, d, e, f, k>

```
188   template < int a, int  b, int c, int d, int e, int f, int k>
189  void MAP<1,a,b,c,d,e,f,k>::process_others(){
190       runCAP_cur = readyMAP_pre[0];
191       runCAP_pre = run;
192       ready = readyCAP_cur;
193   }
```

SystemC Machine Learning Automotive Application

- Introduction
+ DiBA, an RTL Hardware Architecture for LSTM (Top-Down)
– SystemC Implementation of DiBA (Bottom-Up)
  - Multiply and Accumulate (MAC)
  - Convergence Adder Plate (CAP)
  - Matrix-multiplication and Accumulation Plate (MAP)
  - Activation and Accumulation Plate (AAP)
  - n-Dimensional Bitslice Architecture (DiBA)
  - Data Arrangement (Testbench)



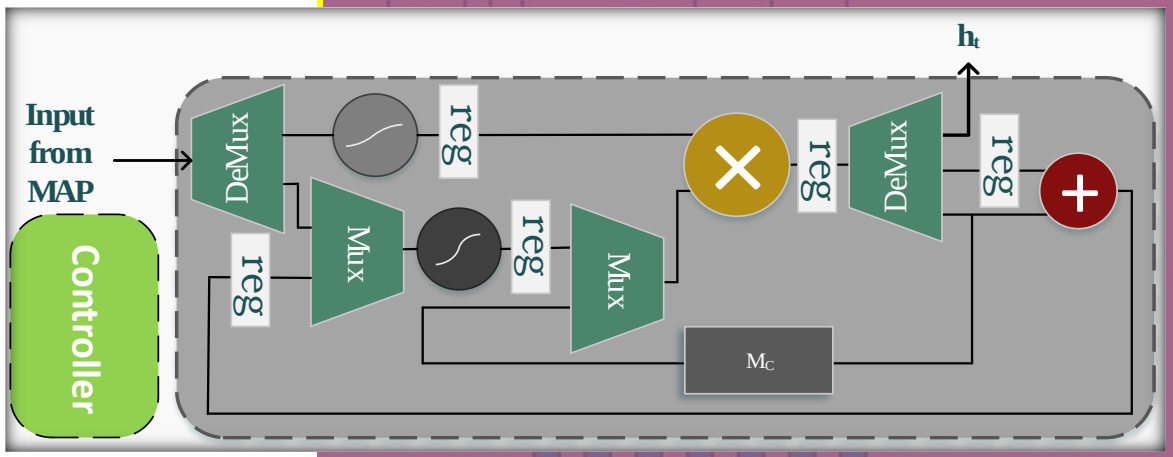


Activation and Accumulation Plate (AAP)

AAP.h

```
#pragma once
#include <systemc.h>
#include "AAP_Datapath.h"
#include "AAP_Controller.h"
using namespace std;

SC_MODULE(AAP) {
    sc_in<sc_logic> clk, rst, run;
    sc_in<sc_lv<32>> df;
    sc_out<sc_logic> ready;
    sc_out<sc_lv<16>> hout;

    sc_signal<sc_lv<32>> h_out;
    sc_signal<sc_lv<10>> pointer_c;
    sc_signal<sc_lv<2>> count_add;
    sc_signal<sc_logic> count_tanh, count_activation, count_mul, en_c, reg_en_sig,
        reg_en_tan, reg_en_mull, reg_en_add_in, reg_en_add_out, reg_en_demux_c;
    AAP_Datapath* AAP_DP;
    AAP_Controller* AAP_CU;
    SC_CTOR(AAP) {
        AAP_DP = new AAP_Datapath("AAP_Datapath_module");
        (*AAP_DP)(h_out, df, pointer_c, count_add, clk, count_tanh, count_activation, count_mul,
        en_c, reg_en_sig, reg_en_tan, reg_en_mull, reg_en_add_in, reg_en_add_out, reg_en_demux_c);
        AAP_CU = new AAP_Controller("AAP_Controller_module");
        (*AAP_CU)(clk, rst, run, ready, pointer_c, count_add, count_tanh, count_activation,
        count_mul, en_c, reg_en_sig, reg_en_tan, reg_en_mull, reg_en_add_in, reg_en_add_out, reg_en_demux_c);
        SC_METHOD(process_h_out);
        sensitive << h_out;
//      trace();
    }
    void process_h_out();
    void trace();
};
```

- Activation and Accumulation Plate (AAP)



Activation and Accumulation Plate (AAP)

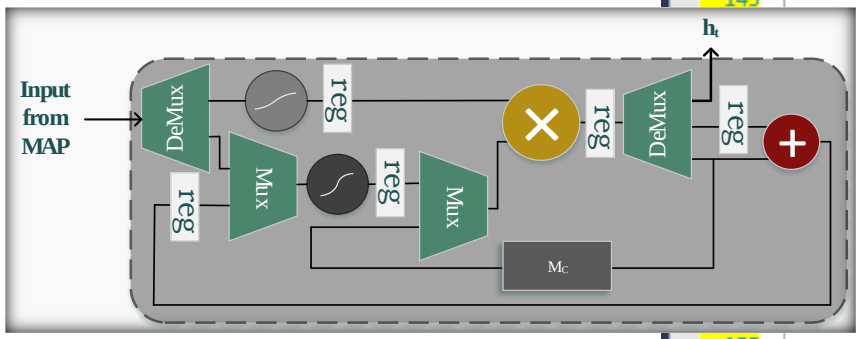
- Activation and Accumulation Plate (AAP): AAP_Datapath

AAP_Datapath.h | AAP_Controller.h | AAP.h

(Global Scope)

AAP_Datapath.h

```
96  SC_MODULE(AAP_Datapath) {
97      sc_out<sc_lv<32>> h_out;
98      sc_in<sc_lv<32>> d;
99      sc_in<sc_lv<10>> pointer_c;
100     sc_in<sc_lv<2>> count_add;
101     sc_in<sc_logic> clk, count_tanh, count_activation, count_mul, en_c, reg_en_sig, reg_en_tan,
102         reg_en_mull, reg_en_add_in, reg_en_add_out, reg_en_demux_c;
103     sc_signal<sc_lv<32>> c_ram[1024];
104     sc_signal<sc_lv<32>> sig_out, sig_in, reg_sig, tan_in, tan_in_1, tan_out, reg_tan,
105         mull_in, reg_mull, demux_c, c_reg, demux_add, add_out, reg_addo;
106     sc_signal<sc_lv<63>> mull_out;
107     sc_signal<sc_lv<32>> c;
108     sc_signal<sc_lv<32>> extra;
109     sc_signal<sc_lv<2>> demux_activation;
110     sc_signal<sc_lv<32>> mull_out_reg;
111     sc_signal<sc_logic> constant_0;
112     sc_signal<sc_lv<1>> mul_out_sig;
113     sc_signal<sc_lv<62>> mull_out_low;
114
115     DeMux1to3<32> *demux_input, *add_demux;
116     sigmoid_32 *sig;
117     Mux2to1<32> *mux2_1, *mux3_1;
118     tanh_32 *tan;
119     dRegister<32> *reg_sig_m, *reg_tan_m, *reg_mull_m, *reg_demux_m, *reg_add_m;
120     unsigned long long int temp1, temp2;
```




```
121     SC_CTOR(AAP_Datapath) {constant_0 = SC_LOGIC_0;
122         demux_input = new DeMux1to3<32>("demux_input_module");
123         (*demux_input)(demux_activation, sig_in, tan_in_1, extra, d);
124         sig = new sigmoid_32("sig_module");
125         (*sig)(sig_in, sig_out);
126         mux2_1 = new Mux2to1<32>("mux2_1_module");
127         (*mux2_1)(count_tanh, tan_in_1, reg_addo, tan_in);
128         tan = new tanh_32("tan_module");
129         (*tan)(tan_in, tan_out);
130         reg_sig_m = new dRegister<32>("reg_sig_m_module");
131         (*reg_sig_m)(constant_0, clk, reg_en_sig, sig_out, reg_sig);
132         reg_tan_m = new dRegister<32>("reg_tan_m_module");
133         (*reg_tan_m)(constant_0, clk, reg_en_tan, tan_out, reg_tan);
134         mux3_1 = new Mux2to1<32>("mux3_1_module");
135         (*mux3_1)(count_mul, reg_tan, c, mull_in);
136         reg_mull_m = new dRegister<32>("reg_mull_m_module");
137         (*reg_mull_m)(constant_0, clk, reg_en_mull, mull_out_reg, reg_mull);
138         add_demux = new DeMux1to3<32>("add_demux_module");
139         (*add_demux)(count_add, demux_c, demux_add, h_out, reg_mull);
140         reg_demux_m = new dRegister<32>("reg_demux_m_module");
141         (*reg_demux_m)(constant_0, clk, reg_en_demux_c, demux_c, c_reg);
142         reg_add_m = new dRegister<32>("reg_add_m_module");
143         (*reg_add_m)(constant_0, clk, reg_en_add_out, add_out, reg_addo);
            SC_METHOD(process_ram);
            sensitive << clk;
            SC_METHOD(process_others);
            sensitive << count_activation;
            SC_METHOD(process_others_2);
            sensitive << c_reg << demux_add;
            SC_METHOD(process_others_4);
            sensitive << mull_in << reg_sig;
            SC_METHOD(process_others_3);
            sensitive << mull_out;

155     void process_others();
156     void process_others_2();
157     void process_others_3();
158     void process_others_4();
159     void process_ram();
160 };
```

Activation and Accumulation Plate (AAP)

- Activation and Accumulation Plate (AAP): AAP_Datapath (cont.)

AAP_Datapath.h

```
AAP_Datapath.h | AAP_Controller.h
(Global Scope)
211 void AAP_Datapath::process_others(){
212     sc_lv<2> demux_activation_concat;
213     demux_activation_concat[1] = constant_0.read();
214     demux_activation_concat[0] = count_activation.read();
215     demux_activation.write(demux_activation_concat);
```

```
AAP_Datapath.h | AAP_Controller.h | AAP.h
(Global Scope)
218 void AAP_Datapath::process_others_2(){
219     if (c_reg.read().range(0, 0) == "Z" || c_reg.read().range(0, 0) == "X") add_out = c_reg;
220     else if (demux_add.read().range(0, 0) == "Z" || demux_add.read().range(0, 0) == "X") add_out = demux_add;
221     else add_out = c_reg.read().to_int() + demux_add.read().to_int();
```

```
AAP_Datapath.h | AAP_Controller.h | AAP.h
(Global Scope)
225 void AAP_Datapath::process_others_3(){
226     mull_out_reg.write((mull_out.read().range(62, 62), mull_out.read().range(60, 30)));
227 }
```

```
AAP_Datapath.h | AAP_Controller.h | AAP.h
AAP_Datapath | process_ram()
229 void AAP_Datapath::process_others_4(){
230     if (mull_in.read().range(0, 0) == "Z" || mull_in.read().range(0, 0) == "X") temp1 = 0;
231     else temp1 = mull_in.read().range(31, 0).to_ulong();
232     if (reg_sig.read().range(0, 0) == "Z" || reg_sig.read().range(0, 0) == "X") temp2 = 0;
233     else temp2 = reg_sig.read().range(31, 0).to_ulong();
234     unsigned long long int mult_out = temp1 * temp2;
235     unsigned long long int temAAP_Datapath = static_cast<unsigned long long int>(mult_out);
236     mull_out_low = temAAP_Datapath;
237     sc_lv<1> mul_out_sig = mull_in.read().range(31, 31) ^ reg_sig.read().range(31, 31);
238     mull_out = temAAP_Datapath;
239     if (mul_out_sig == "1"){
240         sc_lv<63> mul_out_sig_map = "1000000000000000000000000000000000000000000000000000000000000000";
241         sc_lv<63> tmp = temAAP_Datapath;
242         tmp = tmp | mul_out_sig_map;
243         mull_out = tmp;
244     }
245 }
```

```
AAP_Datapath.h | AAP_Controller.h | AAP.h
AAP_Datapath | process_others_4()
248 void AAP_Datapath::process_ram(){
249     if ((clk->event()) && (clk == '1')){
250         if (pointer_c.read().range(0, 0) != "Z" || pointer_c.read().range(0, 0) != "X"){
251             if (en_c == '1')
252                 c_ram[pointer_c.read().to_uint()] = demux_c;
253             else
254                 c = c_ram[pointer_c.read().to_uint()];
255         }
256     }
257 }
```

Activation and Accumulation Plate (AAP)

- Activation and Accumulation Plate (AAP): AAP_Controller

AAP_Datapath.h | AAP_Controller.h | AAP.h

(Global Scope)

AAP_Controller.h

```
36  SC_MODULE(AAP_Controller) {
37      sc_in<sc_logic> clk, rst, run;
38      sc_out<sc_logic> ready;
39      sc_out<sc_lv<10>> pointer_c_out;
40      sc_out<sc_lv<2>> count_add;
41      sc_out<sc_logic> count_tanh, count_activation, count_mul;
42      sc_out<sc_logic> en_c, reg_en_sig, reg_en_tan, reg_en_mull,
43                       reg_en_add_in, reg_en_add_out, reg_en_demux_c;
44      enum state { s0, s1, s1i, s2, s2i, s3, s3i, s4, s4i, s1_1,
45          s1_2, s1_3, s2_1, s3_1, s3_2, s3_3, s4_1, s4_2, s4_3 };
46      sc_signal<state> ps, ns;
47      sc_signal<sc_logic> pointer_en, pointer_rst;
48      sc_signal<sc_lv<10>> pointer;
49      AAP_Counter<10>* cnt;
50
51      SC_CTOR(AAP_Controller) {
52          cnt = new AAP_Counter<10>("AAP_Counter_module");
53          (*cnt)(clk, pointer_rst, pointer_en, pointer);
54          SC_METHOD(controller_comb);
55          sensitive << ps << run;
56          SC_METHOD(controller_seq);
57          sensitive << clk;
58          SC_METHOD(controller_init);
59          sensitive << ps;
60          SC_METHOD(process_others);
61          sensitive << pointer << ps << ns;
62      }
63      void controller_seq();
64      void controller_comb();
65      void controller_init();
66      void process_others();
67  };
```

AAP_Datapath.h | AAP_Controller.h | AAP.h

(Global Scope)

```
69  void AAP_Controller::controller_comb() {
70      ns = s0;
71      switch (ps.read()){
72      case s0:
73          if (run == '1') ns = s1_1;
74          else ns = s0; break;
75      case s1:
76          if (run == '1') ns = s1_1;
77          else ns = s1; break;
78      case s1_1: ns = s1_2; break;
79      case s1_2: ns = s1_3; break;
80      case s1_3: ns = s1i; break;
81      case s1i:
82          if (run == '1') ns = s2;
83          else ns = s1i; break;
84      case s2: ns = s2_1; break;
85      case s2_1: ns = s2i; break;
86      case s2i:
87          if (run == '1') ns = s3;
88          else ns = s2i; break;
89      case s3: ns = s3_1; break;
90      case s3_1: ns = s3_2; break;
91      case s3_2: ns = s3_3; break;
92      case s3_3: ns = s3i; break;
93      case s3i:
94          if (run == '1') ns = s4;
95          else ns = s3i; break;
96      case s4: ns = s4_1; break;
97      case s4_1: ns = s4_2; break;
98      case s4_2: ns = s4_3; break;
99      case s4_3: ns = s4i; break;
100     case s4i:
101         if (run == '1') ns = s1_1;
102         else ns = s4i; break;
103     default: ns = s0; break;
104     }
105 }
```

Activation and Accumulation Plate (AAP)

- Activation and Accumulation Plate (AAP): AAP_Controller (cont.)

AAP_Controller.h

AAP_Datapath.h | AAP_Cor

AAP_Controller

```
107 void AAP_Controller::controller_init() {
108     count_add = "00";
109     count_tanh = SC_LOGIC_0;
110     count_activation = SC_LOGIC_0;
111     count_mul = SC_LOGIC_0;
112     reg_en_sig = SC_LOGIC_0;
113     en_c = SC_LOGIC_0;
114     reg_en_tan = SC_LOGIC_0;
115     reg_en_mull = SC_LOGIC_0;
116     reg_en_add_in = SC_LOGIC_0;
117     reg_en_add_out = SC_LOGIC_0;
118     reg_en_demux_c = SC_LOGIC_0;
119     ready = SC_LOGIC_0;
120     pointer_rst = SC_LOGIC_0;
121     pointer_en = SC_LOGIC_0;
122
123     switch (ps.read()){
124     case s0:
125         pointer_rst = SC_LOGIC_1;
126         break;
127     case s1_1:
128         count_activation = SC_LOGIC_0;
129         break;
130     case s1_2:
131         reg_en_sig = SC_LOGIC_1;
132         count_mul = SC_LOGIC_1;
133         break;
134     case s1_3:
135         count_mul = SC_LOGIC_1;
136         reg_en_mull = SC_LOGIC_1;
137         count_add = "00";
138         reg_en_demux_c = SC_LOGIC_1;
139         reg_en_add_out = SC_LOGIC_1;
140         break;
141     case s2:
142         count_add = "00";
143         reg_en_add_out = SC_LOGIC_1;
144         reg_en_demux_c = SC_LOGIC_1;
145         count_activation = SC_LOGIC_0;
146         break;
```

```
148     case s2_1:
149         reg_en_sig = SC_LOGIC_1;
150         break;
151     case s3:
152         count_activation = SC_LOGIC_1;
153         count_tanh = SC_LOGIC_0;
154         break;
155     case s3_1:
156         reg_en_tan = SC_LOGIC_1;
157         count_mul = SC_LOGIC_0;
158         break;
159     case s3_2:
160         reg_en_mull = SC_LOGIC_1;
161         count_add = "01";
162         break;
163     case s3_3 :
164         count_add = "01";
165         reg_en_add_out = SC_LOGIC_1;
166         break;
167     case s4:
168         count_activation = SC_LOGIC_0;
169         count_tanh = SC_LOGIC_1;
170         en_c = SC_LOGIC_1;
171         break;
172     case s4_1:
173         reg_en_sig = SC_LOGIC_1;
174         reg_en_tan = SC_LOGIC_1;
175         break;
176     case s4_2:
177         reg_en_mull = SC_LOGIC_1;
178         break;
179     case s4_3:
180         count_add = "11";
181         pointer_en = SC_LOGIC_1;
182         ready = SC_LOGIC_1;
183         break;
184     default:
185         ...
186         break;
187     }
}
```

Activation and Accumulation Plate (AAP)

- Activation and Accumulation Plate (AAP): AAP_Controller (cont.)

AAP_Controller.h

AAP_Datapath.h | AAP_Controller.h

AAP_Controller

```
202 void AAP_Controller::controller_seq(){
203     if (clk->event() && clk == SC_LOGIC_1){
204         if (rst == SC_LOGIC_1)
205             ps = s0;
206         else
207             ps = ns;
208     }
209 }
```

AAP_Datapath.h | AAP_Controller.h | AAP.h

AAP_Controller

```
211 void AAP_Controller::process_others(){
212     pointer_c_out = pointer;
213 }
```

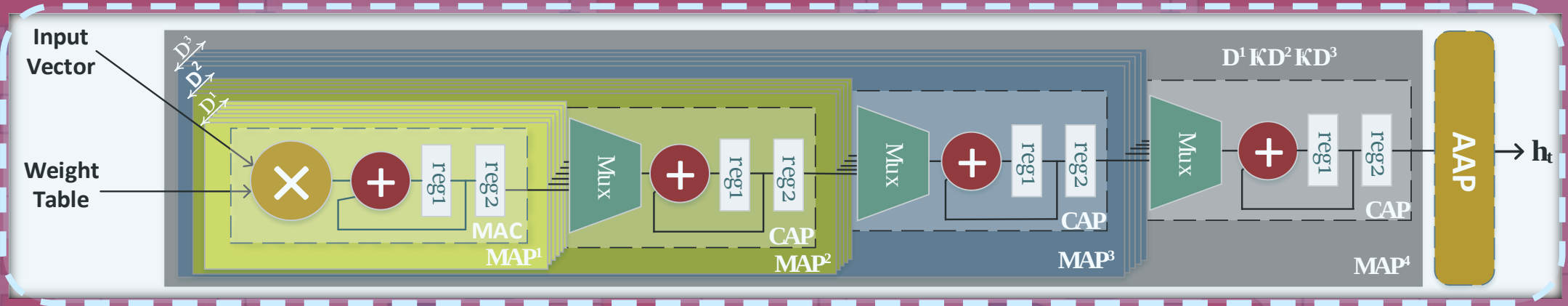
SystemC Machine Learning Automotive Application

Introduction

\+ DiBA, an RTL Hardware Architecture for LSTM (Top-Down)

– SystemC Implementation of DiBA (Bottom-Up)

- Multiply and Accumulate (MAC)
- Convergence Adder Plate (CAP)
- Matrix-multiplication and Accumulation Plate (MAP)
- Activation and Accumulation Plate (AAP)
- n-Dimensional Bitslice Architecture (DiBA)
- Data Arrangement (Testbench)



n-Dimensional Bitslice Architecture (DiBA)

- n-Dimensional Bitslice Architecture (DiBA)

MAP.h | DiBA.h | Sigmoid.h | AAP_Datapath.h

DiBA<h, a, b, c, d, e, f, k>

DiBA.h

```
1   #pragma once
2   #include "MAP.h"
3   #include "AAP.h"
4   int id = 0;
5
6   template <int h, int a, int  b, int c, int d, int e, int f, int k>
7   SC_MODULE(DiBA){
8       sc_in<sc_logic> clk, rst, run, pause;
9       sc_out<sc_logic> reg_out;
10      sc_in<sc_lv<16 * a*b*c*d*e*f>> din, w;
11      sc_out<sc_lv<16>> dout;
12
13      sc_signal<sc_logic> run_tmp;
14      sc_signal<sc_lv<32>> outMAP;
15      MAP< h, a, b, c, d, e, f, k>* MAP_Str;
16      AAP* AAP_Str;
17      SC_CTOR(DiBA) {
18          AAP_Str = new AAP("AAP_module");
19          (*AAP_Str)(clk, rst, run_tmp, outMAP, reg_out, dout);
20          MAP_Str = new MAP< h, a, b, c, d, e, f, k>("MAP_module");
21          MAP_Str->clk(clk);
22          MAP_Str->rst(rst);
23          MAP_Str->run(run);
24          MAP_Str->pause(pause);
25          MAP_Str->ready(run_tmp);
26          MAP_Str->din(din);
27          MAP_Str->w(w);
28          MAP_Str->dout(outMAP);
29          trace();
30      }
31      void trace();
32  };
```

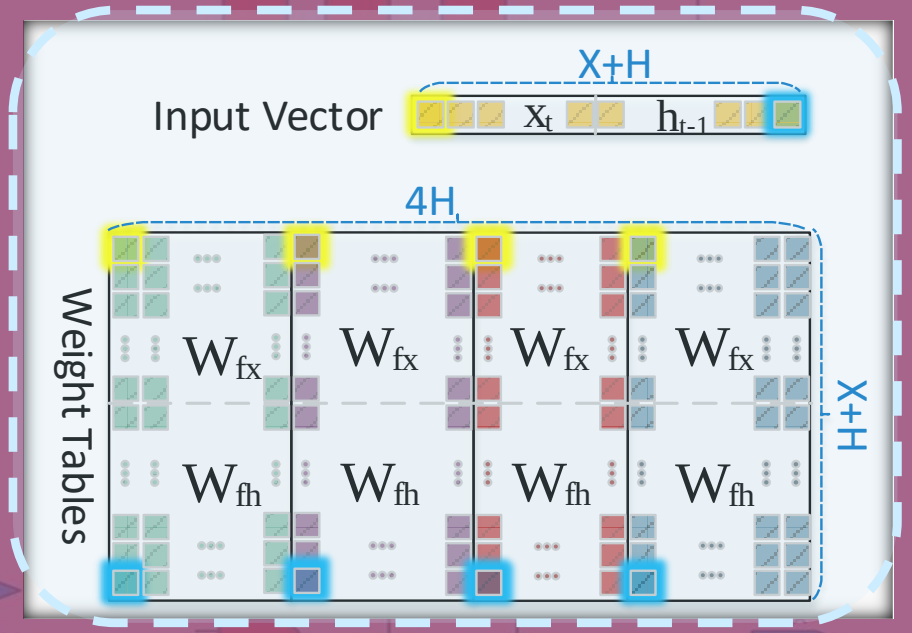
SystemC Machine Learning Automotive Application

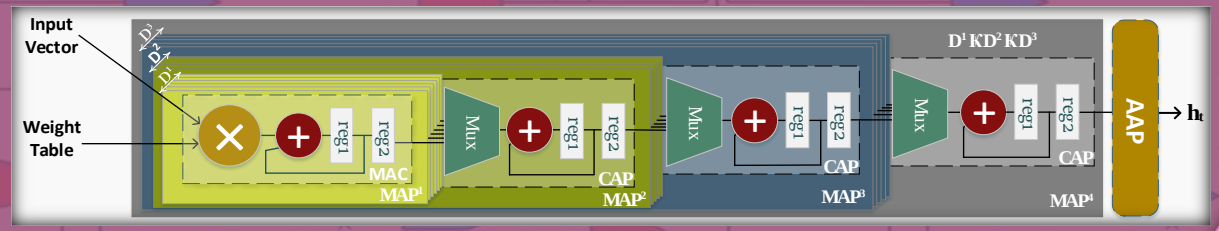
Introduction

\+ DiBA, an RTL Hardware Architecture for LSTM (Top-Down)

– SystemC Implementation of DiBA (Bottom-Up)

- Multiply and Accumulate (MAC)
- Convergence Adder Plate (CAP)
- Matrix-multiplication and Accumulation Plate (MAP)
- Activation and Accumulation Plate (AAP)
- n-Dimensional Bitslice Architecture (DiBA)
- Data Arrangement (Testbench)





Data Arrangement (Testbench)

- Data Arrangement (Testbench)

DiBA_TB.h

```
DiBA_TB.h | Main.cpp | DiBA.h
DiBA_TB<h, a, b, c, d, e, f, k>
 1  #include "DiBA.h"
 2
 3  template <int h, int a, int  b, int c, int d, int e, int f, int k>
 4  SC_MODULE(DiBA_TB){
 5      sc_signal<sc_logic> clk, rst, run, pause;
 6      sc_signal<sc_logic>ready;
 7      sc_signal<sc_lv<16 * a*b*c*d*e*f>> din;
 8      sc_signal<sc_lv<16 * a*b*c*d*e*f>> w;
 9      sc_signal< sc_lv<16>>dout;
10      DiBA<h, a, b, c, d, e, f, k>* DUT;
11      SC_CTOR(DiBA_TB) {
12          DUT = new DiBA<h, a, b, c, d, e, f, k>("DiBA_module");
13          (*DUT)(clk, rst, run, pause, ready, din, w, dout);
14          SC_THREAD(serialgeneration);
15          SC_THREAD(clockgeneration);
16      }
17      void serialgeneration();
18      void clockgeneration();
19  };
```

```
DiBA_TB.h | Main.cpp | DiBA.h
DiBA_TB<h, a, b, c, d, e, f, k>
23  void DiBA_TB<h, a, b, c, d, e, f, k>::clockgeneration() {
24      while (true) {
25          wait(0.5, SC_NS);
26          clk = SC_LOGIC_1;
27          wait(0.5, SC_NS);
28          clk = SC_LOGIC_0;
29      }
30  }
```

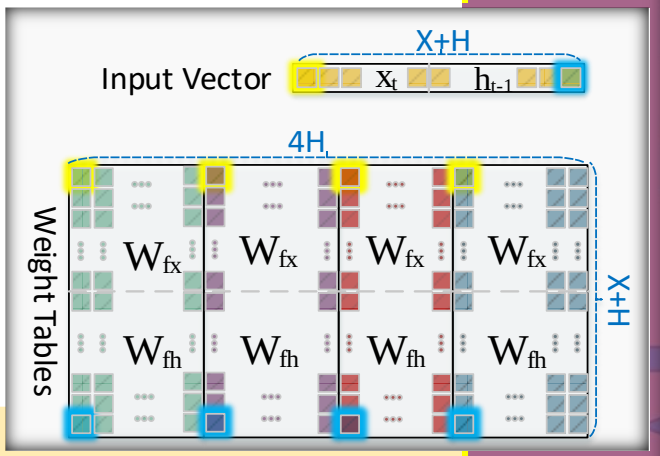
Data Arrangement (Testbench)

- Data Arrangement (Testbench) (cont.)

DiBA_TB.h

```
#include "DiBA.h"

template <int h, int a, int  b, int c, int d, int e, int f, int k>
SC_MODULE(DiBA_TB){
    sc_signal<sc_logic> clk, rst, run, pause;
    sc_signal<sc_logic>ready;
    sc_signal<sc_lv<16 * a*b*c*d*e*f>> din;
    sc_signal<sc_lv<16 * a*b*c*d*e*f>> w;
    sc_signal< sc_lv<16>>dout;
    DiBA<h, a, b, c, d, e, f, k>* DUT;
    SC_CTOR(DiBA_TB) {
        DUT = new DiBA<h, a, b, c, d, e, f, k>("DiBA_module");
        (*DUT)(clk, rst, run, pause, ready, din, w, dout);
        SC_THREAD(serialgeneration);
        SC_THREAD(clockgeneration);
    }
    void serialgeneration();
    void clockgeneration();
};
```

```
template <int h, int a, int  b, int c, int d, int e, int f, int k>
void DiBA_TB<h, a, b, c, d, e, f, k>::serialgeneration() {
    while (true) {
        ifstream file_din("in_test.txt");
        ifstream file_w("w_test.txt");
        string din_line, w_line;
        pause = SC_LOGIC_0;
        rst = SC_LOGIC_0;
        wait(2, SC_NS);
        rst = SC_LOGIC_1;
        wait(10, SC_NS);
        rst = SC_LOGIC_0;
        wait(3, SC_NS);
        run = SC_LOGIC_1;
        wait(1, SC_NS);
        run = SC_LOGIC_0;
        for (int i = 0; i < (4 * 350); i++){
            for (int j = 0; j < 16; j++){
                file_din >> din_line; file_w >> w_line;
                din_line = din_line.substr(1, din_line.length() - 2);
                w_line = w_line.substr(1, w_line.length() - 2);
                din = &din_line[0]; w = &w_line[0];
                wait(1, SC_NS);
                file_din >> din_line; file_din >> din_line; file_din >> din_line;
                file_w >> w_line; file_w >> w_line; file_w >> w_line;
            }
            wait(4, SC_NS);
            run = SC_LOGIC_1;
            wait(1, SC_NS);
            run = SC_LOGIC_0;
        }
        file_din.close(); file_w.close();
        wait();
    }
}
```



- An outer loop for the four weight tables and another loop for the columns of a table
- These loops are peeled off to reach DiBA

Data Arrangement (Testbench)

- Main

DiBA_TB.h DiBA.h Main.cpp

(Global Scope)

Main.cpp

```
1   #include "DiBA_TB.h"
2
3   int sc_main(int argc, char **argv) {
4       sc_trace_file* VCDfile1;
5       VCDfile1 = sc_create_vcd_trace_file("DiBA_TB");
6       DiBA_TB<5, 2, 4, 4, 4, 4, 1, 8>* D = new DiBA_TB<5, 2, 4, 4, 4, 4, 1, 8>("DiBA_TB");
7
8       sc_trace(VCDfile1, D->clk, "clk");
9       sc_trace(VCDfile1, D->rst, "rst");
10      sc_trace(VCDfile1, D->run, "run");
11      sc_trace(VCDfile1, D->pause, "pause");
12      sc_trace(VCDfile1, D->ready, "ready");
13      sc_trace(VCDfile1, D->din, "din");
14      sc_trace(VCDfile1, D->w, "w");
15      sc_trace(VCDfile1, D->dout, "dout");
16
17      sc_start(200, SC_NS);
18      return 0;
19  }
```

Data Arrangement (Testbench)

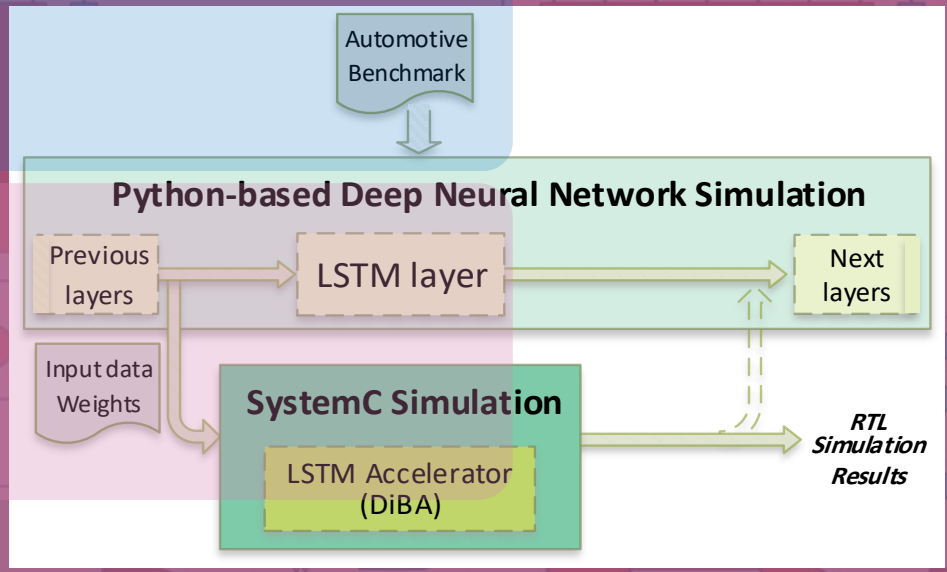
- Result

DiBA Architecture (cont.)

Singular bit implementation of the FA function with timing

Using the timed logic functions

Multiple bit implementation of the FA function with timing



Ref. [1] – Dynamic Memory - P. 74-76