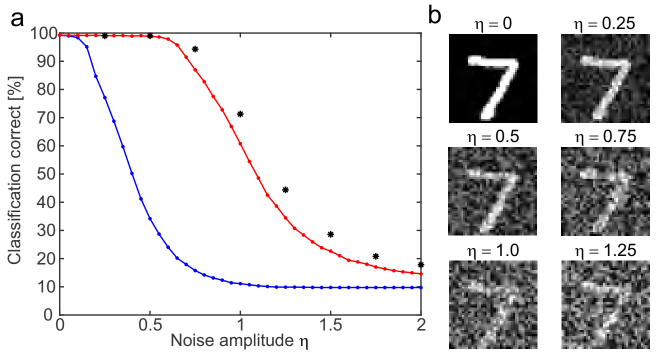

FIGURE 3. (a) Performance classification of SbS NN versus equivalent CNN, and (b) example of the first pattern in the MNIST test data set with different amounts of noise.

10000 patterns in dependency of the noise level for positive additive uniformly distributed noise. The blue curve shows the performance for the tensor flow network, while the red curve shows the performance for the SbS network with 1200 spikes per inference population. Beginning with a noise level of 0.1, the respective performances are different with a p - level of at least $10^{-6}$ (tested with the Fisher exact test). Increasing the number of spikes per SbS population to 6000 (performance values shown as black stars), shows that more spike can improve the performance under noise even more.

## IV. SYSTEM DESIGN

In this section, we revise the system design of [15]. In Ref. [15], we presented a scalable hardware architecture composed of generic homogeneous accelerator units (AUs). This design works entirely with standard floating-point arithmetic (IEEE 754), which represents an unnecessary overhead for error-resilient applications. Furthermore, this architecture does not implement stationary synaptic weight matrices in the hardware AUs, resulting in heavy data movement and longer computational latency.

In this publication, we present an enhanced hardware architecture composed of specialized heterogeneous processing units (PUs) with hybrid custom floating-point and logarithmic dot-product approximation. This approach represents an advantageous design for error-resilient applications in resource-constrained devices due to the reduced computational costs and memory footprint. Furthermore, the proposed approach allows the implementation of stationary synaptic weight matrices. These novelties result in an improved overall system design.

Regarding the software architecture, this is structured as a layered object-oriented application framework written in the C programming language. This offers a comprehensive high level embedded software application programming interface (API) that allows the construction of scalable sequential SbS networks with configurable hardware acceleration. Conceptually this design is modular, reusable, and extensible. The overall structure is depicted in **Fig.** 4.

**Algorithm 1:** SbS network inference.

```
input: Layers of the network as $H^l$, where
    $l$ is the layer index.
input: $N_L$ as the number of layers.
input: $N_X^l, N_Y^l$ as the size of layers.
input: $N_S$ as the number of spikes for inference.
output: Inference.
1:  for $t \leftarrow 0$ to $N_S - 1$ do
        Initialization of $H^l(i_X, i_Y, :)$ :
2:      if $t == 0$ then
3:          for $l \leftarrow 0$ to $N_L - 1$ do
4:              for $i_X \leftarrow 0, i_Y \leftarrow 0$ to $N_X^l - 1, N_Y^l - 1$ do
5:                  for $i_H \leftarrow 0$ to $N_H^l - 1$ do
6:                      $H^l(i_X, i_Y, i_H) \leftarrow 1/N_H^l$
7:                  end for
8:              end for
9:          end for
10:     end if
        Production of spikes :
11:     for $l \leftarrow 0$ to $N_L - 1$ do
12:         if $l == 0$ then
13:             Draw spikes from Input
14:         else
15:             Draw spikes from $H^l$
16:         end if
17:     end for
        Update layers :
18:     for $l \leftarrow 0$ to $N_L - 1$ do
19:         Update $H^l$
20:     end for
21: end for
```

**Algorithm 2:** Spike production.

```
input: Layer as $H_t \in \mathbb{R}^{N_X \times N_Y \times N_H}$, where
    $N_X$ is the layer width,
    $N_Y$ is the layer height
    $N_H$ is the length of $\vec{h}$ (IP vector).
output: Output spikes as $S_t^{out} \in \mathbb{N}^{N_X \times N_Y}$
1:  for $i_X \leftarrow 0, i_Y \leftarrow 0$ to $N_X - 1, N_Y - 1$ do
        Generate spike :
2:      $th \leftarrow MT19937PseudoRandom()/(2^{32} - 1)$
3:      $acu \leftarrow 0$
4:      for $i_H \leftarrow 0$ to $N_H - 1$ do
5:          $acu \leftarrow acu + H_t(i_X, i_Y, i_H)$
6:          if $th \leq acu$ or $i_H == N - 1$ then
7:              $S_t^{out}(i_X, i_Y) \leftarrow i_H$
8:          end if
9:      end for
10: end for
```

### A. HARDWARE ARCHITECTURE

As a hardware/software co-design, the system architecture is an embedded CPU+FPGA-based platform, where the ac-

**Algorithm 3:** SbS layer update.

```
input: Layer as $H \in \mathbb{R}^{N_X \times N_Y \times N_H}$, where
       $N_X$ is the layer width,
       $N_Y$ is the layer height
       $N_H$ is the length of $\vec{h}$ (IP vector).
input: Synaptic matrix as $W \in \mathbb{R}^{K_X \times K_Y \times M_H \times N_H}$, where
       $K_X \times K_Y$ is the size of the convolution/pooling kernel,
       $M_H$ is the length of $\vec{h}$ from previous layer,
       $N_H$ is the length of $\vec{h}$ from this layer.
input: Input spike matrix from previous layer as
       $S_t^{in} \in \mathbb{N}^{N_{Xin} \times N_{Yin}}$, where
       $N_{Xin}$ is the width of the previous layer,
       $N_{Yin}$ is the height of the previous layer.
input: Strides of X and Y as $stride_{Xi}$ and $stride_{Yi}$,
       respectively.
input: Epsilon as $\epsilon \in \mathbb{R}$.
output: Updated layer as $H^{new} \in \mathbb{R}^{N_X \times N_Y \times N_H}$.
       Update layer :
 1:    $i_{Xi} \leftarrow 0$ // X and Y index for $S_t^{in}$
 2:    $i_{Yi} \leftarrow 0$
 3:    for $i_Y \leftarrow 0$ to $N_Y - 1$ do
 4:      for $i_X \leftarrow 0$ to $N_X$ do
 5:        $\vec{h} \leftarrow H(i_X, i_Y, :)$
           Update IP :
 6:        for $j_X \leftarrow 0, j_Y \leftarrow 0$ to $K_X - 1, K_Y - 1$ do
 7:          $s_t \leftarrow S_t^{in}(i_{Xi} + j_X, i_{Yi} + j_Y)$
 8:          $\vec{w} \leftarrow W(j_X, j_Y, s_t, :)$
 9:          $\vec{p} \leftarrow 0$
             Dot-product :
10:          $r \leftarrow 0$
11:          for $j_H \leftarrow 0$ to $N_H - 1$ do
12:            $\vec{p}(j_H) \leftarrow \vec{h}(j_H)\vec{w}(j_H)$
13:            $r \leftarrow r + \vec{p}(j_H)$
14:          end for
15:          if $r \neq 0$ then
               Update IP vector :
16:            for $i_H \leftarrow 0$ to $N_H - 1$ do
17:              $h^{new}(i_H) \leftarrow \frac{1}{1+\epsilon}\left(h(i_H) + \epsilon\frac{\vec{p}(i_H)}{r}\right)$
18:            end for
               Set the new H vector for the layer :
19:            $H^{new}(i_X, i_Y, :) \leftarrow \vec{h}^{new}$
20:          end if
21:        end for
22:        $i_{Xi} \leftarrow i_{Xi} + stride_{Xi}$
23:      end for
24:      $i_{Yi} \leftarrow i_{Yi} + stride_{Yi}$
25:    end for
```

celeration of SbS network computation is based on asynchronous[1] execution in parallel heterogeneous processing units: *Spike* (input layer), *Conv* (convolution), *Pool* (pooling),

[1]The system is synchronous at the circuit level, but the execution is asynchronous in terms of jobs.

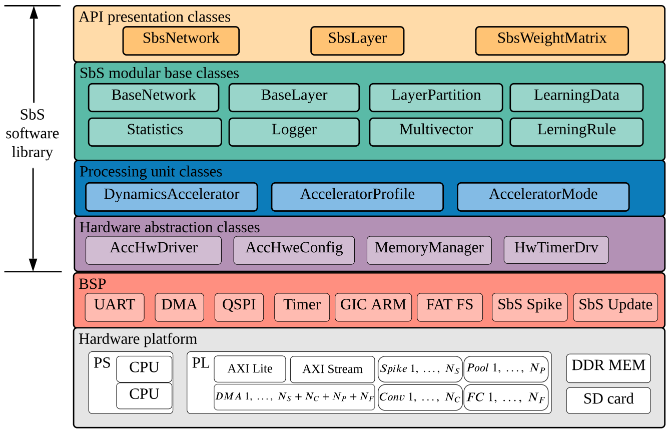

**FIGURE 4.** System-level overview of the embedded software architecture.

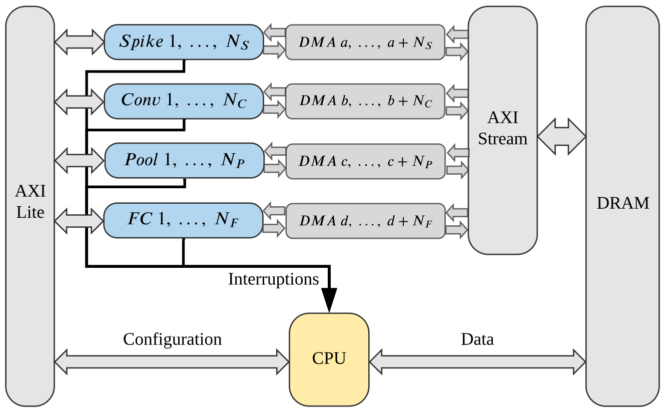

**FIGURE 5.** System-level hardware architecture with scalable number of heterogeneous PUs: *Spike*, *Conv*, *Pool*, and *FC*

and *FC* (fully connected). **Fig.** 5 illustrates the system hardware architecture as a scalable structure. For hyperparameter configuration, each PU uses AXI-Lite interface. For data transfer, each PU uses AXI-Stream interfaces via Direct Memory Access (DMA) allowing data movement with high transfer rate. Each PU asserts an interrupt flag once the job or transaction is complete. This interrupt event is handled by the embedded CPU to collect results and start a new transaction.

The hardware architecture can resize its resource utilization by changing the number of PUs instances prior to the hardware synthesis, this provides scalability with a good trade-off between area and throughput. The dedicated PUs for *Conv* and *FC* implement the proposed dot-product approximation as a system component. The PUs are written in C using Vivado HLS (High-Level Synthesis) tool. In this publication, we illustrate the integration of the approximate dot-product component on the *Conv* processing unit.

## B. CONV PROCESSING UNIT

This hardware module computes the IP dynamics defined by **Eq.** (1) and offers two modes of operation: *configuration* and *computation*.