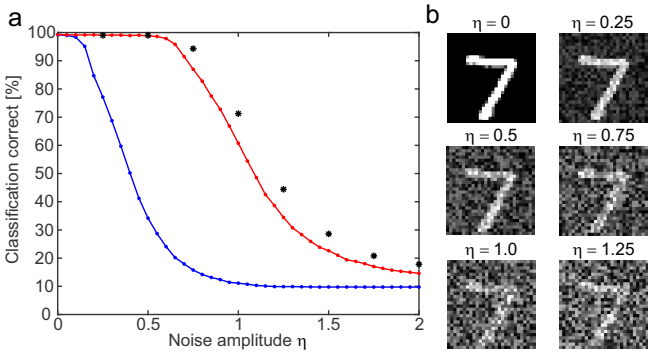

FIGURE 3. (a) Performance classification of SbS NN versus equivalent CNN, and (b) example of the first pattern in the MNIST test data set with different amounts of noise.

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 layer update.

input: $L \in \mathbb{R}^{L_W \times L_H \times N}$, $S_t^{in} \in \mathbb{N}^{L_W \times L_H}$, $N \in \mathbb{N}$, $\epsilon \in \mathbb{R}$
output: $L^{new} \in \mathbb{R}^{L_W \times L_H \times N}$, $S_t^{out} \in \mathbb{N}^{L_W \times L_H}$

Update layer :
for $L_X \leftarrow 0, L_Y \leftarrow 0$ **to** $L_W - K - 1, L_H - K - 1$ **do**
  Generate spike :
  $th \leftarrow MT19937PseudoRandom()$
  $acu \leftarrow 0$
  for $idx \leftarrow 0$ **to** $N - 1$ **do**
    if $th \leq acu$ **or** $idx = N - 1$ **then**
      $S_t^{out}(L_X, L_Y) \leftarrow idx$
      goto *Update IP*
    end if
    $acu \leftarrow acu + \vec{h}_\mu(idx)$
  end for
  Update IP :
  $\vec{h}_\mu \leftarrow L(L_X, L_Y)$
  for $K_X \leftarrow 0, K_Y \leftarrow 0$ **to** $K - 1, K - 1$ **do**
    $s_t \leftarrow S_t^{in}(L_X + K_X, L_Y + K_Y)$
    $\vec{w} \leftarrow W(K_X, K_Y, s_t)$
    $\vec{p} \leftarrow 0$
    Dot-product :
    $r_\mu \leftarrow 0$
    for $j \leftarrow 0$ **to** $N - 1$ **do**
      $\vec{p}(j) \leftarrow \vec{h}_\mu(j)\vec{w}(j)$
      $r_\mu \leftarrow r_\mu + \vec{p}(j)$
    end for
    if $r_\mu(s_t) \neq 0$ **then**
      Update IP vector :
      for $i \leftarrow 0$ **to** $N - 1$ **do**
        $h_\mu^{new}(i) \leftarrow \frac{1}{1+\epsilon}\left(h_\mu(i) + \epsilon\frac{\vec{p}(i)}{r_\mu}\right)$
      end for
      Set the new IP vector on the layer :
      $L^{new}(L_X, L_Y) \leftarrow \vec{h}_\mu^{new}$
    end if
  end for
end for

A. HARDWARE ARCHITECTURE

As a hardware/software co-design, the system architecture is an embedded CPU+FPGA-based platform, where the acceleration of SbS network computation is based on asynchronous[1] execution in parallel heterogeneous processing units: *Spike* (input layer), *Conv* (convolution), *Pool* (pooling), 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

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