Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

CNN Sensor Analytics with Hybrid-Float6 Quantization on Low-Power Embedded FPGAs.

YARIB NEVAREZ[1], ANDREAS BEERING[1], AMIR NAJAFI[1], ARDALAN NAJAFI[1], WANLI YU[1], YIZHI CHEN[2], KARL-LUDWIG KRIEGER[1], ALBERTO GARCIA-ORTIZ[1] (Member, IEEE),

[1]Institute of Electrodynamics and Microelectronics, University of Bremen, Bremen 28359, Germany
[2]School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, 10044 Stockholm, Sweden

Corresponding author: Yarib Nevarez (e-mail: nevarez@item.uni-bremen.de).

This work is funded by the Consejo Nacional de Ciencia y Tecnologia (CONACYT) and the Federal Ministry for Economic Affairs and Climate Action ZIM project IMOP (ZF4176708LP9).

ABSTRACT The use of artificial intelligence (AI) in sensor analytics is entering a new era based on the use of ubiquitous embedded connected devices. This transformation requires the adoption of design techniques that reconcile accurate results with sustainable system architectures. As such, improving the efficiency of AI hardware engines as well as backward compatibility must be considered. In this paper, we present the Hybrid-Float6 (HF6) quantization and its dedicated hardware design. We propose an optimized multiply-accumulate (MAC) hardware by reducing the mantissa multiplication to a multiplexor-adder operation. We exploit the intrinsic error tolerance of neural networks to further reduce the hardware design with approximation. To preserve model accuracy, we present a quantization-aware training (QAT) method, which in some cases improves accuracy. We demonstrate this concept in 2D convolution layers. We present a lightweight tensor processor (TP) implementing a pipelined vector dot-product. For compatibility and portability, the 6-bit floating-point (FP) is wrapped in the standard FP format, which is automatically extracted by the proposed hardware. The hardware/software architecture is compatible with TensorFlow (TF) Lite. We evaluate the applicability of our approach with a CNN-regression model for anomaly localization in a structural health monitoring (SHM) application based on acoustic emission (AE). The embedded hardware/software framework is demonstrated on XC7Z007S as the smallest Zynq-7000 SoC. The proposed implementation achieves a peak power efficiency and run-time acceleration of 5.7 GFLOPS/s/W and $48.3\times$, respectively.

INDEX TERMS Convolutional neural networks, structural health monitoring, hardware accelerator, TensorFlow Lite, embedded systems, FPGA, custom floating-point

I. INTRODUCTION

THERE is a growing demand for ubiquitous AI sensor analytics. Industry 4.0 and smart city infrastructure leverage AI solutions to increase productivity and adaptability [1]. These solutions are powered by advances in ML, compute engines, and big data. Hence, improvements of these should be considered for research, as they are the machinery of the future.

Convolutional neural networks (CNNs) represent the essential building blocks in 2D pattern analytics. Sensor-based applications such as mechanical fault diagnosis [2], [3], structural health monitoring [4], human activity recognition (HAR) [5], hazardous gas detection [6] have been powered by CNN models in industry and academia. CNN-based models, as one of the main types of artificial neural networks (ANNs), have been widely used in sensor analytics with automatic learning from sensor data [7]–[10]. In this context, CNN models are applied for automatic feature learning, usually, from 1D time series as well as for 2D time-frequency spectrograms. CNN models provide advantages such as local dependency, scale invariance, and noise resilience in analytics [11]. However, CNN models are computationally intensive and power-hungry. This is particularly challenging for low-power embedded applications, such as in the Internet of Things (IoT) field.

For ML inference, dedicated hardware architectures are

typically used to enhance compute performance and power efficiency. In terms of computational throughput, graphics processing units (GPUs) offer the highest performance; in terms of power efficiency, ASIC and FPGA solutions are more energy efficient [12]. As a result, numerous commercial ASIC and FPGA accelerators have been proposed, targeting both high performance computing (HPC) for data-centers and embedded systems applications.

However, most FPGA accelerators have been implemented to target mid- to high-range FPGAs for computationally intensive CNN models such as AlexNet, VGG-16, and ResNet-18. The main drawbacks of these implementations are power supply demands, physical dimensions, heat sink requirements, air cooling, and high price. In some cases, these implementations are not feasible for ubiquitous low-power/resource-constrained applications.

To reduce inference hardware there are two types of research [13]: the first one is deep compression including weight pruning, weight quantization, and compression storage [14], [15]; the second type of research corresponds to a more efficient data representation, also known as custom quantization for dedicated hardware implementation. In this group, hardware implementations with customized 8-bit floating-point computation have been proposed [13], [16], [17]. However, these architectures are inadequate for embedded applications, the target devices are high-end FPGA and PCIe devices.

Reducing the compute hardware with more aggressive quantization such as binary [18], ternary [19], and mixed precision (2-bit activations and ternary weights) [20] typically incur significant accuracy degradation for very low precisions, especially for complex problems [21].

In this paper, we present the Hybrid-Float6 quantization and its dedicated hardware design. In this concept, feature maps are represented by a standard FP number representation and the trainable parameters by 6-bit FP. To preserve accuracy, we present a quantization-aware training (QAT) method. For ML compatibility/portability, the 6-bit FP can be wrapped into the standard FP number representation. We propose a parameterized tensor processor implementing a pipelined vector dot-product with HF6. The proposed hardware extracts the 6-bit representation automatically from the standard FP format and performs the computation. The 6-bit FP representation uses 4-bit exponent and 1-bit mantissa. This approach enables an optimized MAC design by reducing the mantissa multiplication to a multiplexer-adder operation. We leverage the intrinsic error tolerance of ANN to further reduce the hardware design with approximation. This approach reduces latency, resource utilization, and power dissipation. The embedded hardware/software architecture is integrated with TensorFlow Lite using delegate interface to accelerate *Conv2D* tensor operations. We evaluate the applicability of our approach with a CNN-regression model and hardware design exploration for sensor analytics of SHM for anomaly localization. The embedded hardware/software framework is demonstrated on XC7Z007S as the smallest and most inexpensive Zynq SoC device, see **Fig.** 1. To the best of our knowledge, this is the first research addressing 6-bit floating-point quantization on CNN models and its dedicated hardware design.

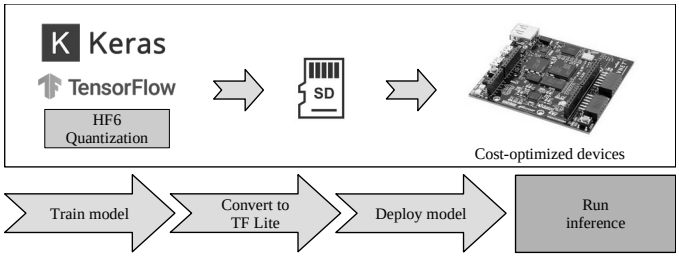


FIGURE 1. The workflow of our approach on embedded FPGAs.

Our main contributions are as follows:

1) We present the Hybrid-Float6 quantization and its dedicated hardware design. We propose an optimized hardware MAC by reducing the mantissa multiplication to a multiplexer-adder operation. We exploit the intrinsic error tolerance of ANN to further reduce the hardware design with approximation. To preserve model accuracy, we present a quantization-aware training method, which provides regularization effects.
2) We develop a custom hardware/software co-design framework for low-power analytics on resource-constrained embedded FPGAs. TensorFlow Lite micro is integrated in this framework.
3) We present a customizable tensor processor as a dedicated hardware for HF6. This design computes *Conv2D* tensor operations employing a pipelined vector dot-product with parametrized on-chip memory utilization. For exploration purposes, the compute engine can be synthesized with the proposed HF6 hardware or with Xilinx LogiCORE IPs (for standard floating-point).
4) We demonstrate the potential of our approach with a CNN-regression model for anomaly localization in SHM based on AE. We address a hardware design exploration. We evaluate inference accuracy, compute performance, hardware resource utilization, and energy consumption.

The rest of the paper is organized as follows. Section II covers the related work; Section III introduces the background for *Conv2D* tensor operation and floating-point number representation; Section IV describes the system design of the hardware/software architecture and the quantized aware training method; Section V presents the experimental results thorough a design exploration flow; Section VI concludes the paper.

This work is available to the community as an open-source project at https://github.com/YaribNevarez/tensorflow-lite-fpga-delegate.git.

II. RELATED WORK

In the literature we find plenty of hardware architectures for CNN accelerators implemented in FPGA. Most of the research implements fixed-point quantization, and very limited research focuses on FP. Moreover, to the best of our knowledge, there is no research work related to FP inference for low-power embedded applications.

A. HYBRID CUSTOM FLOATING-POINT

In [22], Liangzhen Lai et al. proposed a mixed data representation with floating-point for weights and fixed-point for activations. This work demonstrated on SqueezeNet, AlexNet, GoogLeNet, and VGG-16 that 8-bit floating-point quantization (4-bit exponent and 3-bit mantissa) results in constant negligible accuracy degradation. Similarly, in [23], Sean O. Settle et al. presented an 8-bit FP quantization scheme, which needs an extra inference batch to compensate for quantization errors. However, [22] and [23] did not present a hardware architecture.

In [17], Xiaocong Lian et al. proposed a hardware accelerator with optimized block floating-point (BFP). In this design the activations and weights are represented by 16-bit and 8-bit FP formats, respectively. This design is demonstrated on Xilinx VC709 evaluation board. This implementation achieves throughput and power efficiency of 760.83 GOP/s and 82.88 GOP/s/W, respectively. However, this design is not suitable for low-power resource-constrained embedded FPGAs.

B. LOW-PRECISION FLOATING-POINT

In [16], Chunsheng Mei et al. presented a hardware accelerator for VGG16 model using half-precision FP (16-bit). This design is demonstrated on Xilinx Virtex-7 (XC7VX690T) with PCIe interface. This implementation achieves throughput and power efficiency of 202.8 GFLOP/s and 18.72 GFLOP/s/W, respectively. In [13], Chen Wu et al. proposed a low-precision (8-bit) floating-point (LPFP) quantization method for FPGA-based acceleration. This design is demonstrated on Xilinx Kintex 7 and Ultrascale/Ultrascale+. This implementation achieves throughput and power efficiency of 1086.8 GOP/s and 115.4 GOP/s/W, respectively.

C. LOW-POWER

Two research papers have been reported hardware accelerators targeting XC7Z007S. This is the smallest and most inexpensive device from Zynq-7000 SoC family. In [24], Paolo Meloni et al. presented a CNN inference accelerator for compact and cost-optimized devices. This implementation uses fixed-point to process light-weight CNN architectures with a power efficiency between 2.49 to 2.98 GOPS/s/W. In [25], Chang Gao et al. presented EdgeDRNN, a recurrent neural network (RNN) accelerator for edge inference. This implementation adopts the spiking neural network (SNN) inspired delta network algorithm to exploit temporal sparsity in RNNs.

III. BACKGROUND

A. CONV2D TENSOR OPERATION

A convolutional layer aims to learn and extract feature representations from input layers. The convolution layer is made of convolution kernels that are used to compute feature maps. Each unit of a feature map is connected to a region of neighboring units on the input maps (from previous layer). Such a neighborhood of the previous layer is known as the receptive field of the unit. A new feature map can be obtained by first convolving the input maps with a learned kernel and then applying a nonlinear elementwise activation function to the convolved results. All spatial locations on the input maps share a kernel to generate a feature map. All feature maps are obtained by convolving several different kernels [26].

The 2D convolution is performed by the *Conv2D* tensor operation described in **Eq.** (1), where h is the input tensor containing the feature maps, W is the convolution kernels (known as filters), and b is the bias vector used on the output feature maps [27]. $K \times L \times M$ is the receptive field size, $K \times L$ is the 2D convolution kernel, and M is the number of input channels or input feature maps. We denote *Conv* as *Conv2D* operator.

$$Conv\left(W,h\right)_{i,j,o} = \sum_{k,l,m}^{K,L,M} h_{(i+k,j+l,m)} W_{(o,k,l,m)} + b_o \quad (1)$$

B. FLOATING-POINT NUMBER REPRESENTATION

The representation of every numerical value, in any number system, is made of an integer and a fractional part. The border that delimits them is called the radix point. The fixed-point format derives its name from the fact that in this, the base point is fixed at a certain position. For integer numbers, this position is at the right of the least significant digit.

In scientific computation, it is often necessary to represent very large and very small values. This is difficult to achieve using the fixed-point format because the bit size/width required to maintain both the desired precision and the desired range are very large. In such situations, FP formats are used to represent real numbers. Each FP number can be divided into three fields: sign S, exponent E, and mantissa M. Using the binary number system, it is possible to represent any FP number as:

$$(-1)^S \times 1.M \times 2^{E-B} \quad (2)$$

In FP representations the exponent is biased. This bias depends on the bit size of the exponent field in the particular format. This exponent bias is defined by **Eq.** (3), where E_{size} is the exponent bit size.

$$B = 2^{E_{size}-1} - 1 \quad (3)$$

There is a natural trade-off between small bit size requiring fewer hardware resources and larger bit size providing higher precision. Within a given total bit size, it is possible to assign

various combinations of sizes to the exponent and mantissa fields, with wider exponents resulting in a higher range and wider mantissa resulting in better precision.

The most widely used format for FP arithmetic is the IEEE 754 standard [28]. The IEEE single-precision format (32-bit) is expressed by **Eq.** (2) with 8-bits for the exponent, 23-bits for the mantissa, and $B = 127$, see **Fig.** 2(a). In FP formats, the numbers are normalized, the leading one is an implicit bit, and only the fractional part is explicitly stored in the mantissa field.

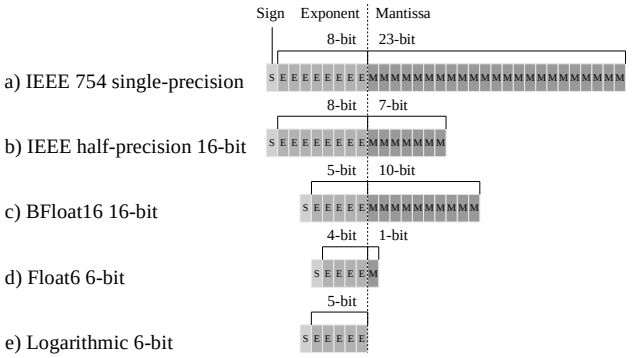


FIGURE 2. Floating-point number representations.

Reduced bit size than those specified in the IEEE 754 standard are often sufficient to provide the desired precision. Reduced designs require fewer hardware resources enabling low-power implementations. In custom hardware designs, it is possible to customize the FP format implemented. In later sections, we use the term EaMb to denote FP formats, where a and b are the exponent and mantissa bit size, respectively. For example, E4M1 means 4-bit exponent and 1-bit mantissa, see **Fig.** 2(d).

There are three special definitions in IEEE 754 standard. The first is subnormal numbers when $E = 0$, then **Eq.** (2) is modified to **Eq.** (4). Infinity and not a number (NaN) are the other two special cases but are not used in our work.

$$(-1)^S \times 0.M \times 2^{1-B} \tag{4}$$

IV. SYSTEM DESIGN

The system design is a hardware/software co-design framework for low-power ML analytics. This architecture allows design exploration for dedicated hardware in embedded systems. For ML compatibility, the proposed framework integrates TensorFlow Lite micro.

A. BASE EMBEDDED SYSTEM ARCHITECTURE

The embedded system architecture consists of a cooperative hardware-software platform. See **Fig.** 3. The embedded CPU delegates low-level compute-bound tensor operations to the TPs. The TPs employ AXI-Lite interface for configuration and AXI-Stream interfaces via Direct Memory Access (DMA) for data movement from of-chip memory. Each

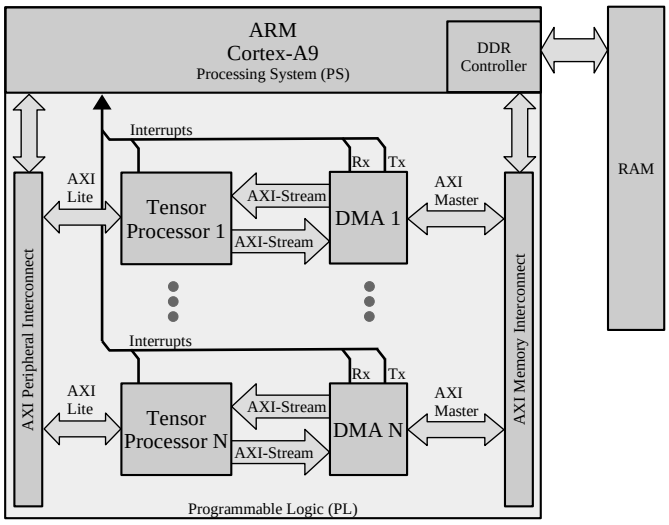


FIGURE 3. Base embedded system architecture.

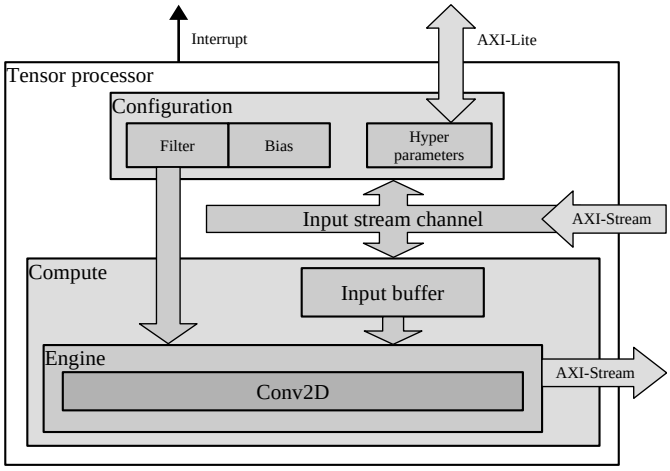


FIGURE 4. High level hardware architecture of the proposed tensor processor.

TP and DMA pair asserts interrupt flags once its compute job/transaction completes. Interrupt events are handled by the embedded CPU to use the results and to start a new compute job/transaction. The hardware architecture can vary its resource utilization by customizing the TPs prior to the hardware synthesis.

B. TENSOR PROCESSOR

The TP is a dedicated hardware module to compute tensor operations. This implements high performance communication with AXI-Stream, direct CPU communication with AXI-Lite, and on-chip storage utilizing BRAM. This hardware architecture is implemented with high-level synthesis (HLS). The tensor operations are implemented based on the C++ TensorFlow Lite micro kernels. See **Fig.** 4. In this paper, we focus on the *Conv2D* tensor operation that computes 2D convolution layers.

1) Modes of operation

The TP has two modes of operation: *configuration* and *execution*.

- In *configuration* mode, the TP receives the hyperparameters of the tensor operation: stride, dilation, padding, offset, activation, depth-multiplier, input shape, filter shape, bias shape, and output shape. Afterwards in the same data stream, the TP receives filter and bias tensors. These are locally stored in BRAM for data re-usage. The filter and bias tensors are transferred using standard FP format wrapping the 6-bit FP representation, which is extracted by the TP for local on-chip storage.
- In *execution* mode, the TP executes the tensor operation according to the hyperparameters given in the configuration mode. During execution, the input and output tensors are moved via DMA.

2) Dot-product with hybrid floating-point

We implement the floating-point computation adopting the dot-product with hybrid custom floating-point [29]. The hardware dot-product is illustrated in **Fig.** 5 and **Fig.** 6(a). This design instantiates an HF6 MAC and an accumulator register of 64-bit fixed-point with 23-bit fraction. During operation, the feature map and filter values are extracted from on-chip memory (BRAM). Both values have to be different than zero to enable the MAC operation. The result is biased by accumulating a denormalized bias value. Since the bias is stored with 6-bit FP, its fractional part has to be aligned with the 23-bit fraction of the accumulator, see **Fig.** 6(b). The ReLu activation is applied and the result is normalized to convert it to IEEE 754 standard FP, see **Fig.** 6(c).

Rather than a parallelized hardware structure, this is a pipelined hardware design suitable for resource-limited devices. The latency in clock cycles of this hardware module is defined by **Eq.** (5), where N is the length for the vector dot-product. This latency equation is obtained from the general pipelined hardware latency formula: $L = (N-1)\,II + IL$, where II is the initiation interval, and IL is the iteration latency. Both II and IL are obtained from the high-level synthesis results. Both the exponent and mantissa bit widths of the filter and bias are set to 4-bit exponent and 1-bit mantissa (E4M1), which corresponds to float6 quantization.

$$L_{hf} = N + 7 \tag{5}$$

3) Multiply-Accumulate

The multiply-accumulate operation calculates the product of two numbers and adds the result to an accumulator register. In FP arithmetics, the area of a hardware multiplier scales with the bit size of the mantissas. In the case of HF6, the 6-bit FP representation allows a reduced hardware multiplicator for mantissas. The 1-bit mantissa enables optimized MAC implementations by reducing the mantissa multiplication to a multiplexed addition, see **Fig.** 7. This MAC produces

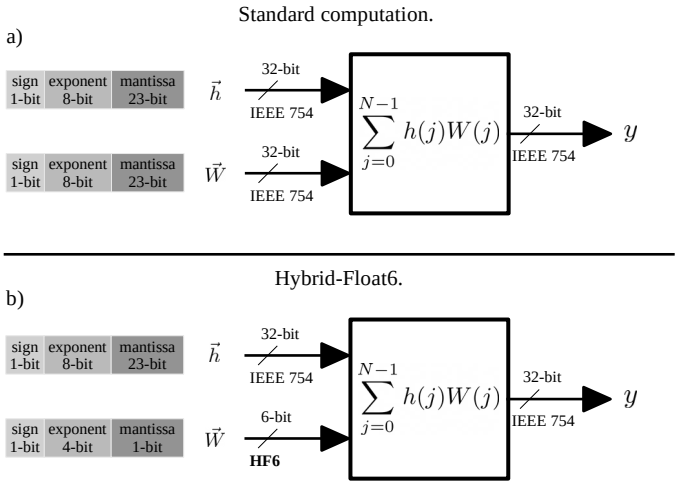


FIGURE 5. Dot-product hardware module with (a) standard floating-point and (b) Hybrid-Float6.

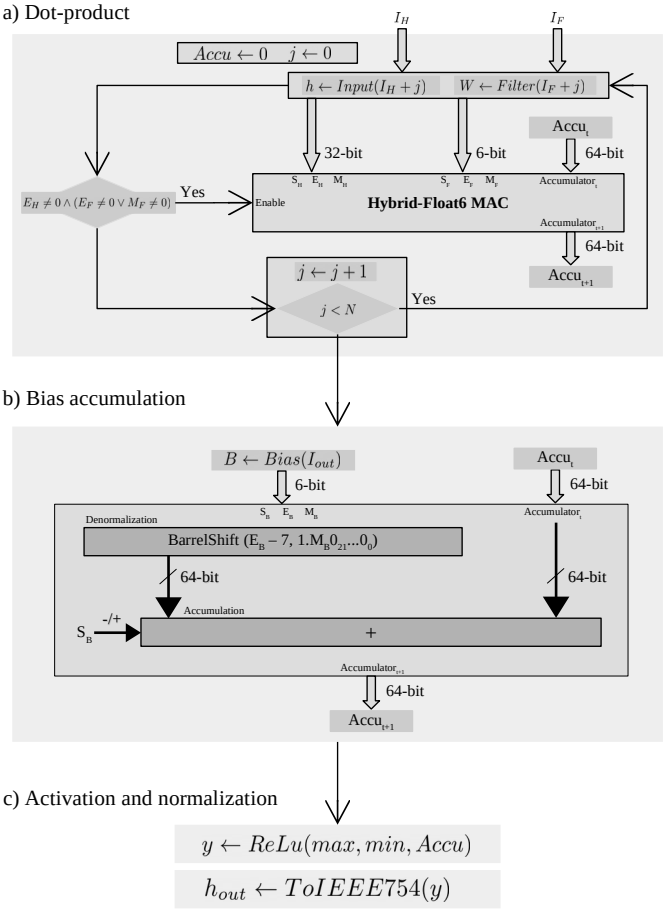


FIGURE 6. (a) Dot-product hardware module with Hybrid-Float6 MAC, (b) bias accumulation, (c) activation and normalization to IEEE754.

denormalized results, which are accumulated in a fixed-point accumulator. This approach reduces latency, energy consumption, and hardware area/resource utilization.

Special cases, such as Infinity and NaN, are not considered

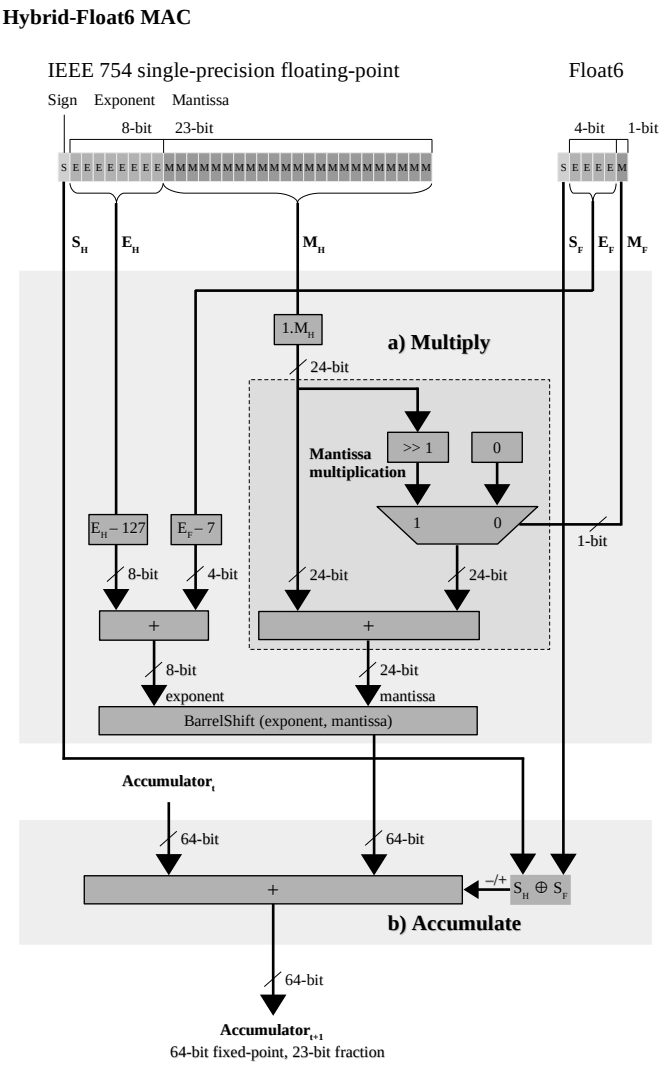


FIGURE 7. Hybrid-Float6 multiply-accumulate hardware design.

in this design for simplicity, since they are not expected for CNN inference. For the subnormal case, the element-wise multiplication is disabled when having a zero entry and is approximated when having subnormal mantissa. The feature map values are considered zero when the exponent is zero ($E_H = 0$). The filter values are considered zero when both exponent and mantissa are zero ($E_F = 0 \wedge M_F = 0$). See **Fig.** 6(a). In the 6-bit FP, the 1-bit mantissa has one subnormal case, which is handled as a normalized case. This exploits the intrinsic error tolerance to reduce the hardware design.

The approximation error is defined by the difference between **Eq.** (2) and **Eq.** (4) when $E = 0$ and $M = 2^{-1}$. The result defines the error as $e = 2^{-B-1}$. Then, from **Eq.** (3) with $E_{size} = 4$, we have $B = 7$. Hence, $e = 3.9\text{e}{-3}$. This error is produced when having the subnormal case $E = 0$ and $M = 2^{-1}$, which corresponds to the value $\pm 7.8\text{e}{-3}$ deviated to $\pm 1.17\text{e}{-2}$. This approximation leverages the intrinsic error tolerance of CNN to reduce hardware resource

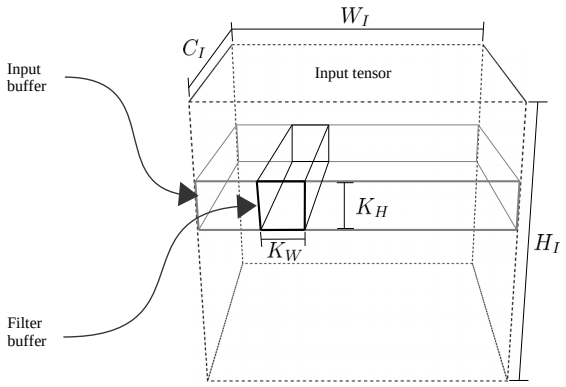


FIGURE 8. Design parameters for on-chip memory buffers on the TP.

utilization and energy consumption [11].

4) On-chip memory utilization

The total on-chip memory utilization on the TP is defined by **Eq.** (6), where TP_B and V_M represent the tensor buffers required for *Conv* operation and local registers required for the logic, respectively. **Eq.** (7) defines the tensor buffers, where $Input_M$ is the *input buffer*, $Filter_M$ is the *filter buffer*, $Bias_M$ is the *bias buffer*. These on-chip memory buffers are defined in bits. **Fig.** 8 illustrates the convolution operation utilizing the on-chip memory buffers.

$$TP_M = TP_B + V_M \tag{6}$$

$$TP_B = Input_M + Filter_M + Bias_M \tag{7}$$

The memory utilization of *input buffer* is defined by **Eq.** (8), where K_H is the height of the convolution kernel, W_I is the width of the input tensor (input feature maps), C_I is the number of input channels, and $BitSize_I$ is the bit size representation used by the input tensor.

$$Input_M = K_H W_I C_I BitSize_I \tag{8}$$

The memory utilization of *filter buffer* is defined by **Eq.** (9), where K_W and K_H are the width and height of the convolution kernel, respectively; C_I and C_O are the number of input and output channels, respectively; and $BitSize_F$ is the bit size representation used by filter values.

$$Filter_M = C_I K_W K_H C_O BitSize_F \tag{9}$$

The memory utilization of *bias buffer* is defined by **Eq.** (10), where C_O is the number of output channels, and $BitSize_B$ is the bit size representation used by bias values.

$$Bias_M = C_O BitSize_B \tag{10}$$

As a design trade-off, **Eq.** (11) defines the capacity of output channels based on given design parameters. The total on-chip memory TP_M determines the TP storage capacity.

$$C_O = \frac{TP_M - V_M - K_H W_I C_I BitSize_I}{C_I K_W K_H BitSize_F + BitSize_B} \tag{11}$$

The floating-point formats implemented in the TP are defined by $BitSize_F$, $BitSize_B$ and $BitSize_I$. The HF6 defines 6-bit for $BitSize_F$ and $BitSize_B$, and 32-bit for $BitSize_I$. These are design parameters defined before hardware synthesis. This allows fine control of BRAM utilization, which is suitable for resource-limited devices.

C. TRAINING METHOD

The training method consists of two separate stages: (1) training with iterative early stop and (2) quantization-aware training.

1) Training with Iterative Early Stop

To achieve better performance on CNN-regression models, we implement a training procedure with an iterative early stop cycle. This allows to reach better local minima. This process consists of four steps:

1) A model is obtained with an initial training with standard early stop monitoring.
2) The model is iteratively re-trained with standard early stop. This process iteratively restarts the optimizer moving averages to search for better local minima.
3) In case of a better local minimum, the model is saved and used as a base for subsequent search iterations, otherwise it is a discarded search.
4) The cyclic process stops automatically with a given number of searches with no better local minimum, this is denoted as stop patience. This allows to set a maximum number of unsuccessful search trials before the stop.

This method is described in **Algorithm** 1.

2) Quantization-Aware Training

The quantization-aware training (QAT) method is integrated into the training process, this operates as a callback on each mini-batch update. The quantization is applied on the trainable parameters of convolution layers. This method is implemented on the ML framework (TensorFlow/Keras), see **Algorithm** 2.

The quantization method uses rounding strategy to reduce the FP representation. This maps the full precision FP values to the closest representable 6-bit FP values, see **Algorithm** 3. This method quantizes the filter and bias tensors of the convolution layers. We have observed that the exponent bit size plays a more predominant influence on the model accuracy than the mantissa bit size. In [22], Lai et al. demonstrated that 4-bit exponent and X-bit mantissa is adequate and consistent across different networks (SqueezeNet, AlexNet, GoogLeNet, VGG-16). In this work, we investigate 4-bit exponent and 1-bit mantissa.

D. EMBEDDED SOFTWARE ARCHITECTURE

The software architecture is a layered object-oriented application framework written in C++, see **Fig.** 9. Description of the software layers is as follows:

Algorithm 1: Training with iterative early stop cycle.

input: $MODEL$ as the input model.
input: D_{train} as the training data set.
input: D_{val} as the validation data set.
input: N_I as the stop patience for iterative training cycle.
input: N_E as the early stop patience (epochs) for training.
input: B_{size} as the mini-batch size.
output: $MODEL$ as the full-precision output model.

```
Train(MODEL, D_train, D_val, N_E, B_size)
mse_i ← Evaluate(MODEL, D_val) // Benchmark
n_I ← 0
while n_I < N_I do
  // Iterative early stop cycle
  Train(MODEL, D_train, D_val, N_E, B_size)
  mse_v ← Evaluate(MODEL, D_val)
  if mse_v < mse_i then
    Update(MODEL)
    mse_i ← mse_v
  else
    MODEL ← LoadPreviousWeights()
    n_I ← n_I + 1
  end if
end while
```

Algorithm 2: OnMiniBatchUpdate_Callback.

input: $MODEL$ as the full-precision input model.
input: E_{size} as the target exponent bits size.
input: M_{size} as the target mantissa bits size.
input: D_{train} as the training data set.
input: D_{val} as the validation data set.
input: N_{ep} as the number of epochs.
input: B_{size} as the mini-batch size.
output: $MODEL$ as the quantized output model.

```
// Quantize
MODEL ← Algorithm 3(MODEL, E_size, M_size)
if 1 < epoch then
  // Update model after first epoch
  mse_v ← Evaluate(MODEL, D_val)
  if mse_v < mse_i then
    Update(MODEL)
    mse_i ← mse_v
  end if
end if
```

- *Application*: As the highest level of abstraction, this software layer implements the analytics application, this invokes the ML library.
- *Machine learning library*: This software layer offers a comprehensive high level API for ML inference. This layer consist of TensorFlow Lite micro, this is modified to implement the delegate software interfaces for the proposed hardware accelerator.

Algorithm 3: Custom floating-point quantization.

input: $MODEL$ as the CNN.
input: E_{size} as the target exponent bit size.
input: M_{size} as the target mantissa bits size.
input: $STDM_{size}$ as the IEEE 754 mantissa bit size.
output: $MODEL$ as the quantized CNN.

```
for layer in MODEL do
  if layer is Conv2D or SeparableConv2D then
    filter, bias ← GetWeights(layer)
    for x in filter and bias do
      sign ← GetSign(x)
      exp ← GetExponent(x)
      fullexp ← 2^(E_size − 1) − 1 // Get full range value
      cman ← GetCustomMantissa(x, M_size)
      leftman ← GetLeftoverMantissa(x, M_size)
      if exp < −fullexp then
        x ← 0
      else if exp > fullexp then
        x ← (−1)^sign · 2^fullexp · (1 + (1 − 2^−Msize))
      else
        if 2^(STDM_size − M_size − 1) − 1 < leftman then
          cman ← cman + 1 // Above halfway
          if 2^M_size − 1 < cman then
            cman ← 0 // Correct mantissa overflow
            exp ← exp + 1
          end if
        end if
        // Build custom quantized floating-point value
        x ← (−1)^sign · 2^exp · (1 + cman · 2^−M_size)
      end if
    end for
    SetWeights(layer, filter, bias)
  end if
end for
```

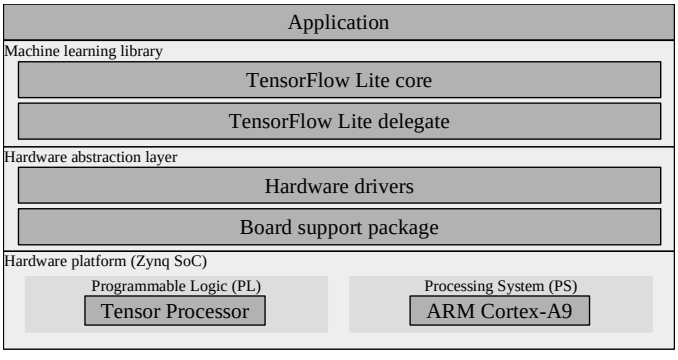


FIGURE 9. High level embedded software architecture.

- *Hardware abstraction layer*: This layer consist of the hardware drivers used in the TFLite delegate software interfaces. This software layer handles initialization and runtime operation of the TP and DMA.

V. EXPERIMENTAL RESULTS

In this section, we present experimental results using a low-power/low-cost sensor analytics application. As a use case, we present a CNN-regression model to predict x- y- coordinates of acoustic emissions based on piezoelectric vibrations. We compare quantitative and qualitative aspects of the analytics using floating-point 32-bit, fixed-point 8-bit, Hybrid-Logarithmic 6-bit, and Hybrid-Float6.

To demonstrate the proposed concept, we deploy the CNN model in the smallest Zynq SoC device for low-power inference. We compare the performance of the TP synthesized with standard FP (using Xilinx LogiCORE IPs) and Hybrid-Float6 design.

A. SENSOR ANALYTICS APPLICATION

The analytics model is designed to predict x- y- coordinates of acoustic emissions on a metal plate. The metal plate is in the presence of noise disturbance to simulate realistic conditions. In this subsection, we present the structure for experimental setup, data sets, and the CNN-regression model.

1) Experimental Setup

The experiment uses eight piezoelectric sensors (Vallen Systeme VS900) attached with magnetic holders on a metal plate (90 cm x 86.6 cm x 0.3 cm). The VS900 devices can operate either in active or passive mode. Six VS900 are used in passive mode as acoustic sensors and two in active mode to produce acoustic emissions. These acoustic emissions simulate anomalies on x- y- coordinates as well as the noise disturbance on the system. See **Fig.** 10(a). To create data sets, the samples of acoustic emissions are labeled with their coordinates.

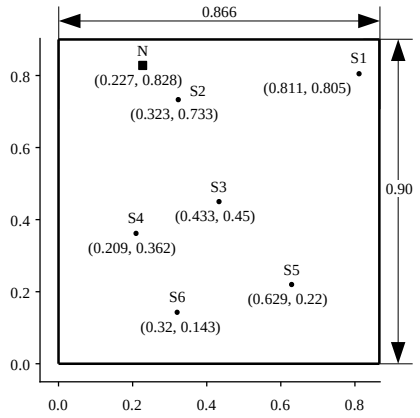
2) Data Sets

The data sets are recorded applying pulses on the metal plate, the x- y- coordinates of these pulses are used as labels. The pulses for training and validation data sets are shown in **Fig.** 10(b) and **Fig.** 10(c), respectively. The pulses for training and validation data sets are mutually exclusive, this exclusion is represented by the cross symbols in **Fig.** 10(c). This creates a grid layout used to collect samples for the data sets. This grid is 10×10. This grid does not consider the four corners as they are used for magnetic holders.

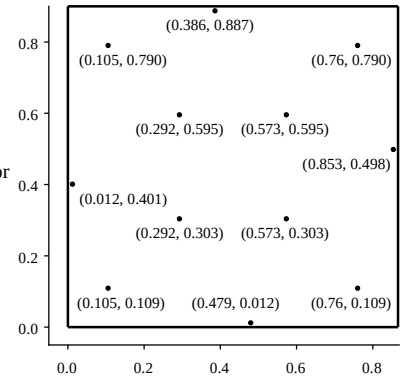
In order to create reproducible acoustic emissions, we use 9-cycle sine pulse in a Hanning window with central frequency f_c (narrow-banded in the frequency domain). We assume guided Lamb waves based on the plate structure. The narrow-band behavior also reduces the dispersion of the acoustic emission waves [30]. The waveform can be expressed as a function of time t as follows:

$$x_{\text{pulse}}(t) = \frac{1}{2}\left(1 - \cos\frac{f_c t}{5}\right) A_0 \sin f_c t. \qquad (12)$$

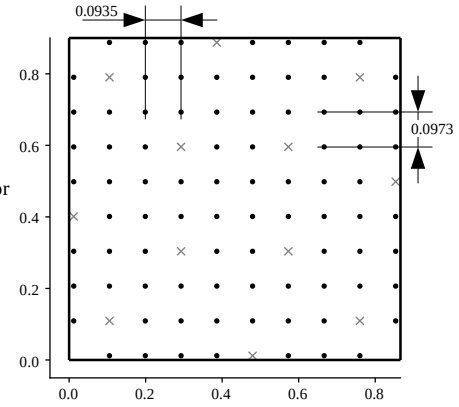
To generate the data sets, we use slightly different pulse amplitudes and frequencies for excitation. The pulse frequency f_c is varied in 1 kHz steps between 300 kHz and



a) Sensor and noise positions



b) Labeled pulses for training data set



c) Labeled pulses for validation data set

FIGURE 10. Experimental setup for sensor analytics on structural health monitoring, all lengths are in meters (m).

349 kHz and the amplitude A_0 is varied in 0.1 V steps between 2.6 V and 3.5 V. This results in 500 different pulses for each of the excitation points.

The signals for labeled pulses and noise disturbance are generated by arbitrary waveform generators (AWGs). The sensor signals are recorded via a Vallen AMSY-6 measurement system with a resolution of 18 bits and a sampling rate of $f_S = 10\,\text{MHz}$. The disturbance signal is gaussian noise with amplitudes between 0-3 V. This noise is applied via the piezoelectric device N at $x = 0.227\,\text{m}$ and $y = 0.828\,\text{m}$, see **Fig.** 10(a).

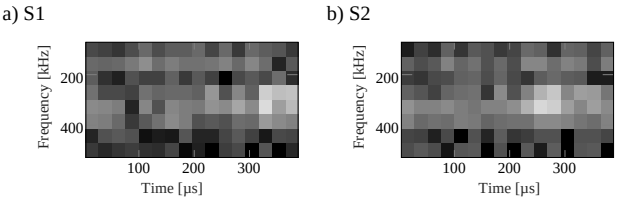


FIGURE 11. Spectrograms of sensors S_1, S_2 converted to grayscale for pulses at $x = 0.105$ m, $y = 0.109$ m with noise disturbance.

To obtain frequency components, the sampled pulses are converted into the frequency-time domain using the Short-Time Fourier Transform (STFT). This is calculated as follows [31]:

$$\mathcal{F}^{\gamma}_{m,k} = \sum_{n=0}^{N-1} x[n] \cdot \gamma^*[n - m\Delta M] \cdot \mathrm{e}^{\frac{-j2\pi kn}{N}} \tag{13}$$

Here $x[n]$ describes a discrete-time signal and $\gamma^*[n - m\Delta M] \cdot \mathrm{e}^{\frac{-j2\pi kn}{N}}$ the time- and frequency-shifted window function inside the considered interval $[0, N-1]$. ΔM describes the time shift and N the transformation window. Since only discrete frequencies and time points are considered, $m = 0, 1, ..., M-1$ is valid. For pictorial representation, the magnitude of the complex-valued STFT is employed in a spectrogram $\mathcal{S}_{m,k}$:

$$\mathcal{S}_{m,k} = \left|\mathcal{F}^{\gamma}_{m,k}\right|^2 = \left|\sum_{n=0}^{N-1} x[n] \cdot \gamma^*[n - m\Delta M] \cdot \mathrm{e}^{\frac{-j2\pi kn}{N}}\right|^2 \tag{14}$$

In addition, these spectrograms are scaled in decibels. The spectrogram in decibels $\mathcal{S}_{m,k,\text{dB}}$ results in $\mathcal{S}_{m,k,\text{dB}} = 20 \cdot \log_{10}(\mathcal{S}_{m,k})$. For the conversion of the data, we use a signal length of 400 µs (75 µs pretrigger and 325 µs post trigger). Thus, the arrival times of the pulses are included in the spectrogram for all channels and labeled positions. We use a Blackman window function [32], a Fast Fourier Transform (FFT) length of 32 samples, and an overlap of 8 samples. The spectrograms are calculated for frequencies in the range of 100 kHz to 500 kHz. This results in a spectrogram size of 8x16 (8 frequency bins, 16 time values).

In order to generate larger data sets, we create four further variants with time shifts of 15 µs/ 30 µs/ 45 µs/ 60 µs. Subsequently, all spectrograms are converted to grayscale with scaling between -100dB and -40dB, see **Fig.** 11.

In overall, the data set has a size of 1,440,000 images. This is the result of 500 (pulses) · 5 (spectrograms) · 6 (listening sensors) · 96 (excitation points).

3) CNN-Regression Model

The data analytics is implemented with a CNN-regression model, see **Fig.** 12. The structure of the model is described below:

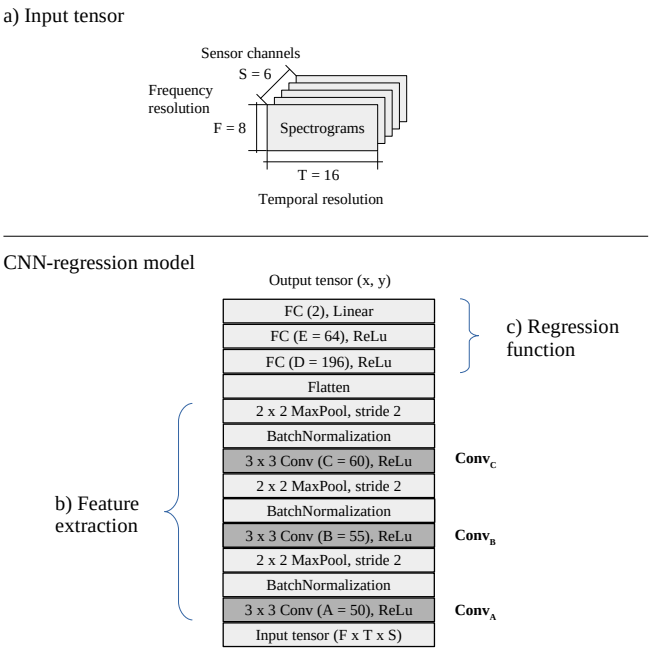


FIGURE 12. CNN-regression model for sensor analytics.

a) Input tensor. This is composed of spectrograms from the sensor signals. The tensor shape is defined by $S \times T \times F$, where S is the number of sensors, and $T \times F$ is the time-frequency resolution of the spectrograms, see **Fig.** 12(a).

b) Feature extraction. This is composed of three blocks of convolution, batch normalization, and max-pooling layers, see **Fig.** 12(b). The number of channels in the convolution layers are defined by the hyper-parameters A, B, and C.

c) Regression function. This is an arbitrary function implemented with two fully connected layers and an output layer with linear activation, see **Fig.** 12(c).

B. TRAINING

1) Base Model

The model in **Fig.** 12 is trained using Adam algorithm with iterative search. The Adam optimizer is configured with the default settings presented in [33]: $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\epsilon = 1\text{e}{-8}$. The training cycle patience is 10 iterations, the optimizer is executed with early stop patience of 10 epochs, and mini-batch size of 512 samples. This is applied using the method described in **Algorithm** 1 with $N_I = 10$, $N_E = 10$, $B_{size} = 512$.

The training results are illustrated in **Fig.** 13(a). In this optimization, the initial and the final models achieve $MSE = 0.0135\,\text{m}^2$ and $MSE = 0.0122\,\text{m}^2$, respectively. The MSE is calculated with the Euclidean distance (loss) between the expected and the predicted coordinates. The initial model is obtained at the first early stop (after 10 epochs). In each stop, the moving averages of the Adam optimizer get re-initialized. This facilitates searching for better local minima. The model gets saved/updated when finding a better minimum.

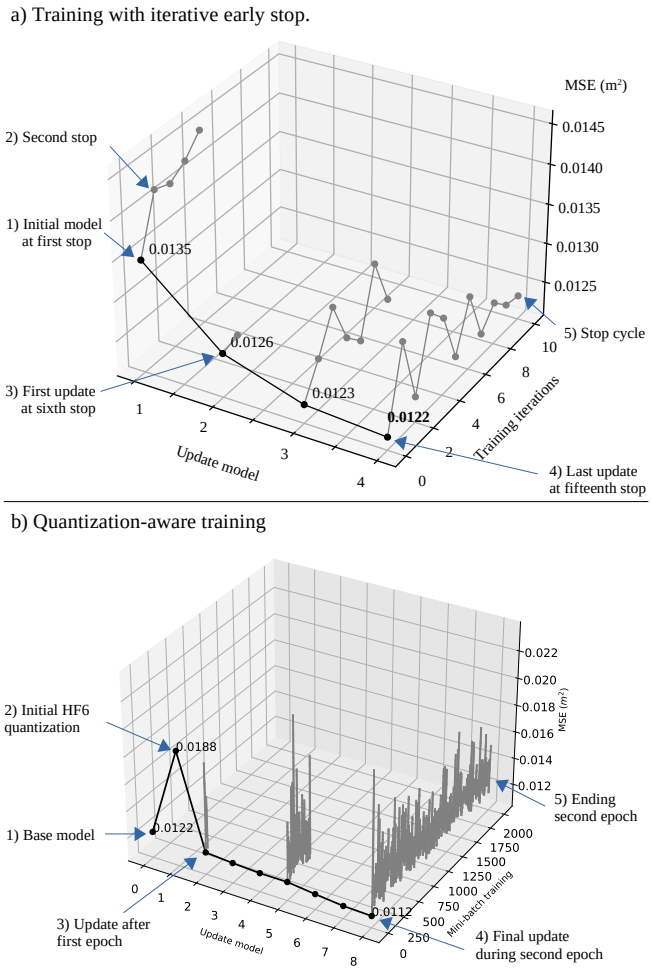


FIGURE 13. Training results.

The final model achieves $MSE = 0.0122\,\text{m}^2$, which corresponds to $MAE = 0.0955\,\text{m}$. See **Fig.** 14(a). In total, the training takes 379 epochs in 25 cycle-search iterations. The first search takes 43 epochs for the initial model and subsequent search iterations take an average of 14 epochs. The total time is 53 minutes using a PC with AMD Ryzen 5 5600H and NVIDIA GeForce RTX 3050.

2) TensorFlow Lite 8-bit Quantization

This optimization method converts filter and bias tensors as well as activation maps to 8-bit integer representation, this allows inference using integer-only arithmetic [30]. In this research, this quantization is applied only to the convolution layers as they are the compute bound operations. Other layers employ 32-bit FP representation.

In the compute graph, the imput and output feature maps are glued with linear quantization at the input and output of the *Conv2D* operations.

The base model is quantized using the TensorFlow Lite library with integer-only quantization on the *Conv2D* tensor

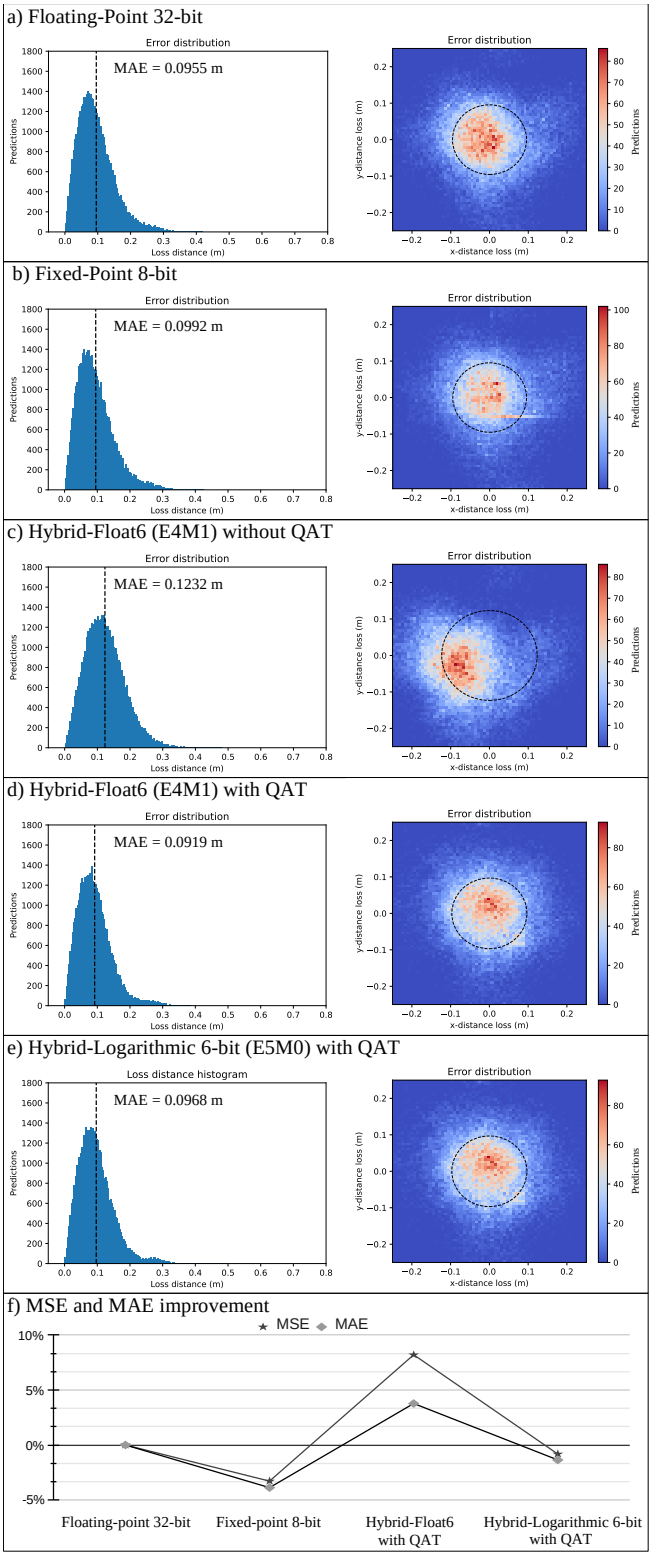


FIGURE 14. Performance of the model with different data representations.

operations. The filter and bias tensors are represented by 8-bit and 32-bit signed integers, respectively. The input and output activation maps are represented by 8-bit signed integer.

The TensorFlow quantization includes two additional vectors (output-multiplier and output-shift coefficients), these two vectors are the same shape as the bias vector with 32-bit integer representation.

This model achieves $MSE = 0.0126\,\mathrm{m}^2$ and $MAE = 0.0992\,\mathrm{m}$. See **Fig.** 14(b). The MAE increases 5.1% of the base model. We attribute this degradation to the 8-bit quantization on the *Conv2D* layers.

3) Inference of non-quantized models on HF6 hardware

We explore the inference of the base model without quantization on the HF6 hardware. See **Fig.** 14(c). This obtains $MSE = 0.0188\,\mathrm{m}^2$ and $MAE = 0.1232\,\mathrm{m}$. The MAE increases 29.5% of the base model. We attribute this degradation to the rounding errors of non-quantized filters and bias in *Conv2D* layers.

4) Quantization-Aware Training for HF6 hardware

The QAT is a post-training optimization. We run the QAT during two epochs with mini-batch size of 10 samples. This a custom floating-point quantization targeting the HF6 format: 4-bit exponent and 1-bit mantissa. This is applied to filter and bias tensors of *Conv2D* layers. This method is described in **Algorithm** 2 with $N_{ep} = 2$, $B_{size} = 10$, $E_{size} = 4$, $M_{size} = 1$. The optimization results are illustrated in **Fig.** 13(b).

The resulting model achieves $MSE = 0.0112\,\mathrm{m}^2$ and $MAE = 0.0919\,\mathrm{m}$. This corresponds to an error reduction of 8.2% and 3.77%, respectively. We attribute this improvement to the regularization effect. See **Fig.** 14(d). The QAT time is 185 minutes.

5) Quantization-Aware Training for Hybrid-Logarithmic 6-bit

For the sake of quality comparison with logarithmic quantization, we generate the model with 6-bit logarithmic representation on trainable parameters of convolution layers. See **Fig.** 2(e). This quantization matches the bit size of HF6. The filter and bias tensors of *Conv2D* layers are quantized with the 6-bit logarithmic format: 1-bit sign, 5-bit signed exponent, and 0-bit mantissa. This is applied using the method described in **Algorithm** 2 with $N_{ep} = 2$, $B_{size} = 10$, $E_{size} = 5$, $M_{size} = 0$.

The model achieves $MSE = 0.0123\,\mathrm{m}^2$ and $MAE = 0.0968\,\mathrm{m}$, which correspond to an error increase of 0.82% and 1.36%, respectively. We attribute this degradation to the 6-bit logarithmic quantization. See **Fig.** 14(e).

A summary of improvement-degradation of MSE and MAE with different data representations is presented in **Fig.** 14(f).

C. HARDWARE DESIGN EXPLORATION

The proposed hardware/software co-design is demonstrated on the Zynq-7007S system-on-chip (SoC) on the MiniZed development board. This SoC integrates a single ARM Cortex-A9 processing system (PS) and a programmable logic (PL) equivalent to Xilinx Artix-7 (FPGA) in a single chip

[34]. The Zynq-7007S SoC architecture maps the custom logic and software in the PL and PS, respectively.

In this platform, we implement the proposed hardware/software architecture to deploy the sensor analytics application. The desired model is converted to TensorFlow Lite (floating-point) and deployed on the embedded software as a hex dump in a C array. The Zynq-7007S SoC executes inference with TensorFlow Lite on the PS. The computational workload of convolution layers is delegated to the dedicated hardware.

1) Benchmark on Embedded CPU

First, we explore the performance of the embedded CPU for inference without hardware acceleration. In this case, TensorFlow Lite creates the CNN model as a sequential compute graph executing all computation on the CPU (ARM Cortex-A9) at 666 MHz and power dissipation of $1,187$ W.

The compute performance and run-time inference of the CPU are shown in **Tab.** 2(a) and **Fig.** 16(a), respectively.

2) Benchmark on Tensor Processor Synthesized with Xilinx LogiCORE IP for Floating-Point Computation

For this design, we implement the TP with standard FP hardware prior synthesis. The design parameters for the maximum required accelerator on-chip size are:

- Max convolution kernel size: $K_W = K_H = 3$.
- Max input tensor width: $W_I = 16$.
- Max input and output channels: $C_I = 55$, $C_O = 60$.
- Filter and bias bit size: $BitSize_F = BitSize_B = 32$.
- Input tensor bit size: $BitSize_I = 32$.

Using equations from Section IV-B4, the on-chip memory buffer utilization are $Input_M = 84,480$b, $Filter_M = 950,400$b, and $Bias_M = 1,920$b. Hence, the required on-chip memory buffer size is $TP_B = 1,036,800$b.

The post-implementation resource utilization and power dissipation are presented in **Tab.** 1(a). The complete hardware platform utilizes 83% of BRAM, this includes the on-chip memory requirements of the TP, DMA, and AXI interconnects. The total available on-chip memory (BRAM) on the Zynq-7007S SoC is 1.8 Mb. After hardware syntheses, the estimated power dissipation of the TP is 85 mW at 200 MHz (this estimation is provided by Xilinx Vivado).

TABLE 1. Resource utilization and power dissipation on the Zynq-7007S SoC.

TP engine	Post-implementation resource utilization				Power (W)
	LUT	FF	DSP	BRAM 36Kb	
(a) Floating-Point	5,578 39%	8,942 31%	23 35%	41.5 **83%**	1.429
(b) Hybrid-Float6	7,313 51%	10,330 36%	20 30%	15 **30%**	1.424

The compute performance and inference schedule of the model on this hardware implementation are shown in **Tab.** 2(b) and **Fig.** 16(b), respectively. During run-time, TensorFlow Lite delegates computation to the TP as dedicated hardware for *Conv2D* tensor operations.

TABLE 2. Compute performance of the CPU and TP on each Conv2D tensor operation. This table presents: tensor operation, computational cost in mega floating-point operations (MFLOP), latency, throughput, power efficiency, and estimated energy consumption as the energy delay product (EDP).

Operation	MFLOP	t (ms)	MFLOP/s	MFLOP/s/W	EDP (mJ)
		a) CPU (ARM Cortex-A9) @666MHz, 1.187 W			
$Conv_A$	0.691	112.24	6.16	5.19	133.23
$Conv_B$	1.584	213.13	7.43	6.26	252.99
$Conv_C$	0.475	46.59	10.20	8.59	55.31
		b) TP (Floating-Point engine) @200MHz, 85 mW			
$Conv_A$	0.691	12.49	55.34	651.11	1.06
$Conv_B$	1.584	16.39	96.66	1,137.20	1.39
$Conv_C$	0.475	3.59	132.44	1,558.13	0.30
		c) TP (Hybrid-Float6 engine) @200MHz, 84 mW			
$Conv_A$	0.691	6.92	99.81	1,188.24	0.58
$Conv_B$	1.584	4.41	358.94	4,273.09	0.37
$Conv_C$	0.475	0.99	482.44	5,743.29	0.08

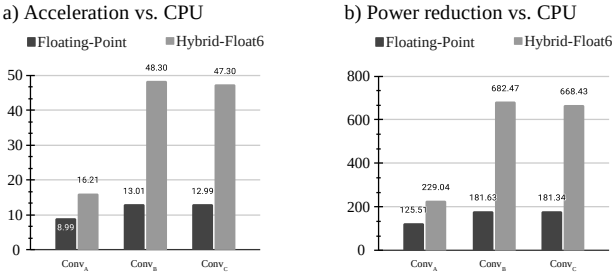


FIGURE 15. Inference acceleration and power reduction on the TP with floating-point and HF6 vs. CPU on the Zynq-7007S SoC.

The implementation of the dot-product with standard FP engine (IEEE 754 arithmetic) utilizes proprietary multiplier and adder FP operator cores. Vivado HLS implements FP arithmetic operations by mapping them onto Xilinx LogiCORE IP cores, these FP operator cores are instantiated in the resultant RTL [35]. In this case, the implementation of the dot-product with the standard FP computation reuses the multiplier and adder cores in different compute sections of the TP. The post-implementation resource utilization and power dissipation of the individual floating-point operator cores are shown in **Tab.** 3.

TABLE 3. Resource utilization and power dissipation of individual multiplier and adder floating-point (IEEE 754) operator cores (Xilinx LogiCORE IP).

Core operation	DSP	FF	LUT	Latency (clk)	Power (mW)
Multiplier	3	151	325	4	7
Adder	2	324	424	8	6

3) Tensor Processor Synthesized with Hybrid-Float6 Hardware Architecture

To demonstrate the proposed design, the TP with HF6 hardware reuses the standard FP design parameters with the following variation for the 6-bit representation in filter and bias: $BitSize_F = BitSize_B = 6$.

Using equations from Section IV-B4, the on-chip memory requirements for the hardware accelerator are $Input_M = 84,480$ b, $Filter_M = 178,200$ b, $Bias_M = 360$ b.

Hence, the required on-chip memory buffer size is $TP_B = 263,040\,\mathrm{b}$.

The post-implementation resource utilization and power dissipation are presented in **Tab.** 1(b). The complete hardware platform utilizes 30% of BRAM, this includes the on-chip memory requirements of the TP, DMA, and AXI interconnects. The estimated power dissipation of the TP is $84\,\mathrm{mW}$ at $200\,\mathrm{MHz}$ (this estimation is provided by Xilinx Vivado).

The compute performance and inference schedule of the model on this hardware implementation are shown in **Tab.** 2(c) and **Fig.** 16(c), respectively. **Fig.** 15 presents a comparison of the acceleration and the reduction of power dissipation between standard FP and HF6 hardware implementations.

This deployment does not require model treatment for hardware compatibility. For backward compatibility, the 6-bit FP representation is wrapped into the standard FP. The dedicated hardware design extracts the 6-bit format automatically to perform computation.

D. DISCUSSION

1) Training and Quantization

The training with iterative early stop obtains a model with enhanced accuracy than standard early stop. This method iteratively resets the moving averages of Adam's optimizer, which helps to iteratively search for better local minima. This iterative search is suitable for models with low computational cost.

The TensorFlow Lite 8-bit quantization preserves the overall model accuracy. In some cases, the associated regularization effect can improve the accuracy. However, the error distribution in CNN linear regressions gets slightly degraded. In particular, 8-bit quantized output layers incur in discrete-degradation patterns, **Fig.** 17(b) shows this effect on three different models. Vertical and horizontal patterns appear in the error distribution of 8-bit fixed-point quantization. We attribute this effect to the 8-bit resolution in the activation maps. In the case of HF6 quantization, the activation maps are represented by floating-point preventing this degradation.

The proposed 6-bit FP representation (E4M1) improves latency, hardware area, and power dissipation, while preserving model accuracy. For comparison, in our application, this number format produces better results than the 6-bit logarithmic representation (E5M0). This is demonstrated in **Fig.** 14(d) and **Fig.** 14(e).

In [22], Lai et al. demonstrated that 4-bit exponent and X-bit mantissa preserves accuracy on SqueezeNet, AlexNet, GoogLeNet, and VGG-16. To contribute on this, we investigate 4-bit exponent and 1-bit mantissa to ALL-CNN-C [36], this produces an accuracy degradation of 1.39% and 0.11% with QAT. While applying 6-bit logarithmic produces a degradation of 11.18% and 7.22% with QAT.

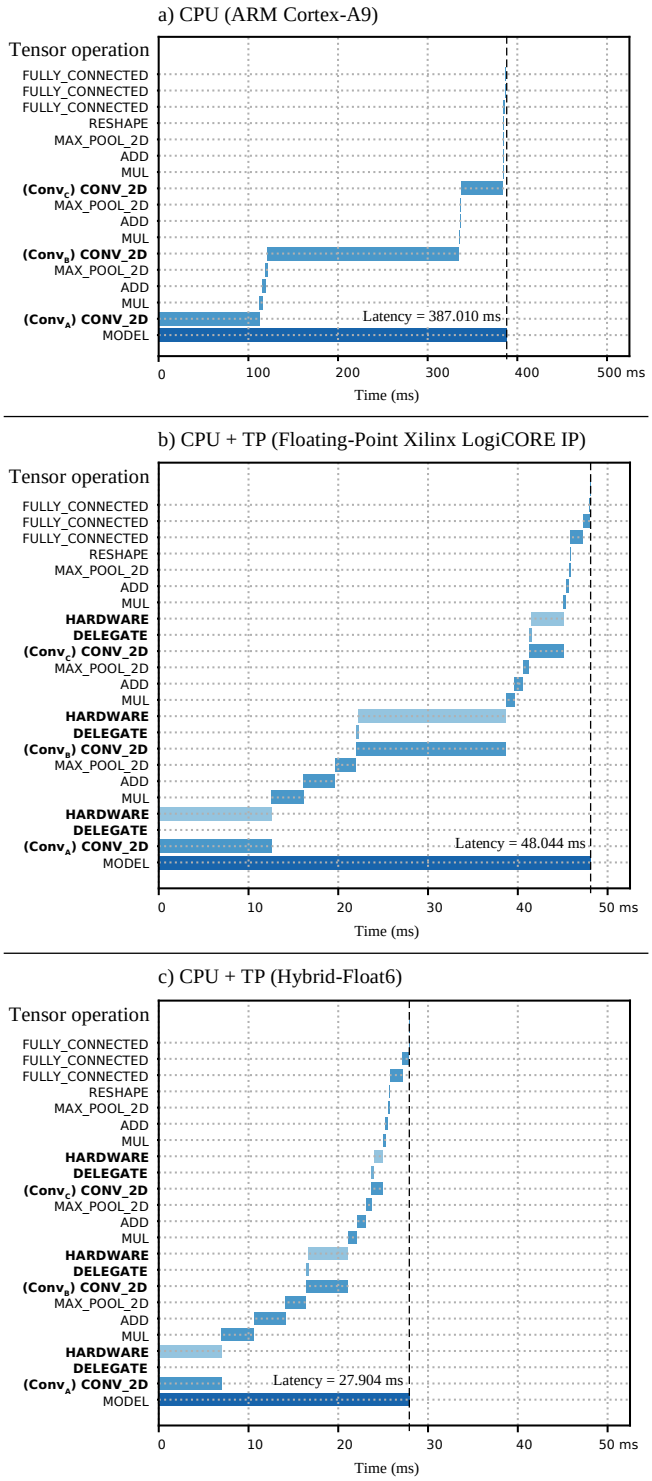


FIGURE 16. Run-time inference of TensorFlow Lite on the Zynq-7007S SoC. (a) CPU ARM Cortex-A9 at $666\,\mathrm{MHz}$, (b) cooperative CPU + TP with floating-point Xilinx LogiCORE IP at $200\,\mathrm{MHz}$, and (c) cooperative CPU + TP with Hybrid-Float6 at $200\,\mathrm{MHz}$.

2) Implementation and Performance

The proposed HF6 implementation reduces on-chip memory and DSP utilization while slightly increasing FF and LUT compared to the standard FP implementation. See **Tab.** 1 and

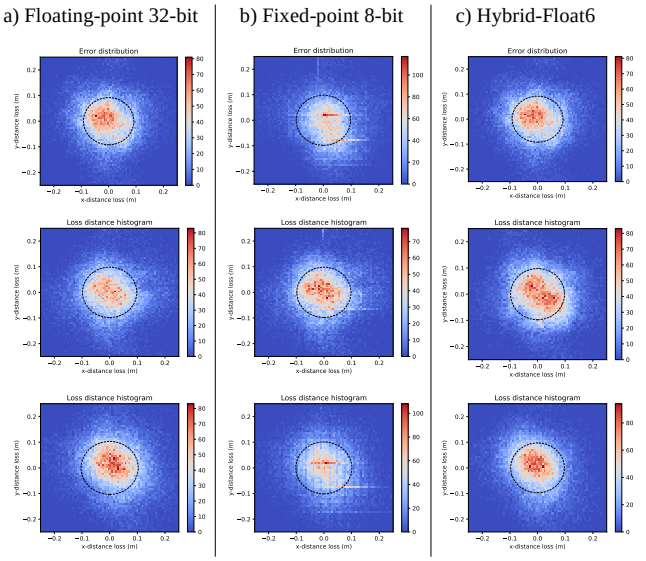


FIGURE 17. 2D error distribution of three CNN-regression models.

Fig. 18. We attribute this to the HF6 logic implementation using FF and LUT, while the FP logic implementation uses Xilinx LogiCORE IPs mainly with DSPs.

The compute performance of the CPU and TP on each convolution layer is presented in **Tab.** 2 and **Fig.** 15. The peak acceleration and power efficiency of the TP with standard FP (Xilinx LogiCORE IP) is $13\times$ and $1,558.13$ MFLOPS/s/W, respectively. While the peak acceleration and power efficiency of the TP with HF6 is $48.3\times$ and $5,743.29$ MFLOPS/s/W, respectively. The HF6 hardware demonstrates an improvement of $3.7\times$ in acceleration and power efficiency with respect to the standard FP hardware. See **Fig.** 15.

The estimated power dissipation on the SoC is presented in **Fig.** 19. This shows a very similar breakdown of power dissipation in both implementations. However, the energy efficiency is increased due to the reduced latency in HF6 hardware. A comparison of related work is presented in **Tab.** 4.

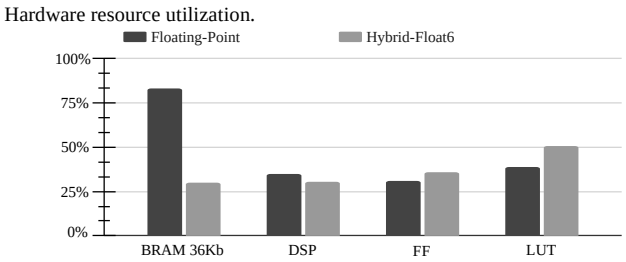


FIGURE 18. Hardware resource utilization on the Zynq-7007S SoC.

The run-time inference of TensorFlow Lite on the SoC is illustrated in **Fig.** 16. This shows the convolution layers as the compute-bound operations. The proposed embedded platform is a cooperative system where the convolution operations are delegated to the dedicated hardware accelerator. The ARM CPU obtains a latency of $387\,\text{ms}$ ($2.58\,\text{FPS}$). The platform with standard FP hardware obtains a latency of $48\,\text{ms}$ ($20.8\,\text{FPS}$), while the implementation with HF6 obtains a latency of $27.9\,\text{ms}$ ($35.84\,\text{FPS}$). These represent an overall acceleration of $8\times$ and $13.87\times$ over the CPU, respectively.

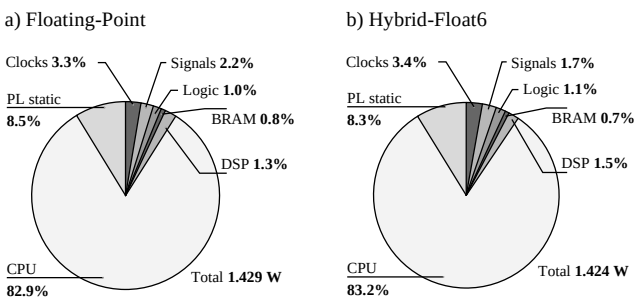


FIGURE 19. Estimated power dissipation on the Zynq-7007S SoC with PS at $666\,\text{MHz}$ and PL at $200\,\text{MHz}$.

This design facilitates ML compatibility/portability as the 6-bit FP is wrapped in the standard FP representation. The dedicated hardware design extracts the 6-bit format automatically and performs computation.

3) SoC Design and Compatibility

The proposed design is an alternative for high accuracy and low-power floating-point inference. The system runs as a cooperative hardware/software mechanism. This architecture delegates compute-bound tensor operations to a hardware accelerator.

The hybrid 32-bit FP and 6-bit FP quantization enables high quality of results and backward ML compatibility. Backwards ML compatibility gives portability from training to inference. This enables to run inference of HF6 quantized models on standard FP hardware and vise versa. The proposed HF6 architecture allows to compute inference of non-quantized floating-point ML models for rapid deployment; however, this will incur in accuracy degradation depending on the resilience of the model, see **Fig.** 14(c).

4) Future Work

In this research, we foresee two lines of future work:

- **To reduce energy consumption.** Activation maps can be represented by Bfloat16 and 8-bit custom floating-point. This would reduce hardware resource utilization, memory footprint, and data transfer while preserving accuracy.
- **To increase performance.** This implementation would require matching computational throughput with memory bandwidth using systolic arrays. This would replace the light-weight pipeline hardware design with a parallelized structure. This requires larger hardware area and energy consumption.

TABLE 4. Comparison of hardware implementation with related work.

Platform	Chunsheng Mei et al. [16]	Chen Wu et al. [13]	BFP [17]	Paolo Meloni et al. [24]	This work
Device	XC7VX690T	XC7K325T	XC7VX690T	XC7Z007S	XC7Z007S
Year	2017	2019	2019	2019	2022
Dev. kit cost	$7,494	$1,299	$7,494	$89	$89
Format (activation/weight)	FP 16-bit	FP 8-bit / 8-bit	FP 16-bit / 8-bit	INT 16-bit	FP 32-bit / 6-bit
Frequency (MHz)	200	200	200	80	200
Peak power efficiency (GFLOP/s/W)	18.72	115.40	82.88	2.98	5.74
Peak throughput (GFLOP/s)	202.42	1086.8	760.83	10.62	0.482
Wall plug power (W)	10.81	9.42	9.18	2.5	2.3
BRAM 36Kb utilization	196.5	234.5	913	44	15
DSP utilization	1728	768	1027	54	20

VI. CONCLUSIONS

In this paper, we present the Hybrid-Float6 quantization for floating-point CNN hardware acceleration. Feature maps and weights are represented by 32-bit and 6-bit floating-point, respectively. The 6-bit floating-point format is composed of 1-bit sign, 4-bit exponent, and 1-bit mantissa. The 1-bit mantissa enables low-power multiply-accumulate implementations by reducing the mantissa multiplication to a multiplexer-adder operation. We exploit the intrinsic error tolerance of neural networks to further reduce the hardware design with approximation. This approach improves latency, hardware area, and energy consumption. To preserve accuracy, we introduce a quantization-aware training method that, in some cases, improves accuracy. We present a lightweight tensor processor implementing a pipelined vector dot-product. For ML compatibility/portability, the 6-bit FP is wrapped in the standard floating-point format, which is automatically extracted by the proposed hardware. The hardware/software architecture is compatible with TensorFlow Lite. We evaluate the applicability of our approach with a CNN-regression model for anomaly localization in a structural health monitoring application based on acoustic emissions. The embedded hardware/software framework is demonstrated on XC7Z007S as the smallest Zynq-7000 SoC. The proposed architecture achieves a peak power efficiency and acceleration on convolution layers of 5.7 GFLOPS/s/W and $48.3\times$, respectively.

REFERENCES

[1] M. Lom, O. Pribyl, and M. Svitek, "Industry 4.0 as a part of smart cities," in *2016 Smart Cities Symposium Prague (SCSP)*. IEEE, 2016, pp. 1–6.

[2] G. Li, C. Deng, J. Wu, X. Xu, X. Shao, and Y. Wang, "Sensor data-driven bearing fault diagnosis based on deep convolutional neural networks and s-transform," *Sensors*, vol. 19, no. 12, p. 2750, 2019.

[3] F. Dong, X. Yu, E. Ding, S. Wu, C. Fan, and Y. Huang, "Rolling bearing fault diagnosis using modified neighborhood preserving embedding and maximal overlap discrete wavelet packet transform with sensitive features selection," *Shock and Vibration*, vol. 2018, 2018.

[4] T. Nagayama and B. F. Spencer Jr, "Structural health monitoring using smart sensors," Newmark Structural Engineering Laboratory. University of Illinois at Urbana . . . , Tech. Rep., 2007.

[5] J. Wang, Y. Chen, S. Hao, X. Peng, and L. Hu, "Deep learning for sensor-based activity recognition: A survey," *Pattern Recognition Letters*, vol. 119, pp. 3–11, 2019.

[6] Y. C. Kim, H.-G. Yu, J.-H. Lee, D.-J. Park, and H.-W. Nam, "Hazardous gas detection for ftir-based hyperspectral imaging system using dnn and cnn," in *Electro-Optical and Infrared Systems: Technology and Applications XIV*, vol. 10433. International Society for Optics and Photonics, 2017, p. 1043317.

[7] T. Ince, S. Kiranyaz, L. Eren, M. Askar, and M. Gabbouj, "Real-time motor fault detection by 1-d convolutional neural networks," *IEEE Transactions on Industrial Electronics*, vol. 63, no. 11, pp. 7067–7075, 2016.

[8] O. Janssens, V. Slavkovikj, B. Vervisch, K. Stockman, M. Loccufier, S. Verstockt, R. Van de Walle, and S. Van Hoecke, "Convolutional neural network based fault detection for rotating machinery," *Journal of Sound and Vibration*, vol. 377, pp. 331–345, 2016.

[9] O. Abdeljaber, O. Avci, S. Kiranyaz, M. Gabbouj, and D. J. Inman, "Real-time vibration-based structural damage detection using one-dimensional convolutional neural networks," *Journal of Sound and Vibration*, vol. 388, pp. 154–170, 2017.

[10] X. Guo, L. Chen, and C. Shen, "Hierarchical adaptive deep convolution neural network and its application to bearing fault diagnosis," *Measurement*, vol. 93, pp. 490–502, 2016.

[11] Z. Du, K. Palem, A. Lingamneni, O. Temam, Y. Chen, and C. Wu, "Leveraging the error resilience of machine-learning applications for designing highly energy efficient accelerators," in *2014 19th Asia and South Pacific design automation conference (ASP-DAC)*. IEEE, 2014, pp. 201–206.

[12] E. Nurvitadhi, G. Venkatesh, J. Sim, D. Marr, R. Huang, J. Ong Gee Hock, Y. T. Liew, K. Srivatsan, D. Moss, S. Subhaschandra *et al.*, "Can fpgas beat gpus in accelerating next-generation deep neural networks?" in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 5–14.

[13] C. Wu, M. Wang, X. Chu, K. Wang, and L. He, "Low-precision floating-point arithmetic for high-performance fpga-based cnn acceleration," *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 15, no. 1, pp. 1–21, 2021.

[14] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[15] S. Han, J. Pool, J. Tran, and W. Dally, "Learning both weights and connections for efficient neural network," *Advances in neural information processing systems*, vol. 28, 2015.

[16] C. Mei, Z. Liu, Y. Niu, X. Ji, W. Zhou, and D. Wang, "A 200mhz 202.4 gflops@ 10.8 w vgg16 accelerator in xilinx vx690t," in *2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. IEEE, 2017, pp. 784–788.

[17] X. Lian, Z. Liu, Z. Song, J. Dai, W. Zhou, and X. Ji, "High-performance fpga-based cnn accelerator with block-floating-point arithmetic," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 27, no. 8, pp. 1874–1885, 2019.

[18] M. Courbariaux, Y. Bengio, and J.-P. David, "Binaryconnect: Training deep neural networks with binary weights during propagations," *Advances in neural information processing systems*, vol. 28, 2015.

[19] Z. Lin, M. Courbariaux, R. Memisevic, and Y. Bengio, "Neural networks with few multiplications," *arXiv preprint arXiv:1510.03009*, 2015.

[20] P. Colangelo, N. Nasiri, E. Nurvitadhi, A. Mishra, M. Margala, and K. Nealis, "Exploration of low numeric precision deep learning inference using intel® fpgas," in *2018 IEEE 26th annual international symposium on field-programmable custom computing machines (FCCM)*. IEEE, 2018, pp. 73–80.

[21] J. Faraone, M. Kumm, M. Hardieck, P. Zipf, X. Liu, D. Boland, and P. H. Leong, "Addnet: Deep neural networks using fpga-optimized multipliers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 1, pp. 115–128, 2019.

[22] L. Lai, N. Suda, and V. Chandra, "Deep convolutional neural network inference with floating-point weights and fixed-point activations," *arXiv preprint arXiv:1703.03073*, 2017.

[23] S. O. Settle, M. Bollavaram, P. D'Alberto, E. Delaye, O. Fernandez, N. Fraser, A. Ng, A. Sirasao, and M. Wu, "Quantizing convolutional neural networks for low-power high-throughput inference engines," *arXiv preprint arXiv:1805.07941*, 2018.
[24] P. Meloni, A. Garufi, G. Deriu, M. Carreras, and D. Loi, "Cnn hardware acceleration on a low-power and low-cost apsoc," in *2019 Conference on Design and Architectures for Signal and Image Processing (DASIP)*. IEEE, 2019, pp. 7–12.
[25] C. Gao, A. Rios-Navarro, X. Chen, S.-C. Liu, and T. Delbruck, "Edgedrnn: Recurrent neural network accelerator for edge inference," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 10, no. 4, pp. 419–432, 2020.
[26] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai *et al.*, "Recent advances in convolutional neural networks," *Pattern Recognition*, vol. 77, pp. 354–377, 2018.
[27] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.
[28] D. Zuras, M. Cowlishaw, A. Aiken, M. Applegate, D. Bailey, S. Bass, D. Bhandarkar, M. Bhat, D. Bindel, S. Boldo *et al.*, "Ieee standard for floating-point arithmetic," *IEEE Std*, vol. 754, no. 2008, pp. 1–70, 2008.
[29] Y. Nevarez, D. Rotermund, K. R. Pawelzik, and A. Garcia-Ortiz, "Accelerating spike-by-spike neural networks on fpga with hybrid custom floating-point and logarithmic dot-product approximation," *IEEE Access*, 2021.
[30] S. Sikdar, S. Banerjee, and G. Ashish, "Ultrasonic guided wave propagation and disbond identification in a honeycomb composite sandwich structure using bonded piezoelectric wafer transducers," *Journal of Intelligent Material Systems and Structures*, vol. 27, 10 2015.
[31] U. Kiencke, M. Schwarz, and T. Weickert, *Signalverarbeitung: Zeit-Frequenz-Analysen und Schätzverfahren*. Oldenbourh, 2008.
[32] R. B. Blackman and J. W. Tukey, "The measurement of power spectra from the point of view of communications engineering — part i," *Bell System Technical Journal*, vol. 37, no. 1, pp. 185–282, 1958.
[33] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
[34] U. Xilinx, "Zynq-7000 all programmable soc: Technical reference manual," 2015.
[35] J. Hrica, "Floating-point design with vivado hls," *Xilinx Application Note*, 2012.
[36] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller, "Striving for simplicity: The all convolutional net," *arXiv preprint arXiv:1412.6806*, 2014.

YARIB NEVAREZ received the B.E. (Hons) degree in electronics from the Durango Institute of Technology, Durango, Mexico, in 2009, and the M.Sc. degree in Embedded Systems Design from the University of Applied Sciences Bremerhaven, Bremen, Germany, in 2017. He is currently pursuing a PhD degree with the Institute of Electrodynamics and Microelectronics, University of Bremen, Germany. His research interest is focused mainly on System-on-Chip architectures and dedicated hardware for artificial intelligence in Embedded Systems.
During his professional experience, he served as a Senior Embedded Software Engineer at Texas Instruments, IBM, Continental Automotive, TOSHIBA, and Carbon Robotics. He has designed and developed software architectures for graphic calculators, automotive systems, robotic drivers, and more.

ANDREAS BEERING received his B.Sc. and M.Sc. degree in Electrical and Information Engineering from the University of Bremen, Germany, in 2015 and 2017, respectively. He is currently working towards a Ph.D. degree at the Institute of Electrodynamics and Microelectronics at the University of Bremen, Germany. His research interests focus mainly on signal processing and classification of vibration signals. In addition to the analysis of passive acoustic emission for monitoring gearboxes and processes, the second focus lies on active ultrasonic investigations. In this context, ultrasonic waveforms for monitoring overmoulding processes are being investigated.

AMIR NAJAFI received the B.Sc. and M.Sc. degree in Electronics Engineering from QIAU, Iran, in 2010 and 2014, respectively. After working for one year as a lecturer in Iran, he started his Ph.D. at the University of Bremen, Germany. In 2021, he received the Ph.D. degree from the department of physics and electronics. He is currently a post-doc researcher in the Institute of Electrodynamics and Microelectronics at the University of Bremen, Germany. His research interests focus mainly on low-power interconnect architectures and approximate communication and computation co-design.

ARDALAN NAJAFI received his M.Sc. degree in digital electronics engineering from Shahid Beheshti University, Tehran, Iran, in 2013. In 2021, he received his PhD from University of Bremen, Germany. He is currently a post-doctoral researcher with the Institute of Electrodynamics and Miroelectronics (ITEM), University of Bremen, Germany. His research interests include: approximate computing design, stochastic computing, low-power digital design, and embedded systems.

WANLI YU received the B. S. degree in information engineering and the M. S. degree in signal and information processing from the China University of Mining and Technology, Xuzhou, China, in 2009 and 2012, respectively and the PhD degree from the University of Bremen, Germany, in 2018.

He is currently a Post-Doctoral Researcher with the Institute of Electrodynamics and Microelectronics, University of Bremen. His research interests include sensor networks, Internet of Things, network performance optimization, energy aware task allocation and scheduling, mobile edge/fog computing, convex optimization, machine learningand nature-inspired population-based optimization.

YIZHI CHEN received the B.E. degree in Electronic and Information Engineering from Wuhan University, China, in 2017 and he received the M.S. degree in Communication and Information Technology from the University of Bremen, Germany, in 2021. Currently, he is working as a Ph.D. student at the Division of Electronics and Embedded Systems, Department of Electrical Engineering, School of Electrical Engineering and Computer Science, KTH Royal Institute of Technology, Sweden. His research interests include hardware accelerator for ML, fault-tolerant hardware for neural networks, Network-on-Chip, and approximate computing.

KARL-LUDWIG KRIEGER received his Ph.D. degree in Electrical Engineering in 1999 from the University of Bremen, Germany. Dr. Krieger worked from 1998-2009 as a manager in the field of function and algorithm development for powertrain systems at Daimler AG in Stuttgart. Since 2009 he has been a full professor for the chair of Electronic Vehicle and Mobility Systems at the University of Bremen, Germany. The research focus of his chair is primarily on the analysis of vibration and ultrasound signals in different application fields, for example, passenger cars, heavy-duty trucks, mobile working machines or industrial machines. Further research priorities of the chair are near-infrared spectral sensors and capacitive sensor systems for different fields of application. These research topics are worked on in close cooperation with leading industrial partners and are mainly financed from third-party funds.

ALBERTO GARCIA-ORTIZ obtained the diploma degree in Telecommunication Systems from the Polytechnic University of Valencia (Spain) in 1998. After working for two years at Newlogic in Austria, he started the Ph.D. at the Institute of Microelectronic Systems, Darmstadt University of Technology, Germany. In 2003, he received from the Department of Electrical Engineering and Information Technology of the university the Ph.D. degree with "summa cum laude." From 2003 to 2005, he worked as a Senior Hardware Design Engineer at IBM Deutschland Development and Research in Böblingen. After that he joined the start-up AnaFocus (Spain), where he was responsible for the design and integration of AnaFocus" next generation Vision Systems-on-Chip. He is currently full professor for the chair of integrated digital systems at the university of Bremen. Dr. Garcia-Ortiz received the "Outstanding dissertation award" in 2004 from the European Design and Automation Association. In 2005, he received from IBM an innovation award for contributions to leakage estimation. Two patents are issued with that work. He serves as editor of JOLPE and is reviewer of several conferences, journals, and European projects.
His interests include low-power design and estimation, communication-centric design, SoC integration, and variations-aware design.