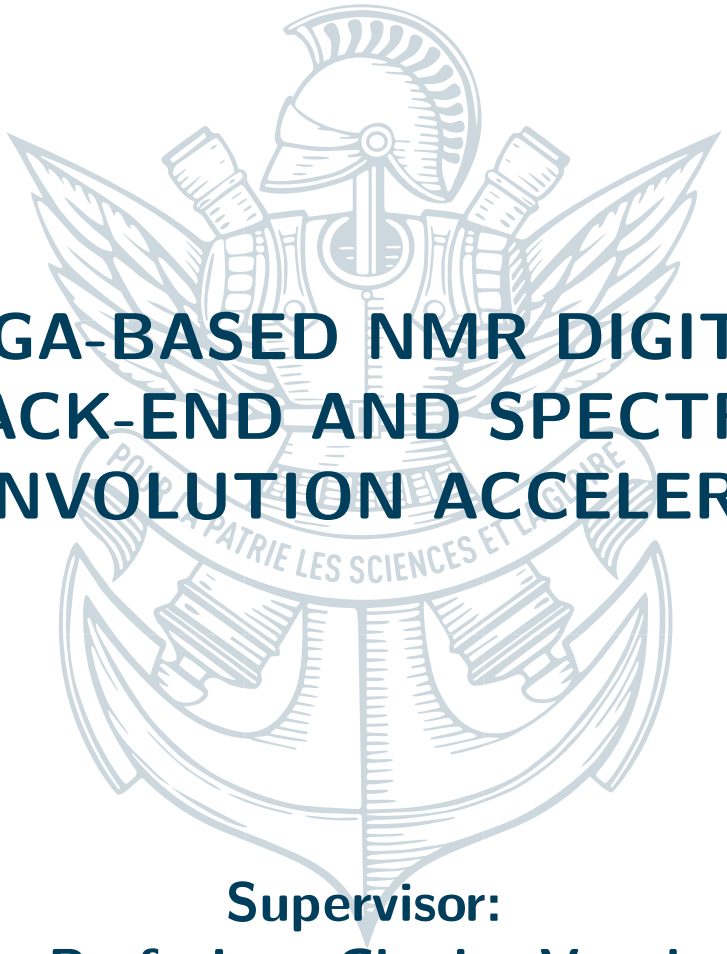

FPGA-BASED NMR DIGITAL BACK-END AND SPECTRA DECONVOLUTION ACCELERATOR

Supervisor:
Prof. Jean-Charles Vanel

January 14, 2024

Yiming Wei

CONTENTS

1 Introduction **2**

2 Portable NMR spectrometer digital back-end design **5**
2.1 Carr-Purcell-Meiboom-Gill (CPMG) Sequence NMR 5
2.2 NMR Controller Design with DE1-SoC 6

3 FPGA based Deconvolution of 1D NMR spectra deep learning accelerator **9**
3.1 LeNet CNN Accelerator 11
3.2 Overall design solution 12

4 Conclusions **16**
4.1 Portable NMR spectrometer design 16
4.2 FPGA based Deconvolution of 1D NMR spectra deep learning accelerator 16

1 INTRODUCTION

Nuclear Magnetic Resonance (NMR) spectroscopy is a powerful and theoretically complex analytical tool that is widely utilized in chemistry, biochemistry, and biology for elucidating the molecular structure of compounds and studying molecular dynamics. The technique is pivotal in various scientific and industrial applications, such as determining the content and purity of a sample as well as its molecular integrity. However, the implementation of NMR spectrometry, particularly in portable and low-cost formats, has been a subject of ongoing research and development.

One viable solution to the challenges of miniaturization and efficiency enhancement lies in the development of Application-Specific Integrated Circuits (ASICs). ASICs, with their specially designed chips, are tailored to maximize both miniaturization and efficiency in specific applications, such as NMR spectrometry. For instance, Bürkle *et al.*[1], Dreyer *et al.*[2], and Krüger *et al.*[3] have designed ASICs that facilitate the creation of notably compact NMR spectrometers without compromising performance(figure 1).

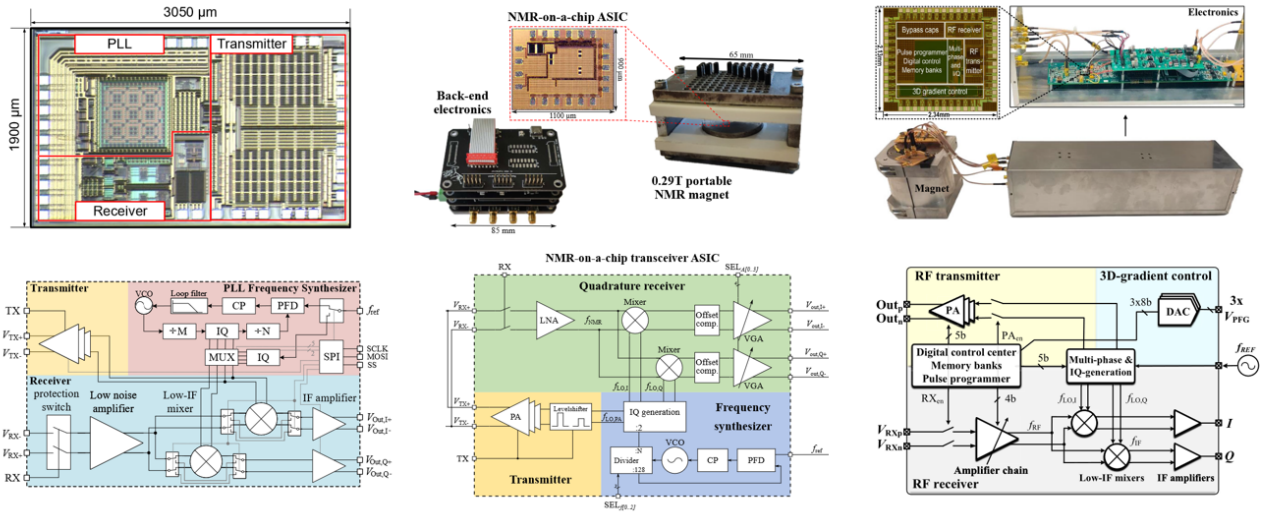

Figure 1: Photographs of ASICs and their architectures. Reprint from [1][3][2].

While the design and manufacturing of ASIC offer a pathway towards miniaturized and efficient NMR spectrometers, the associated costs can be prohibitively high. Consequently, Field-Programmable Gate Arrays (FPGAs) emerge as a compelling alternative, striking a balance between performance and development costs. Pioneering work by Takeda [4][5] around 2007 underscored the viability of FPGAs in constructing a home-built NMR spectrometer. The subsequent, rapid evolution of Integrated Circuit technology, characterized by an exponential growth in transistor count, has precipitated a significant leap in performance capabilities. More

recently, researchers such as Louis-Joseph A *et al.*[6], Webber J B W *et al.*[7], and Ariando *et al.*[8] have demonstrated the construction of a compact NMR spectrometer utilizing a contemporary FPGA. Louis-Joseph A *et al.* provided a comprehensive overview of the ongoing development and prevailing challenges in creating low-cost, portable Fourier Transform NMR (FT-NMR) systems[6]. Furthermore, Ariando *et al.*ingeniously integrated a System-on-Chip (SoC) that amalgamates an ARM microprocessor and an FPGA(figure 2), facilitating fully autonomous operation without necessitating an external computer[8].

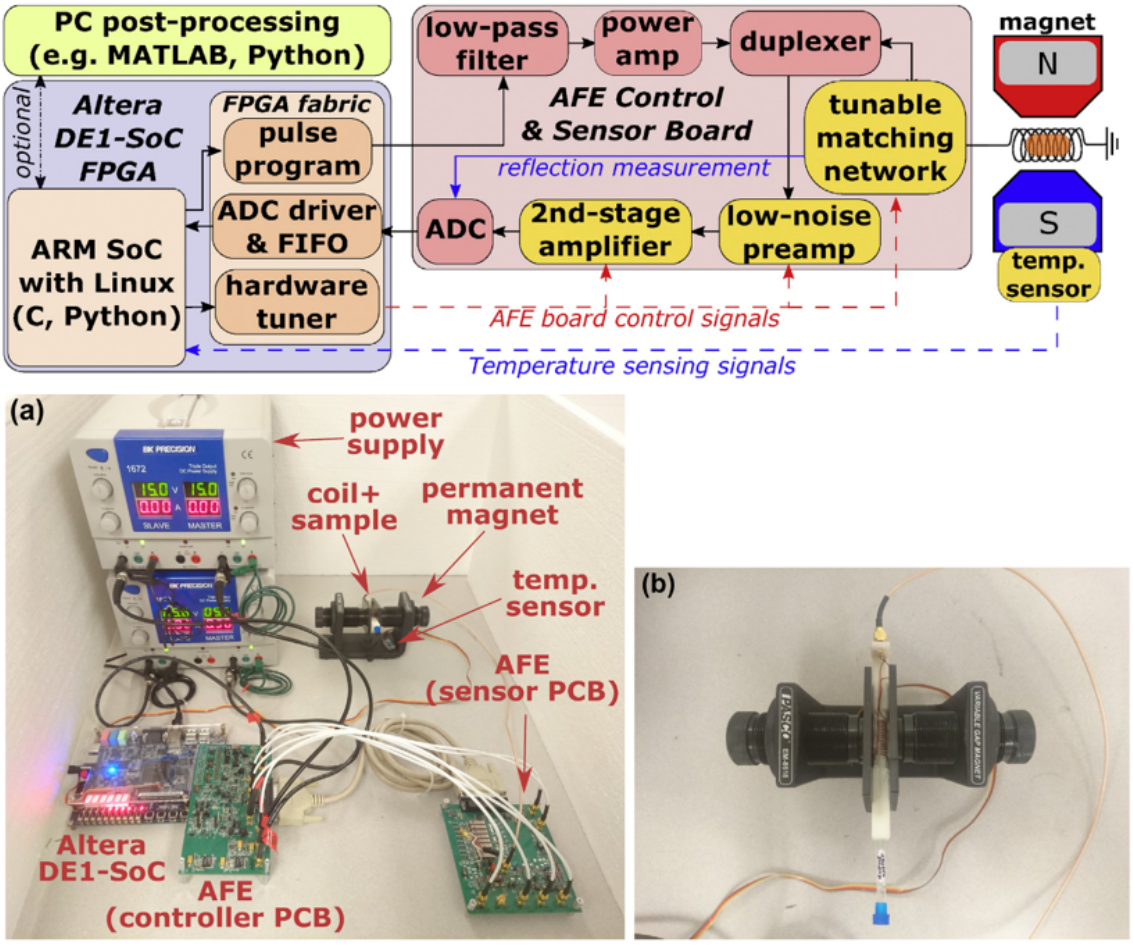

Figure 2: Photograph of SoC and his architecture. Reprint from Ariando *et al.*[8].

Moreover, advancements in artificial intelligence, particularly in machine learning, are beginning to permeate various facets of NMR spectroscopy, enhancing its capabilities and applications. For instance, Manu V S *et al.*[9] has explored the utilization of AI-designed NMR spectroscopy RF pulses to expedite acquisition at both high and ultra-high magnetic fields, showcasing the potential of artificial intelligence in optimizing data acquisition processes. Similarly, Schmid N *et al.*[10] has employed deep learning algorithms to resolve one-dimensional NMR spectra, demonstrating the efficacy of deep learning in enhancing the resolution and interpretability of spectral data(figure 3). Furthermore, Kuhn S *et al.*[11] has leveraged deep

learning to identify substructures in two-dimensional NMR spectra of mixtures, illustrating the potential of artificial intelligence in deciphering complex spectral data and revealing underlying structural information.

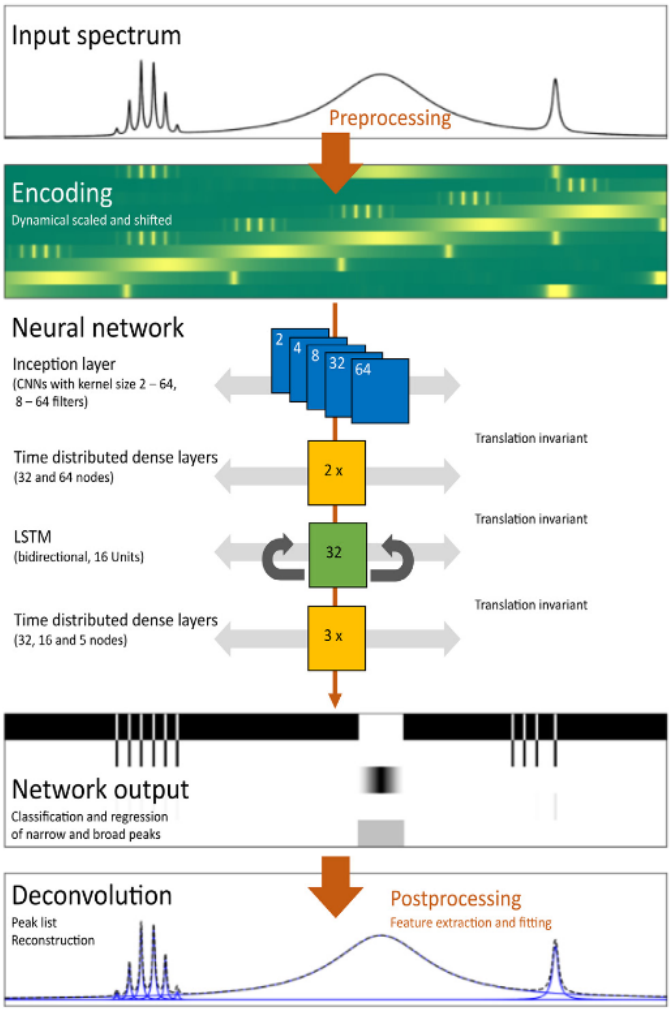

Figure 3: Data flow diagram for the neural network. Reprint from Schmid N *et al.*[10].

2
PORTABLE NMR SPECTROMETER DIGITAL BACK-END DESIGN

2.1 CARR-PURCELL-MEIBOOM-GILL (CPMG) SEQUENCE NMR

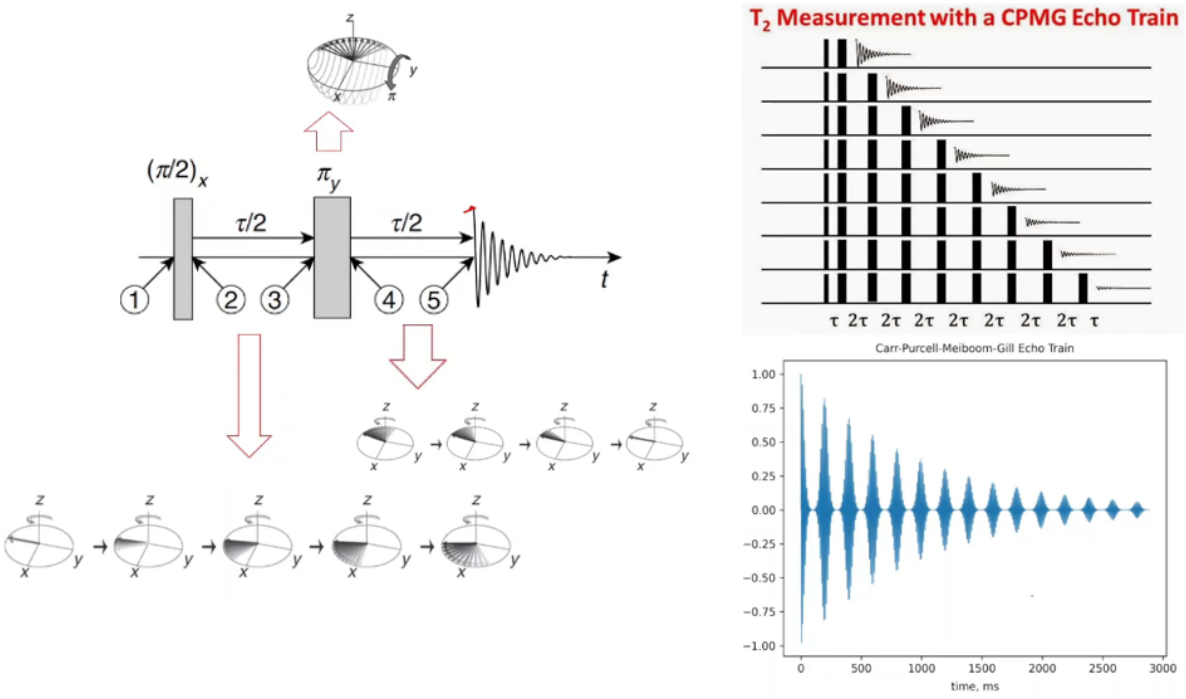

Figure 4: Schematic of CPMG.

The CPMG sequence begins with a $\frac{\pi}{2}$-pulse that tips the magnetization into the transverse plane. After a time interval τ, a π-pulse is applied, which refocuses the dephasing magnetization caused by field inhomogeneities. This π-pulse is typically referred to as the refocusing or echo pulse. After another interval τ, an echo is observed. This process is repeated, with π-pulses applied at subsequent intervals of 2τ, resulting in a series of echoes known as the spin echo train.

The spacing between the π-pulses is crucial; it is set so that any phase errors introduced by inhomogeneities in the main magnetic field B_0 are corrected, which is why the sequence is particularly robust against such imperfections. The sequence can be represented as:

$$\left(\frac{\pi}{2}\right)_x - \tau - (\pi)_y - 2\tau - (\pi)_y - \dots$$

where the subscript x and y denote the axis along which the RF pulse is applied.

Each π-pulse effectively rephases the spin system, compensating for the dephasing that occurs due to the inhomogeneities. This results in a measurable echo signal, which decays over time as a result of true spin-spin relaxation, providing an accurate measurement of the T_2 relaxation time. CPMG is an important NMR technique which allows the true T_2 relaxation time to be measured even with an inhomogeneous B_0 field.

2.2 NMR Controller Design with DE1-SoC

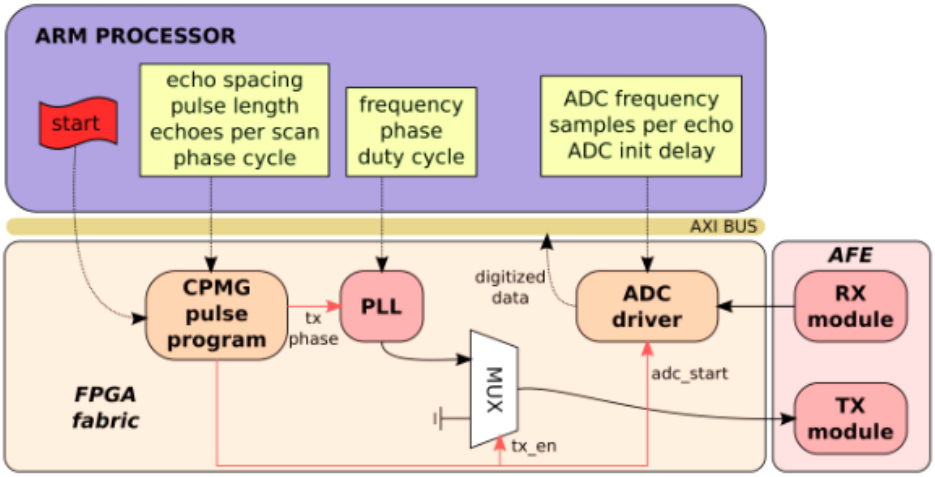

Figure 5: NMR FPGA controller system block diagram on DE1-SoC board. Reprint from Ariando *et al.*[8].

The block diagram of the NMR FPGA controller system, as shown in Figure 5, consists of the CPMG pulse program, ADC driver, and various interconnections. The controller must possess configurability in several parameters such as operating frequency, sampling frequency, duty cycle, delay parameters, phase cycle, and the number of samples. These parameters, once set by the NMR C program inside the ARM processor, remain constant throughout the operation.

The CPMG pulse program, hardcoded into the FPGA fabric, necessitates configurable parameters for execution, including pulse length for both π-pulse and $\frac{\pi}{2}$-pulse, echo spacing, and phase cycle. Additional parameters are also utilized, communicated via the AXI bus. The system utilizes dual PLLs; one for the transmit side interfaced with the transmitter unit, and another within the ADC driver to generate the clock used by the SPI driver for the ADC.

- ### Transmission Driver and ADC Driver

The FPGA's transmit driver includes a PLL block, multiple multiplexers, and parameter registers. The PLL generates four phases from a single frequency, with a $\frac{\pi}{2}$ phase difference,

designated as f_x, f_y, $-f_x$, $-f_y$ to avoid confusion with nutation flip angles. These are then combined to form the differential transmit signal controlled by the CPMG pulse program.

The ADC driver requires parameters such as samples per echo, ADC initialization delay, and ADC frequency. It captures ADC data sequences and adds a starting delay as specified by the CPMG finite state machine. The captured data is transferred to a FIFO buffer for processing.

• NMR signal processing

The signal processing chain starts with the Analog-to-Digital Converter (ADC) operating at the Nyquist sampling rate, which is sufficiently high to capture the essential frequency components of the NMR signal. This is critical as it allows for the direct downconversion of the digitized data using the Larmor frequency, thereby streamlining the signal processing pipeline.

(1)Phase Correction and Summation:

Due to variances in the initial phase of the transmit signal across different scans, a phase correction mechanism is necessary. The process involves summing the echoes from each scan to form a composite signal. The phase of this aggregate signal is then computed, serving as the rotation angle for phase correction.

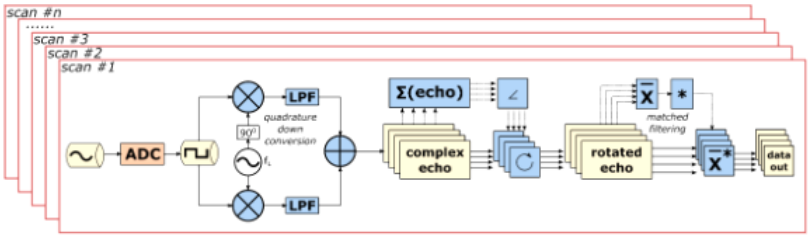

Figure 6: MATLAB NMR signal processing. Reprint from Ariando *et al.*[8].

(2)Detailed Signal Processing Steps:

Figure 6 depicts the detailed steps involved in the signal processing chain:

1. The signal from each NMR scan passes through an ADC, converting the analog signal into a digital format for further processing.
2. Quadrature detection is performed to separate the signal into its in-phase (I) and quadrature (Q) components, each subsequently filtered through a low-pass filter (LPF) to remove high-frequency noise.
3. The filtered I and Q components are then combined to form a complex echo signal.
4. This complex echo undergoes phase correction through a rotation operation, where the previously computed angle is applied to align the phases of echoes from different scans.
5. The rotated echo is then subjected to matched filtering to enhance the signal-to-noise ratio.

6. Finally, the processed signal is outputted as the resulting data, ready for further analysis or display.

The process described ensures that the phase inconsistencies between scans are corrected, resulting in a uniform and coherent data set. Signal processing migration from MATLAB to python, which can be implemented directly in DE1-SoC board, has been done. This will make the system truly portable.

3
FPGA BASED DECONVOLUTION OF 1D NMR SPECTRA DEEP LEARNING ACCELERATOR

Schmid N.*et al.*[10] has developed a highly accurate deconvolutional deep learning network by integrating Convolutional Neural Networks (CNN) and Bidirectional Long Short-Term Memory (Bi-LSTM). The following Python code implements this AI architecture, enabling data analysis on personal computers.

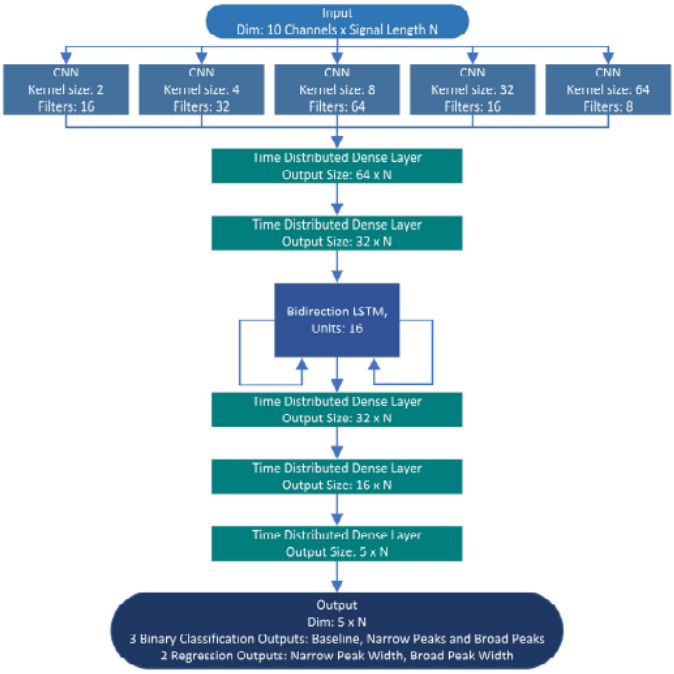

Figure 7: Detailed deep neural network architecture. Reprint from Schmid N *et al.*[10].

The following is reproduced Python code:

```
import tensorflow as tf
from tensorflow.keras.layers import Input, Conv1D, Dense, LSTM,
                                    Bidirectional, TimeDistributed
from tensorflow.keras.models import Model

# Define the input shape
input_shape = (None, 10)  # Assuming variable sequence length N and 10
                                    channels

# Input layer
inputs = Input(shape=input_shape)
```

```
# Inception module with different kernel sizes
conv1 = Conv1D(filters=16, kernel_size=2, activation='relu', padding='same')
                                        (inputs)
conv2 = Conv1D(filters=32, kernel_size=4, activation='relu', padding='same')
                                        (inputs)
conv3 = Conv1D(filters=64, kernel_size=8, activation='relu', padding='same')
                                        (inputs)
conv4 = Conv1D(filters=16, kernel_size=32, activation='relu', padding='same'
                                        )(inputs)
conv5 = Conv1D(filters=8, kernel_size=64, activation='relu', padding='same')
                                        (inputs)

# Concatenate the inception module filters
concatenated = tf.keras.layers.concatenate([conv1, conv2, conv3, conv4,
                                        conv5])

# TimeDistributed Dense layers
td_dense1 = TimeDistributed(Dense(64, activation='relu'))(concatenated)
td_dense2 = TimeDistributed(Dense(32, activation='relu'))(td_dense1)

# Bidirectional LSTM layer
bi_lstm = Bidirectional(LSTM(16, return_sequences=True))(td_dense2)

# More TimeDistributed Dense layers following the LSTM
td_dense3 = TimeDistributed(Dense(32, activation='relu'))(bi_lstm)
td_dense4 = TimeDistributed(Dense(16, activation='relu'))(td_dense3)
td_dense5 = TimeDistributed(Dense(5, activation='softmax'))(td_dense4)

# Create the model
model = Model(inputs=inputs, outputs=td_dense5)

# Compile the model with ADAM optimizer
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=
                                        ['accuracy'])

# Model summary
model.summary()
```

However, to facilitate real-time data measurement and deconvolution in Nuclear Magnetic Resonance (NMR) devices, several enhancements are necessary. Beyond transferring the trained network weights and adapting the network, it is crucial to design a specialized accelerator on the Zynq Programmable Logic (PL) side. This accelerator aims to harness the parallel computing capabilities of Field-Programmable Gate Arrays (FPGAs). Consequently, a streamlined CNN accelerator was constructed. For the Bi-LSTM component, the methodologies presented in the studies by Hao Wang *et al.*[12] offer valuable insights.

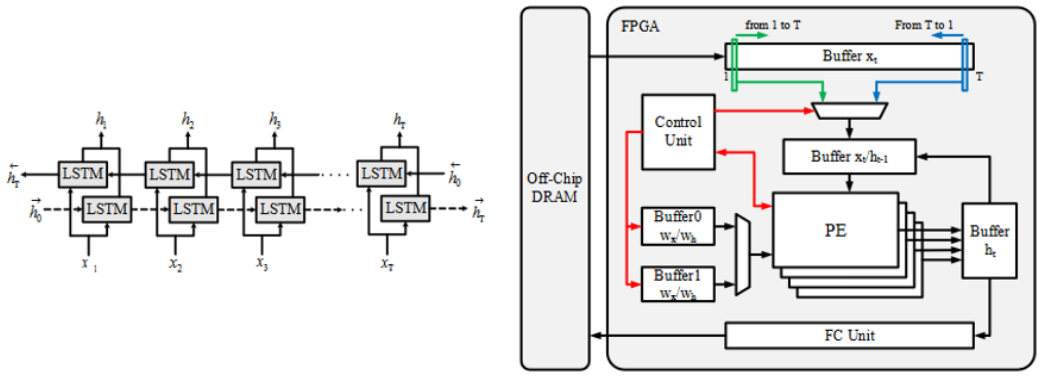

Figure 8: Hardware architecture of the Bi-LSTM accelerator. Reprint from Hao Wang *et al.*[12].

3.1 LeNet CNN Accelerator

LeNet-5, developed by Yann LeCun in 1998[13], is one of the pioneering convolutional neural networks (CNNs) and has significantly influenced the field of machine vision. Designed for handwritten digit recognition, LeNet-5 has been instrumental in advancing the capabilities of deep learning in image classification. It consists of seven layers (excluding the input layer), each playing a specific role in feature detection and recognition. The LeNet-5 architecture which we build is simplified as follows:

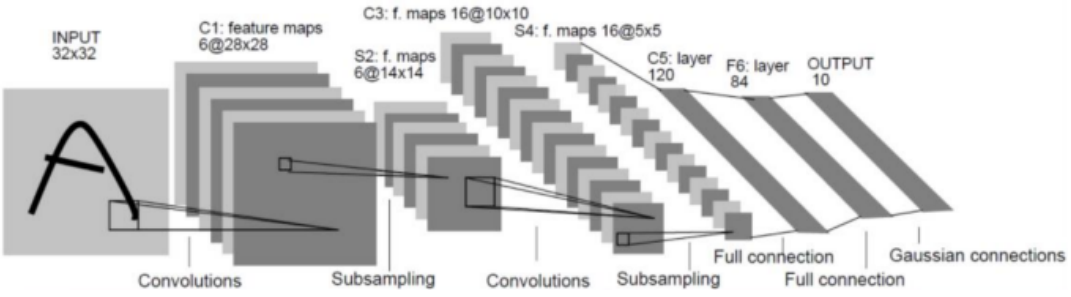

Figure 9: LeNet-5 Convolutional Neural Network Model.

- The first convolutional layer applies a kernel of size 5×5, resulting in 24 feature maps with the dimensions $24 \times 24 \times 6$. This is followed by a subsampling operation with a 2×2 kernel, yielding $12 \times 12 \times 6$ feature maps.
- The second layer increases the depth with a 5×5 kernel, creating 8 feature maps of size $8 \times 8 \times 12$, followed by another convolution with a 4×4 kernel to produce 4 feature maps of dimensions $4 \times 4 \times 12$.
- After flattening, a fully connected layer with 192 neurons is applied, followed by an output layer of 10 neurons for classification, utilizing a softmax activation function for multi-class classification.

- Each convolutional layer prior to the fully connected layer includes operations such as ReLU activation to introduce non-linearity, and padding is applied to maintain the size of the feature maps.
- Specifically, after the first convolutional operation, the size of the feature maps is 24×24, reduced to 12×12 after subsampling with a stride of 2 and no padding.
- The subsequent convolutional layers maintain the size of 12×12 by using 5×5 kernels, with a stride of 1 and no padding, followed by ReLU activation. The feature maps are then reduced to 8×8 and finally to 4×4 after additional convolution and subsampling operations.

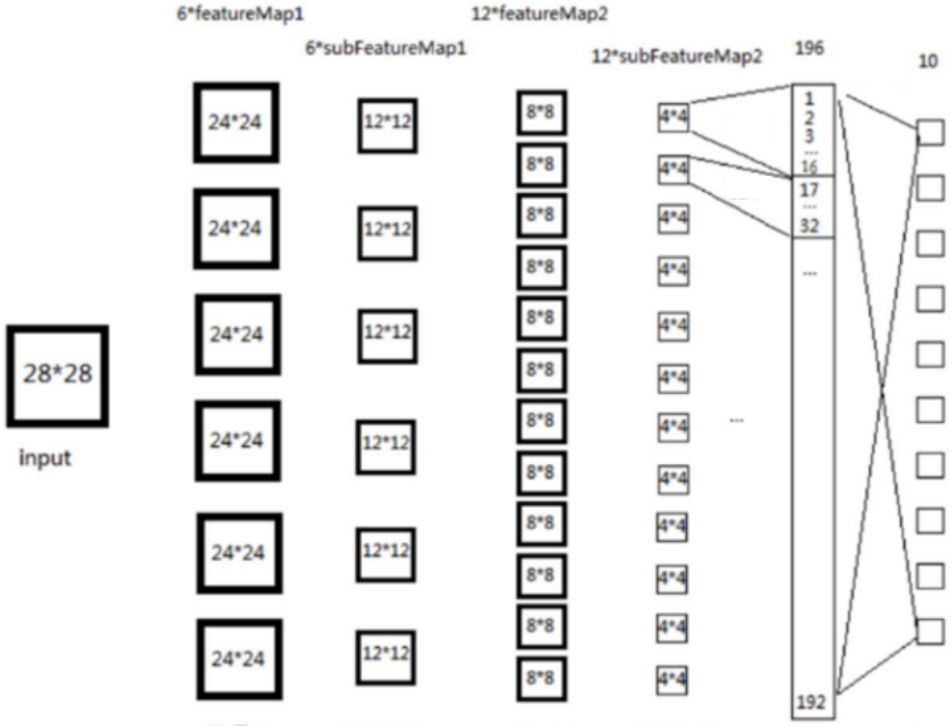

Figure 10: LeNet-5 simplified Model.

3.2 Overall design solution

Resource Utilization and Layer Operations

The FPGA implementation of LeNet-5 involves a meticulous design to fit within the resource constraints of the hardware. The convolution layers F1 to F6 are executed sequentially, each comprising different operational units:

1. The F1 layer is executed with a single operational unit due to the large feature map size of 28×28. The unit utilizes two convolution kernels, each having a size of 5×5, to process the input image.

2. The F2 layer has 6 operational units for the 14×14 feature map size, utilizing 6 convolution kernels of 5×5.

3. The F3 layer has 16 operational units due to the decreased feature map size of 10×10, each using a 5×5 kernel.

4. The F4 layer continues with 16 operational units for the 6×6 feature maps.

5. The F5 layer has 120 operational units for the fully connected layer, transforming the output of the previous layer into a 1-dimensional vector.

6. The F6 layer has 84 operational units, corresponding to the 84 neurons in the fully connected layer of the network architecture.

The data flow between layers is managed through FIFO buffers, ensuring that the processing of one layer can be performed while the next layer's data is being preloaded, optimizing the use of FPGA resources.

FIFO Configuration for Data Flow Management

The FIFO buffers are configured to handle the data flow between layers:

- FIFO 01 manages the output from F1 to F2.
- FIFO 02 manages the output from F2 to F3.
- FIFO 03 manages the output from F3 to F4.
- FIFO 04 manages the output from F4 to F5.
- FIFO 05 manages the output from F5 to F6.
- FIFO 06 manages the output from F6 to the output layer.

Each FIFO buffer is carefully sized to accommodate the data volume from the corresponding layers, ensuring that the output from one layer is synchronized with the input requirements of the subsequent layer. This approach minimizes idle time between operations and maximizes the data throughput within the FPGA-based LeNet-5 implementation.

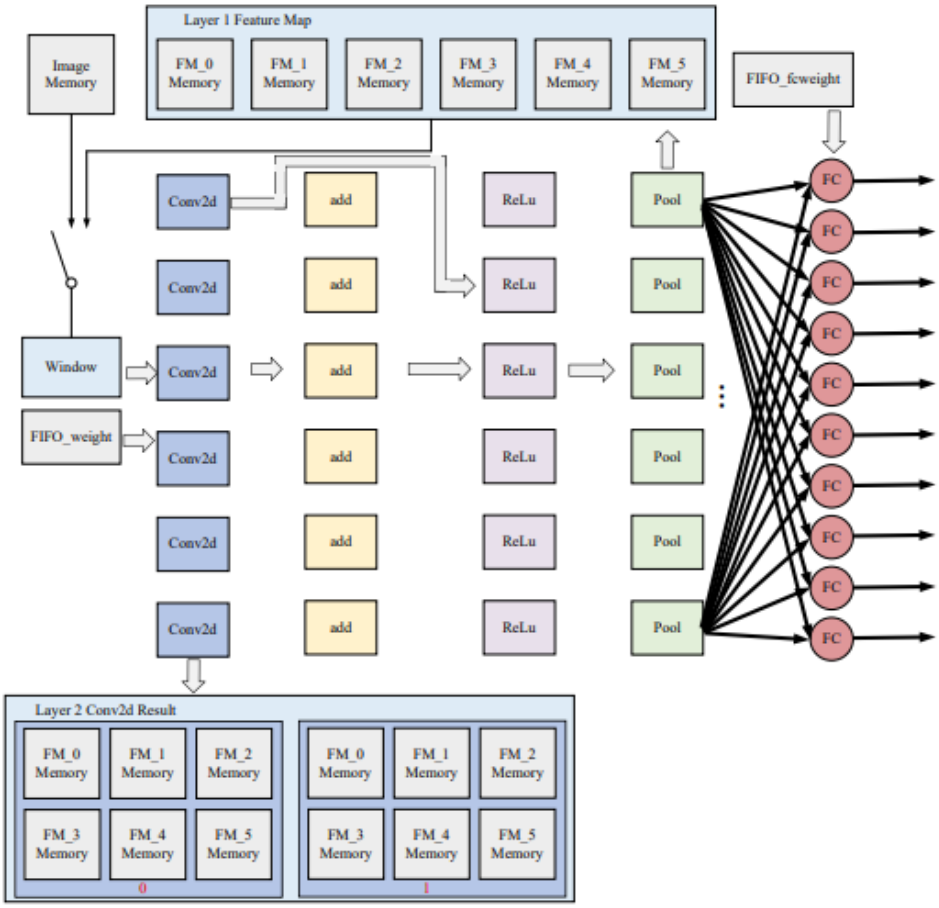

Figure 11: Block Diagram of Accelerator Architecture.

Convolution Implementation

In the convolution operation, a 5×5 kernel is used without padding. The convolution is performed in two steps. The first step processes the data, and the second step combines the processed data into the final result. This two-step process accelerates the convolution operation. Additionally, a shift ram structure is utilized to streamline the storage and retrieval of intermediate data. This structure holds the first layer's 28×28 data and the second layer's 12×12 data, enhancing the efficiency of the convolution operation.

Pooling Implementation

The pooling operation also uses a 5×5 kernel. To optimize the process, a 25 element shift register is employed, which works in tandem with the 5×5 kernel to perform the pooling

operation. This mechanism effectively reduces the feature maps' dimensions, critical for the network's downsampling stages.

Fully Connected Layer Implementation

For the fully connected layers, the FPGA implementation uses a dedicated processing element. Each of the 192 input features is connected to the 10 output features, thus computing the layer in parallel. This results in a significant reduction in processing time, ensuring that the fully connected layer computation is efficient.

Result Calculation and Output

The result calculation takes the 192 input features and connects them to the 10 output features. For each output feature, a 192-dimensional vector is multiplied by a weight matrix of size 192×10 to produce the final result. A shift register arrangement is employed to accelerate the process. The shift registers of size $w(96)$, $w(128)$, $w(144)$, $w(160)$, and $w(176)$ are used to store the intermediate results for the computation of the output features. This implementation allows for the simultaneous calculation of the output features, optimizing the use of FPGA resources and minimizing computation time.

4 CONCLUSIONS

4.1 Portable NMR spectrometer design

In conclusion, the comprehensive study of the autonomous, highly portable NMR spectrometer, as delineated in the paper and further elucidated by the author's open-source System-on-Chip (SoC) design document, has significantly enhanced my understanding of NMR technology. The innovative use of a low-cost SoC in the spectrometer's design not only demystifies the complexities associated with NMR spectroscopy but also exemplifies the potential for wider accessibility and application.

Furthermore, the process of understanding open-source SoC design documents has afforded me a deeper insight into the design techniques essential for NMR. Also signal processing migration from MATLAB to python, which can be implemented directly in DE1-SoC board, maked the system truly portable.

4.2 FPGA based Deconvolution of 1D NMR spectra deep learning accelerator

After that, I performed FPGA hardware gas pedal design for a simple Lenet CNN network, I first learned to design it through HLS tool, which is relatively user friendly and only needs to be written through C, but it also creates redundancy and sacrifices some computational speed. Therefore, after that the design of the gas pedal from HDL was designed, which required a better grasp of timing as well as resources to accomplish the important structural design. In the future, by combining CNN and bidirectional LSTM gas pedals, the NMR deconvolution deep learning network gas pedal can be designed, which in turn realizes the measurement, processing and deconvolution on the same platform, greatly improving the portability of the device and the testing efficiency.

REFERENCES

[1] Heiko Bürkle, Tobias Klotz, Reiner Krapf, and Jens Anders. A 0.1 MHz to 200 MHz high-voltage CMOS transceiver for portable NMR systems with a maximum output current of 2.0 App. In *ESSCIRC 2021 - IEEE 47th European Solid State Circuits Conference (ESSCIRC)*, pages 327–330, September 2021.

[2] Frederik Dreyer, Qing Yang, Belal Alnajjar, Daniel Krüger, Bernhard Blümich, and Jens Anders. A portable chip-based NMR relaxometry system with arbitrary phase control for point-of-care blood analysis. *IEEE Transactions on Biomedical Circuits and Systems*, pages 1–12, 2023. Conference Name: IEEE Transactions on Biomedical Circuits and Systems.

[3] Daniel Krüger, Aoyang Zhang, Behdad Aghelnejad, Henry Hinton, Victor Marrugat Arnal, Yi-Qiao Song, Yiqiao Tang, Ka-Meng Lei, Jens Anders, and Donhee Ham. A Portable CMOS-Based Spin Resonance System for High-Resolution Spectroscopy and Imaging. *IEEE Journal of Solid-State Circuits*, 58(7):1838–1849, July 2023.

[4] Kazuyuki Takeda. A highly integrated FPGA-based nuclear magnetic resonance spectrometer. *Review of Scientific Instruments*, 78(3):033103, March 2007.

[5] Kazuyuki Takeda. OPENCORE NMR: Open-source core modules for implementing an integrated FPGA-based NMR spectrometer. *Journal of Magnetic Resonance*, 192(2):218–229, June 2008.

[6] Alain Louis-Joseph and Philippe Lesot. Designing and building a low-cost portable FT-NMR spectrometer in 2019: A modern challenge. *Comptes Rendus Chimie*, 22(9):695–711, September 2019.

[7] J. Beau W. Webber and Pavel Demin. Credit-card sized field and benchtop NMR relaxometers using field programmable gate arrays. *Magnetic Resonance Imaging*, 56:45–51, February 2019.

[8] David Ariando, Cheng Chen, Mason Greer, and Soumyajit Mandal. An autonomous, highly portable NMR spectrometer based on a low-cost System-on-Chip (SoC). *Journal of Magnetic Resonance*, 299:74–92, February 2019.

[9] V. S. Manu, Cristina Olivieri, and Gianluigi Veglia. AI-designed NMR spectroscopy RF pulses for fast acquisition at high and ultra-high magnetic fields. *Nature Communications*, 14(1):4144, July 2023. Number: 1 Publisher: Nature Publishing Group.

[10] N. Schmid, S. Bruderer, F. Paruzzo, G. Fischetti, G. Toscano, D. Graf, M. Fey, A. Henrici, V. Ziebart, B. Heitmann, H. Grabner, J. D. Wegner, R. K. O. Sigel, and D. Wilhelm. Deconvolution of 1D NMR spectra: A deep learning-based approach. *Journal of Magnetic Resonance*, 347:107357, February 2023.

[11] Stefan Kuhn, Eda Tumer, Simon Colreavy-Donnelly, and Ricardo Moreira Borges. A pilot study for fragment identification using 2D NMR and deep learning. *Magnetic Resonance in Chemistry*, 60(11):1052–1060, 2022. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/mrc.5212.

[12] Hao Wang, Danfeng Qiu, Fen Ge, and Ying Yang. Implementation of Bidirectional LSTM Accelerator Based on FPGA. In *2022 IEEE 22nd International Conference on Communication Technology (ICCT)*, pages 1512–1516, November 2022. ISSN: 2576-7828.

[13] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. ISBN: 0018-9219 Publisher: Ieee.