CENICS 2022

The Fifteenth International Conference on Advances in Circuits, Electronics and Micro-electronics

ISBN: 978-1-68558-009-4

October 16 - 20, 2022

Lisbon, Portugal

CENICS 2022 Editors

Esteve Hassan, Mohawk College, Canada

CENICS 2022

Forward

The Fifteenth International Conference on Advances in Circuits, Electronics and Micro-electronics (CENICS 2022), held on October 16-20, 2022, continued a series of events initiated in 2008, capturing the advances on special circuits, electronics, and micro-electronics on both theory and practice, from fabrication to applications using these special circuits and systems. The topics covered fundamentals of design and implementation, techniques for deployment in various applications, and advances in signal processing.

Innovations in special circuits, electronics and micro-electronics are the key support for a large spectrum of applications. The conference is focusing on several complementary aspects and targets the advances in each on it: signal processing and electronics for high speed processing, micro- and nano-electronics, special electronics for implantable and wearable devices, sensor related electronics focusing on low energy consumption, and special applications domains of telemedicine and ehealth, bio-systems, navigation systems, automotive systems, home-oriented electronics, bio-systems, etc. These applications led to special design and implementation techniques, reconfigurable and self-reconfigurable devices, and require particular methodologies to be integrated on already existing Internet-based communications and applications. Special care is required for particular devices intended to work directly with human body (implantable, wearable, ehealth), or in a human-close environment (telemedicine, house-oriented, navigation, automotive). The mini-size required by such devices confronted the scientists with special signal processing requirements.

We take here the opportunity to warmly thank all the members of the CENICS 2022 technical program committee, as well as all the reviewers. The creation of such a high quality conference program would not have been possible without their involvement. We also kindly thank all the authors who dedicated much of their time and effort to contribute to CENICS 2022.

We also thank the members of the CENICS 2022 organizing committee for their help in handling the logistics and for their work that made this professional meeting a success.

We hope that CENICS 2022 was a successful international forum for the exchange of ideas and results between academia and industry and to promote further progress in the field of circuits, electronics and micro-electronics. We also hope that Lisbon provided a pleasant environment during the conference and everyone saved some time to enjoy the historic charm of the city.

CENICS 2022 Chairs

CENICS 2022 Steering Committee
Junghee Lee, Korea University, Korea
Timm Bostelmann, FH Wedel (University of Applied Sciences), Germany
David Cordeau, XLIM | University of Poitiers, France
Kenneth Skovhede, eScience | Niels Bohr Institute | University of Copenhagen, Denmark

CENICS 2022 Publicity Chairs
Sandra Viciano Tudela, Universitat Politecnica de Valencia, Spain
Jose Luis García, Universitat Politecnica de Valencia, Spain

CENICS 2022

Committee

CENICS 2022 Steering Committee
Junghee Lee, Korea University, Korea
Timm Bostelmann, FH Wedel (University of Applied Sciences), Germany
David Cordeau, XLIM | University of Poitiers, France
Kenneth Skovhede, eScience | Niels Bohr Institute | University of Copenhagen, Denmark

CENICS 2022 Publicity Chairs
Sandra Viciano Tudela, Universitat Politecnica de Valencia, Spain
Jose Luis García, Universitat Politecnica de Valencia, Spain

CENICS 2022 Technical Program Committee

Naeem Abbasi, Qualcomm Technologies Inc., San Diego, USA
Francesco Aggogeri, University of Brescia, Italy
Ahmed Ammar, Ohio Northern University, USA
Amjad Anvari-Moghaddam, Aalborg University, Denmark
Mohammed A. Aseeri, King Abdulaziz City of Science and Technology (KACST), Kingdom of Saudi Arabia
Amirreza Ghadimi Avval, University of Arkansas, USA
M. Ali Aydin, Istanbul University, Turkey
Vincent Beroulle, Grenoble INP-UGA, France
Mahajan Sagar Bhaskar, Prince Sultan University (PSU), Saudi Arabia
Timm Bostelmann, FH Wedel (University of Applied Sciences), Germany
Manuel José Cabral dos Santos Reis, IEETA / University of Trás-os-Montes e Alto Douro, Portugal
Juan-Vicente Capella-Hernández, Universitat Politècnica de València, Spain
Saurabh Chaubey, Analog Devices Inc., Colorado Springs, USA
Spandonidis Christos, Prisma Electronics SA, Greece
Tales Cleber Pimenta, Universidade Federal de Itajuba, Brazil
David Cordeau, XLIM | University of Poitiers, France
Said Drid, University of Batna 2, Algeria
Francisco Falcone, UPNA-ISC, Spain
Laurent Fesquet, TIMA Laboratory | Grenoble Institute of Technology, France
Kamoun Fourati Fourati, University of Sfax, Tunisia
Patrick Girard, LIRMM - University of Montpellier 2 / CNRS, **France**
Victor Grimblatt, Synopsys Chile R&D Center, Chile
Wenkai Guan, Marquette University, USA
Mohammad Haider, The University of Alabama at Birmingham, USA
Amir M. Hajisadeghi, Amirkabir University of Technology (Tehran Polytechnic), Iran
Petr Hanáček, Brno University of Technology, Czech Republic
Abdus Sami Hassan, Chosun University, Korea
Wen-Jyi Hwang, National Taiwan Normal University, Taipei, Taiwan
Malinka Ivanova, Technical University of Sofia, Bulgaria

Table of Contents

High-Level Synthesis of Hardware Accelerators for Deconvolution Engines

Cristian Sestito
Department of Informatics, Modeling, Electronics and System Engineering
University of Calabria
Arcavacata di Rende, Italy
e-mail: cristian.sestito@unical.it

Robert Stewart
Department of Computer Science
Heriot-Watt University
Edinburgh, United Kingdom
e-mail: R.Stewart@hw.ac.uk

Stefania Perri
Department of Mechanical, Energy and Management Engineering
University of Calabria
Arcavacata di Rende, Italy
e-mail: s.perri@unical.it

***Abstract*—Convolutional and Deconvolutional Neural Networks are widespread in several modern computer vision applications, such as high-resolution imaging, object classification and generation, image segmentation and many others. While several efficient hardware architectures are known in literature to accelerate the convolution task, the design of accelerators for deconvolution is still an open problem. The few existing deconvolution engines are customized to exploit in the best possible way specific hardware resources, thus suffering from platform-dependency that certainly allows maximizing speed performances and power-resource efficiency, but, on the other hand makes these designs unsuitable for the high-level synthesis approach. This paper presents a deconvolution structure described in the C++ high-level language and then synthesized at the register-transfer level of abstraction. Results demonstrate that, when characterized within the Xilinx XC7VX980tffg1930-1 device, the described architecture can up-sample a 256×256 input image to the 1024×1024 resolution using less than 3000 LUTs, 1028 18Kb BRAMs and ~640 FFs. The reached 121 MHz running frequency guarantees a frame rate higher than 50 fps to be achieved.**

Keywords-Hardware accelerators; High-Level Synthesis; Deconvolution; Multiply Accumulations; FPGAs.

I. Introduction

Modern deep learning applications [1]-[3], including image segmentation, object generation and high-resolution imaging, exploit both Convolutional and Deconvolutional Neural Networks (CNNs and DCNNs). The former progressively down-sample the digital images received as input to extract relevant features, whereas the latter elaborate the input images to extrapolate new features. As it is well known, Convolution (CONV) and Deconvolution (DECONV) are nothing more than Multiply Accumulations (MACs) performed on the pixels of the received images and the kernel coefficients of $k\times k$ filters. However, despite to their similarity, while CONV has been extensively used in several CNNs, such as AlexNet [4], GoogleNet [5], ResNet [6], VGG16 [7], just to cite some of the most popular models, DECONV has received a great deal of attention only recently: it is an efficient approach to furnish high-resolution images and, therefore, it has become the basic operation of generative neural networks [8][9].

Generally speaking, a DECONV engine receives a low-resolution $H\times W$ image and a $k\times k$ filter and produces a high-resolution $Ho\times Wo$ output image. Several approaches can be exploited to perform such an operation, each having its own pros and cons. As shown in [10], DECONVs can be computed by executing classical CONVs. In order to do this, with S and P being the adopted stride and padding, respectively, the input image is preliminarily strided, by interleaving $S-1$ zeros between each pair of adjacent pixels, and padded by inserting P zeros on the borders. The image obtained in this way is processed through a classical CONV, which is a benefit in terms of design efforts, given that engines designed for CONV can be utilized also to perform DECONV. However, inserted zeros cause useless zeroed MACs and lead to unbalanced workloads. Moreover, the input reorganization, required to stride and pad the input images, limit the achievable speed performances.

As an alternative, the technique proposed in [11] directly multiplies each input pixel by the filter coefficients, thus computing a block of $k\times k$ products. In this way, the blocks of products related to adjacent pixels are overlapped and, to perform DECONV correctly, up to $k-S$ overlapping rows and columns must be properly managed, which increases both the computational complexity and the delay.

The designs presented in [12]-[18] improve the above approach to implement efficient hardware DECONV engines within FPGA-based Systems-on-Chip (SoCs) able to accelerate the complex segmentation and the super-resolution imaging tasks.

As deeply discussed in [19], also the Winograd algorithm can be exploited to perform DECONV. The main benefit of this solution is the very high speed achieved, but, as a drawback, input images and filters must be preliminarily transformed in the Winograd domain, which introduces significant resources and power overheads.

All the previously cited state-of-the-art papers present efficient DECONV engines customized to exploit in the most efficient way the hardware resources available within a specific FPGA device. If on the one hand this choice allows speed performances to be maximized, limiting the power dissipation and the hardware resources requirements, on the other hand it introduces specific realization platform-dependency, which makes such designs unsuitable for the High-Level Synthesis (HLS). The latter allows describing complex tasks, like those performed by DCNNs, in a high-level language (e.g., C/C++) letting the software tool automatically provide the description at the Register-Transfer Level (RTL) of abstraction. The HLS design approach offers a precious aid to the users who: 1) must

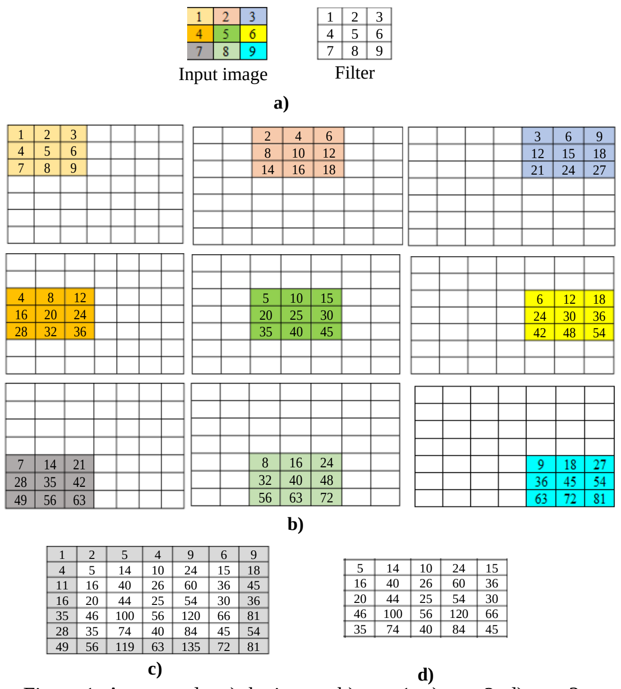


Figure 1. An example: a) the inputs; b) step 1; c) step 2; d) step 3.

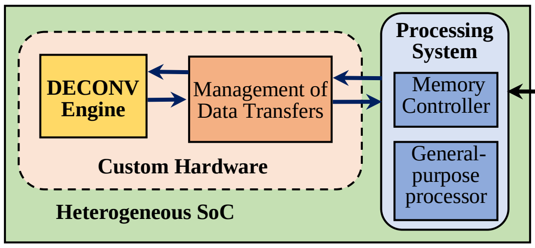


Figure 2. Typical structure of a heterogeneous SoC.

comply with limited realization time; 2) are not familiar with hardware designs at a low-level of abstraction; 3) desire platform-independent portable design descriptions. Indeed, HLS tools can access sets of libraries providing several classes of synthesizable functions that can be exploited to describe complex tasks. Moreover, proper directives and pragmas can be used within the description code to architecturally constrain the synthesis result. Stimulated by these considerations, this paper presents the design of a DECONV engine based on the HLS approach.

The rest of the paper is organized as follows: Section II reviews the adopted DECONV method; Section III details the synthesizable C++ code, written, verified and synthesized with the Xilinx Vivado HLS Tool, and presents post-synthesis results; future works are briefly described in Section IV; finally, conclusions are drawn in Section V.

II. The Adopted DECONV Method

The proposed DECONV engine implements the Input-Oriented-Mapping (IOM) strategy [11]. It performs the generic computation within three steps: 1) multiply each input pixel by the filter coefficients, thus providing a block of $k\times k$ products; 2) sum up the products belonging to the $k–S$ rows (columns) overlapped with adjacent blocks; 3) crop the

```
for (unsigned int i = 0; i < k; i++) {
    for (unsigned int j = 0; j < k; j++) {
        #pragma HLS PIPELINE II=1
        filt[i][j]=filter.read();
for (unsigned int r = 0; r < H; r++) {
    for (unsigned int c = 0; c < W; c++) {
        #pragma HLS PIPELINE II=1
        Pix=InIm.read();
        for (unsigned int i = 0; i < k; i++) {
            for (unsigned int j = 0; j < k; j++) {
                // Multiply the generic pixel by the filter
                Prods[i][j]=Pix*filt[i][j];
                // Store the products to be reused for the column overlap
                if (j >= S)
                  CBuff[i][j–S]=Prods[i][j];
                // Sum up overlapped columns
                if (j<k–S)
                   if (c==0)
                      SumCol [i][j]=Prods [i][j];
                   else SumCol[i][j]=Prods [i][j]+CBuff[i][j];
                else SumCol[i][j]=Prods[i][j];
                // Store the results to be reused for the row overlap
                if (i>=S) {
                   if (j<S)
                     RBuff[i-S][j][c]=SumCol[i][j];
                }
                //Sum up overlapping rows
                if (i < k-S) {
                   if (j < S) {
                     if (r == 0)
                        SumRow[i][j]=SumCol [i][j];
                     else SumRow[i][j]=SumCol[i][j]+RBuff[i][j][c];
                   }
                }
                else SumRow [i][j]=SumCol [i][j];
                // Map the results to the output space
                for (unsigned int i = 0; i < S; i++) {
                    for (unsigned int j = 0; j < S; j++) {
                      OBuff[c+i*W+r*S*H].range(16*j+15,16*j)=SumRow[i][j];
                    }
                }
            }
        }
    }
}
```

Figure 3. The synthesizable C++ code describing the DECONV task.

borders of the output image to modulate its size to $Ho\times Wo$, as given in (1), where P_I and P_O are the input and output padding, respectively.

$$H_O = (H-1)\times S + k - 2P_I + P_O \qquad (1a)$$

$$W_O = (W-1)\times S + k - 2P_I + P_O \qquad (1b)$$

To better explain how the referred method runs, let us examine the example reported Figure 1. It refers to the case in which H=W=3, k=3, S=2, P_I=1, P_O=0. Figure 1b shows how the 3×3 blocks of products obtained by the step 1 (i.e., multiplying each input pixel by the filter) should be arranged into the output space. In this case, adjacent bocks have only 1 overlapping row (column), therefore the accumulations performed in the step 2 lead to the 7×7 provisional image of Figure 1c. Since the size of the output image obtained by (1) is H_O=W_O=5, the gray borders are cropped in the step 3, thus finally producing the output image reported in Figure 1d.

III. The Synthesizable C++ Code and Post-Synthesis Results

The synthesizable C++ routine purposely written to exploit the HLS design approach has been organized assuming that the DECONV engine is the computational core of a custom hardware module exploited within a typical heterogeneous System-On-Chip (SoC) structured as schematized in Figure 2. In such an architecture, data to be processed and produced results are stored in the external memory. As usually happens, read and write memory accesses are managed by the memory controller that communicates directly with the modules responsible for the management of data transfers, like Direct Memory Access modules (DMAs), Central DMAs (CDMAs) or Video DMAs (VDMAs).

From Figure 3, it can be seen that the engine processes the streams *filter* and *InIm* that collect the $k\times k$ filter coefficients and the $H\times W$ pixels of the input image, respectively (lines 1-8). As explained above, the generic pixel *Pix* is multiplied by the filter coefficients, thus providing the block of products *Prods* (lines 9-12). In order to properly manage the overlapping columns between adjacent blocks of products, the 2D array *CBuff* is exploited to provisionally store the overlapping products that must be summed up (lines 13-21) taking into account where the currently processed pixel is located within the input image. To correctly treat also the overlapping rows between adjacent blocks of products, the 3D array *RBuff* is also used. Given that the input image is fed in the raster scan order, the 3D data structure is needed to store: the results obtained by the previous sum of overlapping columns; the results obtained by the current sum of the overlapping rows; and the products that are being computed on the next incoming pixel (lines 22-35). Finally, the results are stored in the output buffer *OBuff* (lines 36-45).

It is worth noting that, in order to architecturally constrain the synthesis result, the C++ code reported in Figure 3 uses the directive *#pragma HLS PIPELINE II=1* several times to introduce pipelining with an Initiation Interval (*II*) equal to 1. The latter ensures that a new input is read at each clock cycle, thus allowing the incoming data and the produced results to be continuously streamed-in and streamed-out.

The above C++ code has been successfully simulated and synthesized using the Vivado HLS 2019.2 CAD tool. Several functional tests have been performed referring to 8-bit unsigned input images and 8-bit signed filters with different image and kernel sizes.

TABLE I. POST-SYNTHESIS RESULTS

Chip		**XC7Z020-clg484-1**					
k	***S***	***H×W, H_O×W_O***	***Tclk* [ns]**	***fps***	**#BRAMs**	**#LUTs**	**#FFs**
3	2	64×64, 128×128	7.81	4878	18	1648	741
		128×128, 256×256	7.81	1219	66	1674	756
		256×256, 512×512	7.81	304	258	1729	771
5	2	64×64, 128×128	7.81	4878	20	2256	1105
		128×128, 256×256	7.81	1219	68	2282	1122
		256×256, 512×512	7.81	304	260	2307	1139
5	4	64×64, 256×256	7.85	840	68	2862	795
		128×128, 512×512	7.85	210	260	2887	817
Chip		**XC7VX980tffg1930-1**					
5	4	256×256, 1024×1024	8.24	53	1028	2917	641
7	4	256×256, 1024×1024	8.01	37	1036	5132	1230

Some post-synthesis results obtained with the XC7Z020-clg484-1 and the XC7VX980tffg1930-1 devices for various

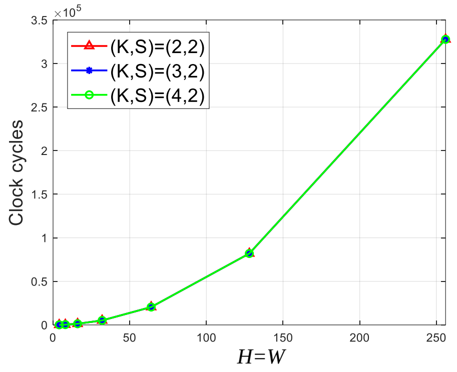


Figure 4. Number of clock cycles versus the input image size.

image and filter sizes and strides are summarized in Table 1. The latter shows how the speed performances, achieved in terms of clock period (*Tclk*) and number of frames produced per second (*fps*), and the hardware resources requirements, represented in terms of occupied Lookup Tables (LUTs), Flip-Flops (FFs) and on chip 18Kb Blocks RAM (BRAMs), change with k, S, $H\times W$ and $H_O\times W_O$.

Obtained results clearly demonstrate that, while the stride S and the output image size $H_O\times W_O$ directly affect the amount of utilized BRAMs, the filter size $k\times k$ impacts on the amount of occupied LUTs and FFs. It can also be observed that the achieved frame rate strictly depends on $H\times W$, which determine how many clock cycles are required to process all the input pixels. Figure 4 plots the number of clock cycles required at various input image size when the stride is set to 2 and the filter size varies from 2 to 4. As expected, the number of clock cycles varies with the image size.

From Table 1, it can also be seen that, due to the limited amount of available BRAMs, the XC7Z020 chip is unsuitable to host the DECONV engine when 256×256 images must be up-sampled to the 1024×1024 resolution (i.e., S=4). For this reason, a different platform has been chosen to synthesize and characterize the proposed architecture in this operating condition. Obtained results confirm the behavior previously discussed.

IV. FUTURE WORKS

It is worth noting that the design presented in the previous Section is the preliminary version of a DECONV engine, which is intended to be used within DCNNs to implement DECONV Layers (DCLs). This means that the deconvolution operation is being performed on M input images (named *ifmaps*) using N different $M\times k\times k$ filters, thus furnishing N output images (named *ofmaps*), each obtained by accumulating M intermediate *ofmaps* in a pixel-wise manner.

Taking this into account, for future works, the architecture above described and characterized will be improved to employ a proper accumulation logic as schematized in Figure 5. Moreover, an adequate level of parallelism will be introduced to process multiple *ifmaps* contemporaneously. Generally speaking, a DCL can be made able to perform multiple deconvolutions in parallel, thus producing O_M intermediate *ofmaps* contemporaneously. A

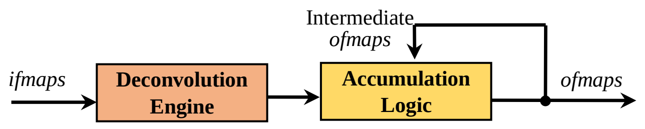


Figure 5. A possible DECONV layer architecture.

certain parallelism may be exploited also at the pixel-level to process multiple pixels of the same *ifmap* at the same time. It is expected that this capability will be introduced by exploiting the Single Instruction Multiple Data (SIMD) paradigm.

Finally, on the basis of the desired behavior other directives and pragmas will be used to use available resources more efficiently, for example including the Digital Signal Processors (DSPs). Obviously, this will further improve the achieve speed performances.

V. Conclusion

This paper presented a deconvolution engine designed using the high-level synthesis approach. In contrast to state-of-the-art designs proposed in literature, the description proposed here avoids specific realization platform-dependency, thus being suitable to be implemented efficiently in different realization platforms. The synthesizable C++ description here described has been characterized at different input and output image sizes, referring to various stride and kernel sizes. Some post-synthesis results have been presented referring to the XC7Z020 low-end device. Then, due to the increasing demand of on-chip memory resources, with the output image being up-sampled to the 1024×1024 resolution, a more expensive chip has been required. Due to its platform independency, the presented code can be synthesized also within different devices families. For future works, the proposed deconvolution engine can be improved to be used within DCNNs and to introduce proper level of parallelism at both frame- and pixel-level.

Acknowledgments

This work was supported by: “POR Calabria FSE/FESR 2014-2020 – International Mobility of PhD students and research grants/type A Researchers” – Actions 10.5.6 and 10.5.12 actuated by Regione Calabria, Italy; The Engineering and Physical Research Council: HAFLANG (EP/W009447/1); Border Patrol (EP/N028201/1); Serious Coding (EP/T017511/1).

References

[1] I. J. Goodfellow *et al.*, “Generative adversarial nets,” in Proc. of the 27th International Conference on Neural Information Processing Systems—Volume 2, Montreal, QC, Canada, 8–13 Dec. 2014, pp. 2672–2680.

[2] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, and J. Garcia Rodriguez, “A review on deep learning techniques for image and video semantic segmentation,” Appl. Soft Comput., vol. 70, pp. 41–65, 2018.

[3] C. Dong, C. C. Loy, K. He, and X. Tang, “Image super-resolution using deep convolutional networks,” IEEE Trans. Pattern Anal. Mach. Intell.,vol. 38, no. 2, pp. 295–307, 2015.

[4] A. Krizhevsky, I. Sutskever, and G. Hinton, ‘‘ImageNet classification with deep convolutional neural networks,’’ in Proc. Neural Inf. Process. Syst. Conf. (NIPS), 2012, pp. 1097–1105.

[5] C. Szegedy *et al.*, “Going deeper with convolutions,” in Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), Boston (MA), USA, 2015, pp. 1-9.

[6] K. He, X. Zhang, S. Ren, and J. Sun, “Deep Residual Learning for Image Recognition,” in Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR), Boston (MA), USA, 2015, pp. 770-778.

[7] K. Simonyan, and A. Zisserman, “Very Deep Convolutional Networks For Large-Scale Image Recognition,” in Proc. Int. Conf. on Learning Representations (ICLR), San Diego (CA), USA, 2015, pp. 1-14.

[8] A. Radford. L. Metz and S. Chintala, “Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks,” in Proc. 4th Int. Conf. on Learning Representations (ICLR 2016), San Juan, Puerto Rico, May 2016.

[9] Y. Yu, T. Zhao, M. Wang, K. Wang, and L. He, “Uni-OPU: An FPGA-Based Uniform Accelerator for Convolutional and Transposed Convolutional Networks,” IEEE Trans. VLSI Syst., vol. 28, no. 7, pp. 1545–1556, 2020.

[10] V. Dumoulin, and F. Visin, “A Guide to Convolution Arithmetic for Deep Learning,” [Online; Retrieved: Jul, 2022] Available: https://arxiv.org/abs/1603.07285.

[11] D. Wang, J. Shen, M. Wen, and C. Zhang, “Efficient Implementation of 2D and 3D Sparse Deconvolutional Neural Networks with a Uniform Architecture on FPGAs,” Electronics, vol. 8, no. 7, pp. 1–13, 2019.

[12] S. Liu, H. Fan, X. Niu, H. C. Ng, Y. Chu, and W. Luk, “ Optimizing CNN-based Segmentation with Deeply Customized Convolutional and Deconvolutional Architectures on FPGA,” ACM Trans. Rec. Technol. Syst., vol. 11, no. 3, pp. 1–22, 2018.

[13] S. Liu, C. Zeng, H. Fan, H. C. Ng, J. Meng, and W. Luk, “Memory-Efficient Architecture for Accelerating Generative Networks on FPGAs,” in Proc. of the IEEE International Conference on Field Programmable Technology, Naha, Okinawa, Japan, 10–14 Dec. 2018, pp. 33–40.

[14] S. Liu, and W. Luk, “Towards an Efficient Accelerator for DNN-Based Remote Sensing Image Segmentation on FPGAs,” in Proc. of the 29th International Conference on Field Programmable Logic and Applications, Barcelona, Spain, 9–13 September, 2019; pp. 187–193.

[15] J. W. Chang, and S. J. Kang, “Optimizing FPGA-based convolutional neural networks accelerator for image super-resolution,” in Proc. of the 23rd Asia and South Pacific Design Automation Conference, Jeju, South Korea, 22–25 January 2018, pp. 343–348.

[16] J. W. Chang, K. W. Kang, and S. J. Kang, “An Energy-Efficient FPGA-Based Deconvolutional Neural Networks Accelerator for Single Image Super-Resolution,” IEEE Trans. Circ. Sys. Video Technol., vol. 30, no. 1, pp. 281–295, 2020.

[17] S. Perri, C. Sestito, F. Spagnolo, and P. Corsonello, “Efficient Deconvolution Architecture for Heterogeneous Systems-on-Chip,” Journal of Imaging, vol. 6, no. 9, pp. 1-17, 2020.

[18] C. Sestito, F. Spagnolo, and S. Perri, “Design of Flexible Hardware Accelerators for Image Convolutions and Transposed Convolutions,” Journal of Imaging, vol. 7, no. 10, pp. 1-16, 2021.

[19] X. Di, H. G. Yang, Y. Jia, Z. Huang, and N. Mao, “Exploring Efficient Acceleration Architecture for Winograd-Transformed Transposed Convolution of GANs on FPGAs,” Electronics, vol. 9, no. 2, pp. 1–21, 2020.

Processing Speed Impact of the Pipeline-Length on a Custom RISC-V CPU for FPGAs

Julian Weihe, Timm Bostelmann and Sergei Sawitzki

FH Wedel (University of Applied Sciences)
Wedel, Germany
Email: {inf104808,bos,saw}@fh-wedel.de

***Abstract*—To achieve a higher processing speed of a Central Processing Unit (CPU), a higher clock frequency can be used. Since the underlying circuit is limited by the switching and signal runtimes, pipeline stages are installed to divide the signal paths. Due to the piecewise processing in the stages, the evaluation of the instruction, which is necessary for the program flow, occurs too late. An example of this are jump instructions in which the target address is not determined until new instructions have already been read. As a result, instructions have to be discarded or the evaluation has to be delayed. This leads to a reduced processing speed and a dependency on the program code. This work shows the difference between a two- and a five-stage CPU with CoreMark. For this purpose, two simple Reduced Instruction Set Computer generation five (RISC-V) CPUs with the instruction set *rv32i* were compared. At the same clock frequency, the two-stage CPU processes 21.358 % more instructions per time than the five-stage CPU, which is slowed down by the pipeline structure. However, a 69.851 % higher clock frequency is possible with the five-stage CPU, which leads to a 39.969 % higher CoreMark score.**

***Keywords*–CPU; FPGA; RISC-V; Pipeline; CoreMark.**

I. Introduction

With RISC-V, an Instruction Set Architecture (ISA) has been developed which, due to its open licensing model, allows modifications and extensions to the underlying hardware. The architecture is particularly widespread in embedded systems and microcontrollers and is also used by companies, such as *Seagate*, *Western Digital Corp.* or *Espressif Systems Corp* [1]. A RISC-V CPU can either be obtained pre-built from companies, such as *SiFive Inc.* or created by the developer [2]. It is precisely the expandability through, as an example, new instructions that makes the development of one's own CPU attractive [3].

The instructions of the ISA must be appropriately converted into hardware when creating a microprocessor with a RISC-V CPU. Since clock speeds and structure depend on the underlying hardware, there is some room for development here. The basic structure of a microprocessor with memory, registers, Arithmetic Logic Unit (ALU) and Program Counter (PC) is always quite similar. However, the interconnection of the components is not trivial and influences runtimes and the size of the design. The execution speed is directly related to the clock used for the CPU. The clock applied to increase performance is limited by the runtimes of the signals and gates. To reduce these, pipeline stages are built into the CPU. The synchronous memories save intermediate results from partial calculations and thus shorten the critical path [4].

In this paper, the performance of a two-stage CPU is compared with a five-stage CPU using *CoreMark* [5]. The number of instructions per time is compared with the increased clock rate of the five-stage pipeline CPU. In addition, the space requirements resulting from the further pipeline stages are also discussed. The development and benchmark of the two designs was implemented on a Field Programmable Gate Array (FPGA). In contrast to other works, which compare complete existing processor designs in different aspects [6][7], here only the effect of the actual number of the two analysed pipeline stages is considered.

In Section II, the basics for implementation and evaluation are described. The RISC-V ISA is described in more detail, as it directly influences the design. The *CoreMark* benchmark is also briefly introduced. Section III describes the implementation of the two CPUs, as well as the compilation of the programme code. The performance losses due to a longer pipeline are also shown here. Section IV compares the results of the two CPUs. As described, space requirements, maximum clock and scores determined by the benchmark are analysed and discussed. Finally, Section V provides a summary of this work and an outlook on further comparisons and analyses.

II. Background

The RISC-V ISA provides a compiler for instructions with an instruction width of 32, 64 and 128 bit. In addition, extensions can be added which, for example, support hardware-supported calculation with floating point values. The list of instructions given by the ISA must be implemented in the hardware. They are roughly divided into logical and arithmetic, load and store and conditional or unconditional jump operations. The instructions determine the structure of the hardware. Modules, such as the logical and arithmetic operations are combined in ALU, the comparator for the conditional jumps or the memory address calculation provide a relatively strict specification for implementation. Also decisive for the compiler are the firmly defined 32 registers, as well as the byte-addressable memory access [8].

A benchmark is used to compare the performance of the two CPUs. There are only a few popular benchmarks for embedded systems [9]. In each of them, different functions are performed to measure performance. In *Whetstone*, the focus is on floating point computation. But especially in small systems often no explicit hardware is implemented for this and therefore it is not used in this project. Also *Dhrystone*, for example, uses the c standard library with functions for memory management, which cannot always be fully implemented

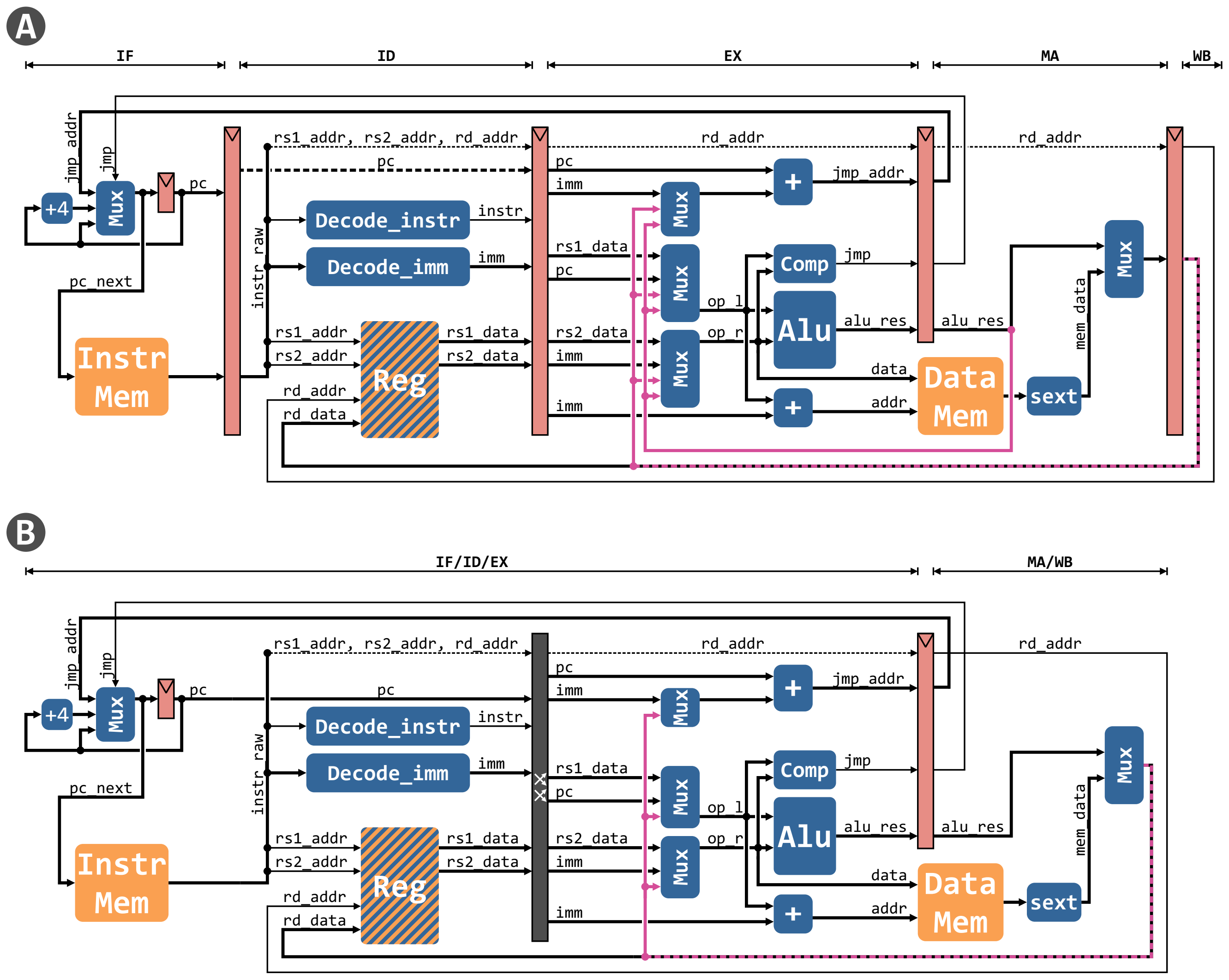


Figure 1. Structure of the implemented five-stage CPU (*A*) and the two-stage CPU (*B*).

due to the memory size [9]. For these reasons, the benchmark *CoreMark* developed by the company *EDN Embedded Microprocessor Benchmark Consortium (EEMBC)* was used. The focus here is on list processing, matrix operations, state machines, and Cyclic Redundancy Check (CRC) calculations [10]. The integration is done without dependencies of other libraries. Regarding the hardware, there are only two requirements. A timer must be integrated for time recording and a communication interface must be implemented to export the results. To use CoreMark with custom hardware, the timer and communication interface must be implemented by the design and made available through functions. The result is output via the serial communication interface after the benchmark has been executed [11].

III. Implementation

The implementation section is divided into four subsections. First, the development environment and conditions are presented. This is followed by an outline of the similarities and then the differences between the two implimented CPUs. Finally, the design decisions that lead to performance losses in the five-stage CPU are described.

A. *Environment*

The development and benchmark was done on an *Intel Cyclone 10LP 10CL025* FPGA. The clock frequency is generated via the FPGA integrated Phase Locked Loop (PLL). These are set at synthesis time. The carrier board of the FPGA also provides a standard clock of 12.000 MHz. Synthesis and timing analyses are provided by the software *Quartus Prime v.20.1.0*. The prebuild tools published by *Sifive* in December 2020 were used to compile the benchmark software [12]. The optimisation *-O1*, as well as selected features, were added to the compiler call. The same compilation was used for all benchmarks.

B. *Uniform structures*

Structurally, the design consists of the CPU and the memory connected to it. The memory is divided into a common program and data memory and a peripheral area that provides Input Output (IO), timer and a serial communication interface used by *CoreMark*. For the simplicity of the system, no external memory is connected. Program and data memory are located on the integrated memory blocks of the FPGA. The access to the *M9K* memory blocks can be done in the used FPGA with a maximum frequency of 200 MHz [13]. As the memory blocks do not support the byte addressing required

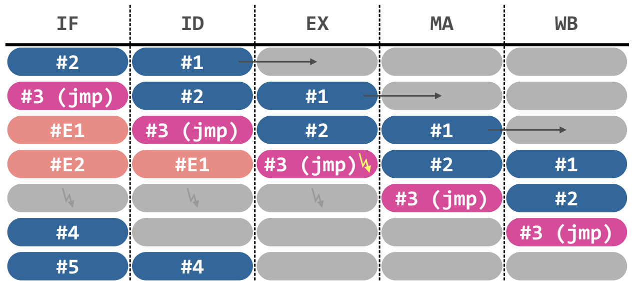


Figure 2. Instruction processing with jump delay.

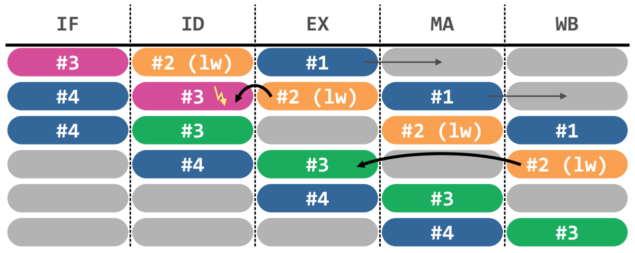


Figure 3. Instruction processing with memory read dependencies.

by the ISA, this is realised by interconnecting four separate byte blocks. Here, the last two bits determine the reading order of the four memory blocks. The memory is identical in both implementations of the microprocessor, so changes in performance are due to differences in the CPU.

Two microprocessors were developed for the comparison. Both implement the instruction set *rv32i* and differ mainly in the pipeline structure. The structure of both implementation are shown in Figure 1. Since this is a fully synchronous design, the clock connection of storing elements has been omitted for a better overview. Likewise, from decoding on, the signal *instr* is not displayed in the further stages, because it is used in almost all places. The pipeline stages are named above the respective design and are described in more detail in the following paragraph. The different stages are separated from each other with synchronous memory blocks represented by the red narrow blocks. In contrast the grey block in the middle of *B* corresponds to a strictly logical linkage and serves the purpose of clarity.

C. Five-stage vs. two-stage structure

In the five-stage pipeline CPU, certain tasks are calculated in each stage. In the Instruction Fetch (IF) stage, the instructions are read from the memory at the address of the programme counter. In the next step, the Instruction Decode (ID) stage is responsible for analysing the command. Operators from the registers are also loaded here. In the following stage Execute (EX), arithmetic, logical and comparison operations are carried out. In addition, the memory address for the memory access and possible jump addresses are calculated. In the next stage Memory Access (MA), write and read accesses to the memory take place. Read signed values are also adjusted to the 32 bit data width. Finally, in the Write Back (WB) stage, the results are written back to the registers.

The two-stage pipeline CPU merges the stages IF, ID and EX and is no longer separated by memory stages. Similarly, the separation of MA and WB has been dropped. Because of the data memory, which writes the address and data to the memory, two stages are also necessary here. The functionality of the pipeline stages are implemented as in the five-stage CPU. The designs differ only in the pipeline memory blocks and the complexity of the multiplexers.

D. Disadvantages of pipeline stages

Due to the reduced signal paths in the five-stage CPU, an increased possible clock frequency can be expected. However, the pipeline stages in particular lead to performance losses due to jumps. Figure 2 shows an example of the instructions in the pipeline stages during a jump. The pipeline stages are displayed vertically and show in each new row the instructions in the respective stage to the corresponding clock cycle. The *#1* and *#2* instruction are fully executed, but the *#3* instruction is a jump, which is only detected in the execution phase. The previous misread *#Ex* instructions are discarded. After the executed jump, the instructions continue to be read sequentially as normal. The late evaluation of the instruction in the EX stage leads to further instructions being read by mistake at first. Since these must be ignored, a gap is created at this point so that the pipeline is not fully utilised. This has the consequence that three instructions are lost per jump instruction.

In the programme sequence, successive calculations can occur on the same register. This would lead to waiting with the following calculation until the result was written back to the WB stage. To speed this up, separate returning connections have been added [4]. The target register corresponds to the one in the MA stage. In the 5-stage pipeline CPU, this affects the results of the ALU operation in the MA and the write back data result of the WB stage. Read operations from memory require another clock cycle before the result can be returned. Therefore, in the sequence, an empty instruction is inserted if the register addresses match, thus delaying the execution by one clock cycle. This is shown in Figure 3. The instruction *#3* coming after the memory-reading instruction *#2* needs the memory value as an operand. Since the memory access is only available one clock cycle later, a stalling instruction is inserted. If instruction *#3* is now applied in the EX stage, the memory value from the WB stage is used. In the design, the returns of the data lines in Figure 1 are recognisable by the pink connections. The two-stage microprocessor has no waiting cycle after a read memory access due to the deliberate reduction of the pipeline stages. This increases the corresponding signal runtimes here.

IV. Results

The evaluation first looks at the performance determined by the benchmark. Then the space requirements of the respective implementation are analysed.

A. Runtime analysis

Both CPUs are not able to perform floating point calculations. Therefore, the ticks determined after the benchmark must be converted into a score for the run. The conversion is shown below.

$$Score_{Iterations\,/\,Sec} = \frac{Iterations \cdot Frequence}{Ticks_{Total}} \tag{1}$$

The score of the benchmark describes the number of iterations per second. At compile time, the number of completed runs was transferred via parameters. In this case, a total measurement of 200 runs was taken. After the benchmark is finished, the number of ticks required for execution is output.

TABLE I. COREMARK SCORES OF THE TWO MICROPROCESSORS WITH THE SAME AND RESPECTIVE MAXIMUM CLOCK.

Stages	Frequence / MHz	Iterations	Ticks	Score
2	12.000	200	203629411	11,786
2	39.670	200	203629411	38,963
5	12.000	200	247103108	9,7125
5	67.380	200	247103108	54,536

TABLE II. ASSIGNMENT OF LOGIC ELEMENTS AND REGISTERS BY THE RESPECTIVE IMPLEMENTATION OF THE MICROPROCESSOR.

Stages	Optimization mode	Logic Elements	Register
2	Balanced	4566	1344
2	Performance	4769	1577
5	Balanced	4821	1670
5	Performance	5009	1833

The score can be calculated from this. The results are shown in Table I.

The series of measurements begins with a synthesis at the same clock rate of 12.000 MHz for both microprocessors. It can be seen that the two-stage CPU with 203 629 411 ticks needs less time to run the benchmark than the five-stage CPU with 247 103 108 ticks. This is also visible in the correspondingly higher score. The two-stage CPU works faster by 21.358 % due to the jumps and also the delays caused by the dependencies of successive instructions with memory accesses.

After synthesis, a time analysis is performed. The development tool provides a maximum clock that may be applied to the circuit. Here, the advantage of the pipeline structure becomes apparent. Whereas the two-stage CPU may clock at a maximum of 39.670 MHz, the maximum clock for the five-stage pipeline CPU is 67.380 MHz, which is 69.851 % higher. With the maximum clocks determined for each microprocessor, a score is calculated again. Although the two-stage pipeline CPU has a higher score at the same clock frequency, the higher clock frequency of the five-stage pipeline CPU leads to a higher score, overall.

From a performance point of view, the increased clock rate due to the pipeline stages is an improvement. However, it should be noted that the function from the benchmark was executed. Since jumps in particular lead to performance losses, it cannot be said in general how efficiently the CPU calculates with the pipeline stages. A compiled program with more jumps, for example, would also perform worse in this respect. It always depends on the application and the compilation.

B. *Space analysis*

In addition to the execution speed, the occupied area on the semiconductor or FPGA is a decisive point, especially for small embedded systems. Table II shows the demand for logic elements and registers of the two implemented designs. This includes, for example, the program and data memory as well as their overlying byte addressing. However, since both implementations use identical assemblies for the implementation of the logic, the difference in number is due to the pipeline structure.

In Table II, two syntheses with different optimisation levels have been carried out in each case. The benchmark values determined in Table I always refer to the *performance* optimisation. As expected, the additional logic through the pipeline requires more logic elements as well as registers.

V. CONCLUSION

In this work, the effect of different numbers of pipeline stages on their performance and space requirements was investigated. It shows that the number of instructions per time decreases with a five-stage CPU, but a higher clock rate is possible. This increases the performance and in this case ultimately works faster than a CPU with two pipeline stages. The space requirement increases with an increasing number of pipeline stages because of the additional logic.

In the end, the application determines the choice between the number of stages. If the application requires the fastest possible execution, a five-stage pipeline CPU is more recommended. But especially when it comes to small embedded systems or the application is not time-critical, the space requirement can also be decisive. Another advantage of the two-stage CPU designed in this work is the guaranteed execution of instructions, which does not depend on the program code.

Further analysis is needed to more accurately assess the efficiency of pipeline stages. In the context of this work, a jump prediction logic was explicitly omitted. Likewise, the memory is directly connected and does not depend on a cache structure. This must be taken into account for the implementation in real systems.

REFERENCES

[1] “Esp32-c3,” https://espressif.com/en/products/socs/esp32-c3, Espressif Inc., accessed: 2022-07-03.

[2] “Sifive processors,” https://www.sifive.com/risc-v-core-ip, SiFive Inc., accessed: 2022-07-12.

[3] J. Hsu, “RISC-V star rises among chip developers worldwide,” https://spectrum.ieee.org/riscv-rises-among-chip-developers-worldwide, April 2021, accessed: 2022-07-12.

[4] H. Miyazaki, T. Kanamori, M. A. Islam, and K. Kise, “RVCoreP: An optimized RISC-V soft processor of five-stage pipelining,” IEICE Transactions on Information and Systems, vol. 103, no. 12, 2020, pp. 2494–2503.

[5] “Coremark,” https://www.eembc.org/coremark/, EEMBC, accessed: 2022-07-03.

[6] A. Dörflinger et al., “A comparative survey of open-source application-class RISC-V processor implementations,” in Proceedings of the 18th ACM International Conference on Computing Frontiers, 2021, pp. 12–20.

[7] P. D. Schiavone et al., “Slow and steady wins the race? a comparison of ultra-low-power RISC-V cores for internet-of-things applications,” in 2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS). IEEE, 2017, pp. 1–8.

[8] K. A. Andrew Waterman, “The RISC-V instruction set manual,” https://riscv.org, January 2021, accessed: 2021-12-18.

[9] P. K. Krause, “Stdcbench: A benchmark for small systems,” in Proceedings of the 21st International Workshop on Software and Compilers for Embedded Systems, 2018, pp. 43–46.

[10] S. Gal-On and M. Levy, “Exploring coremark a benchmark maximizing simplicity and efficacy,” https://www.eembc.org/techlit/articles/coremark-whitepaper.pdf, 2012, accessed: 2022-07-03.

[11] eembc, “coremark,” https://github.com/eembc/coremark, GitHub, accessed: 2022-03-01.

[12] sifive, “freedom-tools,” https://github.com/sifive/freedom-tools, GitHub, accessed: 2022-03-07.

[13] “Intel cyclone 10 lp device datasheet,” https://cdrdv2.intel.com/v1/dl/getContent/666518?fileName=c10lp-51002-683251-666518.pdf, Intel Corp., 2018, accessed: 2022-06-15.

Design of Novel Integrated Data Acquisition System for Multi-Channel Sensing in Landing Gear

Esteve Hassan
Sensor Systems and Internet of Things (IoT)
Centre Mohawk College
Hamilton (ON), Canada
esteve.hassan@mohawkcollege.ca

***Abstract*— In this paper, it is intended to describe the design process, planning, and development of a new Data Logger System (DLS) that can be potentially used in acquiring sensor signals in the landing gear of aircraft. This paper is presenting the new concept and development process of the DLS that employs a novel low-power pulse mode circuit structure. It outlines the overview of the sensor acquisition system and manufacturing of the DLS housing to fit the partner's requirements. The conducted work is broken down into several phases, these include conceptual design, development, production, and testing.**

Keywords—data acquisition; low power; multi-channel sensing; landing gear.

I. INTRODUCTION

Timely and accurate detection of aircraft status signals such as the landing gear is a basic guarantee of its normal operation and also improves aircraft reliability and reduces potential risks through taking timely and effective measures [1]-[6].

The proposed DLS in this paper is adopting a novel data acquisition system to reduce overall system power consumption and provide an effective 32 multi-channel sensing capability that outputs signals with varying ranges. To accommodate the different parts of the DLS including power management and data processing, a modular housing design is being used to meet the requirements of a rigid system that can withstand the harsh environment in aircraft landing gear and provide flexible testing accessibility. The new design is addressing a major sensing data challenge in the aviation industry which required a reliable and sustainable system that runs effectively under severe environmental flying conditions. A new low-power strategy was adopted in our design based on power sampling for sensor array which allows the integration of additional sensors without complicating overall system development.

Section II is outlining the system design overview. The description of the new development will be presented in section IV. The structure design is given in section V, whereas the design verification and testing results are presented in section VI.

II. DESIGN OVERVIEW

The system design requirements obtained from the industry partner are summarized in Table I. The desired low average current could only be achieved by implementing a high-speed configuration that would complete all the action in a small fraction of the sampling period, remaining in sleep mode the rest of the time. This ratio (active/sleep) determines how much is comparatively high active mode consumption reduced to an acceptable low average.

That means, all the procedures (sampling, A/D conversion, and memory update) should be as short, as only possible compared to the sampling period. The description of the system, designed to meet these requirements follows.

To provide the shortest available sampling time, along with minimum power loss, no voltage regulator has been used to power the sensor circuitry (due to long power on/off time). The sensors are powered dynamically (with exponentially rising voltage); A high-speed comparator disconnects the sampling keys from the set of sampling capacitors when the sensor driving voltage achieves the threshold. It is assumed, that the response of the sensor is linear related to driving voltage. To avoid errors related to sensor output impedance, the signals from sensors are applied via buffer amplifiers as seen in the system diagram presented in Figure 1.

The maximum slew rate of sensor driving voltage is limited by the slew rate of buffering amps that could be as high as 2V/us. To avoid errors, related to the response time of the comparator, the driving voltage is also sampled, providing the possibility for further software correction.

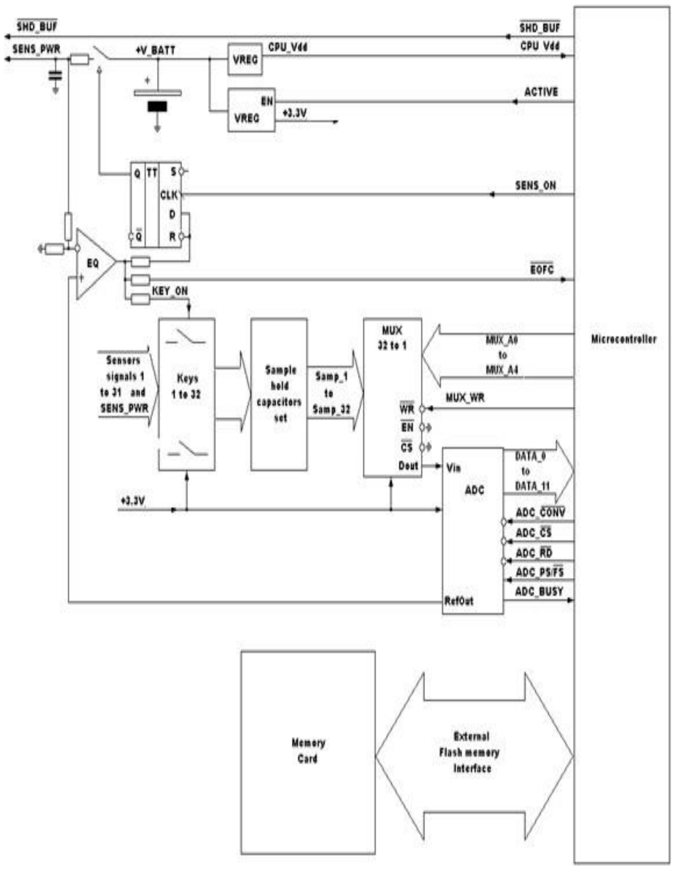


Figure. 1. Data acquisition system diagram

TABLE I. TO BE DEVELOPED SYSTEM REQUIREMENTS

Number of channels	31
Sampling rate	128 Hz
Battery life (in operation mode)	1800 hrs.
Operation temperature range	-40 to +85 C
Overall dimensions	minimize
System weight	minimize
Typical consumption of the set of sensors	400 mA @ 5.0 V
Sensor type	Passive, resistive
Memory type and size	32 Gb, Flash
Data stored	Sensor reading and time stamp
External interface	USB

III. FEASIBILITY ANALYSIS AND SOLUTION APPROACH

Since the system is battery-powered and should provide long battery life, its power consumption becomes the determining factor for setting the system structure. The size and weight of the battery are limited as well, so the available battery of acceptable size is a pack of 4 cells (3.6 V, 2600 mAh, series-parallel), that is, 5200 mAh.

Typically, the capacity of the battery is limited by its discharge to approx. 5.5V is around 60-65% of the total and is actually 3120-3380 mAh. That yields, that the average current consumption should be within 1.73 -1.88 mA (for 1800 hours of operation). The desired low average current could only be achieved if the system is operated in pulse mode, completing all the necessary actions (A/D conversion, memory interfacing, etc.) in a small fraction of the sampling period, remaining in sleep the rest of the time.

The ratio of active/sleep mode of each element of the system will determine this element's average current, the sum of these, being the whole system average current, should be within the required limit. The second important point is, that the sensors used have a significantly different range of output signals (from volts to millivolts) on one hand and that the physical length of cables might be up to 1-1.5m. This makes the use of buffer amplifiers on sensors compulsory, to bring the signal levels to an acceptable range and avoid pickup on signal lines by providing low output impedance. More to the point, some types of sensors have differential output, while the common-mode voltage is high enough – that means, two types of buffering amplifiers are to be designed.

At the first stage of this work, however, to speed up the development, it was planned to test the system with comparatively simple non-inverting amplifiers and imitation of sensor signals from resistor dividers. After completing the conversion, the ADC is brought to the full-sleep mode by the microcontroller, bringing its consumption from 3.3mA in active mode down to 1uA.

IV. SYSTEM DESCRIPTION

This section will be presenting the design methodology and data processing control adopted in this work as illustrated by the following subsections.

A. *Measurement Method*

In order to provide the shortest possible sampling time, the sensors are powered dynamically (with exponentially rising voltage). During this time the keys remain closed, and sensor voltages are transferred to sample holding capacitors as shown in Figure 1.

It is assumed, that due to the passive and purely resistive nature of the sensors, their response is directly proportional to powering voltage. As soon as the powering voltage is applied to the inverting input of the high-speed comparator via the divider to achieve the threshold, the output of the comparator goes low, opening the keys, thus disconnecting sample holding capacitors from the signal lines, and at the same time switching off the sensor power by resetting the flip-flop.

Signal end of conversion (EOFC) is provided to MCU to indicate that the A/D conversion of the voltages present on sample holding capacitors may begin. To avoid any errors related to comparator delay time and threshold variation, the sensor power voltage is sampled in addition to sensor signals to provide the possibility to recalculate the obtained readings.

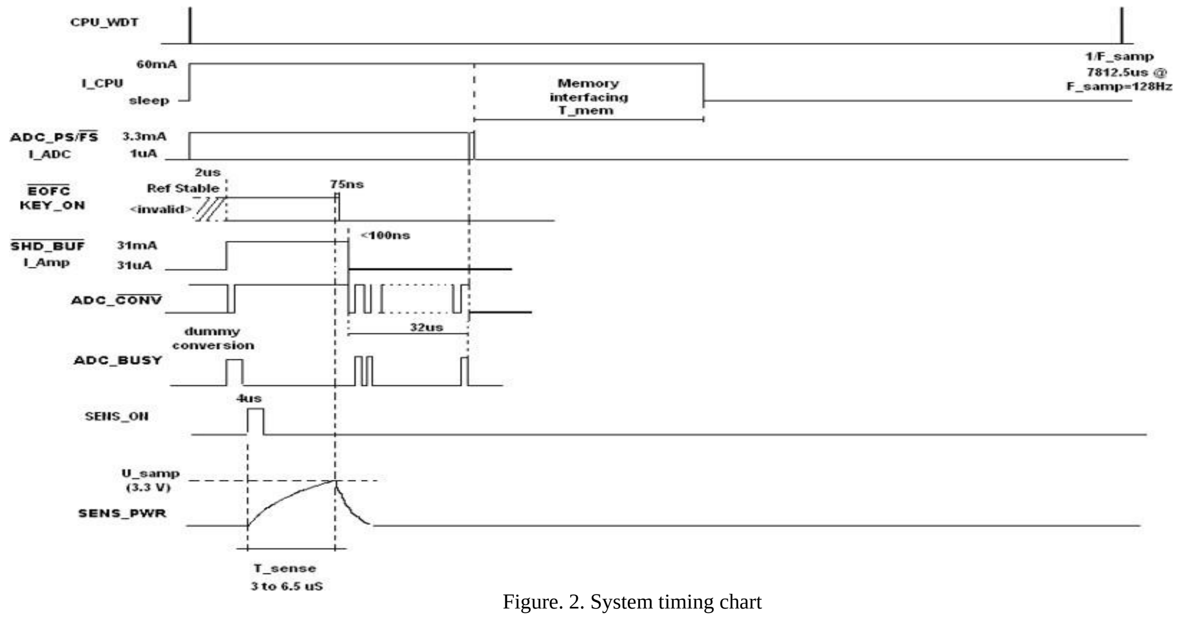


Figure. 2. System timing chart

B. *Considerations for A/D conversion*

Since the time for the conversion should be minimized, usage of ADC built-in to the microcontroller is inapplicable in principle, due to the very long conversion time. The (EOFC) signal is provided to MCU to indicate that the A/D conversion of the voltages present on sample holding capacitors may begin.

To achieve acceptable time from the structure depicted in Figure 1, The voltages, obtained on sampling capacitors, are applied via 32-to-1 multiplexor to a fast (800ns) ADC. The output code is a 12-bit parallel, which allows the controller to get one conversion result in one read instruction. That means, 32 words of data could be obtained in 32us. Additionally, the ADC used has its in-built clock source and stable reference voltage (used to provide threshold for comparator) as given in the system timing chart in Figure 2.

C. *Considerations for microcontroller choice*

The microcontroller should be able to accept the data from the ADC without delay and accept and generate all the necessary control signals without slowing down A/D conversion (that determines its minimum parallel bus speed). However, stricter requirement arises out of necessity to transfer the data to external memory. The data amount for a single transfer is 33 words of 16-bit (32 data and timestamp from RTC). This also requires choosing the fastest available interfacing to external memory.

V. Structural Design

The DLS is designed to be as compact as possible. The original concept, shown below in Figure 3, incorporates two separate modules, the Main module, and the Battery-Memory module.

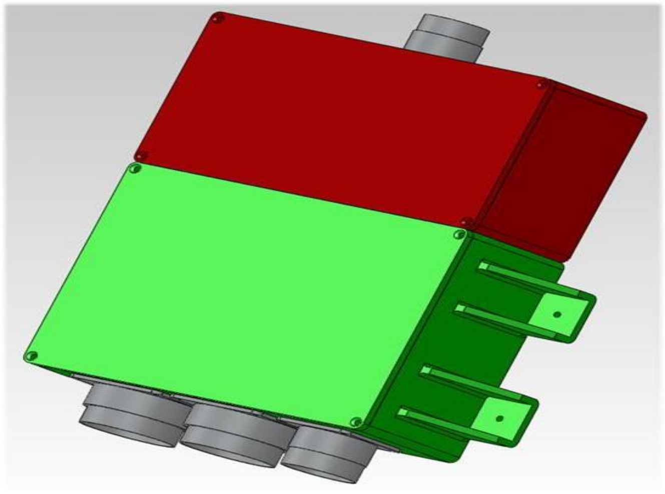


Figure. 3. DLS structural design

Now that the preliminary battery selection and media storage selection was performed and know the approximate volume requirements, a new concept design was proposed. The idea of the concept is that the battery cell can be removed for charging and the media storage also can be removed for data download as shown in Figure 4.

VI. Data Acquisition Verification and Testing Results

To verify the performance of both data logger systems a number of resistive potentiometers of various ranges were used as dummy sensors. From a design perspective, the main focus is to get the dynamic sensors' power circuitry working as desired and then extract the necessary ADC waveforms and match them with the device datasheet. Such practice was useful in debugging and testing to overcome many issues before getting the system prototype functioning as expected. Figure 5 shows some of the sensors' power and conversion signals, where obviously they are very comparable with the ones in Figure 2.

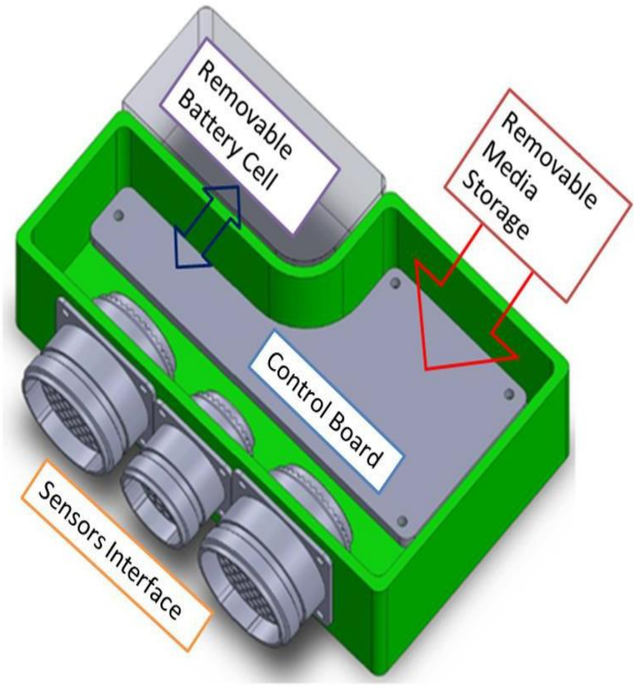


Figure. 4. Actual DLS design with power and data storage

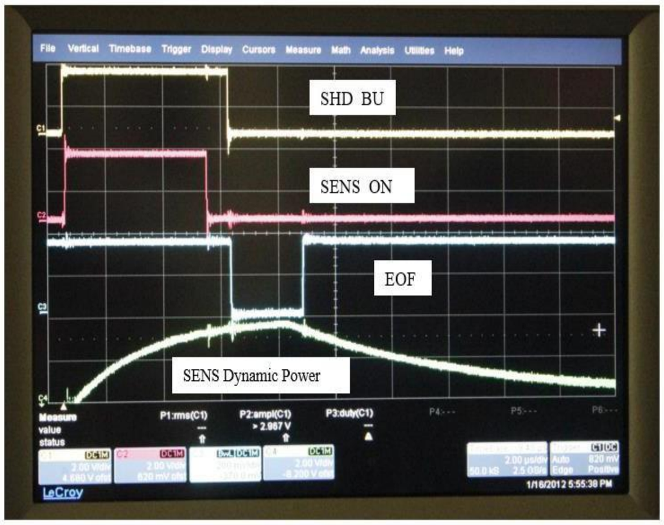


Figure. 5. Dynamic power signals of DLS system-Power unit

A. *Integration with Industrial Sensor and Test Results*

The next phase of the testing process was to use a real industrial sensor interfaced to the Design system and use the developed Graphical Unser Interface (GUI) to monitor and record the multi-channel sensors' data acquisition via serial interface, these sensors are:

- Thermocouple temperature sensor
- Linear Displacement Potentiometer sensor (0-300mm)
- Pressure sensor (5 Ins max)
- Load cell strain gauge sensor

An interface board was manufactured with sensors mounted and all signals and power necessary connections are provided as shown in Figure 6.

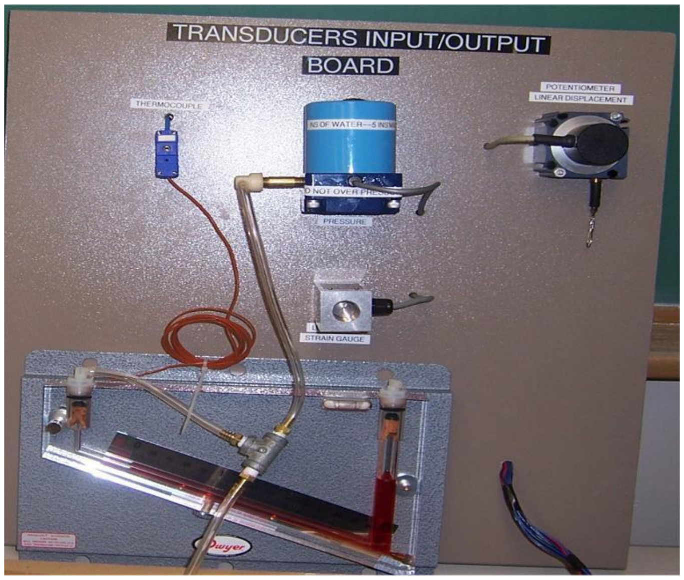


Figure. 6. Industrial sensor system setup using DLS

The DLS system performance was tested and verified using the sensors interface board and sensors readings were displayed by the developed GUI. Special single-ended input and differential amplifiers were designed for low-level sensor signals, like thermocouples and strain gauge sensors.

The enclosure of the DLS was designed using rapid prototyping. Figure 7 shows the side and top views of the designed enclosure where it can be seen the two main units are attached to fit both the system PCBs and battery.

The main functionality of the developed GUI is to show the incoming data from the DLS unit. The programming language used for programming this GUI was Visual Basic.NET from the Microsoft Visual Studio.NET package. The communication is done through COM ports and the data is sent in HEX format in a specific format (frame) which then is read in GUI and interpreted (parsed). The data is shown in a master graph for each channel as given in Figure 8 where each individual graph can be added and associated with specific given user options that can be set to modify the visual display of the GUI.

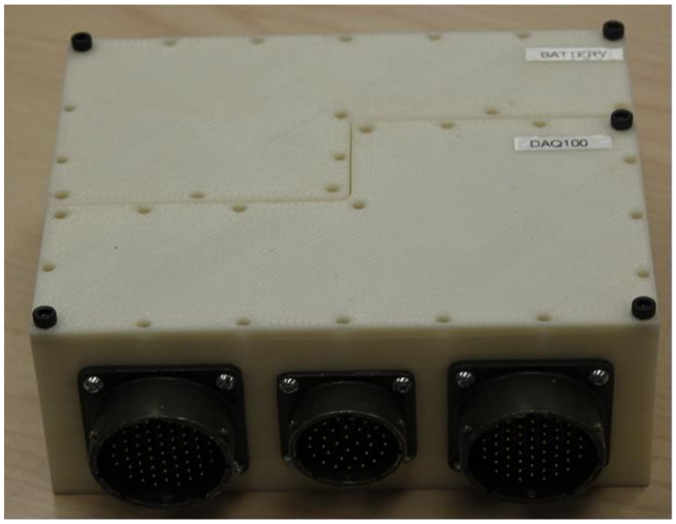
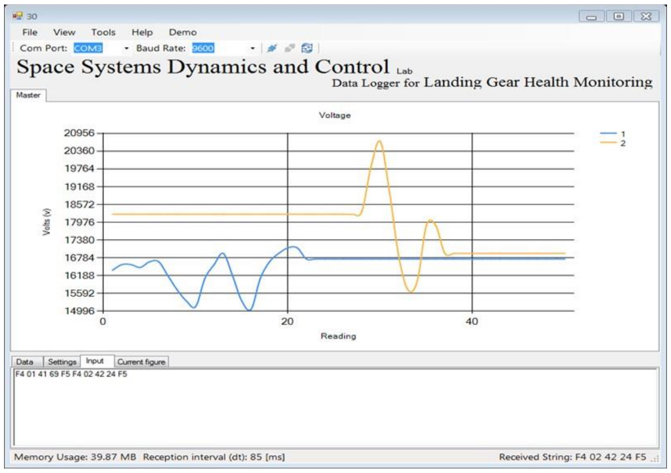


Figure. 7. DLS System enclosure



Com Port: COM3 Baud Rate: 9600

Space Systems Dynamics and Control Lab

Data Logger for Landing Gear Health Monitoring

```
F4 01 41 69 F5 F4 02 42 24 F5
```

Memory Usage: 39.87 MB Reception interval (dt): 85 [ms] Received String: F4 02 42 24 F5

Figure. 8. Developed GUI to monitor the sensor readings recorded by DLS system.

B. *Measuring DLS power consumption*

The average current, consumption by the system is the sum of all quiescent currents of constantly powered elements (CPU in sleep mode (max.25uA), comparator (90uA max), flip-flop (2uA max) and two voltage regulators (2x35=70uA max)), and the average current of the components powered for a short time. Using the time chart, the following equation can be obtained :

$$I_{avg} = 187\text{uA} + \frac{I_{adc}T_{adc} + I_{amp1}T_{amp1+Isens} + \left(T_{adc} + T_{interf}\right) * I_{CPU}}{T_{sample}}$$

The equivalent time for sensor activity (under the condition of powering by constant 5V voltage) is less than 2us. (at 400mA current)

Then, at $T_{sample} = 7812.5us$,

$$I_{avg} = 187\text{uA} + \frac{3.3mA * 40us + 31mA * 4us + 400mA * 2us + \left(40us * T_{interf}\right) * 60mA}{7812.5\text{us}}$$

$$= 622\text{uA} + 60\text{mA}\frac{T_{interf}}{7812.5\text{us}}$$

That guarantees, that for interface time that is less than 143us, the average system current will remain within the limit. At the same time, if interfacing is fast enough, it is possible to reduce the speed of the system (for better accuracy and reliability).

The DLS current consumption was measured using in-circuit metering for two options, with and without pulse-mode operation concept as shown in Table II.

TABLE II. PEAK CURRENT CONSUMPTION MEASURING OF THE DLS

Operating option	Peak current draw (mA)
With Pulse-mode	0.714
Without Pulse-mode	2.26

VII. CONCLUSIONS

In this work, a novel sensor data logging system has been introduced. The system design is utilizing the pulse-mode dynamic power concept where output sensors are sampled through switched capacitors and then sensor power is turned off to start the ADC process. It was found the system power has been significantly reduced by more than (3:1) as can be seen in Table I to meet the requirements of low power consumption and extended battery lifetime.

The overall DLS design has been placed in a developed modular housing that incorporated the power source and storage components. A GUI has been developed to test the system output signals using industrial-rated sensors.

Planning for future work will be involving future testing on the sensing side and improving the power consumption performance. Also, the GUI is to be further developed to include more features on system-measured parameters.

REFERENCES

[1] D. A. Dudina, V. A. Vasiliev, and E. S. Mandrakov, "Smart Tool for Tracing Humidity and Temperature of Products During Transportation," 2021 International Conference on Quality Management, Transport and Information Security, Information Technologies (IT&QM&IS), 2021, pp. 279-281.

[2] J. Kang, K. Choi, Y. Kim, and H. Yang, "A Method of Integrating Information for SWIM," 2017 IEEE 13th International Symposium on Autonomous Decentralized System (ISADS), 2017, pp. 195-198.

[3] S. M. McGovern, and K. F. Chin, "Portable avionics test suite design and operation," Digital Avionics Systems Conference, 2003. DASC '03. The 22nd, 2003, pp. 4-7.

[4] L. Leilei, W. Hongxin, X. Yubing, Y. Zhenshan, Y. Chenyi, and Y. Fan, "Research and development for landing gear test interface unit for one type aircraft," CSAA/IET International Conference on Aircraft Utility Systems (AUS 2018), Guiyang, 2018, pp. 216-219.

[5] Delebarre, Grondel, Dupont, Rouvarel, and Yoshida, "Wireless monitoring system for lightweight aircraft landing gear," 2017 International Conference on Research and Education in Mechatronics (REM), Wolfenbuettel, Germany, 2017, pp. 1-6.

[6] S. Yang, M. Crisp, R. V. Penty, and I. H. White, "RFID Enabled Health Monitoring System for Aircraft Landing Gear," in IEEE Journal of Radio Frequency Identification, vol. 2, no. 3, pp. 159-169, Sept. 2018.