ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-THPDP060

BEAM INSTRUMENTATION SIMULATION IN PYTHON

M. Gonzalez-Berges*, D. Alves, A. Boccardi, V. Chariton, I. Degl'Innocenti, S. Jackson, J. Martínez Samblas, European Organization for Nuclear Research (CERN), Geneva, Switzerland

Abstract

The design of acquisition electronics for particle accelerator systems relies on simulations in various domains. System level simulation frameworks can integrate the results of specific tools with analytical models and stochastic analysis. This allows the designer to estimate the performance of different architectures, compare the results, and ultimately optimise the design. These simulation frameworks are often made of custom scripts for specific designs, which are hard to share or reuse. Adopting a standard interface for modular components can address these issues. Also, providing a graphical interface where these components can be easily configured, connected and the results visualised, eases the creation of simulations. This paper identifies which characteristics ISPy (Instrumentation Simulation in Python) should fulfil as a simulation framework. It subsequently proposes a standard format for signal-processing simulation modules. Existing environments which allow script integration and an intuitive graphical interface have then been evaluated and the KNIME Analytics Platform was the proposed solution. Additionally, the need to handle parameter sweeps for any parameter of the simulation, and the need for a bespoke visualisation tool will be discussed. Python has been chosen for all of these developments due to its flexibility and its wide adoption in the scientific community. The ensuing performance of the tool will also be discussed.

INTRODUCTION

The design of electronic acquisition systems for beam instrumentation can be a lengthy process implying several steps in which the designer has to optimize different parameters. The design steps typically cover architecture definition, component selection, algorithm selection and performance estimation. For systems of a certain size, the number of parameters is not manageable directly, hence simulation tools are required to assist in producing systems that meet the specifications. A number of Python scripts where developed in the past within the CERN Beam Instrumentation (BI) group for each of the individual steps mentioned above. This paper will present the effort to have a simulation tool integrating the functionality in those scripts and extending it to cover further needs.

SIMULATION FRAMEWORK REQUIREMENTS

Several high level requirements were identified for the simulation tool. The tool should be based on components representing the various elements of the acquisition system (e.g. beam characteristics, filters, cables, amplifiers). Users should have the ability to extend the tool with new components without needing detailed knowledge of the tool internal workings.

Users will interact with the tool through a graphical interface, allowing them to describe simulations by connecting the different components, configuring their properties and initiating simulations. The resulting data will be stored in files, along with the simulation schematic and the parameter configuration, ensuring proper tracing of each dataset. Additionally, a visualisation tool will be provided, offering diverse views of the simulation results, complete with selection and filtering capabilities to facilitate result interpretation.

The tools and libraries employed for the implementation should be open source, encouraging collaboration and long-term sustainability. Furthermore, the definition of the simulations and their execution results should be as independent as possible of the specific tools used for their creation. This design principle will enable easy integration with other tools and facilitate the future evolution of the simulation tool.

There was already an existing set of Python scripts for partial simulations where a considerable effort had been invested. Providing methods to reuse these scripts either directly in a Python implementation or in any other compatible way would be necessary.

CHOICE OF ENVIRONMENT

Given the requirements presented in the previous chapter, we considered several possibilities for the actual implementation. An initial idea was to develop a new tool based on Python and PyQt. These are well known in our team and there are many libraries that could be reused for the data processing, storage and visualisation. Another possible approach was to use an existing environment that could be extended to cover our simulations. We identified two possible tools: the KNIME analytics platform [1] and Kepler [2].

After a detailed technical study, it was clear it would have been possible to implement the tool with any of the three proposed solutions. However, using PyQt would have meant a larger development effort and future maintenance. In addition there would not have been any synergies with similar tools.

Out of the two existing platforms, Kepler seemed to have a smaller and less dynamic community. Additionally, the integration with Python seemed somehow more complicated than in KNIME. These points lead us to the selection of KNIME.

* manuel.gonzalez@cern.ch

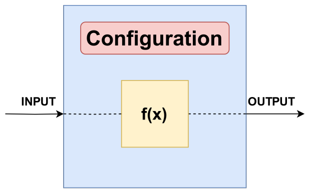


Figure 1: A configurable component that takes an input, executes either a deterministic or stochastic Python function, and subsequently produces an output. The input and output data are stored in binary format on disk.

INTEGRATION IN KNIME

While primarily an open-source data analytics and integration tool, the KNIME Analytics Platform can also enable the easy definition of customised and reusable graphical blocks that can configure and execute Python code. We selected version 4.6.5 as the foundation for building ISPy, ensuring compatibility with the CERN computing environment.

In the development process, the first step involved standardizing the definition of a component, where we established the required structure for source and configuration files to create distinct and reusable components representing the simulated instruments. Figure 1 provides a visual representation illustrating this concept.

To accommodate the users of the simulation tool, the development process put in place ensures that designers can effortlessly create new component types and users can easily utilize them to build simulations. The collection of these components constitutes the ISPyLibrary and resides in a Git repository, enabling updates and version control for the components.

Simultaneously, we faced the challenge of ensuring that ISPy could seamlessly integrate with KNIME while remaining functional as a standalone tool. To address potential issues such as platform inactivity or discontinued support, we had to take extra precautions, which added complexity to the development process but ultimately strengthened the resilience of the simulation framework. Hence, we chose to configure components using reproducible XML-XSD pairs, enabling straightforward configuration and improved component validation both within and outside the KNIME platform. Similarly, the configuration of a simulation chain, consisting of multiple components, is defined using an XML document.

Regarding the transmission of computed data from one component to another, we faced a choice between using a KNIME-specific format, known as KNIME data tables, and simply storing the data in binary format on disk. Performance tests clearly showed that the latter is faster. By storing the data in binary format on the disk, we also ensure that all component outputs are retained until manually removed. This approach facilitates access to simulation data at any time but comes with a trade off in terms of disk space usage, as all component data is preserved and not automatically deleted after component execution.

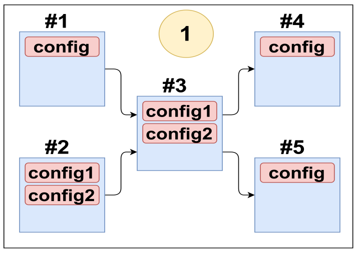
After configuring and executing a simulation chain, data is systematically saved in an individual directory, named according to the time of execution. This data can be loaded, plotted, or deleted using dedicated pre-made or customised components. KNIME provides a wide array of visualisation options within its Node Repository, including line, scatter, histogram, and various other types of plots. However, these options have demonstrated performance limitations when dealing with larger array sizes. To enable the rapid visualisation of larger signal sizes we developed a KNIME custom component that utilizes the Matplotlib library [3]. This custom visualisation component can also serve as a probe, streamlining the debugging process for newly designed chains. Finally, for comprehensive visualisation and analysis of an entire simulation chain, it is recommended to utilize the standalone Data Analysis and Visualisation Tool (DAVIT), which will be discussed later.

PARAMETER SWEEP SIMULATIONS

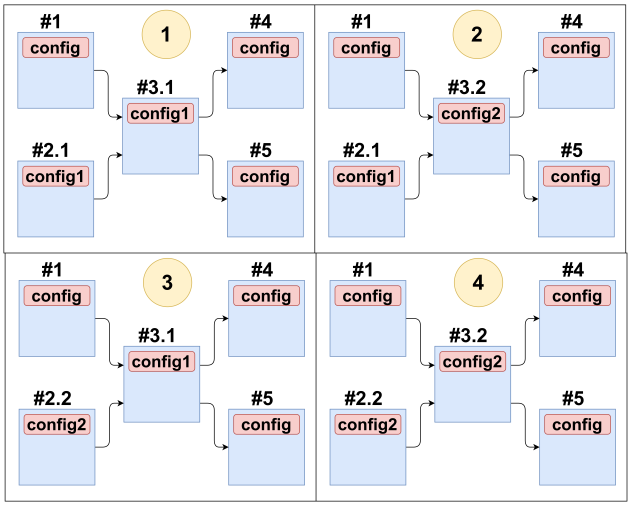
A parameter sweep is a computational technique used in a wide range of simulations to systematically vary input parameters across specified ranges to find an optimal design according to some criteria. In the case of ISPy, a parameter sweep simulation is executed for a chain that consists of components that are uniquely configured multiple times during a single execution. Every component has multiple parameters that can be swept, and all combinations of values need to be considered. The subsequent components can also be further configured multiple times, which leads to an increase in the number of uniquely configured chains. Essentially, for every new component configuration, a new branch of simulations is created.

Figure 2 shows an example of a configuration with parameter sweeps. Within this chain, the second and third components undergo multiple configurations, indicating that they are set up with different parameters more than once. As a result, the original simulation chain is divided into four distinct chains, each demanding separate execution as illustrated in Fig. 3. The total number of unique resulting chains is determined by multiplying the number of configurations chosen for each component.

By examining the interconnections among components defined in the simulation chain's configuration file, we can pinpoint the dependencies of each process and construct the resulting process trees. This involves implementing a modified *Depth First Search (DFS)* algorithm [4] on a directed graph. We initiate this DFS from nodes executed later in the simulation and work our way back to the initial nodes responsible for the simulation's execution. During the DFS traversal, previously visited nodes are continually pushed onto the stack to maintain the correct execution sequence. After completing the search paths, they are reversed, and any revisited nodes are removed. This process generates multiple execution paths for uniquely configured chains. The final step involves identifying repetitions among these DFS paths to avoid re-executing identical sub-chains. The tables in Fig. 3 illustrate the resulting dependencies and process

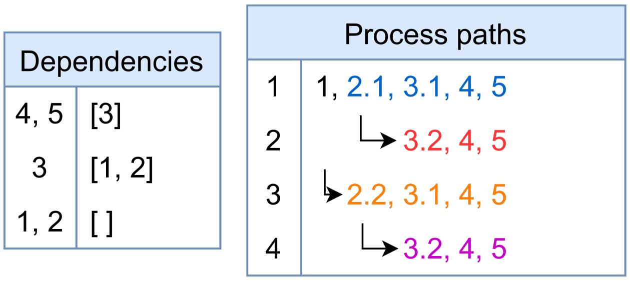


(a) Example chain of components.



(b) Expansion into 4 uniquely configured chains.

Figure 2: Example chain that is configured multiple times.



Dependencies	
4, 5	[3]
3	[1, 2]
1, 2	[ ]

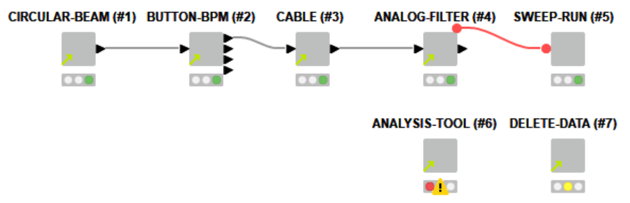
Process paths	
1	1, 2.1, 3.1, 4, 5
2	↳ 3.2, 4, 5
3	↳ 2.2, 3.1, 4, 5
4	↳ 3.2, 4, 5

Figure 3: Process dependencies and process paths. The colors indicate new simulation branches.

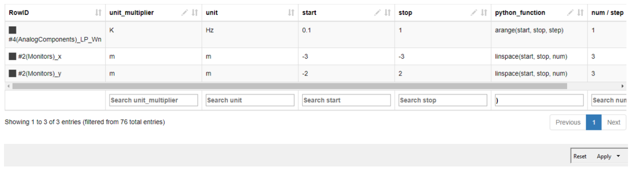
paths of the example in Fig. 2.

To facilitate the execution of a parameter sweep simulation, a custom component named SWEEP-RUN has been developed, which carries out the algorithm described above along with an intuitive UI. This component allows the variation of parameter values in either a logarithmic or linear fashion, or their arrangement within a specified range using a designated step size. Additionally, for user convenience there are options to generate parameters using a multiplication factor or filter them within the table using various criteria.

In Fig. 4, a practical example is presented, featuring a simulation alongside the UI table displaying the sweep parameter values. In this example, the horizontal and vertical positions of the beam with respect to the monitor are



(a) Simulation chain.



RowID	unit_multiplier	unit	start	stop	python_function	num / step
#4(AnalogComponents)_LP_Wn	K	Hz	0.1	1	arange(start, stop, step)	1
#2(Monitors)_x	m	m	-3	-3	linspace(start, stop, num)	3
#2(Monitors)_y	m	m	-2	2	linspace(start, stop, num)	3
	Search unit_multiplier	Search unit	Search start	Search stop	)	Search nun

Showing 1 to 3 of 3 entries (filtered from 76 total entries)

Previous 1 Next

Reset Apply

(b) Parameter sweep user interface. The parameters x and y correspond to the horizontal and vertical displacement of the particle beam with respect to the electrical center of the monitor, while LP_Wn represents the cut-off frequency of the low pass filter. For simplicity, certain columns are hidden from the table.

Figure 4: KNIME workflow designed for conducting a parameter sweep simulation.

swept linearly three times each, while the cut-off frequency is arranged from 0.1 to 1 kHz in increments of 0.1 kHz. This combination results in a total of $3 \times 3 \times 10 = 90$ unique configurations for the simulation chain. This example illustrates how a small amount of configurations can lead to a significant increase in workload.

STORAGE OF RESULTS

The results of the simulation executions are typically a set of scalar values and waveforms gathered at the end of the simulation chain. Values at selected intermediate points can also be included as part of the output data. Ideally one would like to keep all these results together and at the same time add any configuration parameters used as well as any other related metadata. This lead us to the use of the HDF5 format [5], which is designed to store large and diverse amounts of data in a single file. This suits very well the case for our tool, where the simulation can produce large and varied amounts of data. Additional functionality like caching or compression is also included in the extensive set of libraries available in different languages (e.g. C, C++, Java, Python)

Other important factors that lead us to choose this format for the results was its usage within the group in several other projects dealing with large amounts of data and the large community of scientific users behind its development and regular utilisation.

VISUALISATION OF RESULTS

To facilitate the visualisation of the numerous HDF5 files generated by the simulations, we extended a general purpose Python application, known as the Data Analysis and Visualisation Tool (DAVIT), that was developed previously in the group. Accessible both within ISPy and as a standalone application, this tool is designed to filter, analyse and examine the bundle of simulation outputs, presenting the data in a variety of formats including tables, scatter plots, and single or multi-axis line plots.

As illustrated in Fig. 5, the GUI features two principal panels. The left panel accommodates navigation through the internal structure of the hierarchical files, enabling the search of groups, datasets, attributes, and array dimensions. The right panel mainly focuses on visualisation, offering a range of options for graphical representation of the selected data. These options include FFT processing, mean removal, and interactive point inspection upon cursor hover.

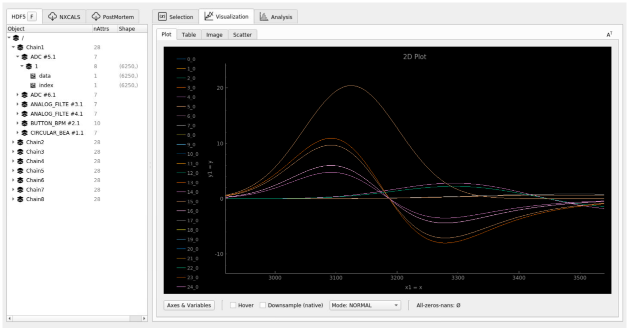
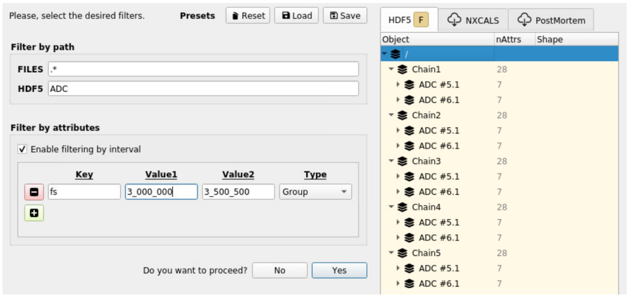


Figure 5: Screenshot of the Data Analysis and Visualisation Tool (DAVIT). The display shows the simulation results for all components across 8 different configuration chains.

A key feature of DAVIT is its generic approach to data processing. Internally, the tool transforms data into Python Pandas DataFrames [6], which enables operations such as transposition, slicing, and the merging of separate datasets. Moreover, the application provides the capability to apply filters to both file names and attributes, a feature that is especially useful for tracking variations in parameter sweeps.



Please, select the desired filters. **Presets** Reset Load Save

Filter by path

FILES	.*
HDF5	ADC

Filter by attributes

✓ Enable filtering by interval

Key	Value1	Value2	Type
fs	3_000_000	3_500_500	Group

Do you want to proceed? No Yes

HDF5 F NXCALS PostMortem

Object	nAttrs	Shape
/		
Chain1	28	
ADC #5.1	7	
ADC #6.1	7	
Chain2	28	
ADC #5.1	7	
ADC #6.1	7	
Chain3	28	
ADC #5.1	7	
ADC #6.1	7	
Chain4	28	
ADC #5.1	7	
ADC #6.1	7	
Chain5	28	
ADC #5.1	7	
ADC #6.1	7	

Figure 6: Filtering use case. The left panel displays available filter options, while the right window shows the HDF5 file explorer post-filtering.

Figure 6 illustrates an example of filter usage. It demonstrates how a user can easily filter simulations to focus on the ADC component given a sampling frequency sweep that ranges from 3 MHz to 3.5 MHz. Note that these configured filters can also be saved as presets, thereby facilitating the effortless replication of specific visualisations in future analyses.

PERFORMANCE - EXAMPLE

The simulation tool can be used to perform a parameter sweep simulation of a beam position monitor (BPM) system's acquisition electronics. The BPM system is an instrument measuring the transverse position of the particle beam in an accelerator. The acquisition part of the system simulated in this example is a direct digitisation system, in which the analog conditioning electronics are minimalistic and the monitor waveforms are sampled at high sampling rates, allowing most of the signal processing to be performed in the digital domain. Figure 7 shows a screenshot of the simulation chain in KNIME. Most of the modules referring to BPMs and electronics are based on existing simulation code developed in the BI group and described in detail in this thesis [7].

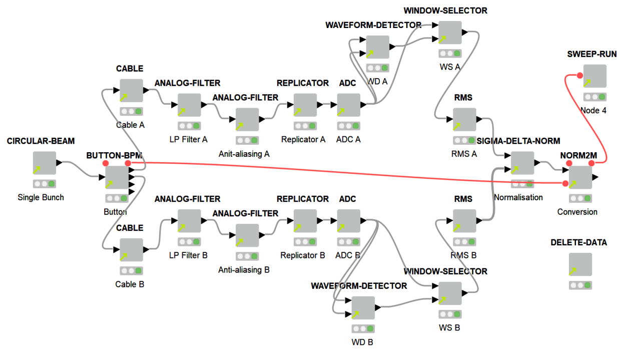


Figure 7: Example of an ISPy simulation chain in KNIME, simulating the acquisition electronics of a BPM.

The first module from the left generates the beam current signal for a circular machine, with parameters such as the number of charges, revolution frequency, number of turns, harmonic number, etc. It is based on an analytic Gaussian model of the beam.

The second module is the button BPM, producing four voltages in response to the beam current. It is parametrised by the button geometry and by the beam's relative transverse position. For the sake of clarity, only the first two of the four outputs, linked to the horizontal plane, are further shown in the diagram. Two identical processing chains follow.

First a cable module, using an analytic parametric model to simulate the effect of dispersion based on the cable length, geometry and material properties. Then two analogue filters, a shaping low pass filter and an anti-aliasing filter; the same module was used but with different configurations for bandwidth and filter order. The module is a wrap of the Bessel function of the Python SciPy signal module [8].

The replicator module generates a parametric number of identical replicas of the input signal, 1000 in this example. This step is needed to feed stochastic simulation modules, which have models based on stochastic functions. The following ADC module is an example of a stochastic module: this module, on top of sampling the input signal in function of the parametric sampling rate, models the converter noise based on the effective number of bits and the uncertainty in the sampling phase. The noise is described by a white Gaussian process and the phase by a uniformly distributed function within the sampling period. Hence, to each of the 1000 input replicas, a different sampling phase and a different noise sequence are applied.

The waveform detector performs a waveform detection algorithm to identify the first sample to use for the following digital processing. The window selector module provides for each replica a selection of points with a parametric offset and length with respect to the identified trigger. Finally, the Root-Mean-Square (RMS) module computes a single power result per replica which is combined with the result of the other electrode by the sigma-delta normalisation module. The normalised value is then converted into a beam position result by the NORM2M module, which takes its configuration from the BUTTON-BPM configuration.

The output is a sequence of 1000 positions, simulating a distribution of possible resulting positions for the defined acquisition chain. The mean value and the standard deviation of this distribution provide an estimation of the offset and resolution performance of the acquisition system.

The simulation is configured to be a parameter sweep simulation by the SWEEP-RUN module: the beam horizontal position is swept with three possible values and the beam number of charges is configured with a series of 5 values. So the simulation is run outside of KNIME with all the 15 combinations of swept parameters, providing a description of how the system's measurement offset and resolution vary with the beam position and intensity. The result, consisting in a folder containing all the configuration files and output signals, is 735 MB of data, produced in less than 4 seconds.

FUTURE WORK

Most of the ISPy core modules have been used for the analysis and initial design of BPM system prototypes for the future LHC upgrade [9], proving its potential and extending its initial development into its current state. During those studies BPM related libraries have been developed and added to the ISPy toolset. Further development of those libraries and the creation of new ones will happen naturally with the adoption of ISPy for more projects, increasing its potential and capabilities with use. One of the next projects to adopt ISPy is the consolidation and upgrade of the AWAKE BPM system [10]. There ISPy will be used to evaluate the potential improvement of the performance of the proton-line BPMs using an RFSoC based DAQ. ISPy will also be used to assess the compatibility of such DAQ with the down mixing front-end system currently used in the electron-line BPM system designed by TRIUMF [11, 12], in an effort to have a more standardised architecture.

In addition to BPM applications, ISPy is expected to play an important role in validating the developments related to the renovation of the front-end electronics, calibration and DAQ systems for the fast beam current transformers foreseen during the 3rd long shutdown of the CERN PS accelerator complex. In particular, it will enable the estimation of the accuracy and signal-to-noise ratio (SNR) of the beam current and intensity measurements used by machine operators to optimize the quality of the beams delivered to the various experiments.

These additional systems will have a threefold impact on ISPy: validation of its flexibility, expansion of the user community seeding the potential for its use outside CERN (the AWAKE studies will be done in in collaboration with the Diamond Light Source), and development of new digital and analogue modules.

CONCLUSION

We have presented the development and application of Instrumentation Simulation in Python (ISPy) as a comprehensive simulation framework for data acquisition systems. The need for such a tool arises from the complex and iterative process of designing acquisition electronics, which involves optimizing an increasing number of parameters.

ISpy offers modularity, a user-friendly interface, and efficient result storage. Built on open-source principles, ISPy maintains adaptability to future tools. Leveraging the KNIME Analytics Platform, it handles parameter sweep simulations effectively. ISPy's use of the HDF5 format streamlines data storage, and the Data & Visualisation Tool simplifies result exploration. It has been demonstrated through a BPM acquisition system example, offering insights into system performance.

Future work includes library expansion, community growth, and application in diverse projects. Although mainly driven by beam instrumentation systems, the tools is generic enough to be used in other environments.

REFERENCES

[1] Knime Analytics Platform, http://www.knime.com

[2] The Kepler Project, https://kepler-project.org/

[3] Matplotlib: A 2D Graphics Environment, https://matplotlib.org

[4] Depth-first search algorithm, https://en.wikipedia.org/wiki/Depth-first_search

[5] The HDF Group, https://www.hdfgroup.org/

[6] Pandas Documentation, https://pandas.pydata.org/docs/index.html

[7] I. Degl'Innocenti, "Direct Digitisation for Position Measurement of Charged Particle Beams: Sampling and Quantisation Effects on Resolution", Ph.D. thesis, CERN, 2021, https://cds.cern.ch/record/2775642

[8] Scipy documentation, https://docs.scipy.org/doc/scipy/

[9] I. Degl, A. Boccardi, L. Fanucci, and M. Wendt, "Direct Digitization and ADC Parameter Trade-off for Bunch-by-Bunch Signal Processing", in *Proc. IBIC'20*, Santos, Brazil, Sep. 2020, pp. 288–294. doi:10.18429/JACoW-IBIC2020-FRAO02

[10] E. Gschwendtner *et al.*, "The AWAKE Run 2 Programme and Beyond.", *Symmetry*, vol. 14, no. 8, p. 1680, 2022. doi:10.3390/sym14081680

[11] V. A. Verzilov *et al.*, “Status of Beam Diagnostic Systems for TRIUMF Electron Linac”, in *Proc. IBIC'13*, Oxford, UK, Sep. 2013, paper TUPC06, pp. 361–364.

[12] M. Barros Marin *et al.*, “Performance of the AWAKE Proton Beam Line Beam Position Measurement System at CERN”, in *Proc. IBIC'17*, Grand Rapids, MI, USA, Aug. 2017, pp. 209–212. doi:10.18429/JACoW-IBIC2017-TUPCF06