D-Matrix: An Approach to Generic Data Acquisition System Design for Nuclear and Particle Physics Experiment

Authors: Zhengyang Sun, Dr. Junfeng Yang, Zeng, Jinrui, Sun, Mr. Ke, Li, Mr. Yi, Li, Miss Lun, Zilong Xiong, Bi, Miss Yulin, Huang, Mr. Yiting, Yang, Dr. Junfeng

Date: 2025-11-25T00:00:00+00:00

Abstract

Data acquisition (DAQ) systems face significant challenges in addressing the high-throughput and real-time processing demands of next-generation nuclear and particle physics experiments. Generic platforms offer solutions to reduce costs and improve efficiency but require innovations to address heterogeneous data formats, hardware interfaces, and processing requirements. This paper introduces the SHARE principles (Standardized, Heterogeneous, Adaptable, Reusable, and Extensible) for generic DAQ systems. Guided by these principles, we proposes D-Matrix, a generic firmware-software co-designed stream processing platform. The key components of D-Matrix include: (1) A standardized data protocol that abstracts spatiotemporal properties to accommodate diverse detector payloads and processing requirements; (2) A hardware abstraction layer that standardizes access to heterogeneous resources and interfaces; and (3) Cascadable and configurable processing modules deployable across these resources under a unified execution model. This platform enables flexible construction of DAQ workflows and the potential for automated design generation. The DAQ system built on this platform has been successfully deployed at the HIRFL-CSR External Target Experiment (CEE), achieving a throughput of 73.6 Gbps and a processing rate of 20,890 events per second. By establishing a foundation of standardization and generalization across data, hardware, and processing paradigms, D-Matrix represents a significant step towards highly usable and generic heterogeneous DAQ systems.

Full Text

Preamble

D-Matrix: An Approach to Generic Data Acquisition System Design for Nuclear and Particle Physics Experiments

Zheng-Yang Sun,[1,2] Jun-Feng Yang,[1,2,†] Jin-Rui Zeng,[1,2] Ke Sun,[1,2] Yi Li,[1,2] Lun Li,[1,2] Zi-Long Xiong,[1,2] Yu-Lin Bi,[1,2] and Yi-Ting Huang[1,2]

[1]State Key Laboratory of Particle Detection and Electronics, University of Science and Technology of China, Hefei 230026, China
[2]Department of Modern Physics, University of Science and Technology of China, Hefei 230026, China

Data acquisition (DAQ) systems face significant challenges in addressing the high-throughput and real-time processing demands of next-generation nuclear and particle physics experiments. Generic platforms offer solutions to reduce costs and improve efficiency but require innovations to address heterogeneous data formats, hardware interfaces, and processing requirements. This paper introduces the SHARE principles (Standardized, Heterogeneous, Adaptable, Reusable, and Extensible) for generic DAQ systems. Guided by these principles, we propose D-Matrix, a generic firmware-software co-designed stream processing platform. The key components of D-Matrix include: (1) A standardized data protocol that abstracts spatiotemporal properties to accommodate diverse detector payloads and processing requirements; (2) A hardware abstraction layer that standardizes access to heterogeneous resources and interfaces; and (3) Cascadable and configurable processing modules deployable across these resources under a unified execution model. This platform enables flexible construction of DAQ workflows and the potential for automated design generation. The DAQ system built on this platform has been successfully deployed at the HIRFL-CSR External Target Experiment (CEE), achieving a throughput of 73.6 Gbps and a processing rate of 20,890 events per second. By establishing a foundation of standardization and generalization across data, hardware, and processing paradigms, D-Matrix represents a significant step towards highly usable and generic heterogeneous DAQ systems.

Keywords: Data acquisition, stream processing, unified data protocol, automated design, hardware abstraction

Introduction

DAQ systems are vital components in nuclear and particle physics experiments, functioning as central hubs for data aggregation, event building, operational control, and more. The rapid pace of experimental upgrades and the growing diversity of detector technologies strain custom-built DAQ systems [1, 2]. Frequent iterations demand costly, time-consuming custom DAQ development, which struggles to efficiently accommodate heterogeneous detectors and varying

physics requirements. Ultimately, this results in extended development timelines, high costs, and limited reuse of expertise, potentially slowing scientific progress.

A generic DAQ platform offers a flexible framework that addresses these issues. It significantly reduces development costs and time by enabling extensive reuse of standardized components (hardware interfaces, protocols, processing modules) across experiments and upgrades. This reuse enhances system maturity, reliability, and performance, while inherent flexibility allows rapid adaptation to new detectors and requirements, boosting scientific productivity [3].

A common and successful approach in modern DAQ systems employs a software-processing-oriented paradigm, where front-end hardware focuses on data collection and transparent transmission to server clusters for computational tasks. This approach builds on standardized components to enhance generality through generic transport channels and unified software frameworks. Generic transport channels, such as GBT (Gigabit Transceivers) [4] and FELIX (Front-End LInk eXchange) [5], reduce custom FPGA development while enhancing system usability and optimizing performance by establishing transparent data transportation pathways from front-end electronics to software processing units. Unified software frameworks like artdaq [6], xDAQ [7], and FairMQ [8] abstract detector-specific complexities and enable algorithm reuse across experiments by defining common data containers and providing a unified approach to constructing generic data processing units.

A key motivation for adopting this paradigm is to reduce the development complexity inherent in FPGA-based processing. Shifting computational loads to software frameworks can accelerate deployment cycles and simplify algorithm maintenance. However, this approach can also increase infrastructure costs, as it typically demands high-performance computational resources and substantial bandwidth capacity. From the perspective of achieving full generality, these established approaches have made significant strides, yet opportunities for further integration remain. A key area for potential enhancement lies in achieving a more complete and synergistic integration between FPGA and server software components. Such deeper integration would allow for a more optimal distribution of processing tasks, potentially better leveraging the inherent parallelism and power efficiency of FPGAs alongside the flexibility of software frameworks.

Given these opportunities for deeper integration, we posit that the next evolutionary step involves establishing a more deeply integrated hardware/software co-design architecture. To guide this evolution, we first propose a more comprehensive definition of DAQ generality through the SHARE principles (Standardized, Heterogeneous, Adaptable, Reusable, and Extensible). To realize these principles, we have developed D-Matrix, a generic firmware-software co-designed stream processing platform. This platform provides a holistic solution by establishing a standardized data protocol, a hardware abstraction layer, and a modular stream processing paradigm, creating a cohesive and extensible DAQ ecosystem.

II. Design Philosophy

This section presents the SHARE generalization principles, along with an overview of the D-Matrix platform, its design philosophy, and motivations.

A. SHARE Principles

A clear definition of generality is a prerequisite for the systematic development of a generic DAQ platform. Thus, we propose a set of five foundational principles, encapsulated by the acronym SHARE. We argue that a generic DAQ platform should exhibit the following characteristics:

1. Standardized

Standardization is the essential pathway to generalization, requiring efforts across data, hardware, and processing aspects. In terms of data, it requires the definition of a globally unified data container and communication protocols. For hardware, it necessitates defining a standardized hardware abstract approach and hardware interconnection interfaces. Regarding processing, it calls for the establishment of a standardized processing module model and standardized communication interfaces between modules.

2. Heterogeneous

Modern large-scale experimental data acquisition and processing systems are invariably built on heterogeneous architectures, most commonly integrating front-end FPGAs and back-end CPU server clusters, potentially with GPU computing cards. A generic platform must provide support for these heterogeneous computing units, and ideally, manage and utilize them in a more unified approach. This represents a key limitation in the current state of general-purpose DAQ research. As mature processing and encapsulation solutions in software already satisfy the basic requirements for generalization, the challenge of unifying heterogeneous architecture usage within the DAQ context shifts primarily to FPGA development. Therefore, a key objective for a general-purpose DAQ system is to create abstracted encapsulations for FPGAs, enabling their processing to follow the same paradigms as CPU software, lowering the barrier for developers to leverage heterogeneous computing resources. Achieving this goal necessitates solving several challenges: FPGA hardware abstraction, interconnecting heterogeneous systems, establishing a general-purpose data processing foundation, and balancing efficiency.

While CPUs utilize mature standardized software abstraction layers (e.g., OS kernel drivers and high-performance runtime libraries) to isolate hardware specifics, FPGAs lack similar encapsulation standards, forcing developers to directly manage details of chips, boards, IP cores, and physical interfaces. Efficient and reliable data exchange requires resolving interface heterogeneity between FPGA-to-FPGA and FPGA-to-CPU interconnects. Enabling algorithm portability and reuse across detectors/tasks necessitates hardware-agnostic fine-grained data frame standards that decouple processing logic

from detector-specific formats. Finally, computational effectiveness must be preserved when enabling general processing capabilities, avoiding performance penalties imposed by high-level synthesis (HLS) tools.

3. Adaptable

Adaptability is the most immediate advantage afforded by a general-purpose design, manifested in several aspects. The platform must adapt to data formats, as different detectors may have varying data output formats, necessitating the design of a unified data container within the generic DAQ platform that also follows the principle of low protocol overhead to conserve precious data bandwidth in nuclear and particle physics experiments. It must also adapt to processing needs, as modern experiments can generally be categorized into triggered and trigger-less modes that often employ different processing workflows, with individual experiments typically having custom requirements. The generic DAQ platform must provide robust adaptability to accommodate these diverse and complex processing needs. Finally, it must adapt to scale and development process, being flexible and scalable to serve experiments of all sizes while supporting the entire development process, from building compact laboratory prototypes for individual detectors to full-scale engineering deployment in large experiments. Therefore, the generic DAQ platform should possess both the capability for large-scale expansion and the agility for rapid deployment of compact test systems.

4. Reusable

The reuse of components is fundamentally enabled by a widely recognized modular design paradigm, which can be implemented in two key aspects: the modular design of hardware boards and the modular design of processing logic (encompassing both software and firmware). To address the common issue of repetitive design in fields like FPGA logic development, a generic design needs to be transformed into truly reusable modules under defined standards or specifications. Such a design enhances reusability, which can be applied within a single experiment and, more importantly, across different experiments. Notably, upgrades to an experiment often coincide with hardware iterations, and in such scenarios, the existing hardware often remains functionally sound. Therefore, strong reusability can significantly reduce the costs associated with these experimental upgrades.

5. Extensible

Extensibility encompasses but is not limited to scalability in size and functionality. We envision a broader sense of extensibility for the generic DAQ platform. Building upon its standardized architecture, the platform should transcend its initial purpose as a dedicated data acquisition system and mature into an open and iterative ecosystem. This ecosystem empowers users to introduce new standards-compliant hardware, develop custom processing modules, and adopt emerging technologies.

B. Overview of D-Matrix Platform

To address the challenges in generalization and to realize the SHARE principles in a unified framework, this paper proposes D-Matrix (Fig. 1): a generic firmware-software co-designed stream processing DAQ platform, structured around three foundational layers. The protocol layer defines standard D-Matrix data frame containers that abstract heterogeneous detector payloads, abstracting spatiotemporal properties into stream data spaces and partitioning hierarchical data domains by granular attributes that directly govern module processing behaviors. The hardware abstraction layer implements a Board Support Package (BSP) and Multiple Point-to-Point (MPP) transmission model, standardizing access to heterogeneous compute resources and transport interfaces. The processing layer defines a model for configurable, reusable, and cascadable modules implemented across FPGA/software platforms and establishes standardized interfaces for inter-module communication within each platform. These foundational layers collectively enable two advanced capabilities: (1) custom DAQ system construction through modular cascading and domain configuration, and (2) the potential for computer-aided design automation. This work serves as a comprehensive architectural overview, explaining the core design philosophy and the overall coordination of the platform' s main layers, while the intricate technical details of individual components are detailed in companion papers [9–12].

C. Stream Processing Paradigm

In large-scale physical experiments, the DAQ system assumes critical responsibilities that include data transmission and aggregation, device access and control, and operational status monitoring. These functionalities are delivered through distinct information carriers: scientific data information, control and feedback information, and operational status information. This operational context naturally leads to the generation of diverse and mixed data, which stems fundamentally from the traditional system' s approach to data organization where the hardware unit serves as the primary packaging entity. A direct outcome of this approach is that a single data frame concurrently contains various elements originating from the same source unit. Such intrinsic data heterogeneity poses a key challenge to DAQ system generalization: excessive variability in data definition and composition makes standardized processing unachievable.

Moreover, this lack of abstraction standardization in data definition inherently prevents unified hardware support. FPGA modules require consistent, well-defined data structures to implement efficient processing pipelines. The intrinsic heterogeneity of mixed data frames violates this requirement, forcing custom-designed FPGA implementations for each specific data mixture. Consequently, system generalization is often limited to the transport architecture level. For instance, a transparent transport layer can be realized via encapsulated protocols, while deeper hardware-level integration remains fundamentally unattainable.

In scenarios with mixed data transmission, all information types share the same physical link while maintaining logical independence. To overcome this data mixture challenge, D-Matrix establishes the stream abstraction as independent information units carrying pure data. Each stream is dedicated to carrying one and only one of these types—for example, scientific data, commands, command feedback, status indicators, or urgent alerts. Furthermore, this purity extends beyond mere categorization by information type, as distinct metadata types within the scientific data stream necessitate their existence as separate streams.

The stream abstraction offers significant advantages. Due to its decoupled nature, the transmission and processing of each stream are largely isolated, minimizing mutual interference. This allows different spatiotemporal generation granularities for different streams, thereby reducing bandwidth waste caused by redundant data duplication in mixed data transmissions. Due to its pure nature, modules within a stream' s processing path focus exclusively on single responsibilities, enabling generalized processing through abstraction of both data and operations. Building on this stream abstraction alongside modular processing, the D-Matrix platform constructs DAQ systems by defining required processing modules within each stream' s execution path.

III. Standard Data Protocol

Based on the characteristics of DAQ data, this section details the data protocol definition adopted by the D-Matrix platform. Constructing a generic DAQ system necessitates a standardized definition for data containers. The incorporation of FPGAs into a generic, unified processing framework imposes stricter requirements on the granularity of data format specifications. However, achieving such generality and integration faces significant challenges arising from the inherent diversity of detector technologies and the divergent spatiotemporal granularities required across different processing stages.

A. Stream Data Space and Data Organization

A fundamental characteristic of scientific data in DAQ systems is its inherent spatiotemporal nature. Consequently, data points can be effectively represented within a two-dimensional matrix defined by time and space coordinates, where time and space constitute the matrix dimensions with scientific data values populating the corresponding points. The D-Matrix platform adopts this matrix-based representation by introducing a unified stream data space, explaining its name "Data Matrix." The data space of a stream comprises all points of interest: temporal occurrences (such as trigger identifiers in triggered systems or time-slice identifiers in trigger-less systems), spatial locations (represented by channel identifiers), and corresponding data values.

Several typical data organization schemes are categorized as follows. Data Array is used for continuous waveform sampling under trigger-less mode, where output data points are continuous in both temporal and spatial dimensions and are

processed as data arrays. Data Triad represents hit-producing detectors (e.g., Time-of-Flight detectors) that output sparse data structures containing elements such as coarse timestamps, channel identifiers, and fine time measurements, which are organized as data triads. Data Cluster describes waveform-recording detectors that output data in clusters, typically structured as a channel identifier followed by a timestamp and a contiguous series of data points representing the full signal waveform. Sub-stream is the fourth organization scheme, which is not specific to a particular detector type. During system processing scenarios such as aggregating diverse information streams (e.g., global event building) or cases where detailed data content is irrelevant, one or multiple data frames within a sub-stream are directly utilized as the data payload.

B. Data Domain

The DAQ data transmission and processing workflows inherently operate at multiple distinct levels, each exhibiting well-defined physical spatiotemporal significance. As illustrated in Fig. 2 [Figure 2: see original paper], the spatiotemporally hierarchical data space is partitioned into nested regions, which constitute formally defined data domains. A key characteristic of this model is the nested relationship between data domains, where a father data domain (representing a coarser spatiotemporal scale) encompasses and aggregates multiple child data domains. Crucially, the raw data points themselves remain unchanged as they are interpreted within these different domains; what evolves is the spatiotemporal context and granularity used to interpret them.

This abstraction effectively separates the immutable data payload from dynamic processing context. For example, during front-end electronics packaging, temporal resolution operates at fine-grained levels (e.g., 10-ps Time-to-Digital Converter bins), with spatial references bound to localized channel identifiers. Conversely, at the event-building phase, temporal context transitions to coarse time slices or global trigger identifiers (e.g., 25-ns Large Hadron Collider bunch-crossing IDs) while spatial references shift to globally remapped identifiers derived from assembly logic.

C. Standard D-Matrix Data Frame

The D-Matrix platform implements a unified frame format specification for all heterogeneous information streams. As shown in Fig. 3 [Figure 3: see original paper], the frame structure comprises three distinct segments: header, data block, and trailer sections. The data block functions as a subordinate data structure within the frame, enhancing the organizational flexibility of the payload. In the payload segment, this data model provides four data organization schemes to address heterogeneous output formats across subsystems as previously described. The detailed specifications of this data frame container are provided in [9].

Within the same data stream, the frame format definitions differ across data

domains. The data block represents the data organization relationship within the current data domain. The Father Data Domain Start Time/Space Index (FDSTI/FDSSI) in the frame header indicate the spatiotemporal coordinate position of the current frame within the father data domain.

D. Redundancy Reduction in Data Representation

In large-scale physics experiments, the demands for high data rates pose a major challenge for transmission bandwidth, computational resources, and storage infrastructure, making it a primary cost driver in the construction of DAQ systems. Therefore, the design of DAQ protocols must thoroughly consider minimizing redundant data transmission to alleviate bandwidth and storage pressures.

The data organization schemes mentioned above directly embody this approach to redundancy reduction and can be categorized based on their spatiotemporal density as follows. Extreme Density occurs when every spatiotemporal coordinate carries transmitted data; in this case, the format need only document the initial time-space coordinates, with subsequent data points sequentially arranged such that each internal point' s coordinates are derivable from its positional offset within the stream. This constitutes the "data array" format. Extreme Sparsity uses data arrays for sparse data that would introduce excessive zero-fill invalid data points; instead, each isolated sparse point is directly represented as a "data triad" containing explicit [time, space, value] parameters, eliminating null data overhead. Locally Dense Clusters apply to globally sparse yet locally dense distributions, where a "data cluster" format is used: one starting coordinate followed by sequential dense data points describes each concentrated region, balancing efficiency and precision.

Furthermore, the two-tier structure (frame and block levels) of the data frame inherently contains high-order temporal-spatial information extraction, which enables substantial reduction in bit-width requirements for temporal-spatial coordinates within the data payload. Additionally, the transmission of specific fields within data frames can be configured per stream. The guiding principle is that D-Matrix frames transmit only mutable fields, while persistently static fields utilize preconfigured default values. For instance, in fixed-length frames, the Stream Frame Length field is configured not to be transmitted. Stream processing modules instead derive frame length details from their default configuration parameters.

E. Configurable Stream Parameters

In the D-Matrix platform, both the data organization schemes and the content/length of data frame fields are highly flexible, requiring a corresponding configuration mechanism. Two configurable properties tables are respectively used to describe the properties of a stream and the properties of a specific data domain within that stream: the Data Space Properties Table (DSPT) and the Data Domain Description Table (DDDT). These configuration tables are final-

ized during the system design phase. In software, they are distributed as global configuration files. In the FPGA implementation, these static configurations are set as parameter values when the stream processing modules are instantiated. When a stream processing module receives a data frame, it uses the Stream ID and data domain number contained in the frame header to look up the corresponding attribute tables and proceeds with subsequent processing.

The DSPT is uniquely determined by the Stream ID and is primarily used to configure the data organization schemes (i.e., data array, data triad, data cluster, and sub-stream) for that stream, as well as basic bit-width parameters (such as the bit-width of a single data point within a data array). The DDDT is jointly determined by the Stream ID and the data domain number. It is mainly used to configure the data frame format within the corresponding data domain, including whether each field is transmitted, its configured bit-width if transmitted, or its default value if not transmitted. Additionally, it contains configuration information regarding data block organization schemes.

IV. Standard Hardware and Interface Abstraction

This section describes the abstraction of transmission interfaces and computing resources, aiming at achieving arbitrary interconnectivity between hardware components and ease of hardware utilization.

A. Unified Transmission Model

To achieve arbitrary interconnectivity among heterogeneous devices, a unified transmission model is indispensable. DAQ systems in large-scale physics experiments incorporate diverse components, including DAQ-specific electronic boards, computing servers, computational accelerators, front-end electronics from detector subsystems, and ancillary devices. Simultaneously, this hardware heterogeneity directly leads to a diversity of interfaces between hardware components, exemplified by optical fibers, electrical ports, PCIe buses, and Ethernet.

To address such interface diversity, application-layer logic requires decoupling from the transport layer, aligning with scalable system design principles that minimize protocol-specific dependencies. As established in the preceding discussion, D-Matrix abstracts heterogeneous information types into distinct data streams. Multiple independent data streams share one physical interface, with each stream connecting exclusively to its own source and destination application modules. D-Matrix introduces this multi-stream transmission over shared interfaces as a Multiple Point-to-Point (MPP) transmission model (Fig. 4) [Figure 4: see original paper] [10]. The MPP module incorporates critical features such as priority scheduling, retransmission, and backpressure.

D-Matrix encapsulates heterogeneous connection types into this unified model, designed to support hardware-to-hardware connections (Low-Voltage Differential Signaling, Multipoint Low-Voltage Differential Signaling, and optical fiber),

hardware-to-software connections (PCIe and IPbus [13], with future extensions for Compute Express Link [14]), and software-to-software connections (ZeroMQ [15], with planned support for high-performance protocols such as Remote Direct Memory Access).

B. Hardware Abstraction Layer

Within DAQ systems, diverse computing resources are often employed, such as CPUs for the most versatile generic processing, GPUs specialized for massive parallelism, and FPGAs providing critical real-time efficiency and pipelining capabilities. This inherent resource heterogeneity introduces significant challenges to unified processing across the system, thereby necessitating an abstraction layer that decouples the underlying hardware specifics from the application layer implementation.

While different architectures exist, abstraction solutions for CPU hardware variations are well-established. Operating systems, device drivers, and high-performance runtime libraries abstract away hardware-specific differences, enabling hardware-agnostic application development. Consequently, driven by these abstraction layers, the same program can be deployed and operated across diverse hardware implementations within the same architecture, while some specific cross-platform libraries can enable its execution even across differing architectures. Unified architectures like CUDA enable GPU computing development in workflows similar to CPU-based systems, while technically, GPUs primarily serve as coprocessors within CPU-centered computing infrastructures.

In contrast, development on FPGAs presents significant challenges due to the lack of standardized tools. The absence of a unified abstraction layer necessitates developers to directly confront heterogeneity across FPGA models, including divergences in physical interfaces, available IP cores, and even computational primitives, while requiring substantial effort to navigate intricate implementation constraints. For the generalized system to integrate these diverse hardware components, the D-Matrix platform introduces the Board Support Package (BSP, Fig. 5) [Figure 5: see original paper], derived from the concept in the VxWorks real-time operating system [16], as the hardware abstraction layer for the FPGA.

The core function of the BSP is to encapsulate the hardware details of the FPGA and provide a consistent interface for FPGA applications. Within a stream processing system where all interactions occur in the form of streams, the BSP consequently requires the provision of a set of standardized stream interfaces as well as some default operations. These interfaces include connections to other entities via multiple MPP ports, interfaces to local devices via local bus or direct signal connections, and an interface to the application layer that exposes pure stream interfaces. By abstracting both local hardware resources and external connections, the BSP enables application layer development to focus exclusively on stream processing logic without concern for underlying hardware details.

In addition to providing interface encapsulation, the BSP also implements a set of default operations specific to each board' s characteristics. These default operations include command and command feedback for configuring, initializing, and controlling the hardware; aggregation of local hardware status (e.g., temperature, humidity, and current); and generation of default urgent messages (e.g., upon temperature or current anomalies). These messages are then routed through their respective hubs for communication with other entities. It is noteworthy that the BSP also manages clock and synchronization signal processing, which can be configured to be generated locally, sourced via dedicated external links, or recovered directly from the data transmission links [17].

The BSP fundamentally constitutes a dedicated code module that abstracts hardware-specific components, which traditionally required custom handling at the application layer, into a cohesive logical design. By performing this individualized design process once for each supported hardware platform, it lays the foundation for reusability, creating modules that can be utilized by subsequent generalized application layer designs. Since the BSP encapsulates hardware-logic mapping, a board comprising multiple hardware components will have a final BSP that is a composition of its individual component-level BSPs. For example, D-Matrix' s hardware designs can adopt the FPGA Mezzanine Card (FMC) standard [18], which uses a carrier board for computational resources and a mezzanine module for interface flexibility. Each of these cards has its own dedicated BSP, and they are combined to form a complete composite BSP for the integrated board.

This hardware encapsulation introduces a critical isolation layer between the core processing logic and the physical hardware. This abstraction provides a unified, standardized interface, which not only simplifies the interconnection of heterogeneous hardware but also significantly enhances its reusability across different experimental setups. By abstracting away low-level details, the BSP transforms hardware from single-use, experiment-specific components into reusable resources. This allows the same hardware to be deployed across diverse scientific missions without requiring duplicative interface development or reapplying complex pin constraints. Consequently, the application logic can be ported across different platforms with minimal effort, significantly accelerating the development and deployment cycles.

C. Expanding FPGA Applications

Compared to other computing platforms, the most defining characteristic of D-Matrix lies in its heterogeneous nature, specifically its support for the generic computing of FPGAs. In the common multi-level aggregation architecture within DAQ systems, FPGA boards typically serve as intermediate nodes in a tree-like structure. However, in a generic computing platform, the topological possibilities for FPGAs extend far beyond this conventional arrangement.

FPGAs are inherently constrained by their available hardware resources in

general-purpose computing, which limits the complexity of processing tasks they can handle. Although FPGA capacities continue to grow, this trend often leads to significantly higher costs. Moreover, even these high-capacity FPGAs can be insufficient for certain demanding applications, such as real-time AI processing. Therefore, by analogy with high-performance computing clusters composed of CPUs and GPUs, one potential approach is to form a computing cluster of FPGAs interconnected by a network. In such an architecture, the interconnection can be implemented using dedicated commercial network switches to form the fabric, or by designating specific FPGA nodes as communication and switching hubs to achieve a more tightly integrated system. This allows the cluster to distribute complex processing tasks across multiple FPGA nodes efficiently.

In another usage scenario, instead of building a cluster purely from FPGAs, the FPGAs are integrated into existing CPU/GPU computing clusters to act as coprocessors. In this model, the CPU remains the primary unit for task scheduling, while the decomposed tasks are offloaded to the FPGAs for processing. The results are then collected and aggregated by the CPU. These usage scenarios heavily rely on the hardware isolation of FPGAs and a well-defined encapsulation of the resources available for general-purpose computing. Furthermore, the stream processing paradigm is inherently suitable for the subtask distribution and result retrieval in coprocessor scenarios. Although these application models have not been fully engineered and deployed, their potential for future expansion should be considered during the architectural design phase.

V. Standard Processing Paradigm

This section presents the modular processing paradigm in D-Matrix, which encompasses standardized inter-module interconnection interfaces and generic computational considerations based on basic module patterns.

A. Module Processing Paradigm

The processing architecture of a generic DAQ system is simultaneously constrained by two fundamental sets of requirements derived from its scientific application scenarios: those on performance and those on functional complexity. On one hand, these systems must satisfy inherently stringent real-time constraints and exceptionally high throughput demands, necessitating high-speed data acquisition and immediate processing capabilities. On the other hand, they must implement core data processing functionalities such as event building, High-Level Trigger (HLT) processing, and real-time data compression, which require substantial and flexible computational resources.

In response to the challenge of processing complexity, a modular design provides a fundamental approach for achieving key system objectives such as enhanced flexibility, scalable architecture, and clear separation of responsibilities among components. As previously mentioned, D-Matrix employs the spatiotemporal data matrix as the fundamental form for data representation. Correspondingly,

the stream processing modules within D-Matrix also utilize the spatiotemporal data matrix as the basic unit for data processing. By defining the spatiotemporal attributes of the data that a module handles, the same module can demonstrate different specific behaviors, fulfilling diverse processing requirements and enhancing reusability. Each module is responsible for a single specific duty. For a given complex processing task, its implementation can be approached by decomposing the task into a cascading series of simpler stream processing modules, thereby constructing the intended functionality through their sequential or parallel arrangement. This requires that modules have the property of arbitrary interconnectivity; therefore, the definition of standard module interfaces is a prerequisite.

In heterogeneous systems where the underlying computing hardware is heterogeneous, D-Matrix enhances adaptability by providing hardware-specific implementations for different modules. Many modules are available in both FPGA firmware and CPU software versions, enabling flexible task placement and execution during system design. In the modular stream processing paradigm, where data flows continuously through a chain of modules, the primary performance metric is often throughput, which can take precedence over latency. Provided the throughput requirement is met, a tolerable increase in latency is acceptable within buffer limits to ensure an uninterrupted data flow. At the software level, methods like increased parallelism and load balancing are employed to enhance throughput. However, these approaches are not directly applicable to FPGA processing. Instead, we impose a controllable pipelining requirement on FPGA modules to boost their throughput.

B. Standard Module Interface

To ensure the arbitrary interconnection capability between modules required for such flexible construction, standardized inter-module interfaces are a prerequisite. For intra-FPGA module communication, the Standard D-Matrix FPGA (SDMF) port [12] is implemented. The SDMF port is extended from the AXI4-Stream interface, with its key characteristic being that the spatiotemporal information of the data frame is transmitted synchronously with the data payload, ensuring that downstream modules can simultaneously access all information required for processing, thereby enhancing the pipeline performance of modules within the FPGA.

For intra-server module communication, the Standard D-Matrix Software (SDMS) port [9] governs software interaction. The core design principle of SDMS is to minimize data copying overhead. It employs a message queue coupled with a dual shared memory architecture, which partitions a shared memory segment into a payload area and a descriptor area. Within a single server, all modules operating on the same data stream utilize the same shared memory segment. Data transfer between these modules is achieved by sending the starting address of the current data frame within the shared memory through the message queue, thereby eliminating the copying overhead for data

transmission within the node. This design allows operations such as data frame reorganization to be performed by modifying only the descriptor area. By altering the order in which descriptors point to the data payloads, the logical relationship of the data can be restructured without physically moving large volumes of data, enhancing processing efficiency.

As for inter-entity communication, transcoder modules at the transmission boundary handle protocol conversion between the SDMF port and AXI4-Stream. Similarly, for software modules, universal interfaces like files or ZeroMQ sockets are provided for external interaction, with data being serialized. Furthermore, we plan to develop an RDMA-based interface in the future to enable direct memory-to-memory data transfer between nodes, thereby significantly reducing the overhead associated with memory copy and serialization/deserialization. These interface implementations are categorized in Table 1 .

Such standard module interface definitions grant the flexibility of free interconnection between modules, thereby allowing the construction of data processing paths for stream processing systems through the cascading of modules.

C. Standard Stream Processing Modules

The D-Matrix platform categorizes standard DAQ modules into two primary classes: dataflow control modules and data processing modules. Dataflow control modules encompass: (1) 1-to-N distributors for fan-out operations, (2) N-to-1 multiplexers for data concentration, (3) fully-connected switching modules for N-to-N routing, and (4) transcoding modules for interface conversion and cross-node transmission. The data processing modules are primarily responsible for the actual implementation of functionalities, including split module, feature extractor module, trigger module, and flowmeter module, among others. Furthermore, event building constitutes the core functionality of the DAQ system. In D-Matrix, the merge module implements a generic event building algorithm based on the configuration of data domain attributes, as detailed in [11].

D. Basic Module Pattern and Complex Task Decomposition

Beyond the basic functions of data distribution and event assembly in DAQ, this streaming framework actually possesses the capabilities to implement more complex processing. Implementing such complexity directly in FPGA firmware often relies on custom programming, which challenges processing generality and reuse. To overcome this, we draw upon a fundamental systems theory: the behavior of a complex real-time system is determined by both its current internal state and sequences of historical states. This state-dependent nature implies that a complex task can be conceptually decomposed into a series of simpler, more manageable subtasks or states.

However, mere theoretical decomposability does not guarantee efficient hardware implementation. The critical engineering challenge lies in standardizing the implementation of these common subtask patterns to avoid reinventing the

wheel for each new application. It is this need for standardization that logically leads to the adoption of a template-based methodology. Analogous to the template-derivation paradigm, D-Matrix abstracts three base patterns [12, 19]. During the design phase of each basic module pattern, we first ensure the pipelining characteristics of its individual components and the overall structure. Then, different algorithmic functions are incorporated into this base pattern to derive various concrete functional modules. This approach is fundamentally different from HLS. While HLS focuses on translating a computational function into hardware, our basic pattern framework primarily emphasizes the satisfaction of pipelining constraints first.

Base patterns and their exemplary instantiations include: Extractor (e.g., 1-D Peak Finder, Multiplicity extractor), Filter (e.g., Trigger filter, 2-D weighted average calculator), and Reorganizer (e.g., Merge, Incremental nearest neighbor clusterer). In addition to basic pattern derivation, more complex tasks can also be accomplished through the cascading of multiple derived modules. Although it has not been theoretically proven that these three basic templates can be combined to address all stream processing tasks, their practical validation has been demonstrated in applications such as Time Projection Chamber cluster reconstruction and two-dimensional Hough transforms [12, 19], among others.

E. Module Configurability

To fulfill generic design requirements, all modules feature standardized configurability, categorized into interfacial adaptability and behavioral configurability. Behavioral configurability encompasses two aspects: predefined configuration rules bound to specific data characteristics (implemented via data stream and data domain properties), and module-customized configuration parameters. In addition to the parameters configured via the configuration file, standard modules also feature a command interface that can connect to the command network, enabling dynamic updates to their operational parameters.

VI. Unified Control Mechanism

Control mechanisms in DAQ systems face operational variability due to heterogeneous hardware resources and diverse experimental modes, demanding standardized approaches. Generic requirements encompass device-agnostic hardware control and generic execution of preset operational procedures.

A. Component Access

To achieve device-agnostic hardware control, the platform necessitates unified access protocols and standardized identification mechanisms. As previously discussed, large-scale physics experiments incorporate numerous connection types. While each connection method itself may possess mature communication and node identification schemes, a more universally unified solution is still essential for a heterogeneous platform. The connections between these devices are

typically wired bidirectional links. Benefiting from the communication and encapsulation capabilities of the MPP model in the D-Matrix generic platform, component access is achieved without establishing additional physical links.

To address the hierarchical topology commonly found in DAQ systems, we have developed a multi-root tree-like topology model complemented by a command forwarding and routing mechanism [10]. This design enables automatic node traversal, yielding a routing-based identification outcome that fulfills clustering and hierarchical node management requirements. Within this node identification algorithm, CPU-based servers are treated as distinct nodes, while GPUs function as attachments to CPU nodes and do not possess independent identifiers. In contrast, FPGA boards are consistently regarded as independent nodes, whether they are standalone units in a chassis or installed in server slots. Theoretically, this approach can be generalized to arbitrary graph-based topological structures, provided the graph remains connected without isolated nodes.

B. Run Control

A key design principle is the decoupling of the DAQ software from detector-specific front-end electronics control, facilitated by open and extensible interfaces. Simultaneously, given the variability in operational modes and detector configurations, the DAQ platform must deploy generic workflow control mechanisms to minimize redundant development efforts. D-Matrix implements a universal control module leveraging Python' s dynamic loading capability, which resolves functions by name at runtime to achieve architectural decoupling between the control kernel and operational logic components. Designers can extend functionality by adding new functional groups or extending existing groups. The control module dynamically invokes these functions through a standardized syntax: group_name + command_name [+ optional_parameters] [20].

Furthermore, during experimental operations, D-Matrix partitions workflows into distinct operational phases including initialization, parameter configuration, synchronization, acquisition start/stop, and global reset. Configuration and control behaviors for individual detectors across these phases are defined via phase-specific functions in Python scripts. These scripts, authored entirely by detector designers, implement device-specific configurations through register read/write commands. Scripts for different operational modes can be independently configured and modified, enabling run-time switching based on mode selection. Additionally, DAQ functionalities leverage script-based approaches for enhanced flexibility in implementing simpler tasks.

C. Online Reconfiguration

Changing experimental requirements and scales between runs demand online reconfigurability. For FPGA-based deployments, remote logic updates are implemented using vendor-specific partial reconfiguration mechanisms, with Xilinx devices utilizing MultiBoot [21] technology and Intel FPGAs employing Remote

System Update (RSU) [22] schemes. For software reconfiguration, modules utilize configuration file swaps during operation, with execution management handled by Supervisor [23], which determines active modules and configuration file paths per server. Batch updates across cluster nodes are orchestrated using Ansible automation tools [24].

VII. DAQ System Construction

A. Construction Process

Based on the D-Matrix components introduced earlier for generalized design, the construction process of an actual DAQ system is illustrated through a multi-level merging scenario as follows:

1. Requirements Assessment: Evaluates overall system data attribute requirements and processing demands to determine the number of merging stages and hardware units required for satisfying performance and I/O constraints.

2. Hardware Connection: Establishes the physical hardware topology by interconnecting hardware components according to the determined merging hierarchy and hardware quantity.

3. Stream Attribute Planning: Partitions scientific data streams into data domains based on design-phase processing requirements, generating a data domain attribute table while maintaining consistent attributes for status, command, feedback, and urgent message streams across systems.

4. Core Processing Module Design: Identifies hardware levels for each merging stage and designs configuration parameters for merging modules through functional module selection and configuration file generation from the library.

Note: Steps 3 (Stream Attribute Planning) and 4 (Core Processing Module Design) can be executed before or after Step 2 (Hardware Connection), depending on the design scenario. In requirements-driven approaches, stream attributes and modules are defined first to guide hardware integration. Conversely, hardware connections may be prioritized based on factors like cost or scalability. This flexibility allows the D-Matrix platform to adapt to various experimental needs without compromising modularity or performance.

5. Auxiliary Module Design: Supplements the design with inter-node dataflow control modules, transcoding modules, and frame checkers associated with merging modules, alongside command/status/urgent message modules per hardware node. Additionally, this step requires comprehensively considering the performance requirements and processing capabilities of each data processing path to configure an appropriate buffer size for the inter-module interfaces.

6. Node-Level Configuration File Generation: Generates final module lists and configuration files for each hardware node, including inter-module connection relationships based on preceding module designs.

7. Code Generation and Deployment: The software phase initiates with pre-deployment of binary executables using infrastructure tools such as Ansible, followed by per-node customization of module and supervisor configuration files. For FPGA cards, this involves combining card-level BSP and connecting functional modules to generate application-layer HDL code. Subsequently, the BSP and application-layer code are integrated to construct the complete logic implementation, which is ultimately deployed through remote bitstream update capabilities.

B. Towards Automation in DAQ Design

Leveraging computers to assist human designers is a common practice, exemplified by CAD software in the industrial domain and EDA software in the electronics field. The adoption of computer-aided methods in DAQ design can also greatly enhance development efficiency. The D-Matrix platform already inherently possesses some prerequisites for realizing Computer-Aided System Design for DAQ (CASD-DAQ). The realization of CASD can be broadly divided into two phases. The first phase involves using computers to accelerate the development process, while the second phase aims for fully automated computer-driven design.

1. Computer-Aided Design Phase: During this phase, the overall system-level design is still undertaken by the DAQ designer. The designer must plan the hardware connectivity topology and the system processing flow, select the specific processing modules to be employed, and then interconnect and configure these processing modules via GUI operations or configuration files. The subsequent development steps, including automatic code generation and deployment, can be assumed by computational tools. Realizing this process relies on several prerequisites. First, the configuration and connectivity of processing modules are ensured by D-Matrix's standardized abstraction of modules and the standardization of module interfaces. Secondly, automated code deployment depends on a unified access model and online reconfigurability, both of which are topics addressed in preceding sections. Regarding automated code generation, the software domain inherently supports system deployment via configuration files since the software modules adopted by D-Matrix are already configured through these files, eliminating the need for code generation or modification. In the FPGA domain, however, automated code generation is fully achievable with the support of the BSP, and relevant developments are under active investigation.

2. Computer-Driven Design Phase: Within this phase, after the system designer defines coarse-grained processing requirements, the computer autonomously refines the processing flow, selects the necessary processing modules, and determines their interconnections and configurations. One viable structure for such automated design is outlined below (Fig. 6) [Figure 6: see original paper]: Resources maintain the resources required for system design through a resource library, including available hardware boards, functional modules, and

their detailed attributes. Input for the target system' s hardware connection topology and a coarse-grained flow graph (including core processing module configurations) is provided by means of configuration files. An optimizer performs iterative optimization based on resource constraints and optimization strategies, automatically allocating processing modules to target hardware entities. During iteration, a simulator outputs behavioral simulation and performance simulation results via the simulator for designer supervision. Finally, the deployment configuration for each target entity module is similarly output in the form of configuration files. Subsequent code generation and deployment follow an identical procedure to that of the preceding design phase.

VIII. Conclusion

This paper has presented D-Matrix, a generic firmware-software co-designed stream processing platform for constructing DAQ systems in large-scale physics experiments. Built upon the SHARE principles, D-Matrix provides a holistic solution to critical challenges such as heterogeneous data formats, diverse hardware interfaces, and stringent real-time processing demands. D-Matrix employs a strategy of standardization and generalization across its core components. This approach, applied to data protocols, hardware and interface abstraction, and processing paradigms, is key to significantly enhancing the generality and usability of heterogeneous systems, especially those integrating FPGAs. The platform significantly reduces development costs and time-to-deployment by enabling extensive reuse of standardized components while maintaining adaptability to diverse detectors and experiments.

Based on the development of the D-Matrix platform, the DAQ system has been deployed at the CEE experiment [25], achieving 73.6 Gbps throughput and processing 20,890 events/s. Future implementations target the High Energy Fragment Separator (HFRS) [26] and Super Tau-Charm Facility (STCF) [27]. Furthermore, the inherent modularity and standardization within D-Matrix provide essential foundations for automated DAQ design and an extensible ecosystem. By standardizing component definitions, the platform enables third-party contributions of compliant hardware, firmware and software modules, ensuring long-term extensibility and iterative evolution. With the enhanced generality of the platform, significant improvements have been achieved in its applicable scenarios, the diversity of hardware connectivity, and the complexity of processing tasks it can support. Consequently, such a platform now transcends its original purpose, evolving into a generic heterogeneous real-time computing platform rather than being confined to the single application scenario of data acquisition.

Future work will expand module libraries, explore new high-performance communication methods, enhance cross-platform portability, and advance CASD-DAQ capabilities.

References

[1] V. Friese, CBM Collaboration, et al., The high-rate data challenge: computing for the CBM experiment. *Journal of Physics: Conference Series*, 898, 112003 (2017). doi: 10.1088/1742-6596/898/11/112003

[2] R. Bartoldus, C. Bernius, D. W. Miller, Innovations in trigger and data acquisition systems for next-generation physics facilities. arXiv preprint (2022). arXiv: 2203.07620 [hep-ex]

[3] J. Gutleber, S. Murray, L. Orsini, Towards a homogeneous architecture for high-energy physics data acquisition systems. *Computer Physics Communications*, 153, 155–163 (2003). doi: 10.1016/S0010-4655(03)00161-9

[4] P. Moreira, R. Ballabriga, S. Baron, et al., The GBT Project. In: *Proceedings of the Topical Workshop on Electronics for Particle Physics*. (CERN, Geneva, Switzerland, 2009), p. 342–346. doi: 10.5170/CERN-2009-006.342

[5] W. Wu, FELIX: the New Detector Interface for the ATLAS Experiment. *IEEE Transactions on Nuclear Science*, 66, 986–992 (2019). doi: 10.1109/TNS.2019.2913617

[6] K. Biery, C. Green, J. Kowalkowski, et al., artdaq: An Event-Building, Filtering, and Processing Framework. *IEEE Transactions on Nuclear Science*, 60, 3764–3771 (2013). doi: 10.1109/TNS.2013.2251660

[7] J. Gutleber, L. Orsini, Software Architecture for Processing Clusters Based on I2O. *Cluster Computing*, 5, 55–64 (2002). doi: 10.1023/A:1012744721976

[8] T. Stockmanns, PANDA Collaboration, et al., FairMQ for Online Reconstruction - An example on PANDA test beam data. *Journal of Physics: Conference Series*, 898, 032021 (2017). doi: 10.1088/1742-6596/898/3/032021

[9] T. Wang, J. Yang, H. Wang, et al., A Generic Streaming Software Platform Design for High-Energy Physics Data Acquisition Systems. *IEEE Transactions on Nuclear Science*, 68, 101–109 (2021). doi: 10.1109/TNS.2021.3050140

[10] Z. Sun, J. Yang, L. Zhang, T. Wang, R. Liu, K. Song, A generic node identification and routing algorithm in a distributed data acquisition platform: D-Matrix. *Journal of Instrumentation*, 18, P12012 (2023). doi: 10.1088/1748-0221/18/12/P12012

[11] L. Zhang, J. Yang, T. Wang, Z. Sun, K. Sun, J. Zeng, Event Building Algorithm in a Distributed Stream Processing Data Acquisition Platform: D-Matrix. *IEEE Transactions on Nuclear Science*, 70, 105–112 (2023). doi: 10.1109/TNS.2023.3235904

[12] L. Zhang, J. Yang, Z. Sun, et al., D-Matrix: FPGA-Based Solutions for General Stream Processing in High-Energy Physics Experiments. *IEEE Transactions on Nuclear Science*, 71, 2209–2218 (2024). doi: 10.1109/TNS.2024.3467107

[13] C. G. Larrea, K. Harder, D. Newbold, D. Sankey, A. Rose, A. Thea, T. Williams, IPbus: a flexible Ethernet-based control system for xTCA hardware. *Journal of instrumentation*, 10, C02019 (2015). doi: 10.1088/1748-0221/10/02/C02019

[14] D. Das Sharma, R. Blankenship, D. Berger, An Introduction to the Compute Express Link (CXL) Interconnect. *ACM Computing Surveys*, 56, 1–37 (2024). doi: 10.1145/3669900

[15] ZeroMQ: An open-source universal messaging library. (2025). Available: https://zeromq.org/

[16] Wind River Systems, *VxWorks 7 BSP Development Guide*. (2025). Available: https://www.vxworks6.com/bsp/vxworks-7-bsp-development-guide/

[17] J. Zeng, J. Yang, L. Zhang, Z. Sun, K. Sun, FPGA Implementation of Fixed-Latency Command Distribution Based on Aurora 64B/66B. *IEEE Transactions on Nuclear Science*, 71, 1348–1356 (2024). doi: 10.1109/TNS.2024.3400378

[18] R. Seelam, I/O design flexibility with the FPGA mezzanine card (FMC). Xilinx White Paper WP315 (2009). Available: https://docs.amd.com/v/u/en-US/wp315

[19] L. Zhang, *Research on the Design of a General Stream Processing Architecture Based on FPGA in the Nuclear and Particle Physics Experiment Data Acquisition System*. Ph.D. dissertation, University of Science and Technology of China, Hefei, China (2025).

[20] Z. Sun, J. Yang, A Generic Main Control Software Structure in a Distributed Data Acquisition Platform: D-Matrix. *PyHEP 2023: Python in High Energy Physics Users Workshop*. (CERN and PyHEP Collaboration, Online, Oct. 2023), Lightning talk. Available: https://indico.cern.ch/event/1252095/contributions/5593987/

[21] B. Li, K. K. Gakhal, MultiBoot with 7 Series FPGAs and SPI (XAPP1247). AMD (Xilinx), Application Note XAPP1247 (2017). Available: https://docs.amd.com/v/u/en-US/xapp1247-multiboot-spi

[22] Intel Corporation, *Agilex™ 7 Configuration User Guide*. (Intel Corporation, 2025), Document 683673, Chapter 5: Remote System Update (RSU). Available: https://www.intel.com/content/www/us/en/docs/programmable/683673/25-1/remote-system-update-rsu.html

[23] Agendaless Consulting, Supervisor: A Process Control System. (2025). Available: https://supervisord.org/

[24] Red Hat, Inc., Ansible Collaborative. (2025). Available: https://www.redhat.com/en/ansible-collaborative

[25] L. Lü, H. Yi, Z. Xiao, et al., Conceptual design of the HIRFL-CSR external-target experiment. *Science China Physics, Mechanics & Astronomy*, 60, 012021 (2017). doi: 10.1007/s11433-016-0342-x

[26] L. Sheng, X. Zhang, J. Zhang, et al., Ion-optical design of High energy FRagment Separator (HFRS) at HIAF. *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms*, 469, 1–9 (2020). doi: 10.1016/j.nimb.2020.02.026

[27] M. Achasov, X. Ai, L. An, et al., STCF conceptual design report (Volume 1): Physics & detector. *Frontiers of Physics*, 19, 14701 (2024). doi: 10.1007/s11467-023-1333-z

Note: Figure translations are in progress. See original paper for figures.