MeerKAT correlator-beamformer: a real-time processing back-end for astronomical observations

Andrew van der Byl,[a,*] James Smith,[a] Andrew Martens,[a] Jason Manley,[a] Tyrone van Balla,[a] Alec Rust,[a] Amish Patel,[a] Gareth Callanan,[b] Adam Isaacson,[a] Wesley New,[a] Robin van Wyk,[a] Francois Kapp,[a] Henno Kriel,[a] and Omer Mahgoub[a]
[a]South African Radio Astronomy Observatory, Cape Town, South Africa
[b]Lund University, Faculty of Engineering, Lund, Sweden

Abstract. The MeerKAT radio telescope consists of 64 Gregorian-offset antennas located in the Karoo in the Northern Cape in South Africa. The antenna system consists of multiple subsystems working collaboratively to form a cohesive instrument capable of operating in multiple modes for defined science cases. We focus on the channelizing subsystem (F-engine), the correlation subsystem (X-engine), and the beamforming subsystem (B-engine). In the wideband instrument mode, the channelizing can produce 1024, 4096, or 32,768 channels with correlation up to 64 antennas. Narrowband mode decomposes sampled bandwidth into 32,768 channels. The F-engine also performs delay compensation, equalization, quantization, and grouping and ordering. The X-engine provides both correlation and beamforming computations (independently). This document is intended to be a stand-alone entity covering the channelizing, correlation, and beamforming processes for the MeerKAT radio telescope. This includes data reception, pre- and post-processing, and data transmission. © *The Authors. Published by SPIE under a Creative Commons Attribution 4.0 International License. Distribution or reproduction of this work in whole or in part requires full attribution of the original publication, including its DOI.* [DOI: 10.1117/1.JATIS.8.1.011006]

Keywords: MeerKAT; channelizer; polyphase filter bank; correlation; beamforming; radio astronomy.

Paper 21076SS received Jul. 29, 2021; accepted for publication Nov. 1, 2021; published online Nov. 22, 2021.

1 Introduction

The MeerKAT radio telescope is a Square Kilometre Array (SKA) precursor instrument, a 64-element interferometer using Gregorian-offset antennas, built by SARAO in South Africa's Karoo Desert.[1]

The MeerKAT digital back-end uses an *FX*-style correlation process. The *F* in *FX* represents the channelization process and is performed by a logical device known as an F-engine. The *X* represents cross-correlation performed in the frequency domain. The B-engine performs beamforming and is colocated within the processing silicon of the X-engine but logically distinct from it. The B-engine receives a duplicate data feed from the X-engine and processes independently from the X-engine correlation process. Both the B-engine and the X-engine share common networking logic. This paper will cover both the F-engine and X-engine systems in logical processing order collectively known as a correlator-beamformer (CBF).

2 MeerKAT: At a Glance

To create broader context for MeerKAT, Fig. 1 shows an information graphic of the system. MeerKAT consists on many subsystems that all operate collaboratively, each with their own depth and merit. This paper will focus on the CBF subsystems and will detail their operations

*Address all correspondence to Andrew van der Byl, avanderbyl@sarao.ac.za

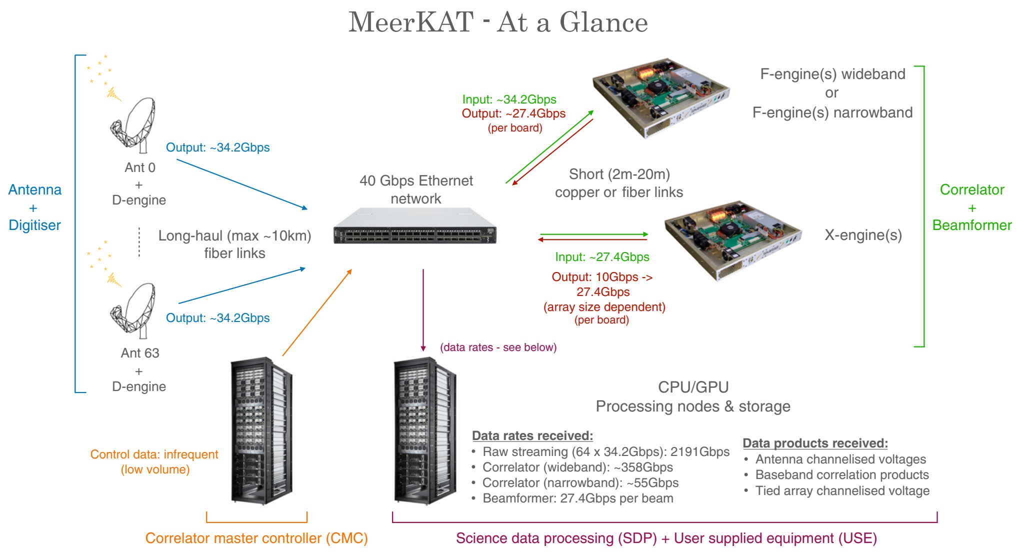
MeerKAT - At a Glance
Antenna + Digitiser
Ant 0 + D-engine
Ant 63 + D-engine
Output: ~34.2Gbps
Output: ~34.2Gbps
Long-haul (max ~10km) fiber links
40 Gbps Ethernet network
Input: ~34.2Gbps
Output: ~27.4Gbps (per board)
F-engine(s) wideband or F-engine(s) narrowband
Short (2m-20m) copper or fiber links
X-engine(s)
Input: ~27.4Gbps
Output: 10Gbps -> 27.4Gbps (array size dependent) (per board)
Correlator + Beamformer
(data rates - see below)
CPU/GPU Processing nodes & storage
Control data: infrequent (low volume)
Data rates received:
Raw streaming (64 x 34.2Gbps): 2191Gbps
Correlator (wideband): ~358Gbps
Correlator (narrowband): ~55Gbps
Beamformer: 27.4Gbps per beam
Data products received:
Antenna channelised voltages
Baseband correlation products
Tied array channelised voltage
Correlator master controller (CMC)
Science data processing (SDP) + User supplied equipment (USE)

Fig. 1 MeerKAT at a glance. MeerKAT consists of 64 Gregorian-offset antennas and connected via a 40-Gbps Ethernet switching network. Real-time backend processing is performed by F-, X-, and B-engines, and the data products produced stored and processed offline by SDP compute nodes.

and finer details. Some additional information regarding processed data and science data processing (SDP) is given in Sec. 2.6.

2.1 *System Size*

The MeerKAT CBF comprises 288 Square Kilometer Array Reconfigurable Application Board (SKARAB) processing nodes (see Sec. 4.1), two correlator master controllers (CMC) (made up of commercial-off-the-shelf servers), and 54 Mellanox SX1710 (36-port) 40 Gigabit Ethernet Network Switches. The bulk of this hardware is installed across ten 42U 19-in. server racks, kitted with managed rack power distribution units, and located in a single row within the Karoo Array Processing Building (KAPB). Ten of the 40 Gigabit switches are installed in other rows to provide interconnect ports for other data subscribers.

2.2 *Power Consumption*

The entire CBF was designed to fit within a total power budget of 45 kW, with each individual rack drawing up to 4.5 kW. Actual power consumption during operation varies as the number of processing nodes required depends on the mode of operation, the size of the instrument, and how many concurrent instruments are running.

2.3 *Cooling*

The CBF host building (KAPB) (see Appendix) is fitted with a twin redundant air conditioning cooling system. The CBF is cooled via the natural convection of air moving from the cold aisle to the hot aisle. This same cooling system cools all control and monitoring as well as SDP servers.

2.4 *Radio Frequency Interference Mitigation*

The CBF is housed within the KAPB, which was specifically designed to offer sufficient electromagnetic interference and RFI shielding from the electrically noisy hardware (servers, switches, Ethernet cables, etc.) located inside the KAPB.

2.5 *Reliability and Maintenance*

The CBF is still currently being deployed, although it has been operational since 2018. To mitigate against reliability issues, spare processing nodes (20%) are deployed and ready for use. Two CMCs are provisioned to provide control redundancy. The system is not fully deployed, so accurate reliability statistics are presently not available. To assist with maintenance, all of the major physical components making up the CBF (e.g., processing nodes, servers, and switches) are line replaceable units and may be easily swapped out in the event of failure.

2.6 *Science Data Processing and Data Storage*

Digitized data captured by the antenna(s) are processed real-time through the CBF. Once complete, the various data products (see Sec. 2.7) end up in the SDP subsystem. The data storage is provisioned using Ceph, a data object store, using the Ceph RADOS gateway.[2] This supports a RESTful API that is compatible with the basic data access model of the Amazon S3 API.[3] The volume of data produced by MeerKAT is dependent on the observation type, and the MeerKAT telescope accumulates on average 6 PiB per annum. The bulk of this data is made up of uncalibrated visibility data, continuum cubes, and spectral image cubes.

2.7 *Data Products*

The CBF utilizes a publish-subscribe mechanism for data access. At all subsystem levels (D-, F-, X-, and B-engines) (see Fig. 2), data that is produced at those stages can be subscribed to if required. The following data products are produced:

- D-engine–baseband voltage: raw sampled voltage produced at each antenna (Sec. 3.1).
- F-engine (wide and narrow)–antenna channelized voltages: channelized voltages computed from baseband voltage of each antenna (Sec. 6).
- X-engine–baseline correlation products: correlation product for each antenna baseline pair (Sec. 8).
- X-engine–tied array channelised voltage: beamformed product of antenna channelized data (Sec. 9).

3 MeerKAT Correlator in Context

MeerKAT's design as an interferometer requires a powerful correlator system to be able to achieve its science objectives. Figure 2 shows the digital back-end of the MeerKAT radio telescope conceptually.

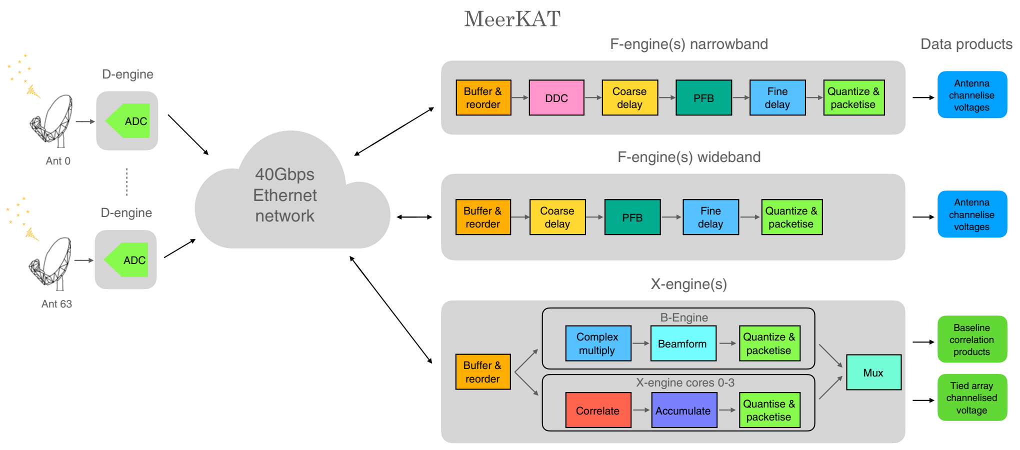

Fig. 2 MeerKAT signal-chain overview, showing the interconnection of the digitizer subsystem, and the F- and X-engines of the correlator via a 40-GbE network. This paper will only focus on the F- and X-engines. The D-engine is included for context. Data products produced by the various CBF subsystems are included and detailed in Sec. 2.7. Downstream subsystems are not shown.

One of MeerKAT's strengths is its flexibility. The correlator enables MeerKAT to observe in various modes of operation, depending on the science requirements:

- array size: MeerKAT's 64 antennas can be deployed in subarrays of 4, 8, 16, or 32 antennas, for parallel operations, if all 64 are not required for the observation. A full array of 64 antennas is also possible.
- bandwidth: MeerKAT currently has capability to observe in UHF band (544 MHz of bandwidth), L-band (856 MHz), or S-band (875 MHz). In addition, narrowband modes can trade bandwidth for spectral resolution by a factor of 16 or 32. An indexer positioned at the feed of each antenna permits changeover of receivers for different frequency-band operation.
- channelization: the observed band can be channelized into 1024, 4096, or 32,768 channels. (Narrowband modes currently only support 32,768 channels).

The processing pipeline starts with RF data digitized at an antenna feed by direct-sampling at RF frequency and sent into the 40-GbE network through a publish-subscribe mechanism where the data are subscribed to by an F-engine to channelize and perform delay compensation and equalization. The processed channelized data are then transmitted back into the Ethernet network and subscribed to by X-engines, which in turn perform correlation and beamforming functions. Finally, this correlated and beamformed data are transmitted back to the network where they may be subscribed to by other subsystems for further processing.

3.1 *Digitizer*

During an observation, antenna voltage data are digitized directly by one of several selectable digitizers. The digitizer samples at 10 bits at a rate commensurate to the mode (UHF, L-band, S-band). In the sampling process, the resulting time-series data are translated to base-band, correcting for spectral inversion.

3.2 *Ethernet Interconnect*

Telescope data are transported between processing components by a 40-GbE switching network configured in a folded-Clos configuration.[1] All processing nodes (the digitizers, the F- and X-engines of the correlator, and downstream subscribers) are connected to the leaf switches in the network. The number of leaf and spine switches is chosen to enable nonblocking, full-crossbar interconnect between any two nodes on the network.[4]

3.2.1 *Multicast*

The data are transported on the network using Ethernet multicast, with a publish-subscribe mechanism. In principle, any logical subsystem connected to the network is able to subscribe to any data stream. In the case of MeerKAT, this is typically the next processing stage; however, external parties can connect user supplied equipment (USE) to the network and subscribe to any of the intermediate stages of data.

Practically within MeerKAT, each F-engine node subscribes to antenna digitzer data streams from precisely one antenna. The output from the F-engine is transported to multiple X-engines using multicast (discussed further in Sec. 6.8). For information on the X-engine data ingest see Sec. 8.1.

3.2.2 *SPEAD*

Transmission of data between elements of the signal chain over the network uses a streaming protocol termed streaming protocol for the exchange of astronomical data (SPEAD).[5] SPEAD is built on top of UDP and is used for sending and receiving logical collections of data. Data generated within subsystems can be logically grouped for processing further down the pipeline.

The SPEAD protocol offers flexibility through variable-length fields and assists in the grouping of processed data.[5]

4 Processing Platform

4.1 *Hardware*

The hardware and software platforms used in the design and implementation of the correlator are noted here as important design choices and were constrained by the limitations of available hardware.

The main processing element of the MeerKAT CBF is the SKARAB. SKARAB is a reconfigurable field programmable gate array (FPGA)-based platform developed specifically for MeerKAT; however, it is not limited to radio astronomy (RA) processing pipelines. The SKARAB platform is available commercially.[6]

The SKARAB is designed around a Xilinx FPGA (Virtex-7 XC7VX690T) and is equipped with three mezzanine sites capable of housing mezzanine cards for 40 GbE, hybrid memory cube (HMC),[7] or analog-to-digital conversion (ADC). In the MeerKAT case, ADCs are not required in the SKARAB as digitization is performed at the antenna feed. These mezzanine sites are therefore populated with one 40 GbE card and three HMC cards. More information on the SKARAB platform can be found in other literature.[6,8]

Each mode of operation is implemented by SKARABs running specifically compiled *gateware*, which enables the SKARAB to perform its assigned processing task. Each SKARAB is identical in configuration and hardware features and can take on either an F-engine or X-engine role depending on observation requirements.

4.2 *FPGA Gateware*

The Collaboration for Astronomy Signal Processing and Electronics Research[9] (CASPER) provides a community-supported, open-source software toolflow to streamline and simplify the design of RA instrumentation. The combination of readily available hardware and open-source software enables design reuse from generation to generation and across a wide range of instrumentation types and scales.

The toolflow currently uses Mathworks' MATLAB/Simulink™ as an interface. CASPER provides libraries for DSP, networking, memory, and other related logic, which include configurable fast Fourier transforms (FFTs), FIR filters, 40 GbE cores, etc. The toolflow's back-end is driven by Xilinx Vivado™ for logic place-and-route for the FPGA, controlled by a set of Python scripts.

The SKARAB's CASPER-based board support package includes a Microblaze soft-core microcontroller. This allows the SKARAB to act as a node on the network by implementing a DHCP client. This further facilitates reprogramming of the FPGA with the required gateware, as well as maintaining multicast subscriptions during operation. In MeerKAT, there is no out-of-band management included in the gateware configured on a SKARAB, and all communications occur on the 40-GbE links.

4.3 *Hybrid Memory Cube*

Due to the limited memory available on the FPGA, an off-chip memory was required to perform the DSP operations. HMC (at the time of design of the SKARAB) was the only available external memory with the right combination of size, speed, and pin-count for this purpose. Several characteristics of the HMC impose limitations in the application. These include

- variable latency between requests and responses;
- order of the responses not being guaranteed;
- bus width and speed of the memory being fixed.

The HMC has a fixed bus width of 256 bits. To maximize efficiency of data transfers and storage in the DSP pipeline, the data must be packaged to make full use of the HMC.

In addition, FPGA logic and memory resources are utilized to reassemble the data in order after retrieval.

5 Timestamps and Synchronization Management

Correlation relies on coherency of the data from all antennas. A timestamp is transferred with every SPEAD packet of data in the system. These are 48-bit counters with units that are in digitizer samples that start at a globally determined epoch in the digitizers. They are used to synchronize and control operation throughout the signal-processing pipeline.

5.1 *Timestamps: F-Engine*

Packet timestamps indicate the time of the oldest sample in the packet or of the oldest digitizer samples associated with the data in the packet. For example, the output of the F-engines is tagged with the timestamp of the first sample used to generate the first spectrum in the block of data being transferred. These timestamps enter with the data and are used to determine where to write the data in the reorder buffer and are regenerated on read-out when necessary. The delay compensation system uses this mechanism to determine when to load coefficients, and it also generates the timestamps added to packets output to X-engines. At the beginning of an observation, all F-engines are synchronized. There is logic to ensure that the resynchronization control path operates as expected, and if the system has not been synchronized within a deterministic time period, watchdog logic causes a resynchronization. Logic to determine error states is included, which can also trigger a resynchronization if required.

5.2 *Timestamps: X-Engine*

At each point in the X-engine pipeline, the timestamp is updated relative to the rate at which the data flows. The timestamps are an important factor to be considered, and the rate of update will also differ depending on the mode of operation (for frequency decomposition of 1k, 4k, and 32k channels in the F-engine) in the case of wideband and narrowband (bandwidth of 107 or 53.5 MHz). In the case of wideband modes (both F-engine and X-engine), there is no downsampling as the bandwidth (856 MHz total) remains the same; however, in the case of narrowband, the data are downsampled in the F-engine. Timestamp regeneration within the X-engine is adjusted based on the downsample factor required if narrowband mode is operated.

6 F-Engine: A MeerKAT Channelizer

A data-flow representation of the F-engine is shown in Fig. 3. Digitizer data are ingested into the F-engine through 40 GbE. Egress data are through the same physical connection to the switching network.

6.1 *Ethernet Receive*

Data enter the FPGA through a 40-GbE link via a 256-bit-wide data path. Both polarizations for each antenna are processed on a single F-engine. The F-engine buffers and realigns the independent polarizations (which are received in separate multicast streams) before processing them simultaneously. Data from each polarization are received by subscribing to all Ethernet multicast groups transmitted by the appropriate digitizer.

6.2 *Network Data Reorder*

Retransmits, data scrambling, and nondelivery (typically associated with UDP) can cause problems for F-engines, which must process digitizer data in the order in which they were sampled. As a result, it is necessary to buffer and reorder the incoming data stream to accommodate missing or out-of-order UDP packets. The system is designed to be tolerant of dropped packets and

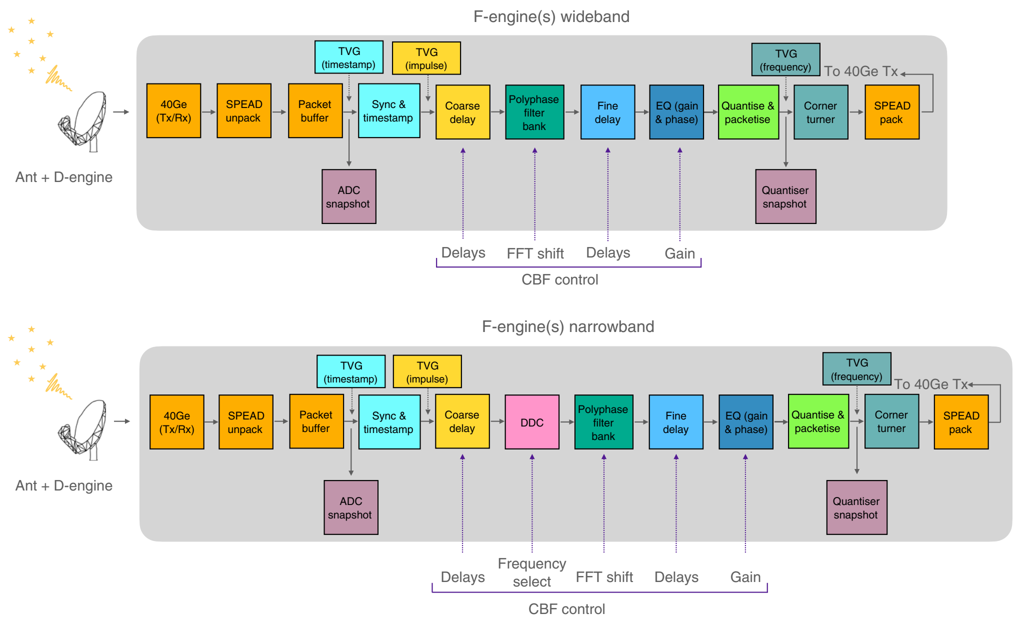

Fig. 3 Data flow block diagram of the wideband and narrowband F-engine(s). Data flow from left to right. Output data are fed back into the Ethernet block for transmission. The CBF control labels indicate the control parameters available for real-time adjustments. Section 11 details the control interfaces. The TVGs are discussed in Sec. 7.

performance degrades gracefully under these conditions. Missing data from dropped packets is zero-padded.

As part of this process, when packets are received from the network, they are ordered in the buffer based on received timestamps, and missing packets' data is flagged before being processed. The reordering and buffering mechanism is implemented as a gated circular buffer in on-chip BRAM. The buffer is divided into windows, where each window contains a single packet for each polarization. There is a buffer per polarization.

6.3 *Delay Compensation*

To correlate the received data from multiple antennas, compensation is required for the geometric delays experienced by the signals received at each antenna in the array. There are two mechanisms used to implement this functionality: coarse delay (CD) and fine delay.

The majority of the delay to be compensated for is done by the coarse-delay portion of the signal-processing pipeline. This adjusts the delay by an integer multiple of the digitizer sampling period. The remaining fraction of a sample's worth of time to delay must also be removed. This is handled in the fine delay.

The CD function accepts five control inputs:

- delay (units: samples);
- rate of change of delay (unitless, or samples per sample);
- phase offset (-1 to $+1$, representing $-\pi$ to $+\pi$ radians);
- rate of change of phase offset (in phase offset step change per sample);
- load time (in ADC sample-count timestamp).

Delay compensation is performed by equalizing the total delay experienced by a single antenna i after it arrives at the array reference position by adding a non-negative compensating delay $\tau_{d,i} = \tau_{\max} - \tau_{r,i} - \tau_{g,i}$, where $\tau_{\max}$ is the maximum delay experienced by any antenna for any source position, $\tau_{r,i}$ is the receiver chain delay (very small for MeerKAT), and $\tau_{g,i}$ is the geometric delay. The compensation delay has an upper bound $\tau_{d,i} \leq 2\tau_{\max}$. MeerKAT has a maximum baseline of 8000 m, giving a $\tau_{\max}$ of 26.7 μs. The maximum compensation delay is therefore 53.4 μs.

The F-engine is capable of compensating for sidereal objects at 15 deg elevation at a rate of 0.5 ns/s. For a sidereal object at 90 deg elevation, the delay rate is 1.9 ns/s. MeerKAT has a control and monitoring subsystem that sends the F-engine a Taylor expansion description of $\tau_{d,i}$ taken at time t_n.

The F-engine may split the desired delay into a CD $\tau_{c,i}$ (described in Sec. 6.3.1) and a fractional-sample fine delay $\tau_{f,i}$ (Sec. 6.3.2), such that $\tau_{d,i} = \tau_{c,i} + \tau_{f,i}$. The CD is non-negative and is implemented by a sample buffer in the F-engine. The fine or fractional delay may be positive or negative and is implemented as a per-channel phase correction that changes linearly across the band.

In addition to delay compensation, compensating for the phase change incurred due to frequency translation is required. The RF signal will have undergone a phase change of $2\pi f_{\mathrm{RF}}\tau_i$ where $\tau_i = \tau_{r,i} + \tau_{g,i}$ by the time it reaches the F-engine input. If both coarse and fractional delay were applied at an intermediate frequency (IF), we are left with a remaining local oscillator (LO) fringe rotational phase $\phi_{\mathrm{fringe}} = 2\pi f_{LO}\tau_i$, where a constant term has been omitted since it is fixed per frequency and will be removed by baseline calibration. The purpose of LO fringe stopping is to counter the LO fringe rotation by the application of an inverse phase rotation, based on the already available compensation delay and the effective LO frequency, f_{LO}. The sign is based on the effective sideband (upper or lower) that is employed. The correlator has to allow for a constant phase offset per channel $\phi_{r,i}$, which will be calculated during instrument calibration. The fringe phase range is between $-\pi$ and $+\pi$, with a resolution of <1 deg. The fringe rate is between 0 and 186 rad/s for sidereal objects, with a resolution of <7 mHz.

6.3.1 *Coarse delay*

The CD focuses on the shifting or delaying of the data stream by a runtime-programmable integer number of samples.

The CD data pipeline is thus implemented in two parts:

1. The primary delay in HMC, which operates on multiples of eight samples and includes a small BRAM memory to reorder the HMC responses.
2. A barrel shifter to shift between 1 and 8 samples to cater for delays that are not a multiple of the width of the data bus of the HMC. In fact, this component handles delays between 0 and 16 samples.

In the case of MeerKAT, the signal sample rate and resulting signal bandwidth is larger than the FPGA clock rate, so the delay component is complicated by the fact that samples are processed in parallel (eight samples at 10 bits per sample per FPGA clock cycle). Data cannot, thus, just be written-to and read-from a single, wide, monolithic memory. A barrel-shifter is needed to reorder the parallel streams, in the event that the number of samples to be delayed is not a multiple of the number of parallel streams (Fig. 4).

To cater for the variable latency of the HMC, the input timestamp is not propagated in a parallel path, like with many of the other components in the DSP pipeline, but rather regenerated on the output of the HMC memory. In addition to the data pipeline, two other components form part of this CD subsystem: timed-load latches and linear interpolating address generators. These components are duplicated in the downstream fine delay function. In the CD application, two instances are used: one to generate the instantaneous delays that must be applied by the HMC and another to generate the delays that the barrel shifter should apply.

There is a fundamental limitation with this split, serial implementation. It arises because the barrel-shifter sometimes requires the preceding cycle's data, which the barrel-shifter does not always have in a streaming application when the preceding CD (eight sample step implemented in the HMC) has stepped. The CD may erroneously process eight old samples when the requested delay is decremented. In reality, this is not of significant concern as even at the fastest required delay rates, this will only affect eight in 224 samples. It can also arise when the delay increments by a multiple of 16 samples in one step. The "old samples" are the last values that

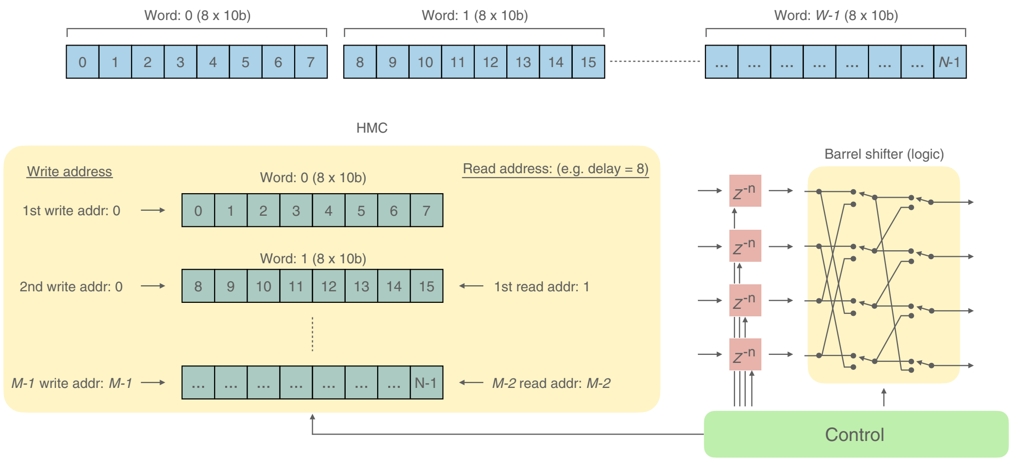

Fig. 4 CD: This is made up of HMC and a barrel shifter (in logic). Note that each data word consists of 8 samples of 10 bits. The barrel shifter is used if an integer delay is less than eight samples or the delay is not an integer multiple of 8 (the delay can be split between the HMC and barrel shifter, e.g., for a delay of 9).[10]

were in the CD pipeline. Exactly which samples these are depend on the step-change in the CD (e.g., stepping by 32 samples would result in processing eight erroneous samples from 32 samples ago). Since delay tracking is used for tracking, the CD will step in single-sample increments or decrements, so the erroneous samples will be adjacent, statistically similar numbers and the error will actually be much smaller than suggested.

It should be noted that there are separate sets of registers for each polarization. All registers are 32 bit, but bits are sliced as needed to obtain the desired precision for the given mode. The resolution provided is 0.0015 rad ($2\pi/4096$). The delay is loaded throughout the delay pipeline (coarse and fine delay) as the designated timestamp's data pass that component. The timestamps in all MeerKAT designs are 48 bit sample counters. The delay model continues to execute indefinitely until a new delay is loaded. If a new delay is armed before a prior delay is loaded, the new delay will replace the previous one (i.e., there is no queue/FIFO).

6.3.2 *Fine delay*

By the Fourier identity $x(t - t_0) \Leftrightarrow a_k e^{jk(\frac{2\pi}{\tau_0})}$, a temporal delay of a signal is equivalent to multiplication by a rotating phasor in the frequency domain. While fine delays are also sometimes implemented in the time-domain using a phase-shifting FIR filter, this requires an entire additional wideband filter. In this FX correlator, however, channelization is already being performed as part of the F-engine process, so this fine-delay functionality can be implemented simply by multiplying the channelizer output with an indexed sine/cosine lookup table where the index has a linear slope corresponding to the delay to be introduced.

The F-engine has a requirement to update these delays while the system is operating to accommodate changing geometric delays in the telescope. The adjustment of the coarse and fine delay components must be coordinated to ensure that the appropriate delay is applied to each sample. Also, when incrementing or decrementing the fine-delay shift to the point of a digitizer sample boundary, the fine-delay component must wrap, while the CD must be incremented or decremented by one sample on the same data. In the data path, the CD takes place before the FFT. If the latency through the FFT is L data samples, the CD must thus pre-emptively adjust L data samples before the fine delay wraps. The F-engine needs to account for fringe rotation due to Nyquist sampling and downconversion in the receiver stages or processing pipeline. The combined effect of both fringe stopping and fine delay compensation is implemented as a phase lookup index in a cos–sine table on the FPGA in the form phase $= \omega f + \theta$ for a given spectral channel of index f, where ω is the delay and θ is the instantaneous phase offset due to downconversion.

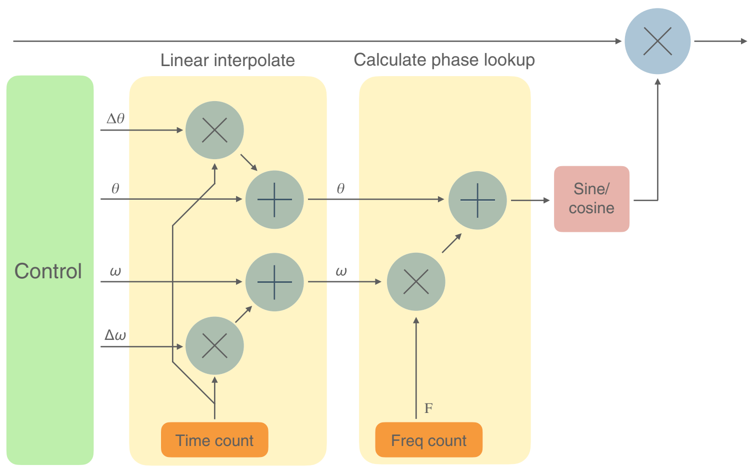

Fig. 5 Fine delay. This is implemented in FPGA logic. Updates are periodically supplied by the control and monitoring subsystem. Linear interpolation is performed in between updates.[10] θ represents the instantaneous phase and ω the delay. $\Delta\theta$ and $\Delta\omega$ represent the change in these respective fields and are used in interpolation process between updates.

Hardware linear interpolation is supported, so software can periodically update the phase offset and delay values, along with update rates for each, and then leave the hardware to continue interpolating independently. This reduces the software processing load and greatly reduces the control and monitoring bandwidth required for each input during runtime. Figure 5 shows the data path for fine delay and fringe stopping.

6.4 *Wideband Channelization*

The channelization operation employs a polyphase filterbank (PFB) constructed from a polyphase FIR filter followed by a discrete Fourier transform. MeerKAT's wideband coarse PFB employs a Hann window polyphase FIR filter together with a computationally efficient FFT. There are three wideband coarse channelization modes specified for MeerKAT. Each has a different number of taps in the FIR and a different number of FFT output channels:

- F-engine 1k has a 16-tap polyphase FIR with a 2048-point real FFT (1024 output channels).
- F-engine 4k has a 16-tap polyphase FIR with a 8192-point real FFT (4096 output channels).
- F-engine 32k has a eight-tap polyphase FIR with a 65,536-point real FFT (32,768 output channels).

An N-point real CASPER FFT[11] produces $\frac{N}{2}$ complex output channels per spectrum, and only the positive frequency half is output (since the other half is a mirror image) to save logic resources.

The wideband real FFT that is used is a streaming FFT. This consumes additional multiplier and adder resources but requires less buffering (thereby reducing memory requirements on an already memory-constrained device). It accepts input samples in parallel, allowing for processing of very wideband signals on low-clock-rate FPGAs, provided sufficient resources are available. The PFB signal path uses fixed-point arithmetic to make best use of the FPGA's onboard DSP multipliers and BRAM. To prevent numerical overflows in our fixed point data path due to FFT processing gain, bit growth is limited through the FFT butterfly stages by an optional downshift at the output of each stage before quantization. Received sky signals are noise-like where a growth factor of $\sqrt{2}$ is expected per butterfly stage. In these cases, the shift schedule can be relaxed to allow the noise to occupy a significant fraction of the available dynamic range. However, in the presence of narrowband RFI, such as that received from satellites and terrestrial

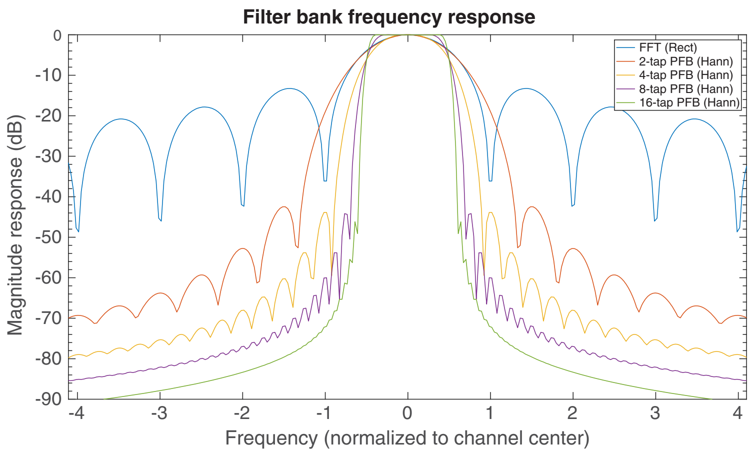

Fig. 6 Channel filter response for the polyphase filter bank. The traces shown illustrate the differences for various taps. Increasing the number of taps improves the per channel roll-off but comes at the expense of increased logic and memory resources utilized on the FPGA.

broadcast towers, significant power can accumulate in individual bins, so in RFI-rich environments, it may be necessary to be more aggressive with this shifting schedule.

Care must be taken not to be too aggressive with the shifting schedule, especially with longer FFTs, and risk dividing down the signal so much that SNR is degraded. At the extreme, the signal of interest could be lost entirely. The numerical overflow of a value in any FFT stage ruins the results in multiple output channels due to the employed algorithm's subsequent reuse of intermediate results in later stages. Over-ranges and under-ranges inside the FFT must thus be avoided and spectra discarded if these do occur. A status bit is provided to indicate if an overflow has occurred, so the shift schedule can be appropriately re-adjusted.

In the 1k and 4k wideband F-engines, our input data samples are 10 bits. In the polyphase FIR, there are 18-bit FIR coefficients and the output data path grows to 18 bits (36-bit complex) to minimize quantization errors while using the onboard multipliers and BRAMs efficiently (SKARAB multipliers can do 18×25-bit operations and BRAMs have 9/18/36-bit interfaces). Coefficients in the FFT are 18 bits. The data path through the FFT is 18 bits. This allows eight bits of bit growth for the 10-bit input values and allows for a sufficient dynamic range to prevent numerical overflows due to RFI, while preserving enough signal to provide the efficiency required (Fig. 6).

In the 32k wideband F-engines, it was discovered that an 18-bit data path did not provide sufficient dynamic range. RFI could grow more than eight bits in the 16-stage FFT, and shifting to prevent numerical overflow would remove too much signal. The data path was increased to 22 bits, which improved performance sufficiently. Coefficient resolution was kept at 18 bits.

6.5 *Narrowband Channelization*

The narrowband systems are variants of the wideband design, with added components to select-out and only process a subset of the band. The direct digital synthesizer (DDS) (Sec. 6.5.1) and digital downconverter (DDC) (Sec. 6.5.2) are positioned before the PFB in a narrowband design.

6.5.1 *Direct digital synthesizer*

A complex DDS is required for the mixing stage in the digital downconvert stage. The DDS is constructed around multiple cos–sin look up tables (LUTs), stored in BRAM on the FPGA. The output of the DDS is eight complex samples per FPGA clock as this stage is part of the wideband (i.e., multiple samples per FPGA clock tick) processing pipeline that forms the initial stages of the narrowband F-engine. DDS word length is 18 bits (18 bits total with 17 bits allocated to precision keeping values in the range $-1:1$) per sample (i.e., 36-bit for cos–sin). The LUTs are referenced using the lower 32 bits of the incoming timestamp. The timestamp is first scaled by a multiplier derived from the required frequency scalar.

6.5.2 *Digital downconverter*

This function digitally downconverts and filters a sub-band of the pass band for further channelization by the PFB. The DDC consists of a complex mixer, followed by a low-pass filter. After filtering, the signal is downsampled to reduce processing requirements in the PFB. Two variants are available:

- 107 MHz filter, with downsampling by 16 and
- 53.5-MHz filter with downsampling by 32.

6.5.3 *Channelization*

The narrowband PFB behaves just like the wideband PFB where the only difference is that the narrowband PFB does not process multiple samples in parallel. The first stage of the PFB is a Hann window FIR filter followed by a shallow FIFO. The FIFO depth is 16 words and is used to buffer the input data to the FFT stage (which follows). The stage following the FIFO is a complex 32k-point FFT. The input data are complex, so no spectrum symmetry occurs (unlike that in the wideband real input case). The FFT output is reordered as the channel data ordering is required to be from $\frac{-fs}{2}$ to $\frac{+fs}{2}$ (i.e., ch0 $= -53.5$ MHz and ch32767 $= +53.5$ MHz in the 107-MHz band case).

6.6 *Equalization*

The MeerKAT specification calls for gain flatness of ± 5 dB across the pass band. This can be corrected for in the F-engine, following flux calibration on a known source, by providing per-channel gain correction of at least ± 4 dB. The resolution and accuracy with which to set the gain is highly dependent on the number of bits used for the correlation operation. The MeerKAT implementation uses an 8-bit correlator with 16-bit linear gain coefficients, giving a linear resolution of half a step (0.5/32, 768), providing a 90-dB range, with a variable resolution over the gain range of 6 dB (low end) to 0.00027 dB (high end).

In addition to the ability to adjust the amplitude, the F-engine provides the ability to adjust the phase of individual frequency channels on any input. This facility can be used to adjust for phase imperfections across the band of individual receivers. The steering of tied array beams is based on the assumption that all antennas have been phased-up and gain-corrected. These corrections are not calculated by the F-engine. It is expected that other subsystems will be responsible for calculating the complex gain correction coefficients per frequency channel for each antenna, from the baseline correlation products, for all of the required baselines.

6.7 *Quantization*

To meet the F-engine efficiency requirement, at least 4 bits are required for correlation, with the signal amplitude carefully controlled to maintain quantization efficiency.[12] To reduce quantization noise and preserve signal integrity in high-dynamic-range signals, a wide data path is preserved through most of the signal chain in the FPGA. This starts in the polyphase FIR. Due to the limited bandwidth of the HMC memories and network links, the data precision is reduced before the corner turn operation and transfer to X-engines. MeerKAT employs an 8-bit correlator. The gain must be applied to the signal before quantization to ensure that the signal of interest occupies the top 8 bits. In ordinary signed 2s-complement arithmetic, there is one more value below zero than above. For a low precision signal, this extra value can introduce a significant DC bias when signals are represented. This is prevented by rounding values having the largest possible negative value to the second largest negative value.

6.8 *Corner Turn*

The quantized datastream is prepared for transport by the network, prior to packetization, for distribution to X-engines. This is primarily a matrix transpose operation to group a selection of time series samples from a single FFT channel into each packet. The number of time samples in a packet is chosen to match the internal accumulation in the X-engine core, typically 256 time samples per packet for MeerKAT. The accumulation length within the X-engine core must increase for systems with larger numbers of antennas to keep the data rates flowing from the X-engine cores to the accumulators manageable, which in turn increases memory requirements within the F-engines as larger packets must now be buffered as the matrix size increases. In MeerKAT, an HMC is used to accommodate this large buffering requirement. This reordering requires a matrix-transpose operation, which is typically double-buffered, writing into one memory while reading from another in a different order. This requires a memory capacity of $2MP$ for each input, where M is the number of frequency channels and P is the packet size. The bandwidth capability of the HMC infrastructure requires that one HMC is used per polarization.

The matrix transpose operation on SKARAB is complicated by various aspects when using HMC memory:

- HMCs do not guarantee that the result of read operations will be output in order;
- operations do not have guaranteed latency;
- HMC bus width and speed are fixed.

Output order is dealt with by having a small buffer on the output of each HMC to reorder packets. We read data out of each buffer simultaneously and interleave it to satisfy the interleave requirement. The maximum reorder that is possible is 512 operations, so these buffers are 512 words deep and read out delayed by 512 values. To deal with fixed bus width and speed, a two-stage transpose operation is required. A large, primary corner-turn with dimensions of $2 \times M \times 256$ values, hosted in an off-FPGA HMC device, followed by a second smaller transpose in on-chip BRAM of dimensions $2 \times 2 \times 16 \times 256$ values (where 256 is the packet size, M is the number of frequency channels, and 16 is the number of parallel values accommodated by the width of the HMC data bus). 16 complex 16-bit (8-bit real and 8-bit imaginary) values are transported via the 256-bit wide data bus to each HMC on every second FPGA clock cycle taking $M/8$ clock cycles to write in a single spectrum. After 256 spectra are written, readout begins. 256 values are read from both HMCs, each address containing 16 FFT channels. Each 256-bit wide value is divided up and reordered such that polarization data are packed together. These 16 32-bit values are then each written to a separate buffer. When 256 values have been written, readout from each buffer in order begins to output the 16 packets containing frequency data as the other half of the buffer is written to.

A double-buffered corner turn allows for arbitrary reordering. This is leveraged to smooth the switch load: packets are not output sequentially by frequency, but indexed modulo the number of X-engines. Thus, the first set of 16 packets is sent to the first X-engine, and the second set of 16 packets goes to the second X-engine, rather than to the first engine (X-engines process a contiguous chunk of the band and expect data packets containing data for both polarizations from a single antenna). Otherwise, all of the F-engines would be sending a lot of sequential packets to one X-engine, rather than just 16, filling switch buffers while the data are emptied out over a single link. So, instead, F-engine output packets "hop" through the band to round-robin through the available X-engines. An initial offset is added such that each F-engine begins sending packets to a different X-engine. This further reduces the requirement on switch port buffers.

7 Test Vectors

Test vector generators (TVGs) are used to inject data into the data pipeline to confirm correct operation or to aid with debugging. A multiplexer allows the test data to be chosen instead of the usual data and is controlled by bits from a register that can be written to from software. There are

TVGs at various locations in the signal processing chain. Figure 3 shows where TVGs are present within an F-engine subsystem.

7.1 *Timestamps*

The timestamp TVG is used to insert a timestamp into the F-engine data stream. If enabled, incoming data are ignored and the current timestamp as unpacked from the incoming datastream is used instead. This input switch is useful for tracing timestamps as they propagate through the data pipeline and can be used for verifying reordering of data as required within the CD.

7.2 *Impulse Generator*

The impulse generator can be used to replace the incoming digitizer data with test data as it exits the reorder block and before it enters the CD function. The test data contain zeros except for an impulse of specified amplitude at a specific time offset. Data generated for every spectrum replace incoming data and contain an impulse at the specified offset. Only data words are replaced, and sync and data valid signals are ignored. The impulse generator is useful for testing the operation of the FFT (exercising known Fourier transform pairs for an impulse and time shifted impulse)[13] and the delay- and phase-compensation system. Impulses generated are periodic and positioned to occur at the required timestamp based on the FFT length and are synchronized with the system sync generated from incoming timestamps.

7.3 *Tone Targets*

The tone target TVG inserts zeros into all data entering the corner turner except for a value into a particular frequency channel. The output should be zeros in all packets, with a value filling a packet depending on the frequency selected. These values determine the offset of the impulse for polarization 0 and 1, respectively. This is useful in troubleshooting data reordering and accumulations later in the correlator processing chain such as X-engine network data reorder. It is also useful to have a known deterministic input to the X-engine to verify the vector accumulation process.

8 X-Engine: A MeerKAT Correlator

The principal function of the X-engine is to multiply the channelized data from each antenna with every other antenna (cross-correlation) as well as with themselves (autocorrelations) in the configured instrument. This results in the number of products (or baselines) for N inputs to be $n_{\mathrm{bls}} = \frac{N(N-1)}{2}$ and is a $O(N^2)$ compute problem. The correlation processes are described in Sec. 8.3. A dataflow representation of the X-engine is shown in Fig. 7.

8.1 *Ethernet Receive and Data Format*

Data enter the FPGA through a 40-GbE link via a 256-bit-wide data path. The data ingest on an X-engine is channelized data as received from the F-engines. It is important to note that, while an F-engine will produce a full spectrum based on the selected mode (e.g., 32,768 channels in a wideband mode), only a subset of this spectrum is subscribed to and correlated on any given X-engine. Each X-engine receives channelized data from all F-engines for the subset range allocated to it. Data intended for each X-engine are received by subscribing to all relevant Ethernet multicast groups transmitted by the appropriate F-engines. Figure 8 shows the F-engine to X-engine data mapping and Fig. 9 shows the data format as received by the X-engine via 40 GbE.

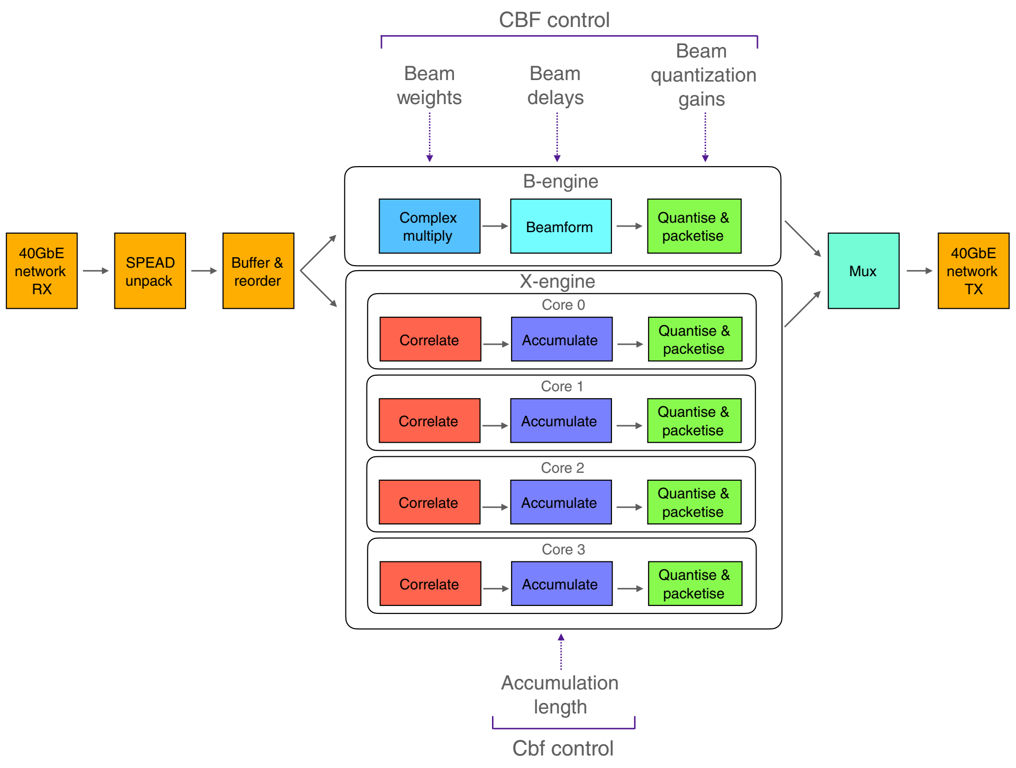

Fig. 7 Block diagram illustrating the data pipeline for both the correlation (X-engine) and beamforming (B-engine) processes. It is worth noting that both the X-engine and the B-engine are colocated on the same SKARAB processing hardware for the range of frequencies that are assigned to be processed. This has the benefit of sharing resources such as 40 GbE as well as data reordering when packets are received out-of-order due to network latency variations.

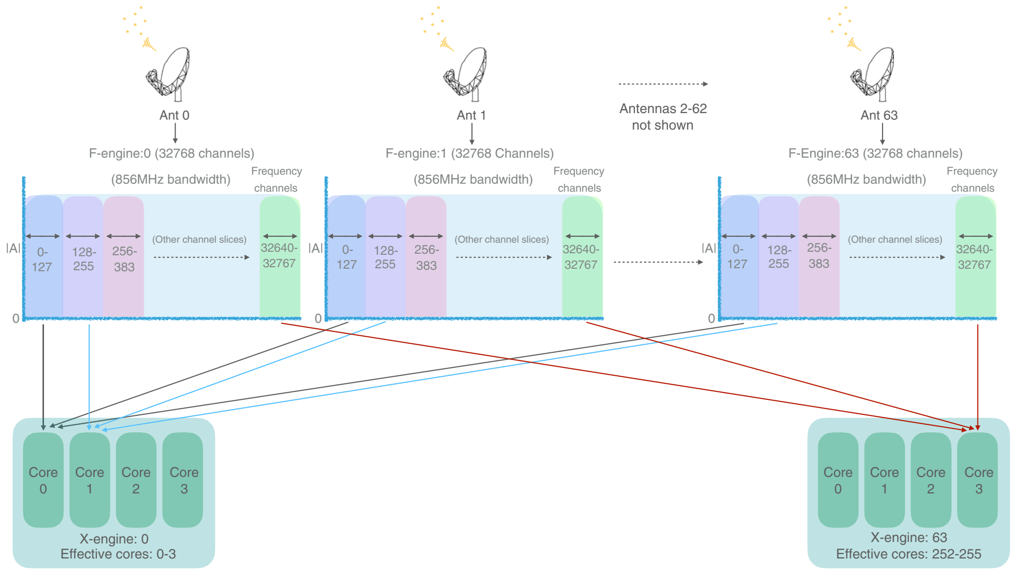

Fig. 8 F-engine to X-engine data mapping. Each F-engine produces a full spectrum that is divided up into subsets and distributed among the X-engine cores. This figure illustrates a 64-antenna system channelizing 856-MHz bandwidth (L-band) in 32,768 channels. A total of 64 X-engines are required (256 cores in total), and each core will process combined bandwidth from all antennas. Figure 9 shows how the channelized data are packaged and received by each X-engine core.

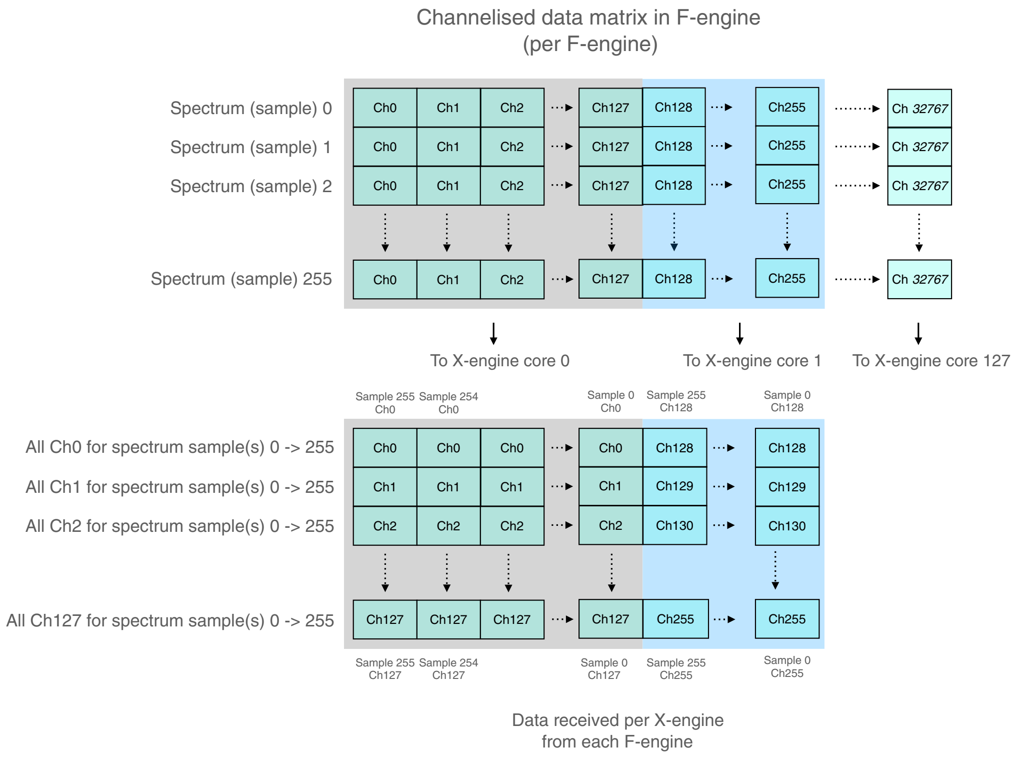

Fig. 9 Receiver data format per X-engine. The data ordering in this illustration are as it is formatted in the F-engine. Each X-engine receives a subset of channelized data to process. In this example illustration, each X-engine core will receive and process 128 channels. Each core will receive the allocated channelized data from all F-engines (64 in this case for a 32,768-channel wideband system). The F-engines each store 256 spectra with 32,768 channels per spectra. Each X-engine core receives all 256 samples from each F-engine for each of the channel subset ranges. The F-engines perform this matrix transpose termed a "corner turn" prior to transmission.

8.2 *Network Data Reorder*

Similarly to the F-engine, resend, data-scrambling, and nondelivery of packets can cause problems for the X-engines, which must process F-engine data in the order in which it was produced. Section 6.2 covers network data reordering applicable to the X-engine.

8.3 *X-Engine Core*

Each X-engine consists of four X-engine cores with each operating on a dedicated range of frequencies as received from the F-engines. The number of channels to operate on is decided at the time that the instrument is built and configured. The number of X-engine cores required depends primarily on the bandwidth and the number of antennas in the system. This can be computed as $n_x = N\frac{B}{f_x}$, where N is the number of antennas, B is the analog bandwidth, and f_x is the bandwidth per X-engine core. For example, a 64-antenna system with 856-MHz bandwidth and a per core bandwidth of 214 MHz would require $n_x = 256$ cores. Each logical X-engine contains four cores per board, therefore requiring 64 X-engines (64 SKARAB FPGA boards).

The number of channels per X-engine core is computed from the number of total channels to be processed (F-engine) as well as the total number of X-engine cores in the system (n_x). Using the same 64-antenna array channelizing into 32k channels in L-band using wideband mode would require each X-engine core to operate on $\frac{\text{Number channels}}{n_x} = \frac{32{,}768}{256} = 128$ channels. Data entering each core contain complex data from both polarizations of an antenna.[10]

Once an instrument is configured, antenna data from a specific frequency channel (coming from an F-engine) enter as a block of 256 time samples, with the oldest sample arriving first

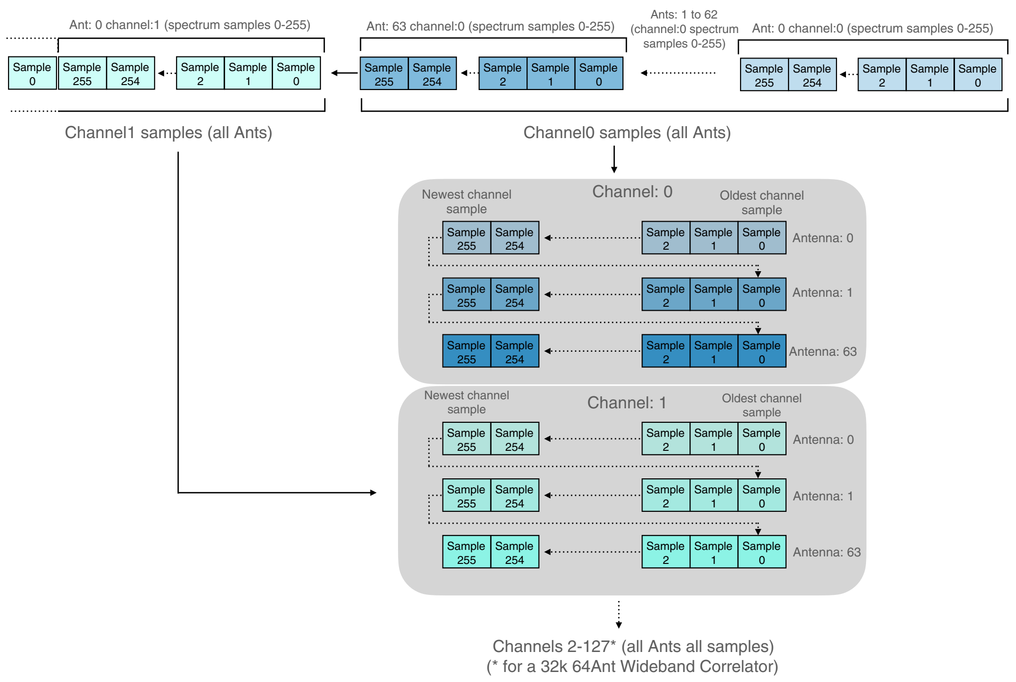
Ant: 0 channel:1 (spectrum samples 0-255)
Ant: 63 channel:0 (spectrum samples 0-255)
Ants: 1 to 62 (channel:0 spectrum samples 0-255)
Ant: 0 channel:0 (spectrum samples 0-255)
Channel1 samples (all Ants)
Channel0 samples (all Ants)
Channel: 0
Channel: 1
Newest channel sample
Oldest channel sample
Sample 255
Sample 254
Sample 2
Sample 1
Sample 0
Antenna: 0
Antenna: 1
Antenna: 63
Channels 2-127* (all Ants all samples)
(* for a 32k 64Ant Wideband Correlator)

Fig. 10 Received data order for the X-engines and B-engines. Each X-engine core is responsible for processing a range of channels. In this illustration, each core is responsible for a subset of 128 channels. The F-engines store and send the respective subset of channels each with 256 spectra per transmission, so each sample is a complex valued frequency-domain sample from the 256 stored spectra. After transmission, the F-engines collect another 256 spectra prior to transmission. The B-engines utilize the same data; however, processing occurs independently. Section 9 discusses B-engine data processing.

followed by the block from the next antenna in sequence and so on through all antennas. Figure 10 shows the X-engine core ingest data order. Following this, the next frequency channel in the band assigned to this X-engine enters and so on until all frequency channels have been transferred.

8.3.1 *Correlation*

To perform the correlation process, data are shifted through a long shift register where there are $\frac{N}{2}$ stages (N is the number of antenna). The data are combined with data from the other antennas to produce the correlation products. A total of $\frac{N(N-1)}{2}$ correlation products is produced. The product is the multiplication of channelized data from one antenna with the complex conjugate of the other, with the result being the product in magnitude and difference in phase between the two vectors.[14] The data enters the X-engine with a precision of 8 bits per component of a complex value. The complex multiplication produces a 16-bit value, and accumulating this value 256 times requires a precision of 25 bits.

At the end of $\frac{N}{2}$ shift register stages, the data loop around, so they can be combined with the second $\frac{N}{2}$ antenna's data. As the correlation values are produced, they are summed together such that the block of 256 products is reduced to a single value even as the number of baselines increases. This pre-accumulation helps reduce the data rate being passed to the HMC vector accumulator (VACC). Figure 11 shows the basic structure of the X-engine that produces correlation products.

8.3.2 *Scaling*

The number of X-engine cores required to process F-engine data is dependent on the number of antennas used in the observation. For example, a wideband four-antenna system requires

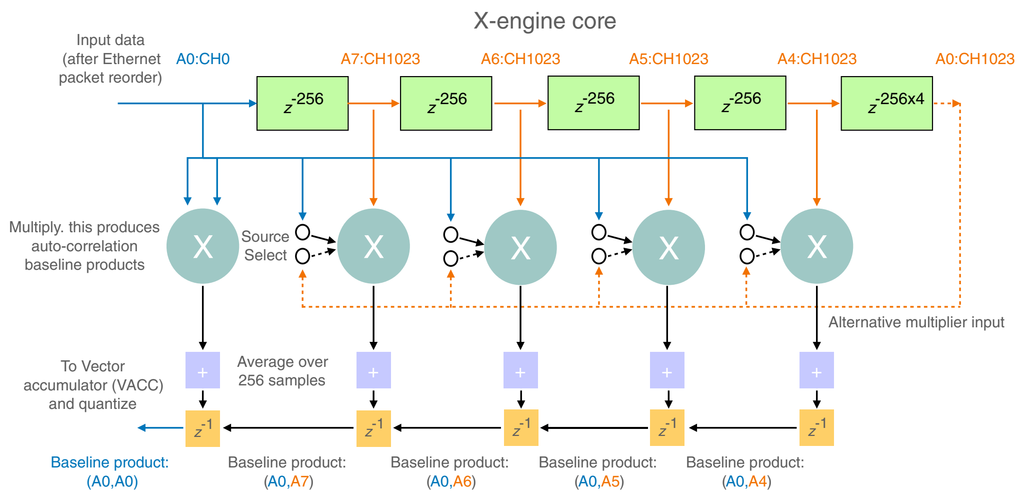
X-engine core
Input data (after Ethernet packet reorder)
A0:CH0
A7:CH1023
A6:CH1023
A5:CH1023
A4:CH1023
A0:CH1023
z^{-256}
z^{-256x4}
Multiply. this produces auto-correlation baseline products
Source Select
Alternative multiplier input
To Vector accumulator (VACC) and quantize
Average over 256 samples
z^{-1}
Baseline product: (A0,A0)
Baseline product: (A0,A7)
Baseline product: (A0,A6)
Baseline product: (A0,A5)
Baseline product: (A0,A4)

Fig. 11 An illustration of an X-engine core for an eight-antenna correlator, 32,768 channels in wideband (856-MHz bandwidth). The example in the illustration produces the first baseline (blue) of antenna 0 with itself (autocorrelation). The other products are not valid as the product is the current window (A0 in blue) with the previous window (A7:A0 in orange). The baseline product matrix showing valid baseline products is shown in Table 1. It should be noted that the X-engine cores operate on a range of channels. In this case, the number of channels per core is 1024. In the case of baseline (0,0) (antenna 0 autocorrelation), each sample that is clocked in is multiplied with itself and pre-accumulated (before the larger HMC-based accumulation). At the same time, baseline products ($A0_{\text{current}}, A7_{\text{previous}}$) as well as baseline products ($A0, A6$); ($A0, A5$); ($A0, A4$) are calculated ($A6$; $A5$; $A4$ are all from the previous window). When the current window $A0:CH0$ (and the remaining 255 samples in the sequence for antenna 0, channel 0) reaches the output of the delay element $z^{-256\times4}$ the "source select" switches over to permit the $\frac{N}{2}$ baseline computations (first one being $A0_{\text{current}}, A7_{\text{current}}$).[10]

4 X-engine boards (each with four X-engine cores—16 total cores in this case). Each X-engine core processes one dual-polarization complex data stream. The number of X-engine cores maps 4:1 to the number of antennas (or 1:1 with the number of X-engine configured SKARAB boards). The number of boards (SKARABs) required for narrowband mode is dependent on the mode and scales according to the reduction in bandwidth. For example, the 107-MHz narrowband mode requires a factor of 8 less X-engine resources. An eight-antenna system (107 MHz) would require one X-engine (four cores). A corner case exists when less than eight antennas are used as one board would still be required. Logic in the four-antenna X-engine handles this at compilation time.

8.3.3 *Vector accumulator*

Correlation products exit the X-engine and are passed to an HMC-based VACC for long-term accumulation. Signal components that are coherent will increase in amplitude quicker than those that are noise-like during accumulation. The number of bits in the accumulator must support this increasing difference in amplitude. The HMC supports a maximum bitwidth of 32, but one bit is used as a synchronization bit, so 31 bits is used, being the largest number of bits possible. Incoming data enter via an FIFO, and the associated timestamp for each value is registered. The VACC can be configured to begin accumulation at a specified timestamp to allow, for example, accumulations to start at second boundaries or other interesting points in time. During accumulation, if the incoming timestamps do not increase as expected, a re-arm is triggered. This resets the accumulator and flushes the current accumulation. Accumulating will then automatically begin again, beginning two accumulations ahead of when the error was detected.

Once the current accumulation is complete, a read out of the final vector will occur in between the next accumulation's operation. Double buffering is used to ensure that the current accumulation does not overwrite the previous before being read out. A small buffer is used to

Table 1 A correlation results matrix representing the order of output baseline correlations. It should be noted that not all outputs are valid as the X-engine cores process multiple windows of data. The baseline correlations (paired antenna) are read out from top to bottom, right to left. The entries in bold indicate valid correlated baselines. The italics entries are also valid; however, they need to be conjugated as the antenna pairing order is reversed. Each row represents the baseline product pairings for a set of 256 samples for the antenna pair.

0,0	7,0	6,0	5,0	4,0
1,1	**0,1**	7,1	6,1	5,1
2,2	**1,2**	**0,2**	7,2	6,2
3,3	**2,3**	**1,3**	**0,3**	7,3
4,4	**3,4**	**2,4**	**1,4**	**0,4**
5,5	**4,5**	**3,5**	**2,5**	**1,5**
6,6	**5,6**	**4,6**	**3,6**	**2,6**
7,7	**6,7**	**5,7**	**4,7**	**3,7**
0,0	**7,0**	**6,0**	**5,0**	*4,0*
1,1	0,1	**7,1**	**6,1**	*5,1*
2,2	1,2	0,2	**7,2**	*6,2*
3,3	2,3	1,3	0,3	*7,3*
4,4	3,4	2,4	1,4	0,4
5,5	4,5	3,5	2,5	1,5
6,6	5,6	4,6	3,6	2,6
7,7	6,7	5,7	4,7	3,7
4,4	3,4	2,4	1,4	0,4
5,5	4,5	3,5	2,5	1,5
6,6	5,6	4,6	3,6	2,6

reorder the output values before they are written to an output FIFO as the HMC may reorder read transactions. Once this FIFO fills up to half capacity, a request is sent to the bus arbiter. The bus arbiter grants the bus access to this data source; data are read from the FIFO, and a SPEAD packet is created for transmission via multicast Ethernet.

9 B-Engine: A MeerKAT Beamformer

In addition to the correlation performed by the X-engine cores, a beamformer (B-engine) is colocated with the X-engine. The beamformer's operations occur independently of the X-engine cores; however, it is an advantageous pairing of computation as the B-engine and X-engine share the required input data format. For this reason, after Ethernet packet reordering, the data (8-bit complex) is buffered separately (duplicated) to provide an independent stream to the B-engine.

The data ingested by the B-engine must appear as the lowest frequency and antenna first. The data from the next antenna for the same frequency are then expected. After the final antenna's data for a frequency have been supplied, the data for the next frequency can be applied and so on. Figure 10 shows this data-ordering relationship.

The first process performed on the data is to duplicate it, so multiple simultaneous beams can be formed and pointed at different locations. In MeerKAT, each polarization is duplicated four

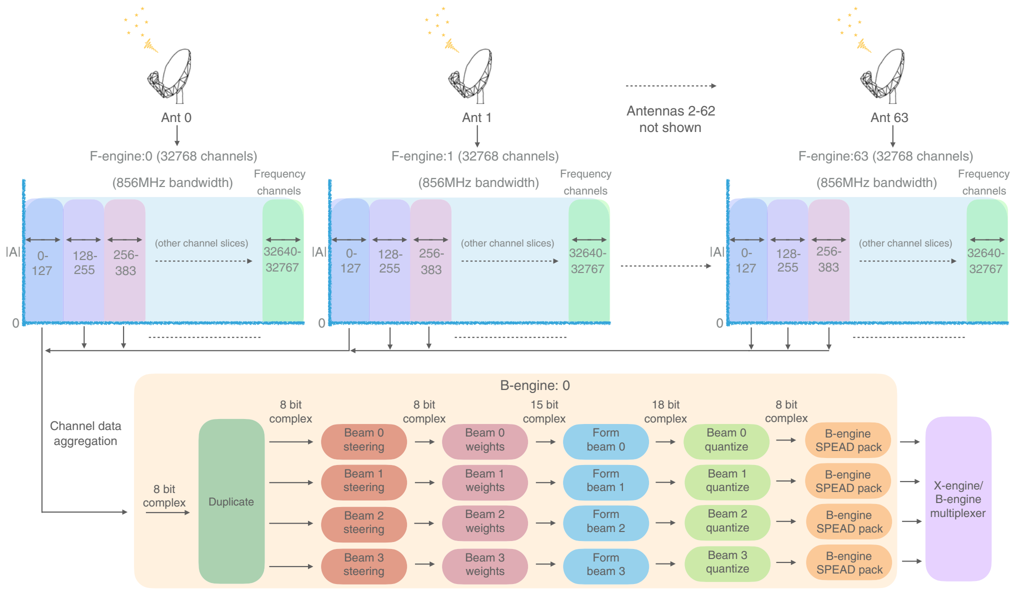

Fig. 12 An illustration of the data pipeline for the B-engine. The B-engine is colocated with the X-engine on the same SKARAB board. Like the X-engine, the B-engines operate on a subset of the full band. In this illustration, a 64-antenna system each with an F-engine producing a total of 32,768 channels are divided up into 256 subsets of 128 channels per subset. As per Fig. 8, each X-engine core processes one 128 channel subset (four subsets per X-engine board); however, the B-engine per X-engine board operates on all four subsets allocated to the X-engine. The channel data aggregation shown in the illustration is conceptual in that it is to indicate that all channelized data received by the X-engine are also passed to the single B-engine core for processing.

times so that there are eight unique beams (four dual polarization beams). The second function performed is to point each beam at a unique position within the primary beam. This is accomplished by modifying the phase of each complex value by means of complex multiplication with another complex value with a phase that corresponds to the delay to be applied. The delay value applied is unique to each antenna in every beam. The complex value associated with the phase change at the lowest frequency is supplied from software (via a control and monitoring system), as well as the delta of this value with respect to frequency. This change is then applied to the complex value before the value is applied to each frequency.

After beam-steering, a scalar weight is applied to each antenna before summation. Beams are then formed by summing the data from all antennas, reducing the output data rate by a factor of N (where N is the number of antenna). Finally, quantization is required to reduce the data transport requirements. A unique scalar per beam is used first to scale the data so as to reduce signal loss. As with the X-engine core, a block of 256 time values are processed at a time in the B-engine. The timestamp of the first data sample, used to construct the first spectrum in the block, is transferred with the processed B-engine data. This timestamp originates in the F-engine (at the output of the corner turner) and is attached to the packet transferring the data. This timestamp is regenerated for use in the X-engine (Fig. 12).

10 Bus Arbitration

To accommodate the colocation of X-engines and B-engines on a single SKARAB, bus arbitration must be provided to handle transmissions over the shared 40 GbE interface. As data become ready for transmission in the X- and B-engines, a request to transmit is sent to the bus arbiter. This arbiter cycles through the data sources in a round-robin fashion until it encounters a source requiring the bus. Once it does, a signal is sent allowing data transmission, and the arbiter holds the bus for that source until the end-of-frame (EOF) signal is received. This EOF signal indicates the end of the packet, informing the arbiter that the bus can now be released.

The 40-GbE logic empties the TX data FIFO at a lower rate (156.25 MHz) than the FPGA fills it (∼240 MHz). Logic within the arbiter forces dead cycles between packets such that there are sufficient dead cycles to prevent FIFO overflows due to bursts. If a data source has not sent any data at any point within a defined window period, a watchdog timer times out and the bus is released to prevent lock-ups.

11 CBF Control

CBF subsystems each have control interfaces for real-time parameter adjustments. These parameters are controlled by a higher level control and monitoring subsystem (not detailed in this work). The control parameters are broken down per CBF subsystem:

F-engine:

- Delays: (Secs. 6.3.1, 6.3.2)
- FFT shift (Sec. 6.4)
- Gain (equalization): (Sec. 6.6)

X-engine:

- Accumulation length: (Sec. 8.3.3)

B-engine:

- Beam weights: (Sec. 9)
- Beam delays: (Sec. 9)
- Beam quantization gains: (Sec. 9)

12 CBF Monitoring

Low-level system monitoring within the F- and X-engines is possible through the inclusion of debug and monitoring logic. There are two primary logic subsystems used to provide this functionality: snapshots (Sec. 12.1) and counters (Sec. 12.2).

12.1 *Snapshots*

Within the F-engine, semi-real-time monitoring of a live system is possible through included logic in the design termed snapshots. Snapshots (implemented in BRAM in the FPGA) are statically connected to various points within the internal pipeline and are used to capture windows of data. Sections 12.1.1, 12.1.2, and 12.1.3 cover the three capture points within an F-engine. These test points are useful in debugging a live system if required.

12.1.1 *40 GbE snapshots*

The SKARAB's 40-GbE core provides hardware support for capturing raw packet data as well as packet and data rates as it enters the F- and X-engines. Capture is triggered by a valid EOF signal. These capture buffers are included for verifying the data received for debugging purposes if required.

12.1.2 *Input data snapshots*

Inside of the F-engine snapshots is available to capture raw sample data from both polarizations and is located just before the PFB, which can be used to analyze raw digitizer data before channelization. There is circular capture and extra value capture support functionality. The extra value captured is 32 bits, 12 bits from the MSB (the timestamp entering the F-engine increases by 12 bits from 12 bit boundaries). Various modes including circular capture mode as well as capture at a specific time can be enabled.

12.1.3 *Quantizer snapshots*

These snapshots in the F-engine allow the capture of data after quantization for polarizations 0 and 1, respectively. This is configured so that the whole spectra is captured and can confirm PFB operation and gain settings. For very large numbers of frequency channels, the start of capture can be delayed. This allows a full spectrum to be captured by successively capturing different portions and then combining them.

12.2 *Counters*

Various events in the processing chain cause counters to increment. These can be used to localize errors to particular blocks, as well as to determine more systematic errors. These counters are a fixed size and wrap. They can generally be reset to clear them if necessary. The value of the counter itself is often not useful, but the increase in value, and the rate thereof, on subsequent reads conveys information about various events in the processing pipeline. Some useful counters include:

- counts of valid packets received and transmitted. These can show if data are being received and transmitted;
- counts of the number of syncs that have passed points in the DSP pipeline;
- counts of the number of valid data items that have passed various points in the pipeline. These can similarly provide a clue as to whether there are errors preventing the proper operation of the block.

13 Resource Utilization

The MeerKAT telescope has multiple operating modes for both F- and X-engines, where the choice of mode is dependent on the type of observation and number of antennas to be used in the array. FPGA gateware is not one-size-fits-all, and multiple pre-built gateware files are available for deployment based on the instrument to be built. Each FPGA design has a different parameter set and has different resource utilization within the FPGA. Tables 2 and 3 list the resources required per design.

14 Conclusion

This paper presents the design of the CBF as deployed for the MeerKAT radio telescope in South Africa. The paper provides an overview of the MeerKAT radio telescope before focusing on the channelization followed by the correlation and beamforming processes, which form the

Table 2 Resources utilization for the various F-engine modes. Wideband (WB) F-engine modes do not differ significantly in resource usage with the exception being Block RAM (BRAM) in the 32k channel version. The 32k design timing constrained and required additional effort to the pipeline to reduce timing paths. The two narrowband (NB) designs have a significant jump in resource utilization, particularly in DSP slices for the 53.5-MHz mode. This is due to a doubling of the number of samples in parallel required for processing per FPGA clock cycle in the DDC.

F-engine mode	DSP (%)	LUTs (%)	Registers (%)	Slice (%)	BRAM (%)
WB: 1k	16	47	40	85	67
WB: 4k	17	47	41	79	68
WB: 32k	14	47	44	82	95
NB—107 MHz: 32k	33	47	44	78	89
NB—53.5 MHz: 32k	54	60	54	91	89

Table 3 Resource utilization for various X-engine modes. X-engines exist for antenna array sizes of 4, 8, 16, 32, and 64 antennas for the various F-engine modes. Utilization between different X-engines for the same array size but different F-engines (i.e., 1k, 4k, 32k) does not change significantly. Resource utilisation for all X-engine modes for use with F-engine(s) with 4096 channels is listed.

X-engine mode	DSP (%)	LUTs (%)	Registers (%)	Slice (%)	BRAM (%)
4 Ant	17	45	40	83	72
8 Ant	20	43	37	80	67
16 Ant	27	46	40	83	70
32 Ant	42	49	43	76	76
64 Ant	70	57	51	91	94

processing stages for the CBF. The correlator deployed is a *FX* style correlator, and the channelization process as well as delay compensation and equalization is performed by a subsystem termed an F-engine. The correlation (X-engine core) and beamforming (B-engine) processes are performed by a subsystem termed the X-engine. The X-engine and B-engine are colocated and operate independently; however, they share a common data ingest. Processed data from the F-engine, X-engine, and B-engine are sent into an Ethernet data switching network for other subsystems to subscribe through multicast transmissions. The X-engine(s) subscribe to F-engine data and other data processing system(s) further downstream subscribe to X-engine and B-engine data. It is also possible for third-party USE to subscribe to any data stream on the 40-GbE switching network if required.

15 Source Code

MeerKAT CBF is built using a variety of open-sourced hardware, software, and FPGA gateware, all of which are available on Github. Table 4 lists the relevant software and Github URL.

Table 4 Description of various hardware and software libraries as used in MeerKAT. It should be noted that the SKARAB citation[8] only refers to the design files for the SKARAB. FPGA gateware (source and built bit files) for the F-engine, X-engine, and B-engine are colocated in the same Github repository.[15] Programming of the SKARAB (FPGA) and associated software access registers and read memory in the designs can be performed using *casperfpga*.[16] Software to configure and start a FX correlator can be performed using *corr*2.[17] All other supporting software such as control and monitoring and data processing is available on Github[18] but not detailed in this paper.

Description	Library	Github URL
SKARAB	SKARAB	8
F-engine, X-engine, and B-engine design files	mkatfpga	15
Library to interact and interface with CASPER Hardware	casperfpga	16
Configure FX correlators, beamformers	corr2	17

16 Appendix

Photographs showing the MeerKAT radio telescope. MeerKAT is located in the South African Karoo in the Northern Cape. MeerKAT dishes are assembled onsite. All back-end processing is performed in the Karoo Array Processing Building (KAPB) which houses control servers, SKARAB's and networking equipment. Backup power generation is also located onsite.

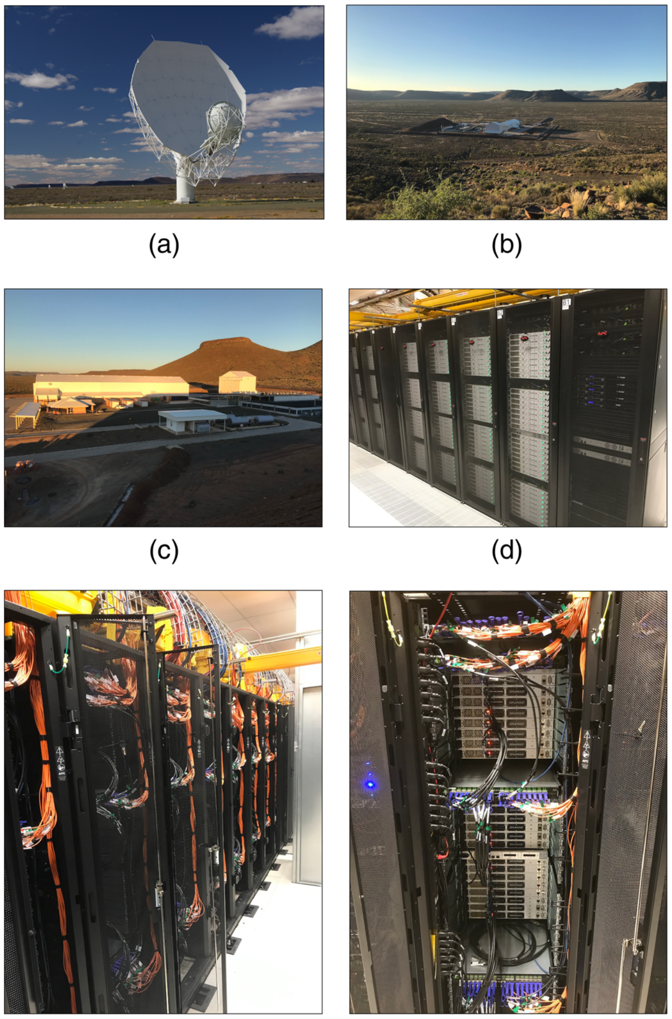

Fig. 13 (a) MeerKAT 13.5-m Gregorian-offset antenna. The MeerKAT telescope consists of 64 identical units. (b) MeerKAT site. The site buildings consist of two antenna assembly sheds, processing building (KAPB), and temporary accommodation. (c) MeerKAT site : the two large buildings (distant) are the antenna assembly sheds. The KAPB is partly suppressed below ground (middle-far right of frame). (d) Front view of a rack assembly inside the KAPB. The white paneled units are SKARABs. (e) Rear view of SKARAB rack assembly. All orange colored cables are 40-Gbps active optical cables. (f) Inside a SKARAB rack assembly. The black cables are 40-Gbps short-length (2 m) copper cables. A 40-Gbps leaf switch can be seen at the base of the frame.

Acknowledgments

The South African Radio Astronomy Observatory (SARAO), a facility of the National Research Foundation, is responsible for managing all RA initiatives and facilities in South Africa, including the MeerKAT Radio Telescope in the Karoo and the Geodesy and VLBI activities at the HartRAO facility. SARAO also coordinates the African Very Long Baseline Interferometry

Network (AVN) for the eight SKA partner countries in Africa, as well as South Africa's contribution to the infrastructure and engineering planning for the Square Kilometre Array Radio Telescope. The Collaboration for Astronomy Signal Processing and Electronics Research (CASPER) is a research community promoting open-source hardware, software, and programming tools.

References

1. F. Camilo et al., "Revival of the magnetar PSR j1622–4950: observations with MeerKAT, Parkes, XMM-Newton, Swift, Chandra, and NuSTAR," *Astrophys. J.* **856**, 180 (2018).
2. "Ceph storage cluster," https://docs.ceph.com/en/pacific/rados/index.html.
3. "Amazon S3 REST API," https://docs.aws.amazon.com/AmazonS3/latest/API/Welcome.html.
4. M. J. Slabber et al., "MeerKAT data distribution network," *Proc. SPIE* **10707**, 107070H (2018).
5. J. Manley et al., "SPEAD: streaming protocol for exchanging astronomical data," 2020, https://casper.ssl.berkeley.edu/wiki/SPEAD.
6. "Peralex: Radio Astronomy (SKARAB)," https://www.peralex.com/radio-astronomy/.
7. "Hybrid Memory Cube - HMC Gen2," https://www.micron.com
8. "Square Kilometre Array Reconfigurable Application Board (SKARAB)," https://github.com/casper-astro/casper-hardware/tree/master/FPGA_Hosts/SKARAB.
9. "Collaboration for Astronomy Signal Processing and Electronics Research (CASPER)," https://casper.berkeley.edu/.
10. J. Manley, "A scalable packetised radio astronomy imager," PhD thesis, University of Cape Town (2015).
11. "CASPER FFT," https://casper-toolflow.readthedocs.io/en/latest/src/blockdocs/Fft.html.
12. A. R. Thompson, D. T. Emerson, and F. R. Schwab, "Convenient formulas for quantization efficiency," *Radio Sci.* **42**(3) (2007).
13. A. Oppenheim and R. Schafer, *Discrete-Time Signal Processing*, Prentice-Hall Signal Processing Series, Prentice Hall (1989).
14. W. Urry et al., "The ATA correlator," 2007, https://www.seti.org/ata-memo-series.
15. "Github Repository: MKATFPGA," https://github.com/ska-sa/mkat_fpga.
16. "Github Repository: CASPERFPGA," https://github.com/ska-sa/casperfpga.
17. "Github Repository: Corr2," https://github.com/ska-sa/corr2.
18. "Github Repository: SKA-SA," https://github.com/orgs/ska-sa/repositories.

Andrew van der Byl is a signal processing engineer at SARAO. He received his PhD from the University of Cape Town, South Africa, in 2012. His research interests include signal and image processing, machine learning, and reconfigurable hardware. Prior to SARAO, he held a research position at the University of Cape Town.

James Smith received his BEng degree in 2013 and his MEng degree in 2019, both in electronic engineering, from the University of Pretoria. Currently, he is a digital signal processing (DSP) engineer with the SARAO in Cape Town, South Africa.

Tyrone van Balla received his BSc (Eng) degree in electronic engineering from the University of Cape Town in 2014. He joined SARAO in 2015 as an electronics engineer working on software and hardware for the MeerKAT Telescope's Digital Backend. In 2017, he assumed the role of systems engineer for the DSP group and now serves as the functional manager for DSP, leading DSP-related activities across the MeerKAT, MeerKAT Extension, and AVN Projects.

Alec Rust received his MSc degree in electronic engineering from the University of Stellenbosch in 2000. He joined SARAO in 2005 as a digital design engineer for the Karoo Array Telescope digital systems. He now manages qualification and performance testing of digital systems that are deployed on the MeerKAT telescope and related systems.

Amish Patel received his BSc (Hons) degree in electronic engineering from the University of KwaZulu-Natal. Since 2017, he has been working on FPGA and GPU software and firmware development for DSP applications in the MeerKAT correlator-beamformer (CBF). His current work is focused on developing DSP pipelines, using NVIDIA's latest GPUs, for the next-generation MeerKAT CBF.

Gareth Callanan received his MSc degree in electronic engineering from the University of Cape Town, South Africa, and is currently working toward his PhD at Lund University, Sweden. Between 2018 and 2021, he worked on various FPGA and GPU pipelines for the MeerKAT CBF subsystem. His current research involves developing tools to accelerate the design of embedded systems for streaming applications.

Adam Isaacson received his MSc degree in electrical engineering from the University of Cape Town. Since 2000, he has been involved in various roles for a radar company, and he joined SARAO in February 2016. He has been involved as an FPGA design engineer for the SKARAB hardware, which is part of the digital back-end for the MeerKAT radio astronomy project, and now serves as the hardware manager for the Electronics Group.

Henno Kriel received his MSc degree in electronic engineering from the University of Cape Town, South Africa. Since 2012, he has been involved in various roles with the development of the digital backend and frontend subsystems for the MeerKAT radio astronomy project. His work focuses on FPGA-based processing platforms designs, wideband direct RF digitization, and array system integration.

Omer Mahgoub received his BEng (electronic) from the University of Pretoria in 2007. From 2008 to 2012, he was a hardware engineer at the Radar and Electronic Warfare group of the Council for Scientific and Industrial Research. Since 2012, he has been involved in the development of various hardware for the Digital Back End and Receiver Groups at SARAO on projects such as Meerkat, AVN, and MeerKat Extension.

Biographies of the other authors are not available.