Radar Hardware Accelerator - Part 1

The *Radar Hardware Accelerator User's Guide* (in two parts) describes the Radar Hardware Accelerator architecture, features, and operation of various blocks and their register descriptions. The purpose is to enable the user to understand the capabilities offered by the Radar Hardware Accelerator and to program it appropriately to achieve the desired functionality.

This user's guide is divided into two parts. The first part (this document) provides an overview of the overall architecture and features available in the Radar Hardware Accelerator. The main features, such as, windowing, FFT, and log-magnitude are covered in this part.

The second part of the user's guide covers additional features like CFAR-CA and other advanced usage possibilities. The second part of the user's guide is optional and can be skipped if the user is interested only in the FFT computation capability.

This user's guide is organized as follows: Section 1 covers the introduction and high-level architecture. Section 2, Section 3, and Section 4 describe the state machine, trigger mechanisms, input/output formatting, and general framework for using the accelerator. Section 5 describes the primary computational unit features, namely, windowing, FFT, and log-magnitude. Section 6 presents a use-case example.

Contents

1 Radar Hardware Accelerator – Overview 3
2 Accelerator Engine – State Machine 9
3 Accelerator Engine – Input Formatter 15
4 Accelerator Engine – Output Formatter 21
5 Accelerator Engine – Core Computational Unit 26
6 Radar Hardware Accelerator – Use Case Example 35
Appendix A 47

List of Figures

1 Radar Hardware Accelerator (mmWave 14xx Device) 4
2 Accelerator Engine Block Diagram 6
3 Parameter-Set Configuration Memory (512 Bytes) 7
4 State Machine 9
5 Input Formatter 15
6 Input Formatter Source Memory Access Pattern (Example) 17
7 Invalid Configuration Example 18
8 Input Formatter Data Scaling 18
9 Output Formatter 21
10 Output Formatter Destination Memory Access Pattern (Example) 23
11 Output Formatter Data Scaling 24
12 Core Computational Unit 26
13 Core Computational Unit Block Diagram 27
14 Windowing Computation 28
15 Butterfly Stage Fixed-Point 30
16 FFT SFDR Performance With and Without Dithering 30
17 Accuracy of Log2 Computation 32

Copyright © 2017, Texas Instruments Incorporated

18 Layout of Samples in ADC Buffer (Ping) 36
19 Layout of First-Dimension FFT Output Samples in Accelerator Local Memory 38
20 Layout of First-Dimension FFT Output Samples in Radar Data Cube Memory 39
21 Layout of Second-Dimension FFT Input Samples 42
22 Layout of Second-Dimension FFT Output Samples 43
23 Layout of Zero-Padded, Third-Dimension, FFT Output Samples 45
24 Register Layout of Parameter-Set Registers 47

List of Tables

1 State Machine Registers 12
2 Input Formatter Registers 19
3 Output Formatter Registers 24
4 FFT Computation Time 31
5 Core Computational Unit Registers 32
6 Chirp Configuration Used for Illustration 35
7 Key Register Configurations for First-Dimension FFT 37
8 Key Register Configurations for Second-Dimension FFT 41
9 Key Register Configurations for Third Dimension FFT 44
10 Key Register Configurations for Log-Magnitude Processing 46

Trademarks

ARM, Cortex are registered trademarks of ARM Limited.
All other trademarks are the property of their respective owners.

Copyright © 2017, Texas Instruments Incorporated

1 Radar Hardware Accelerator – Overview

This section provides an overview of the Radar Hardware Accelerator. The section covers the key features of the accelerator and overall architecture.

1.1 Introduction

The Radar Hardware Accelerator is a hardware IP that enables off-loading the burden of certain frequently used computations in FMCW radar signal processing from the main processor. It is well known that FMCW radar signal processing involves the use of FFT and log-magnitude computations to obtain a radar image across the range, velocity, and angle dimensions. Some of the frequently used functions in FMCW radar signal processing can be done within the Radar Hardware Accelerator, while still retaining the flexibility of implementing other proprietary algorithms in the main processor.

1.2 Key Features

The main features of the Radar Hardware Accelerator are as follows.

- Fast FFT computation, with programmable FFT sizes (powers of 2) up to 1024-pt complex FFT
- Internal FFT bit width of 24 bits (for each I and Q) for good SQNR performance, with fully programmable butterfly scaling at every radix-2 stage for user flexibility
- Built-in capabilities for simple pre-FFT processing – specifically, programmable windowing, basic interference zeroing-out, and basic BPM removal
- Magnitude (absolute value) and log-magnitude computation capability
- Flexible data flow and data sample arrangement to support efficient multidimensional FFT operations and transpose accesses as required
- Chaining and looping mechanism to sequence a set of accelerator operations one-after-another with minimal intervention from the main processor
- CFAR-CA detector support (linear and logarithmic)
- Miscellaneous other capabilities of the accelerator:
  - Stitching two or four 1024-point FFTs to get the equivalent of 2048-point or 4096-point FFT for industrial level sensing applications where large FFT sizes are required
  - Slow DFT mode, with resolution equivalent to 16K size FFT, for FFT peak interpolation purposes (for example, range interpolation)
  - Complex vector multiplication and Dot product capability for vectors up to 512 in size

This user's guide is divided into two parts. The first part covers the high-level architecture and key features such as windowing, FFT, and log-magnitude. The (optional) second part covers additional features such as CFAR-CA, complex multiplication, and so forth.

Copyright © 2017, Texas Instruments Incorporated

1.3 High Level Architecture

The Radar Hardware Accelerator module is loosely coupled to the main processor (ARM® Cortex®-R4F in the mmWave 14xx device). The accelerator is connected to a 128-bit bus that is present in the main processor system, as shown in Figure 1.

The Radar Hardware Accelerator module comprises an accelerator engine and four memories, each of 16KB size, which are used to send input data to and pull output data from the accelerator engine. These memories are referred to as *local memories* of the Radar Accelerator (ACCEL_MEM). For convenience, these four local memories are referred to as ACCEL_MEM0, ACCEL_MEM1, ACCEL_MEM2, and ACCEL_MEM3.

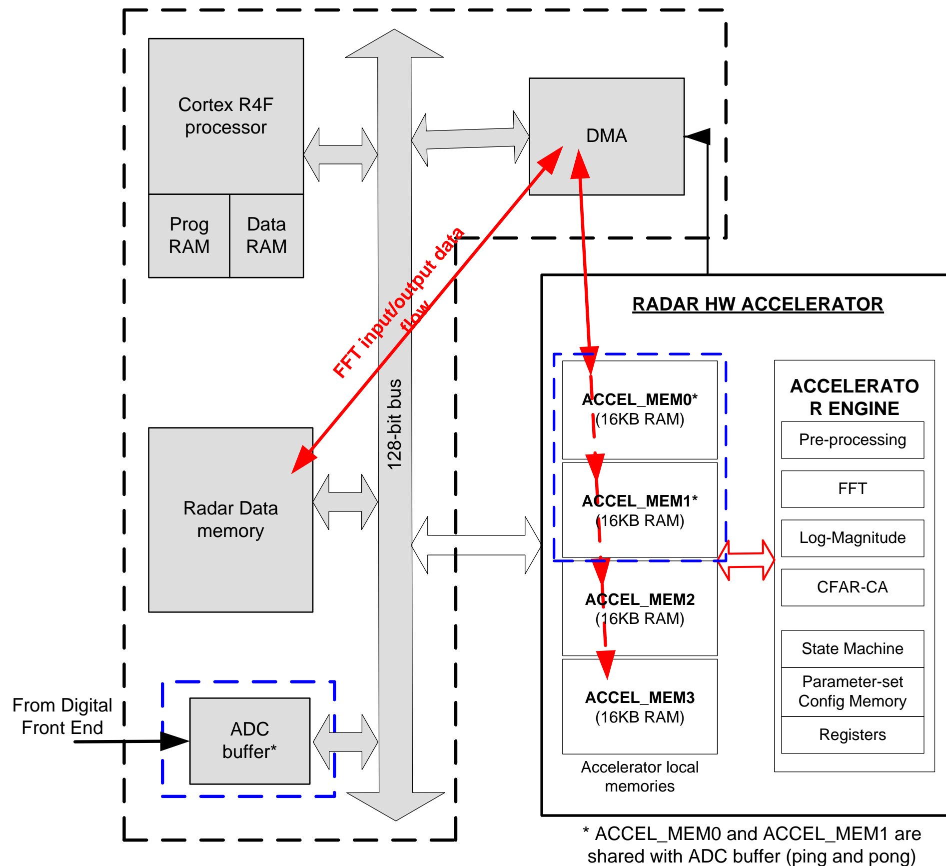


Figure 1. Radar Hardware Accelerator (mmWave 14xx Device)

Copyright © 2017, Texas Instruments Incorporated

1.3.1 High-Level Data Flow

The typical data flow is that the DMA module is used to bring samples (for example, FFT input samples) into the local memories of the Radar Hardware Accelerator, so that the main accelerator engine can access and process these samples. Once the accelerator processing is done, the DMA module reads the output samples from the local memories of the Radar Hardware Accelerator and stores them back in the Radar data memory for further processing by the main processor. In Figure 1, the red arrow shows data movement from the Radar data memory into the accelerator local memories for the FFT and other processing steps. The red arrow also shows the output samples from the accelerator being picked up by the DMA and written back into the Radar data memory for further processing by the main processor.

Note that in the mmWave 14xx device, the Radar Hardware Accelerator is included as part of a single chip along with the mmWave RF and analog front end. In this device, two of the accelerator local memories, namely ACCEL_MEM0 and ACCEL_MEM1, are directly shared with the ping and pong ADC buffers (which are 16KB each) – such that the ADC output samples for first-dimension FFT processing are directly and immediately available to the Radar Hardware Accelerator at the end of each chirp, without needing a DMA transfer. After the first-dimension FFT processing is complete (typically, at the end of the active transmission of chirps in a frame), it is possible to freely use these memories for second-dimension FFT processing by bringing in data to these memories through DMA transfer.

The purpose behind the four separate local memories (16KB each) inside the Radar Hardware Accelerator is to enable the *ping-pong* mechanism, for both the input and output, such that the DMA write (and read) operations can happen in parallel to the main computational processing of the accelerator. The presence of four memories enables such parallelism. For example, the DMA can be configured to write FFT input samples (ping) into ACCEL_MEM0 and read FFT output samples (ping) from ACCEL_MEM2. At the same time, the accelerator engine can be working on FFT input samples (pong) from ACCEL_MEM1 and writing FFT output samples (pong) into ACCEL_MEM3. However, both the DMA and the accelerator cannot access the same 16KB memory at the same time. This would lead to an error (refer to the STATERRCODE register description in Table 3.

The Radar Hardware Accelerator and the main processor (Cortex-R4F) in the mmWave 14xx device operate on a single clock domain and the operating clock frequency is 200 MHz.

The accelerator local memories are 128-bits wide, for example, each of the 16KB banks is implemented as 1024 words of 128 bits each. This allows the DMA to bring data into the accelerator local memories efficiently (up to a maximum throughput of 128 bits per clock cycle, depending upon the DMA configuration).

It is important to note that any of the four local memories can be the *source* of the input samples to the accelerator engine and any of the four local memories can be the *destination* for the output samples from the accelerator engine – with the important restriction that the source and destination memories cannot be the same 16KB bank. Note also that the accelerator local memories do not necessarily need to be used in ping-pong mode and can instead be used as larger 32KB input and output memories, if the use case requires. The address space for the four 16KB memories is contiguous and thus the source and destination memory can effectively be larger than 16KB.

1.3.2 Configuration

The operations of the Radar Hardware Accelerator are configured using registers, which are of two types – *parameter sets* and *common* (common for all parameter sets) registers. The purpose of the parameter sets is to enable a complete sequence of various accelerator operations to be preprogrammed (with appropriate source and destination memory addresses and other configurations specified for each operation in that sequence), such that the accelerator can perform them one after the other, with minimal intervention from the main processor.

The parameter-set register configurations are programmed into a separate 512-byte *parameter-set configuration memory*. A state machine built into the accelerator handles the loading of one parameter-set configuration at a time and sequences the preprogrammed operations one after another. This process is further explained in later sections of this user's guide. Section 6 shows a use-case example, which illustrates this well.

Copyright © 2017, Texas Instruments Incorporated

1.4 Accelerator Engine Block Diagram

As previously mentioned, the Radar Hardware Accelerator module consists of four local memories of 16KB each (ACCEL_MEM) and the main accelerator engine. The accelerator engine has the following five components (as shown in Figure 2) – a state machine, input formatter block, output formatter block, core computational unit, and the 512-byte parameter-set configuration memory.

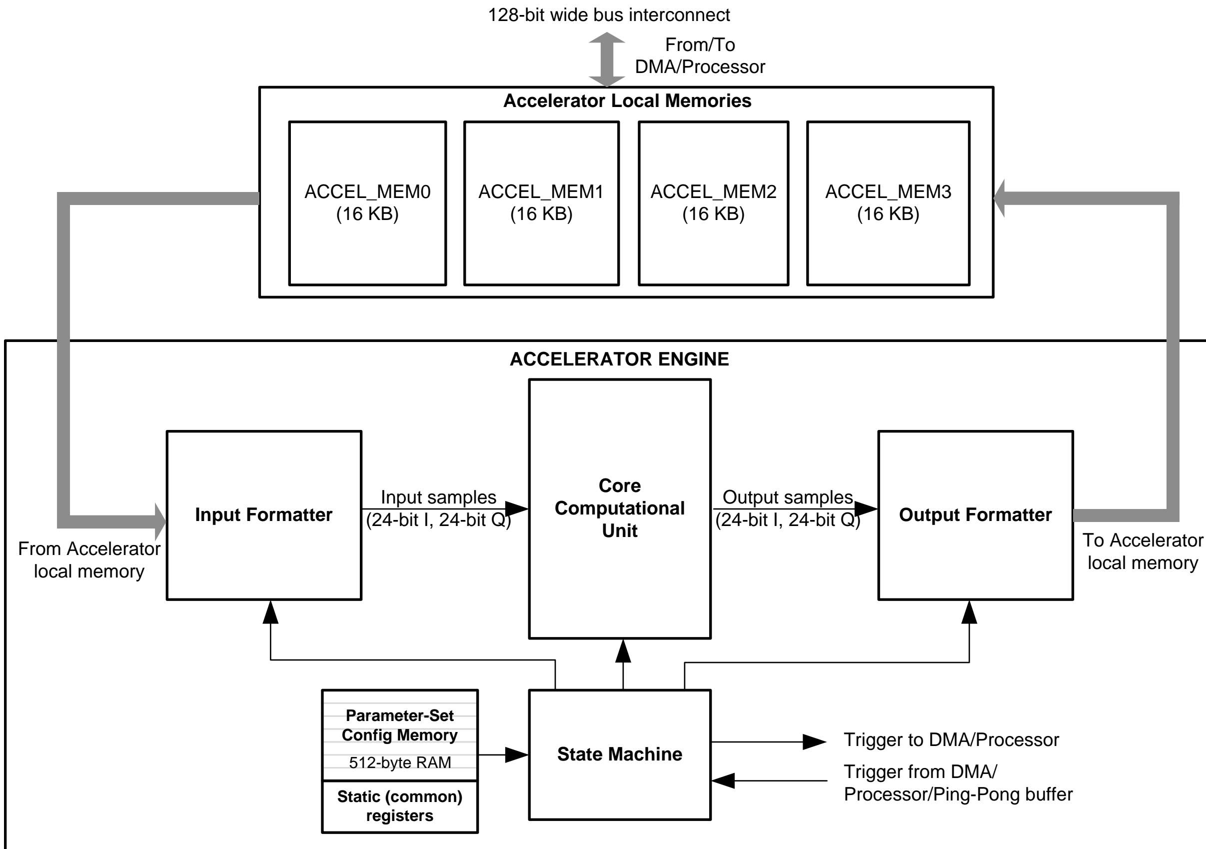


Figure 2. Accelerator Engine Block Diagram

The purpose of these components is as follows.

- State machine: the state machine is responsible for controlling the overall operation of the accelerator – specifically, the starting, looping, stopping, as well as triggering and handshake mechanisms between the accelerator, DMA, and main processor. The state machine is also closely connected to the parameter-set configuration memory and takes care of sequencing and chaining a sequence of multiple accelerator operations as programmed in the parameter-set configuration memory.
- Input formatter: the input formatter block is responsible for reading the input samples from any one of the local memories and feeding them into the core computational unit. In this process, this block provides flexible ways of accessing the input samples, in terms of 16-bit versus 32-bit aligned input samples, transpose read-out, flexible scaling, and sign extension to generate internal bit-width of 24 bits, and so on. Lastly the input formatter block provides 24-bit complex samples as input to the core computational unit. The local memory (memories) from which the input formatter reads the input samples is called the *source* memory.
- Output formatter: the output formatter block is responsible for writing the output samples from the core computational unit into the local memories. This block also provides flexible ways of formatting the output samples, in terms of 16-bit versus 32-bit aligned output samples, transpose write, flexible scaling from internal bit-width of 24 bits, to 16-bit or 32-bit aligned output samples, sign-extension, and so on. The local memory (memories) to which the output formatter writes the output samples is called the *destination* memory.

Copyright © 2017, Texas Instruments Incorporated

- Core computational unit: the core computational unit contains the main computational logic for various operations, such as windowing, FFT, magnitude, log2, and CFAR-CA calculations. The unit accepts a streaming input from the input formatter block (at the rate of one input sample per clock cycle), performs computations, and produces a streaming output to the output formatter block (typically at the rate of one output sample per clock cycle), with some initial latency depending on the nature of the computations involved.
- Parameter-set configuration memory: this is a 512-byte RAM that is used to preconfigure the sets of parameters (register settings) for a chained sequence of accelerator operations, which can then be executed by the state machine in a loop. This allows the accelerator to perform a preprogrammed sequence of operations in a loop without frequent intervention from the main processor.

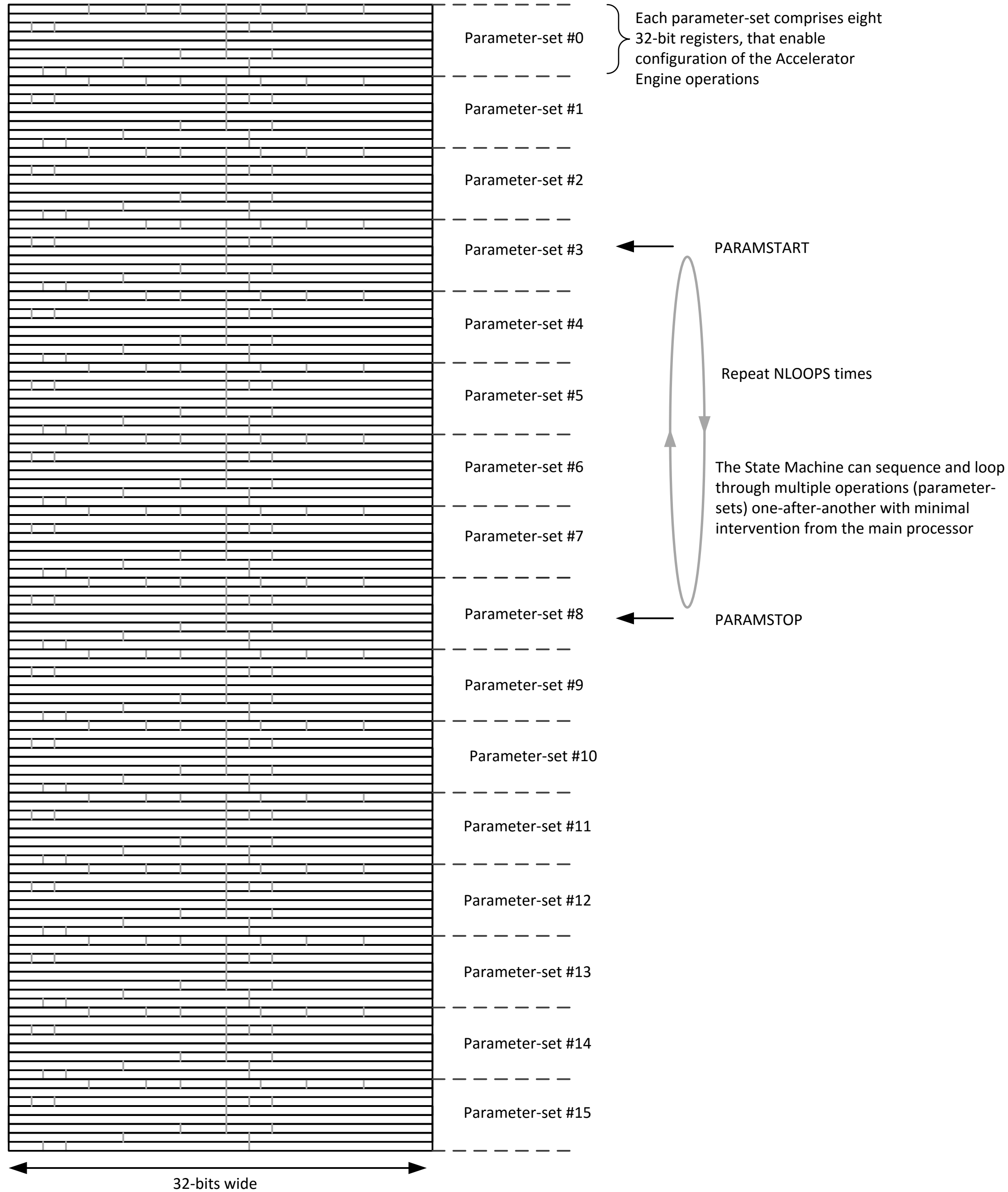


Figure 3. Parameter-Set Configuration Memory (512 Bytes)

Copyright © 2017, Texas Instruments Incorporated

The number of parameter sets that can be preconfigured and sequenced (chained) is 16. This means that up to 16 accelerator operations can be chained together and these can then be looped as well, with minimal intervention from the main processor. For example, operations like FFT, log-magnitude, and CFAR-CA detection can be preconfigured in the parameter-set configuration memory and the state machine can be made to sequence them one after another and run them in a loop for specified number of times. There is a provision available to interrupt the main processor and/or trigger a DMA channel at the end of each parameter set if required. This allows various ways by which the accelerator, DMA, and the main processor can work together to establish a data and processing flow. As shown in Figure 3, each parameter set contains the equivalent of eight 32-bit registers, which corresponds to total RAM size of 16 × 8 × 32 bits = 512 bytes for the parameter-set configuration memory.

The layout of the parameter-set register map is provided in Appendix A. The detailed descriptions of the registers is provided in the various sections, as and when the functionality of each component is presented.

1.5 Accelerator Engine Operation

The accelerator engine and the local memories run on a single clock domain. The overall operation of the accelerator can be summarized as follows. The accelerator engine is configured by the main processor through common configuration registers (common for all parameter sets), as well as the parameter-set configuration memory. As explained earlier, the former comprises common register settings for overall control of the accelerator engine, and the latter comprises the 16 parameter-set specific settings which control the functioning of the accelerator for each of its *chained* sequence of operations.

When the accelerator engine is enabled, the state machine kicks off and controls the overall operation of the accelerator, which involves loading the parameter sets one at a time from the parameter-set configuration memory into various internal registers of the accelerator engine and running the accelerator as per the programmed configuration for each parameter set one after another. The entire procedure then repeats in a loop for a programmed number of times (NLOOPS described later).

Each parameter set includes various configuration details such as the accelerator mode of operation (FFT, Log2, and so on), the source memory address, number of samples, the destination memory address, input formatting, output formatting, trigger mode for controlling the start of computations to ensure proper handshake with the DMA, and so on.

1.5.1 Data Throughput

Once the state machine has loaded the registers corresponding to the current parameter set to be executed, the data flow happens as follows: at each clock cycle, one sample from the source memory is read by the input formatter and fed into the core computational unit with appropriate scaling and formatting as configured. The data interface between the input formatter and the core computational unit is a 24-bit complex bus (24-bit for each I and Q) which streams one input sample every clock cycle. The core computational unit processes this streaming sequence of input samples and in general, produces a streaming output also at one sample every clock cycle, after an initial latency period. Thus for most operations (FFT, log-magnitude, CFAR-CA, and so on), in steady state the core computational unit maintains a streaming data rate of one sample per clock cycle. The data interface between the core computational unit and the output formatter is also a 24-bit complex bus (24-bit for each I and Q) and the output formatter is responsible for writing into the destination memory, with appropriate scaling and formatting as configured.

The next section provides more details regarding the state machine, including its detailed operation, registers, trigger mechanisms, and so on.

Copyright © 2017, Texas Instruments Incorporated

2 Accelerator Engine – State Machine

This section describes the state machine block present in the accelerator engine (see Figure 4). This block, together with the input formatter and output formatter blocks described in the next two sections, provides the overall framework for establishing the data flow and using the accelerator for various computations.

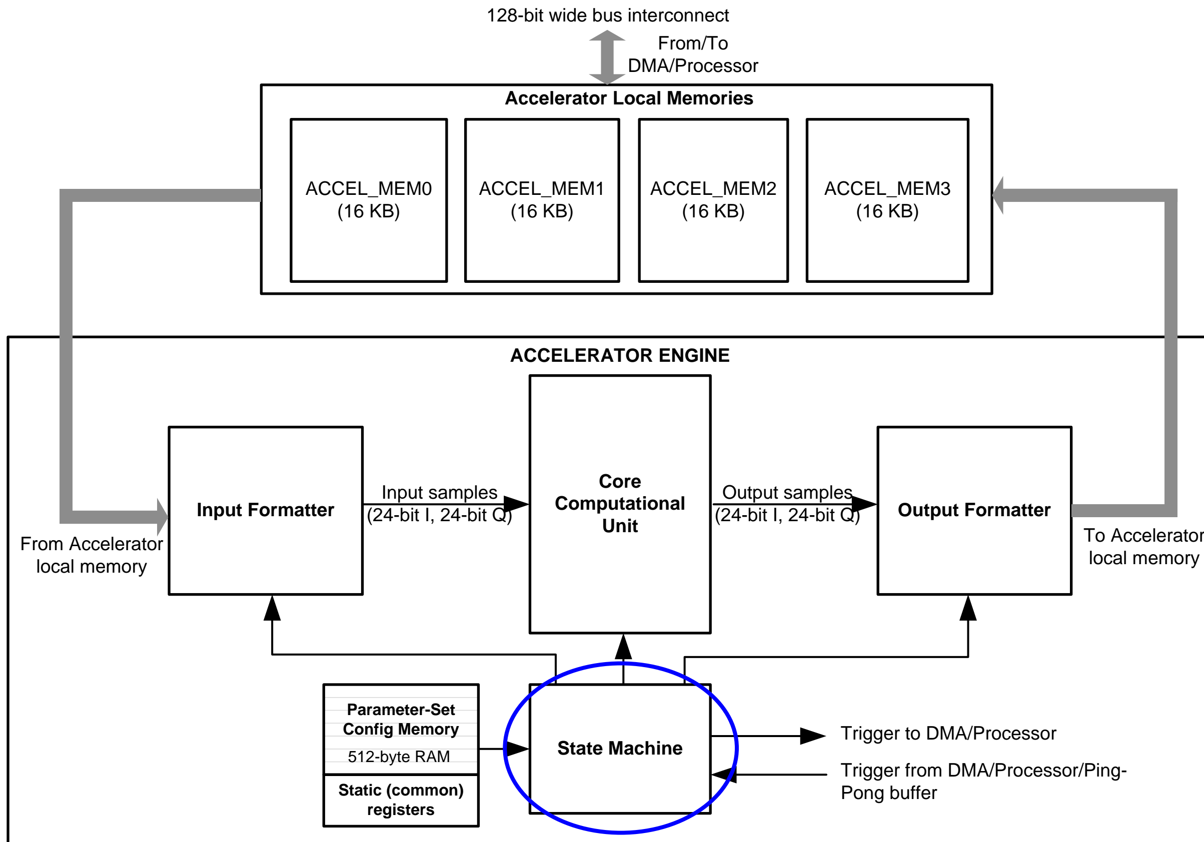


Figure 4. State Machine

2.1 State Machine

The state machine controls the overall functioning of the Radar Hardware Accelerator. The state machine controls the enabling and disabling of the accelerator, as well as supports sequencing an entire set of operations (configured using parameter-set configuration memory), and looping through those operations one after another without needing frequent intervention from the main processor.

2.1.1 State Machine – Operation

The state machine block and the entire accelerator remain in reset and disabled state by default. The state machine (and hence the accelerator in general) is enabled by setting the ACCCLKEN register bit, followed by writing 111b into the ACCENABLE register.

Note that a complete list of registers pertaining to the state machine is provided in Table 1. Some of the registers are common (common for all parameter sets) registers, whereas some other registers are parameter-set registers, which as explained in the previous section means that they can be uniquely programmed for each of the 16 parameter sets. For each register, Table 1 lists whether it is part of the parameter set or not. Table 1 also provides a brief description of each register.

Copyright © 2017, Texas Instruments Incorporated

When enabled, the state machine steps through (one after another) the parameter sets programmed in the parameter-set configuration memory and executes the computations as per the configuration of each parameter set. The registers PARAMSTART and PARAMSTOP define the starting index and ending index within the 16 parameter sets, so that only those parameter sets between the start and end indices are executed by the accelerator, as shown in Figure 3. The state machine also loops through these parameter sets for a total of NLOOPS times (unless NLOOPS is programmed as 0 or 4095, in which case the loop does not run or runs infinite times respectively). As an example, if the state machine needs to be configured to run the first four parameter sets in a loop 64 times, then the registers should be programmed as follows: PARAMSTART = 0, PARAMSTOP = 3, and NLOOPS = 64.

For each parameter set, there is a TRIGMODE register, which is used to control when the state machine starts executing the computations for that parameter set. This control is useful, for example, to ensure that the input data is ready in the accelerator local memory (source memory) before the computations are started. Specifically, it is possible to trigger the start of computations after completion of a DMA transfer, or, after a ping-pong switch happens in the ADC buffer, and so on. The TRIGMODE register setting thus controls when the accelerator operation is triggered for the current parameter set and there are four trigger mechanisms supported as listed in the next subsection. Once triggered, the state machine loads all the registers from the parameter-set configuration memory for the current parameter set into corresponding internal registers of the accelerator and starts the actual computations for that parameter set. After completion of computations of the current parameter set, it moves to the next parameter set.

After a sequence of operations as programmed in the parameter set(s) for the specified number of loops is complete, the accelerator provides a completion interrupt (ACC_DONE_INTR) to the processor. The accelerator can be reconfigured as desired. For reconfiguration, the following procedure must be followed. The accelerator must be disabled by writing 000b to the ACCENABLE register. Then, a reset must be asserted by writing 111b followed by 000b to the ACCRESET register. The new configurations can now be written in to the accelerator, and then the accelerator can be enabled again by writing 111b to ACCENABLE.

2.1.2 State Machine – Trigger Mechanisms (Incoming)

As mentioned in the previous subsection, for each parameter set, the start of the computations can be triggered based on specific events. Four trigger mechanisms are supported as follows.

- Immediate trigger (TRIGMODE = 000b): In this case, the state machine does not wait for any trigger and starts the accelerator computations immediately for the current parameter set. This mode is applicable when chaining (sequencing) a set of operations one after another in the accelerator without any need for control handshake or data exchange outside the accelerator (for example, when chaining FFT and log-magnitude operations) with no need to wait for a trigger in between.
- Wait for processor-based software trigger (TRIGMODE = 001b): This is a software-triggered mode that is useful when the main processor must directly control the data flow and start or stop of accelerator computations. In this trigger mode, the state machine waits for a software-based trigger, which involves the main processor setting a separate self-clearing bit in a CR42ACCTRIG register (single-bit register). The state machine keeps monitoring that register bit and waits as long as the value is zero. When the value becomes 1 (set), the state machine gets triggered to start the accelerator operations for the current parameter set.
- Wait for the ADC buffer ping-to-pong or pong-to-ping switch (TRIGMODE = 010b): This trigger mode is specific to the mmWave 14xx device, which has RF and analog front end integrated in the same chip with the main processor and the Radar Hardware Accelerator. Recall that in the mmWave 14xx device, the ADC ping and pong buffers are shared with the accelerator local memories (ACCEL_MEM0 and ACCEL_MEM1), such that the ADC data is directly available to the accelerator for processing during active chirping portion of the frame. This sharing mode is enabled by setting the FFT1DEN register bit before the start of the frame. In this trigger mode, the state machine of the accelerator starts the computations for the current parameter set as soon as the ADC buffer switches from ping-to-pong or pong-to-ping. As an example, during the active chirping portion of a frame, the mmWave 14xx digital front end and ADC buffer can be configured to switch from ping-to-pong or pong-to-ping buffer at the end of every chirp or at the end of every few chirps or at the end of every specified number of ADC samples. This mmWave 14xx digital front-end configuration is accomplished using other registers unrelated to the Radar Hardware Accelerator and not described in this document.

Copyright © 2017, Texas Instruments Incorporated

Now, using this trigger mode (TRIGMODE = 010b) allows the accelerator computations to start whenever the ping-to-pong or pong-to-ping switch happens in the ADC buffer, thus enabling inline per-chirp processing. It is important to mention here that the user must take care to ensure that processing of the current ping data is completed by the accelerator, before the next switch/trigger happens on the ADC buffer. In other words, the chirp duration (ping-pong switch frequency) must be configured to be so fast that the accelerator cannot complete its configured operations within that duration

- Wait for the DMA-based trigger (TRIGMODE = 011b): This trigger mode is useful when a DMA transfer completion must be used to trigger the start of the accelerator computations for the current parameter set. The primary purpose of this trigger mode is as follows; when performing second dimension FFT, the DMA is used to bring the FFT input samples from the Radar data memory to the local memory of the accelerator. Upon completion of each DMA transfer, it is useful to automatically trigger the accelerator to perform the FFT.

  To achieve this, the state machine of the accelerator has a 16-bit register called the DMA2ACCTRIG register, where each register bit maps to one of 16 DMA channels that are associated with the accelerator. To use the DMA-based trigger mode, the DMA2ACC_CHANNEL_TRIGSRC register in the current parameter set must be programmed to the DMA channel whose completion we wish to monitor. The state machine then monitors the corresponding register bit in the DMA2ACCTRIG register, and triggers the execution of the current parameter set only when that register bit gets set. For e.g. if DMA2ACC_CHANNEL_TRIGSRC is programmed to 5, then the current parameter set will execute only once the register bit #5 gets set in DMA2ACCTRIG.

  The user may utilize the EDMA's linking capability to set the appropriate register bit in DMA2ACCTRIG. Linking is a programmable feature of the EDMA, where the completion of a DMA transfer can automatically trigger a second DMA transfer. In the present context, the DMA transfer that moves data to the local memory of the accelerator can be linked to a second DMA whose purpose is to write a one-hot signature into DMA2ACCTRIG to set a specific register bit and trigger the accelerator. Note that there are 16 read-only, one-hot, signature registers (SIG_DMACH1_DONE, SIG_DMACH2_DONE, and more) that are available. These registers are simply read-only registers which contain hard-coded values (each register is a one-hot signature – 0x0001, 0x0002, 0x0004, 0x0008, and so on). For convenience, these hard-coded 16 read-only signatures can be used, so that the second DMA can simply copy from one of these SIG_DMACHx_DONE registers into the DMA2ACCTRIG register to set the appropriate register bit.

2.1.3 State Machine – Trigger Mechanisms (Outgoing)

After the accelerator computations for the current parameter set are triggered (using one of the four incoming trigger mechanisms mentioned in the previous subsection), it performs the actual computation operations for that parameter set. These computations typically take several tens or hundreds of clock cycles, depending on the nature of the configuration programmed. Once the accelerator completes its computation operations for the current parameter set, the state machine advances to the next parameter set and repeats the same process. But before advancing to the next parameter set, it can interrupt the main processor and/or trigger a DMA channel. This provision is useful if the main processor is required to read or write registers or memory locations at the end of the current parameter set. Also, this provision is useful for triggering a DMA channel, so that the output of the accelerator can be copied out of the accelerator local memories.

There are two trigger mechanisms provided as follows:

- Interrupt to main processor (CR4INTREN = 1): The accelerator interrupts the main processor at the end of completion of computations for the current parameter set, if the register bit CR4INTREN is set.
- Trigger to DMA (DMATRIGEN = 1): The accelerator gives a trigger to a DMA channel at the end of completion of computations for the current parameter set, if the register bit DMATRIGEN is set. If DMATRIGEN is set, then the particular DMA channel as specified in a separate ACC2DMA_CHANNEL_TRIGDST register (valid values are 0 to 15, for the 16 DMA channels dedicated for the accelerator) is triggered. Thus, it is possible to preconfigure up to 16 DMA channels and trigger the appropriate one at the end of the computations of the current parameter set. The trigger from accelerator to the DMA channels can also be faked by the processor, by writing to a CR42DMATRIG register.

Copyright © 2017, Texas Instruments Incorporated

This can be used by the processor to kick-start a full/repetitive chain of operations, that are then subsequently managed between the DMA and the accelerator without further processor involvement – for example, the processor writes to the CR42DMATRIG register to trigger a DMA channel for the first time, and this kicks off a series of back-to-back data transfers and accelerator computations, with the DMA and accelerator hand-shaking with each other.

2.1.4 State Machine – Register Descriptions

Table 1 lists all the registers of the state machine block. As explained previously, some of the registers are common (common for all parameter sets) registers, whereas some others are *part of each parameter set*. For each register, this distinction is captured as part of the register description in Table 1.

Table 1. State Machine Registers

Register	Width	Parameter Set	Description
ACCENABLE	3	No	Enable and Disable Control: This register enables or disables the entire Radar Hardware Accelerator. The reason for a 3-bit register (instead of 1-bit) is to avoid an accidental bit-flip (for example, transient error caused by a neutron strike) from unintentionally turning on the accelerator engine. A value of ACCENABLE = 111b enables the Radar Hardware Accelerator and any other value of the register keeps the accelerator engine in disabled state.
ACCCLKEN	1	No	Clock-gating Control: This register bit controls the enable/disable for the clock of the Radar Accelerator. This register bit can be set to 0 to clock-gate the accelerator when not using the accelerator. Before enabling the accelerator or before configuring the accelerator's registers, this register bit should be set first, so that the clock is available.
ACCRESET	3	No	Software Reset Control: This register provides software reset control for the Radar Hardware Accelerator. The assertion of these register bits by the main processor will bring the accelerator engine to a known reset state. This is mostly applicable for resetting the accelerator in case of unexpected behavior. Under normal circumstances, it is expected that whenever the accelerator is enabled (from disabled state), it always comes up in a known reset state automatically. The recommended sequence to be followed in case software reset is desired is to write 111b to this register and then a 000b, before the clock is enabled to the accelerator.
NLOOPS	12	No	Number of loops: This register controls the number of times the state machine will loop through the parameter sets (from a programmed start index till a programmed end index) and run them. The maximum number of times the loop can be made is run is 4094. A value of 4095 (0xFFF) programmed in this register should be considered as a special case and it should be interpreted as an infinite loop mode, for example, keep looping and never stop the accelerator engine unless reset by the main processor. A value of zero programmed in this register means that the looping mechanism is disabled. In this case, the accelerator engine can still be used under direct control of the main processor (without the state machine looping provision coming into the picture).
PARAMSTART	4	No	Parameter-set Start and Stop Index: These registers are used to control the start and stop index of the parameter set through which the state machine loops through. The state machine starts at the parameter set specified by PARAMSTART and loads each parameter set one after another and runs the accelerator as per that configuration. When the state machine reaches the parameter set specified by PARAMSTOP, it loops back to the start index as specified by PARAMSTART.
PARAMSTOP	4	No	
FFT1DEN	1	No	ADC buffer sharing mode (mmWave 14xx): This register is relevant in mmWave 14xx, where the Radar Hardware Accelerator is included in a single device along with the mmWave RF front-end. In such a case, during active chirp transmission and inline first dimension FFT processing, the ACCEL_MEM0 and ACCEL_MEM1 memories of the accelerator are shared as ping-pong ADC buffers. This register bit needs to be set during this time, so that while the digital front end writes ADC samples to the ping buffer, the accelerator automatically accesses (only) the pong buffer, and vice versa. At the end of the active transmission portion of a frame, this bit can be cleared, so that the accelerator has access to all the four local memories independently.

Copyright © 2017, Texas Instruments Incorporated

Table 1. State Machine Registers (continued)

Register	Width	Parameter Set	Description
TRIGMODE	3	Yes	Trigger mode control: This parameter-set register is used to control how the state machine and the operations of the accelerator are triggered for each parameter set. The following modes are supported: • 000b – Immediate trigger • 001b – Software trigger • 010b – Ping-pong switch based trigger (applicable only when FFT1DEN is set) • 011b – DMA-based trigger The trigger modes are described in Section 2.1.2.
CR42ACCTRIG	1	No	Software trigger bit: This register bit is relevant whenever software triggered mode is used (for example, TRIGMODE = 001b). Whenever software triggered mode is configured for a parameter set, the state machine keeps monitoring this register bit and waits as long as the value is zero. The main processor software can set this register bit, so that the state machine gets triggered and starts the accelerator operations for that parameter set.
DMA2ACCTRIG	16	No	DMA trigger register: This register is relevant whenever DMA triggered mode is used (for example, TRIGMODE = 011b). Whenever a DMA channel has finished copying input samples into the local memory of the accelerator and wants to trigger the accelerator, the procedure to follow is to use a second linked DMA channel to write a 16-bit one-hot signature into this register to trigger the accelerator. In DMA triggered mode, the state machine keeps monitoring this 16-bit register and waits as long as a specific bit (see DMA2ACC_CHANNEL_TRIGSRC) in this register is zero. The second linked DMA channel writes a one-hot signature that sets the specific bit, so that the state machine gets triggered and starts the accelerator operations for that parameter set.
DMA2ACC_CHANNEL_ TRIGSRC	4	Yes	DMA channel select for DMA completion trigger: This parameter-set register is relevant whenever DMA triggered mode is used (for example, TRIGMODE = 011b). This register selects the bit number in DMA2ACCTRIG for the state machine to monitor to trigger the operation for that parameter set.
CR4INTREN	1	Yes	Completion interrupt to main processor: This parameter-set register is used to enable/disable interrupt to the main processor upon completion of the accelerator operation for that parameter set. If enabled, the main processor receives an interrupt from the Radar Hardware Accelerator at the end of operations for that parameter set, so that the main processor can take any necessary action.
PARAMDONESTAT (read-only)	16	No	Parameter-set done status: This read-only status register can be used by the main processor to see which parameter sets are complete that led to the interrupt to the main processor. The individual bits in this 16-bit status register indicate which of the 16 parameter sets have completed. These status bits are not automatically cleared, but they can be individually cleared by writing to another 16-bit register PARAMDONECLR.
PARAMDONECLR	16	No	
DMATRIGEN	1	Yes	Completion trigger to DMA: This parameter-set register is used to enable DMA channel trigger upon completion of the accelerator operation for that parameter set. This trigger mechanism enables the accelerator to hand-shake with the DMA so that output data samples are copied out of the accelerator local memory. If enabled, the accelerator triggers a specified DMA channel, so that the output samples can be shipped from the local memory to Radar data memory.
ACC2DMA_CHANNEL_ TRIGDST	4	Yes	DMA channel select for accelerator completion trigger: This parameter-set register is used to select which of the 16 DMA channels allocated to the accelerator should be triggered upon completion of the accelerator operation for that parameter set. This register is to be used in conjunction with DMATRIGEN.
CR42DMATRIG	16	No	Trigger from processor to DMA: This register can be used by the processor to trigger a DMA channel for the first time, so that a full sequence of repeated operations between the DMA and the accelerator gets kick-started.

Copyright © 2017, Texas Instruments Incorporated

Table 1. State Machine Registers (continued)

Register	Width	Parameter Set	Description
PARAMADDR	4	No	Debug register for current parameter-set index: This read-only status register indicates the index of the current parameter set that is under execution. This is useful for debug, where parameter sets can be executed in single-step manner (one-by-one) using SW trigger mode for each of them. In such a debug, this register indicates which parameter set is currently waiting for the SW trigger.
LOOPCNT	12	No	Debug register for current loop count: This read-only status register indicates what is the loop count that is presently running. When the state machine is programmed for NLOOPS loops, this register shows the current loop count that is running.
ACC_TRIGGER_IN_STAT	19	No	Debug register for trigger status: This is a read-only status register, which indicates the trigger status of the accelerator, for example, whether a specific DMA trigger or a Ping-pong trigger or a SW trigger was ever received (refer TRIGMODE). The MSB 16 bits of this register indicate whether a trigger was received via DMA trigger method. The next two bits (for example, bit indices 2 and 1) indicate the status of DFE ping-pong switch-based trigger and SW trigger respectively. The LSB bit is always 1 and can be ignored.
ACC_TRIGGER_IN_CLR	1	No	Clear trigger status read-only register: This register-bit when set clears the trigger status register ACC_TRIGGER_IN_STAT described above.

The next two sections cover the Input Formatter and Output Formatter blocks, including their detailed operation, registers and usage procedure.

Copyright © 2017, Texas Instruments Incorporated

3 Accelerator Engine – Input Formatter

This section describes the input formatter block present in the accelerator engine (see Figure 5).

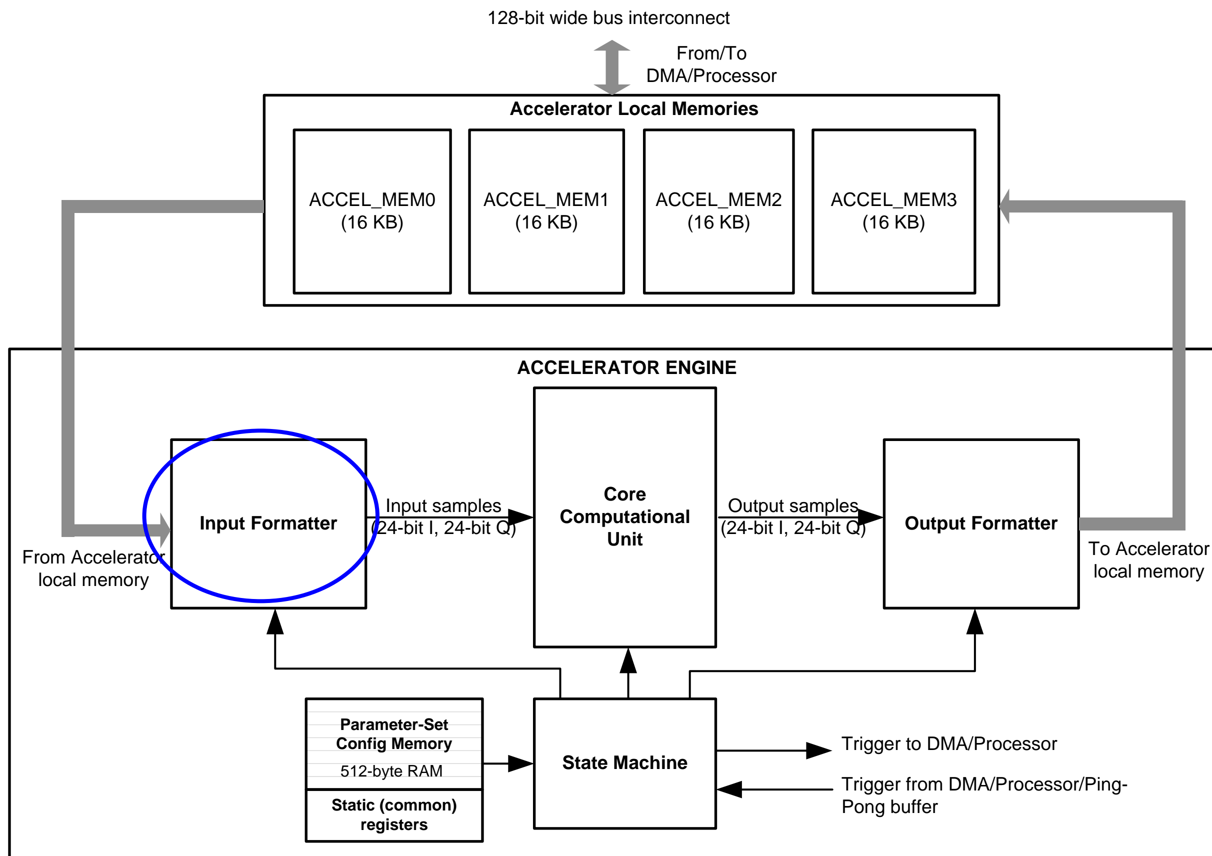


Figure 5. Input Formatter

3.1 *Input Formatter*

The input formatter is used to access, format, and feed the data from the local memories of the accelerator as 24-bit I and 24-bit Q samples into the core computational unit. The input formatter provides various capabilities to access and format the samples from the local memories – especially, various multidimensional access patterns (for example transpose access), 16-bit or 32-bit aligned word access, scaling using bit-shifts to generate 24-bit wide samples from 16-bit or 32-bit words, real versus complex input, sign extension, conjugation, and more.

3.1.1 Input Formatter – Operation

The input formatter block is responsible for reading the input samples from the accelerator local memory and feeding them into the core computational unit (see Figure 2). The data flow from the input formatter, through the core computational unit, to the output formatter is designed to sustain a steady-state throughput of one complex sample per clock cycle. The input formatter thus feeds one sample (24-bit I and 24-bit Q) into the core computational unit every clock cycle.

To make the best use of the capabilities of the core computational unit and to allow meaningful chaining of radar signal processing operations with minimal intervention from the R4F processor, the input formatter supports flexibility in how the input samples are accessed from the memory and how they are formatted and fed into the core computational unit.

Copyright © 2017, Texas Instruments Incorporated

The memory from which the input formatter picks up the data is referred to as *source memory*. Note that any of the four accelerator local memories can be the source memory. However, as will be described in a subsequent section, there is an important restriction which explains that the source memory cannot be the same as the destination memory (which is the memory to which the output formatter writes the output data).

3.1.2 Input Formatter – 2D Indexed Addressing for Source Memory Access

The 16-bit parameter-set register SRCADDR specifies the start address at which the input samples must be accessed. This register is a byte-address, and a value of 0x0000 corresponds to the first memory location of ACCEL_MEM0 memory. The 16-bit SRCADDR register maps to the entire 64KB address space of the four accelerator local memories (4x16KB).

The input data can be read from the memory as either 16-bit wide samples or 32-bit wide samples. Also, they can be read as real samples or complex samples. These two aspects are configured using register bits SRC16b32b and SRCREAL. See Table 2 for a description of these and other registers pertaining to the input formatter block. As an example, if SRC16b32b = 0 and SRCREAL = 0, then the input samples are read from the memory as 16-bit complex samples (16-bit I and 16-bit Q), shown in Figure 6. In the mmWave 14xx device, the ADC buffer is always filled with complex samples from the digital front end – this is true even if the device is configured for real-only operation, in which case the Q-channel output is written with zero values. Therefore, for all purposes of part one of the user guide, SRCREAL can be configured as 0.

An important feature of the input formatter block is that it supports flexible access pattern to fetch data from the source memory, which makes it convenient when the data corresponding to multiple RX channels are interleaved or when performing multi-dimensional (FFT) processing. This feature is facilitated through the SRCAINDX, SRCACNT, SRCBINDX, and REG_BCNT registers, which are part of each parameter-set configuration.

The register SRCAINDX specifies how many bytes separate successive samples to be fetched from the source memory and the register SRCACNT specifies how many samples need to be fetched per iteration. An iteration is typically one computational routine, such as one FFT operation. It is possible to perform multiple iterations back-to-back – for example, four FFT operations corresponding to four RX channels. The register SRCBINDX specifies how many bytes separate the start of input samples for successive iterations and REG_BCNT specifies how many iterations to perform back-to-back. These registers can be better understood using the example given in Figure 6. Also, a complete use case is illustrated in Section 6, which provides further clarity on this aspect.

Copyright © 2017, Texas Instruments Incorporated

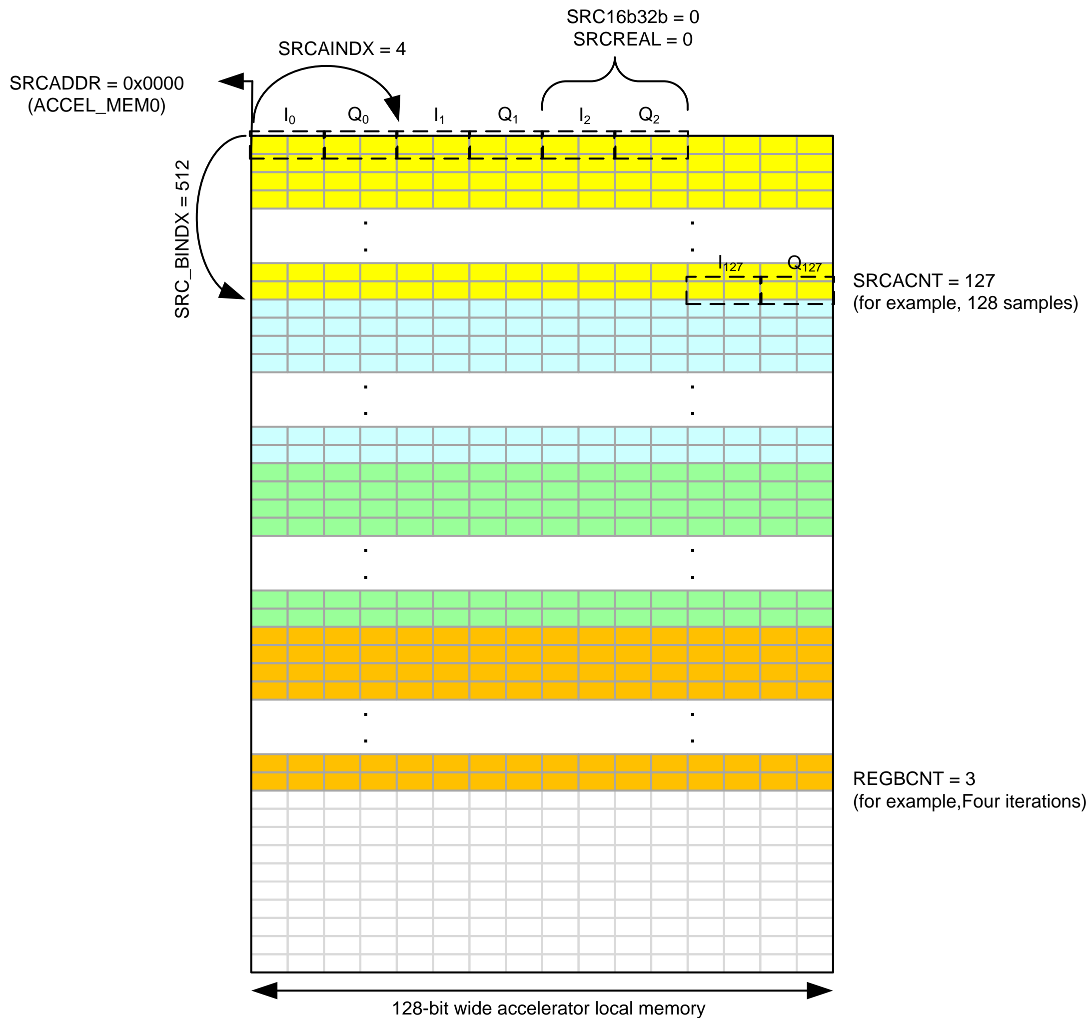


Figure 6. Input Formatter Source Memory Access Pattern (Example)

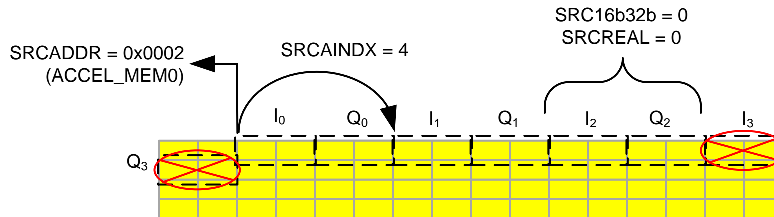
In Section 6, the input data consists of complex data (16-bit I and 16-bit Q) that is contiguously present in ACCEL_MEM0. The data in memory consists of four sets of 128 samples each (say, corresponding to four RX antennas) and these are shown in four different colors. Because each sample occupies 4 bytes and the samples are contiguously placed in the memory starting at the beginning of ACCEL_MEM0, values of SRCADDR = 0x0000 and SRCAINDX = 4 are used to fetch these samples.

In each clock cycle, the input formatter fetches one complex sample from the memory and feeds it into the core computational unit (with appropriate scaling, as described later). Because there are 128 samples to be fed for the first iteration (computational routine), a value of SRCACNT = 127 is used. For the second iteration, the samples are fetched starting from a memory location that is SRCBINDX (=128 × 4 = 512) bytes away from SRCADDR.

This process repeats for the programmed number of iterations as per the REG_BCNT register. For example, the value of REG_BCNT = 3 used in this example corresponds to four iterations. Note that the registers shown here are part of parameter-set configuration registers and the four iterations described here can be performed using a single parameter set.

Copyright © 2017, Texas Instruments Incorporated

An important restriction in programming the registers related to source memory access pattern is that the input formatter can only read data from one memory row (128-bit memory location) in a clock cycle. Therefore, if a sample is placed in memory such that the real-part (I value) is at the end of one memory location and the imaginary part (Q value) is at the beginning of the next memory location, then that would be an invalid configuration (see Figure 7).



(1) Single sample getting divided into two rows

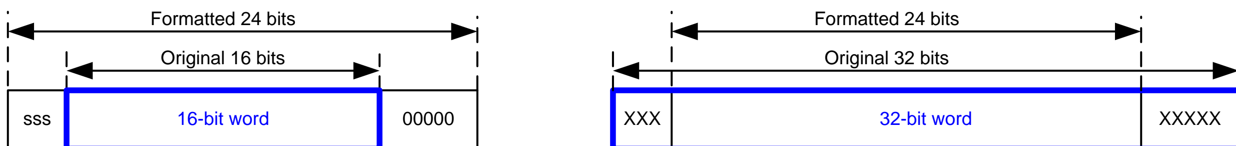
Figure 7. Invalid Configuration Example

3.1.3 Input Formatter – Scaling and Formatting

The input formatter allows the input samples read from the source memory to be scaled and formatted before feeding them as 24-bit complex samples into the core computational unit.

Even though the data read from the source memory is initially 16-bits or 32-bits wide (for each I and Q), the samples expected by the core computational unit are 24-bit complex samples (24-bits each for I and Q). There is a REG_SRCSCAL register which provides scaling options using bit-shift to generate 24-bit samples from the original 16- or 32-bit data (see Figure 8).

For the 16-bit case, the 24-bit sample is generated by padding (8-REG_SRCSCAL) zeros at the LSB and REG_SRCSCAL redundant MSBs. For the 32-bit case, the 24-bit sample is generated by dropping REG_SRCSCAL bits at the LSB and clipping (8-REG_SRCSCAL) bits at the MSB. Note that the register bit SRCSIGNED is used to indicate whether the input samples are signed or unsigned. When this register bit is set, the input samples are treated as signed numbers and hence any extra MSB bits are sign-extended and any clipping of MSB bits takes care of signed saturation. In most cases of interest in part one of this user guide (for example, when performing FFT operation), the input samples would be signed and hence SRCSIGNED should be set (for example, equal to 1).



For 16-bit case, if REG_SRCSCAL = 3, then 5 zeros are padded at the LSB, and 3 redundant (extension) bits are padded at the MSB

For 32-bit case, if REG_SRCSCAL = 3, then 5 bits are dropped at the LSB, and 3 bits are clipped (with saturation) at the MSB

(1) 16- or 32-bit words to 24-bit samples

Figure 8. Input Formatter Data Scaling

When the input samples are complex (for example, SRCREAL = 0), there is a provision to conjugate the input samples. Setting the register bit SRCCONJ conjugates the input samples before feeding them to the core computational unit. This feature (together with a corresponding DSTCONJ register bit in the output formatter block) enables an IFFT mode from the FFT engine. Note that conjugating the input and output of an FFT block is equivalent to an IFFT function.

There are other registers in the input formatter, such as BPM_EN, BPMPATTERNLSB and BPMPATTERNMSB, BPMRATE, CIRCIRSHIFT, CIRCSHIFTWRAP, and so on, which are beyond the scope of part one of this user's guide and these registers are described in part two. For the immediate purpose of the first part of the user's guide, it is important to note that BPM_EN and CIRCIRSHIFT registers must be kept 0.

Copyright © 2017, Texas Instruments Incorporated

3.1.4 Input Formatter – Zero Padding

The input formatter has provision for *zero padding*, which is important when performing FFT of a set of samples whose length is not a power of 2. The input formatter automatically feeds the required number of zeros into the core computational unit, whenever the FFT size (as programmed using the FFTSIZE register, which is described in a later section) does not match the SRCACNT setting.

For example, if the number of input samples read by the input formatter is 56 (for example, SRCACNT = 55) and the FFT size is programmed to be 64 (for example, FFTSIZE = 6), then the input formatter feeds 8 zeros at the end of each iteration, before starting to read the input samples for the next iteration from the source memory. This zero-padding provision enables the core computational unit to perform 64-point FFT with the correct set of zero-padded input samples. It is important for the user to note that SRCACNT should never be larger than 2^FFTSIZE-1.

The zero padding is effective only when performing FFT operation in the core computational unit (i.e., when FFT_EN = 1) and not otherwise. Please refer to section 6 for further information regarding the registers relevant for FFT operation.

3.1.5 Input Formatter – Register Descriptions

Table 2 lists all the registers of the input formatter block.

Table 2. Input Formatter Registers

Register	Width	Parameter Set	Description
SRCADDR	16	Yes	Source start address: This register specifies the starting address of the input samples, for example, it specifies the source memory start address from which input samples have to be fetched by the input formatter. This is a byte-address and this 16-bit register covers the entire address space of the four local memories (4 × 16KB = 64 KB). The four accelerator local memories are contiguous in the memory address space and any of them can act as the source memory (as long as the same memory bank is not configured to be used as destination memory at the same time).
SRCACNT	12	Yes	Source sample count: This register specifies the number of samples (minus 1) from the source memory to process for every iteration. The sample count is in number of samples, not number of bytes. For example, the sample count can be specified as 255 (SRCACNT = 0x0FF) in a case where a 256-point FFT is required to be performed. Note however that the sample count register does not always match the FFT size. This can happen when zero-padding of input samples is required. For example, a sample count of 192 could be used with an FFT size of 256, in which case, the input formatter will automatically append 64 zeros.
SRCAINDX	16	Yes	Source sample index increment: This register specifies the number of bytes separating successive samples in the source memory. For example, a value of SRCAINDX = 16 means that successive samples are separated by 16 bytes in memory. The maximum value allowed for this register is 32767.
REG_BCNT	12	Yes	Number of iterations: This register specifies the number of times (minus 1) the processing should be repeated. This register can be used to process the four RX chains back-to-back – for example, a value of REG_BCNT = 3 means that the processing (say first dimension FFT processing) is repeated four times. Note the distinction between the NLOOPS register of the state machine block and the REG_BCNT register of the input formatter block. The NLOOPS register specifies how many times the state machine loops through all the configured parameter sets (with each time possibly awaiting a trigger), whereas the register REG_BCNT specifies how many times the input formatter and the computational processing of the accelerator is iterated back-to-back for the current parameter set (without any intermediate triggers).
SRCBINDX	16	Yes	Source offset per iteration: This register specifies the number of bytes separating the starting address of input samples for successive iterations. For example, when using four iterations to process the four RX chains, this register can be used to specify the offset in the starting address between the successive RX chains. Note the distinction that SRCAINDX specifies the number of bytes separating successive samples for a particular iteration, whereas SRCBINDX specifies the number of bytes separating the starting address of the first sample for successive iterations. The maximum value allowed for this register is 32767.

Copyright © 2017, Texas Instruments Incorporated

Table 2. Input Formatter Registers (continued)

Register	Width	Parameter Set	Description
SRCREAL	1	Yes	Complex or real Input: This register-bit specifies whether the input samples are real or complex. A value of SRCREAL = 0 implies complex input and a value of SRCREAL = 1 implies real input. When real input is selected, the input formatter block automatically feeds zero for the imaginary part into the core computational unit.
SRC16b32b	1	Yes	16-bit or 32-bit input word alignment: This register-bit specifies whether the input samples fetched from source memory are to be read as 16-bits or 32-bits wide. A value of SRC16b32b = 0 implies that the input samples are 16-bits wide each (in case of complex input, real and imaginary parts are each 16 bits wide). A value of SRC16b32b = 1 implies that the input samples are 32-bits wide each.
SRCSIGNED	1	Yes	Input sign-extension mode: This register-bit, when set, specifies that the input samples are signed numbers and hence, sign-extension or signed-saturation at the MSB is required when converting 16-bit or 32-bit input words to the 24-bit wide samples to be fed into the core computational unit.
SRCCONJ	1	Yes	Input conjugation: This register-bit specifies whether the input samples should be conjugated before feeding them into the core computational unit. If SRCCONJ is set, then the input samples are conjugated. Setting this register-bit only makes sense if the samples are complex numbers (for example, SRCREAL = 0). This register, together with its counterpart in the output formatter block, enable an IFFT mode for the FFT engine. Note that conjugating the input and output of an FFT block is equivalent to an IFFT function.
REG_SRCSCAL	8	Yes	Input scaling: This register specifies a programmable scaling using bit-shift, when converting the 16-bit or 32-bit wide input data to 24-bit wide samples before feeding into the core computational unit. See Figure 8 and its description for more details regarding this register.
CIRCIRSHIFT	–	–	Described in part two of this user's guide. For the immediate purposes relevant to part one of this user's guide, all of these registers must be kept as 0.
CIRCSHIFTWRAP	–	–	
BPMPATTERNLSB and BPMPATTERNMSB	–	–	
BPMRATE	–	–	
BPMPHASE	–	–	

Copyright © 2017, Texas Instruments Incorporated

4 Accelerator Engine – Output Formatter

This section describes the output formatter block present in the accelerator engine (see Figure 9).

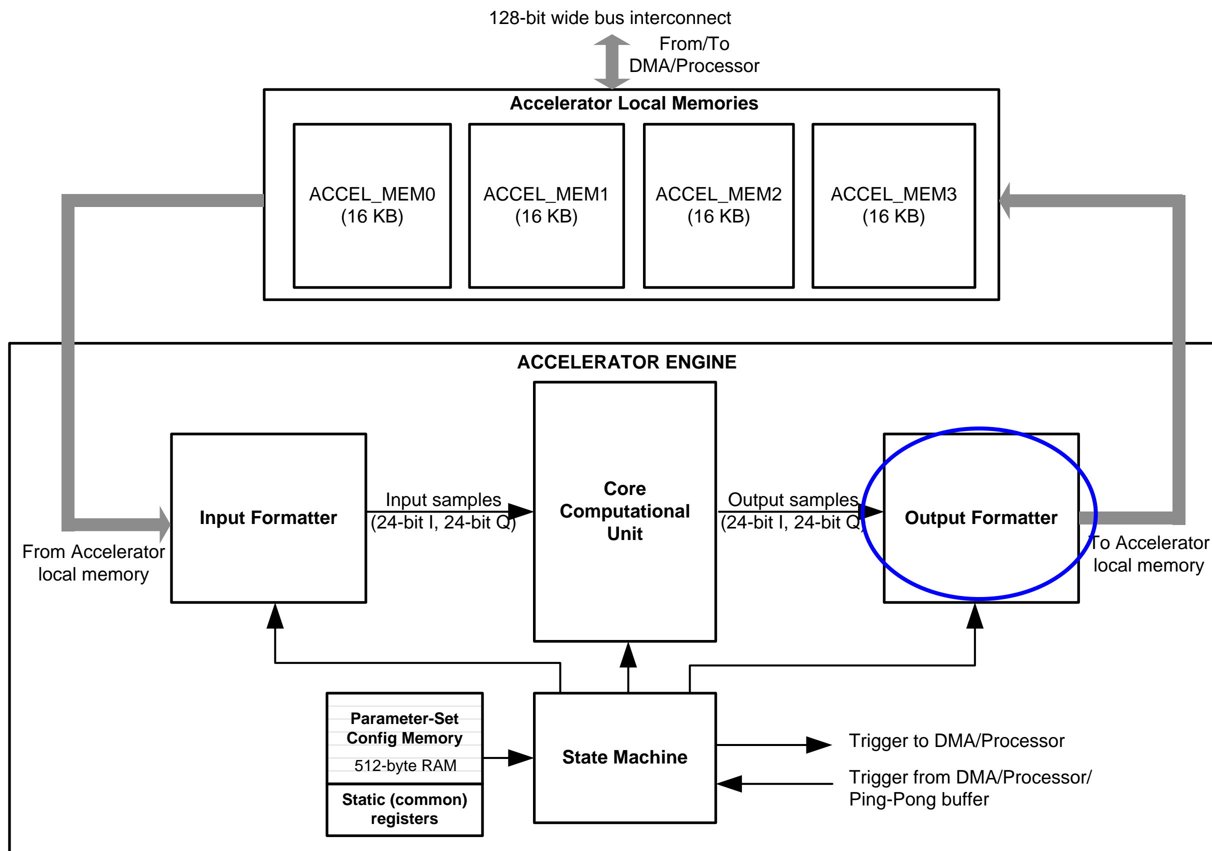


Figure 9. Output Formatter

4.1 *Output Formatter*

The output formatter is used to format and write the data coming out of the core computational unit into the accelerator local memory. Similar to the input formatter block discussed in the previous section, the output formatter block also provides various capabilities to format and write the samples written to the local memory – especially, various multidimensional access patterns (for example, transpose writes), 16-bit or 32-bit aligned word writes, scaling using bit-shifts to generate 16-bit or 32-bit words from 24-bit wide samples, real versus complex output write, and more.

4.1.1 Output Formatter – Operation

The output formatter block is responsible for storing the samples coming out of the core computation unit into the accelerator local memory (see Figure 2). As mentioned in the previous section, the data flow from the input formatter, through the core computational unit, to the output formatter, is designed to sustain a steady-state throughput of one complex sample per clock cycle. Thus, typically, the output formatter accepts one sample (24-bit I and 24-bit Q) from the core computational unit every clock cycle and writes it to the accelerator local memory. Just like the input formatter, the output formatter also supports lot of flexibility in how the samples are formatted and written into the memory.

The memory into which the output formatter writes the data is referred to as *destination memory*. Note that any of the four accelerator local memories can be the destination memory, with the important restriction that the source memory cannot be same as the destination memory. In other words, each of the four 16KB memory banks can either function as source memory, or as destination memory at any time (for example, in any given parameter set).

Copyright © 2017, Texas Instruments Incorporated

4.1.2 Output Formatter – 2-D Indexed Addressing for Destination Memory Access

The 16-bit parameter-set register DSTADDR specifies the start address at which the output samples must be written into the accelerator local memory. Similar to the SRCADDR register of the input formatter, the DSTADDR register of the output formatter is a byte-address and a value of 0x0000 corresponds to the first memory location of ACCEL_MEM0 memory. The 16-bit DSTADDR register maps to the entire 64KB address space of the four accelerator local memories (4 × 16KB). As mentioned in the previous paragraph, in a given parameter set, SRCADDR and DSTADDR cannot be configured such that the input samples being fetched and the output samples being written out are accessing the same memory bank.

Even though the core computational unit produces a 24-bit complex output stream, this output data can be written to the memory as either 16-bit wide samples or 32-bit wide samples. Also, they can be written out as complex samples or real samples (for example, drop imaginary part – applicable when performing log-magnitude computation). These two aspects are configured using register bits DST16b32b and DSTREAL. See Table 3 for a description of these and other registers pertaining to the output formatter block. As an example, if DST16b32b = 0 and DSTREAL = 0, then the output samples are written to the memory as 16-bit complex samples (16-bit I and 16-bit Q), shown in Figure 10.

Similar to the input formatter block, the output formatter block also supports flexible patterns to write multidimensional data to the destination memory and this makes it convenient when the data corresponding to multiple RX channels must be interleaved, or when performing multidimensional (FFT) processing. This feature is facilitated through the DSTAINDX, DSTACNT, DSTBINDX, and REG_BCNT registers, which are part of each parameter-set configuration.

The register DSTAINDX specifies how many bytes separate successive samples to be written to the destination memory and the register DSTACNT specifies how many samples must be written per iteration. Note that DSTACNT can be different from SRCACNT – this is useful when only a subset of the output samples need to be stored in the output memory (for example, if some FFT output bins must be discarded). The register DSTBINDX specifies how many bytes separate the start of output samples for successive iterations and REG_BCNT specifies the number of iterations. The REG_BCNT register is common for input formatter and output formatter. These registers can be better understood using the example given in Figure 10. Also, a complete use case is illustrated in Section 6 which provides further clarity on this aspect.

Copyright © 2017, Texas Instruments Incorporated

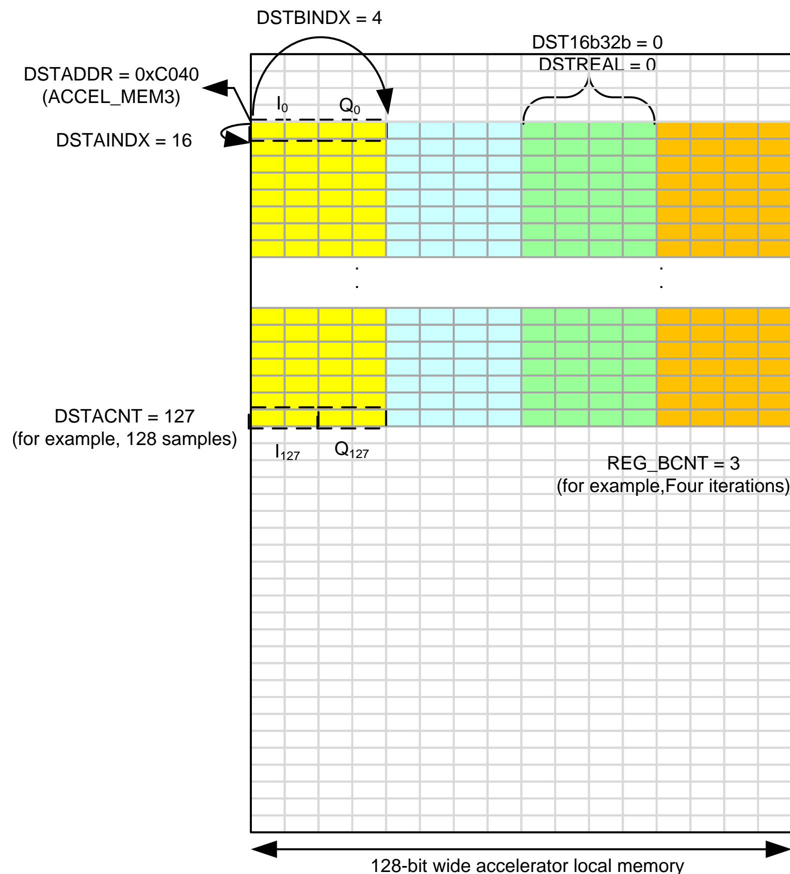


Figure 10. Output Formatter Destination Memory Access Pattern (Example)

In the example shown in Figure 10, the output data consists of complex data (16-bit I and 16-bit Q) that is written to ACCEL_MEM3. The output data consists of four sets of 128 samples each (say, corresponding to FFT output of four RX antennas) and these are shown in four different colors. Each sample occupies 4 bytes and the samples are written to the output memory at a specific start address inside ACCEL_MEM3, as shown in Figure 10. The samples for the four RX antennas are written to the memory in an interleaved manner. Thus, for this example, a value of DSTADDR = 0xC040, DSTAINDX = 16, DSTACNT = 127, and DSTBINDX = 4 are used. The register REG_BCNT (common for input formatter and output formatter) is configured with a value of 3, corresponding to the four iterations required (for the four RX antennas). In steady state, for each clock cycle, the output formatter accepts one complex sample from the core computational unit and writes it into the memory as per the 2-D indexed addressing pattern programmed.

The register DSTACNT, which corresponds to the number of samples written to the destination memory for each iteration does not need to be equal to SRCACNT. This is useful in cases where some of the output samples (for example, some FFT bins at the end) can be dropped and do not need to be written into the destination memory. Another register, REG_DST_SKIP_INIT is also available, which can be used to skip some samples in the beginning as well. The number of samples written to the destination memory for each iteration is equal to (DSTACNT + 1) – REG_DST_SKIP_INIT.

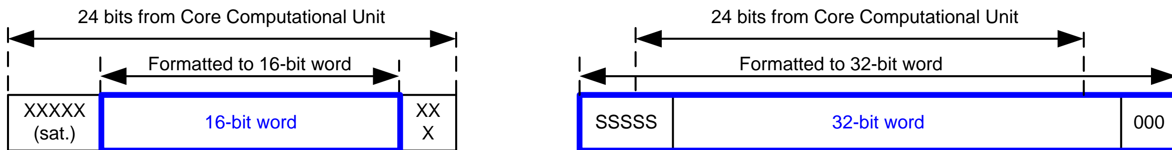
Note that when performing FFT operations, internally the core computational unit sends out FFT output data in bit-reversed addressing order, but this is automatically handled in the output formatter, such that when the FFT output samples are written into the destination memory, they are written out in the correct normal order. Therefore, no special procedure is required on the part of the main processor to read the FFT output samples in the right sequence.

Copyright © 2017, Texas Instruments Incorporated

4.1.3 Output Formatter – Scaling and Formatting

The output formatter allows the 24-bit output samples from the core computational unit to be scaled and formatted before writing them to the destination memory as 16-bit or 32-bit words. There is a REG_DSTSCAL register which provides scaling options using bit-shift, to take the 24-bit samples and convert them to 16-bit or 32-bit data.

For the 16-bit case, the 24-bit sample (24-bits for each I and Q) is converted to 16-bit word by dropping REG_DSTSCAL bits at the LSB and by clipping with saturation (8-REG_DSTSCAL) bits at the MSB. For the 32-bit case, the 24-bit sample is padded with REG_DSTSCAL extra bits at the MSB and with (8-REG_DSTSCAL) extra zeros at the LSB. Note that the register bit DSTSIGNED is used to indicate whether the output samples are signed or unsigned. When this register bit is set, the output samples are treated as signed numbers and therefore any extra MSB bits are sign-extended and any clipping of MSB bits handles signed saturation. In most cases of interest in part one of this user's guide (for example, when performing FFT operation), the output samples would be signed and therefore DSTSIGNED should be set (for example, equal to 1). However, if the log-magnitude operation in the core computational unit is enabled, then the output samples are unsigned and therefore DSTSIGNED is cleared (for example, equal to zero).



For 16-bit case, if REG_DSTCAL = 3, then 3 btis are dropped at the LSB, and 5 bits are clipped (saturated) at the LSB

For 32-bit case, if REG_DSTCAL = 3, then 3 zeros are padded at the LSB, and 5 bits are extended at the MSB

(1) 24-bit samples to 16- or 32-bit words

Figure 11. Output Formatter Data Scaling

When the output samples are complex (for example, DSTREAL = 0), there is a provision to conjugate the output samples. Setting the register bit DSTCONJ conjugates the output samples before writing them to the destination memory. This feature (together with a corresponding SRCCONJ register bit in the input formatter block) enables an IFFT mode from the FFT engine.

4.1.4 Output Formatter – Register Descriptions

Table 3 lists all the registers of the output formatter block.

Table 3. Output Formatter Registers

Register	Width	Parameter Set	Description
DSTADDR	16	Yes	Destination start address: This register specifies the starting address of the output samples, for example, it specifies the destination memory start address at which the output samples have to be written by the output formatter. This is a byte-address and this 16-bit register covers the entire address space of the four local memories (4 × 16KB = 64 KB). The four accelerator local memories are contiguous in the memory address space and any of them can act as the destination memory (as long as the same memory bank is not configured to be used as source memory at the same time).
DSTACNT	12	Yes	Destination sample count: This register specifies the number of samples (minus 1) to be written to the destination memory for every iteration. The sample count is in number of samples, not number of bytes. For example, the sample count can be specified as 191 (DSTACNT = 0x0BF) in a case where 192 samples must be written. Note that the DSTACNT register can be different from SRCACNT or even the FFT size. This is useful when only a part of the FFT bins must be written to memory and the remaining (far-end FFT bins) can be discarded. This register description is true when the REG_DST_SKIP_INIT register value is zero (see further for more information related to REG_DST_SKIP_INIT).

Copyright © 2017, Texas Instruments Incorporated

Table 3. Output Formatter Registers (continued)

Register	Width	Parameter Set	Description
DSTAINDX	16	Yes	Destination sample index increment: This register specifies the number of bytes separating successive samples to be written to the destination memory. For example, a value of DSTAINDX = 16 means that successive samples written to the destination memory should be separated by 16 bytes. The maximum value allowed for this register is 32767.
DSTBINDX	16	Yes	Destination offset per iteration: This register specifies the number of bytes separating the starting address of output samples for successive iterations. For example, when using four iterations to process four RX chains, this register can be used to specify the offset in the starting address between the successive RX chains. Note the distinction that DSTAINDX specifies the number of bytes separating successive samples for a particular iteration, whereas SRCBINDX specifies the number of bytes separating the starting address of the first sample for successive iterations. The maximum value allowed for this register is 32767.
REG_DST_SKIP_ INIT	10	Yes	Destination skip sample count: This register specifies how many output samples should be skipped in the beginning, before starting to write to the destination memory. This is useful if only a certain part of the FFT output (skipping the first several bins) need to be stored in memory. The total number of samples written to destination memory is equal to DSTACNT+1-REG_DST_SKIP_INIT.
DSTREAL	1	Yes	Complex or real output: This register-bit specifies whether the output samples are real or complex. A value of DSTREAL = 0 implies complex output and a value of DSTREAL = 1 implies real output. When real output is selected, the output formatter block automatically stores only the real part into the destination memory. This is useful when the core computational unit is configured to output magnitude or log-magnitude values.
DST16b32b	1	Yes	16-bit or 32-bit output word alignment: This register-bit specifies whether the output samples are to be written as 16-bits or 32-bits wide in the destination memory. A value of DST16b32b = 0 implies that the output samples are to be written as 16-bit words (in case of complex output, real and imaginary parts are each 16 bits wide). A value of DST16b32b = 1 implies that the output samples are 32-bits wide each.
DSTSIGNED	1	Yes	Output sign-extension mode: This register-bit, when set, specifies that the output samples are signed numbers and therefore, sign-extension or signed-saturation at the MSB is required when converting the 24-bit wide samples coming from the core computational unit into 16-bit or 32-bit output words to be written to the destination memory.
DSTCONJ	1	Yes	Output conjugation: This register-bit specifies whether the output samples must be conjugated before writing them into the destination memory. If DSTCONJ is set, then the output samples are conjugated. Setting this register-bit only makes sense if the samples are complex numbers (for example, DSTREAL = 0). This register, together with its counterpart in the output formatter block, enables an IFFT mode for the FFT engine.
REG_DSTSCAL	8	Yes	Output scaling: This register specifies a programmable scaling using bit-shift, when converting the 24-bit samples coming from the core computational unit into 16-bit or 32-bit wide words to be written to the destination memory. See Figure 11and its description for more details regarding this register.
STATERRCODE	4	No	Memory access error: This 4-bit read-only register indicates if there is a memory access error caused by incorrect configuration or usage of the accelerator, where both the DMA and the accelerator are attempting to access the same 16KB memory at the same time. The 4-bit register indicates the error status for the 4 16KB memories (MSB bit corresponds to ACCEL_MEM0).
ERRCODEMASK	4	No	Mask for memory error: This register can be used to mask the memory access error. If set, the memory access error indication is disabled.
ERRCODECLR	4	No	Clear memory access error: This register can be used to clear the memory access error indication. Setting this register clears the error indication.

Copyright © 2017, Texas Instruments Incorporated

5 Accelerator Engine – Core Computational Unit

This section describes the core computational unit present in the accelerator engine (see Figure 12).

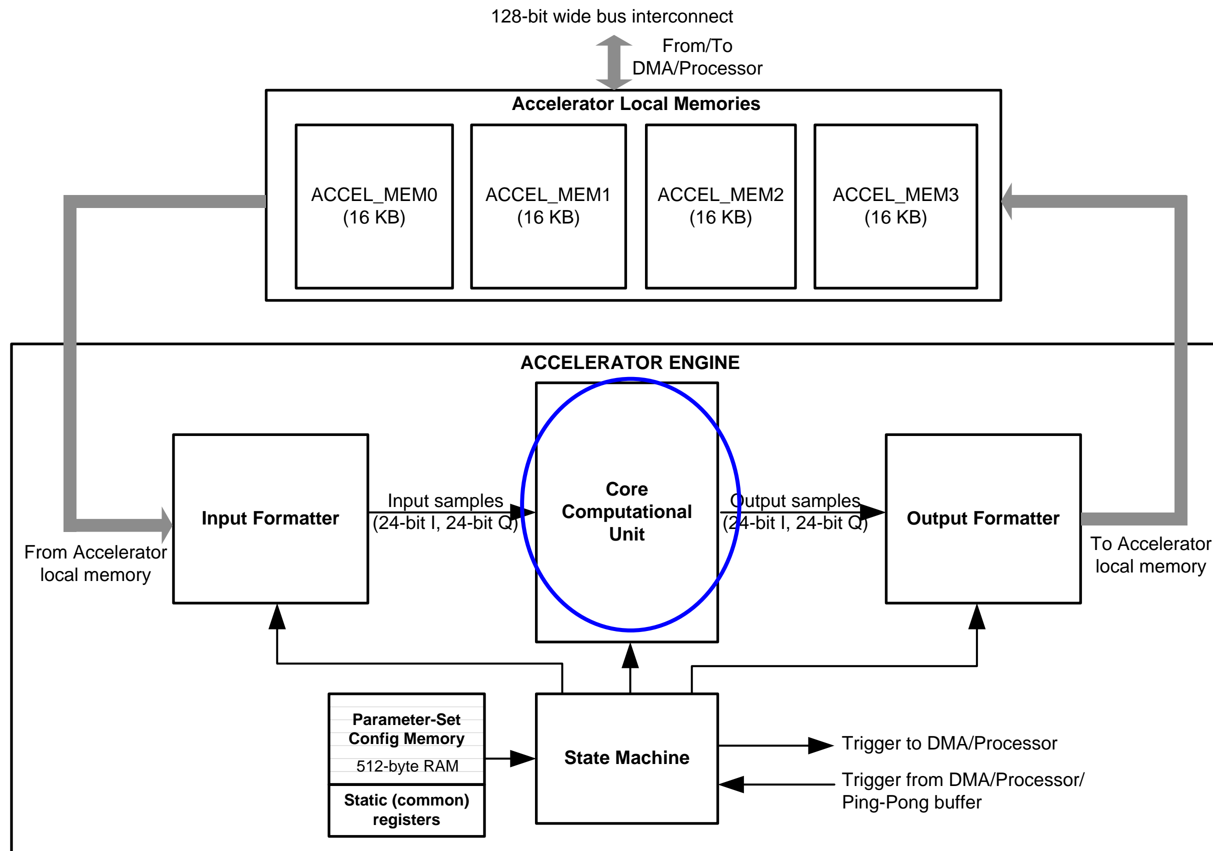


Figure 12. Core Computational Unit

5.1 Core Computational Unit

The core computational unit performs the mathematical operations required for the key functions, such as FFT, log-magnitude, and so on. The core computational unit accepts a streaming 24-bit complex input (24 bits for each I and Q) from the input formatter block and it outputs a streaming 24-bit complex output (24 bits for each I and Q) to the output formatter block. In addition to FFT and log-magnitude, the core computational unit has provision for simple pre-FFT processing, such as zeroing out large interference samples, complex derotation, and windowing prior to FFT. The core computational unit also contains a CFAR-CA detector unit for detecting peak samples (for example, radar targets).

Copyright © 2017, Texas Instruments Incorporated

Figure 13 shows the block diagram of the core computational unit. The core computational unit has two main paths – namely the FFT Engine path and the CFAR Engine path. Only one of these two paths can be operational at any given instant. However, in separate parameter sets, different paths can be configured and used, so that multiple parameter sets executing one after another can accomplish a sequence of computational operations as desired. The register ACCEL_MODE controls which path gets used in a given parameter set.

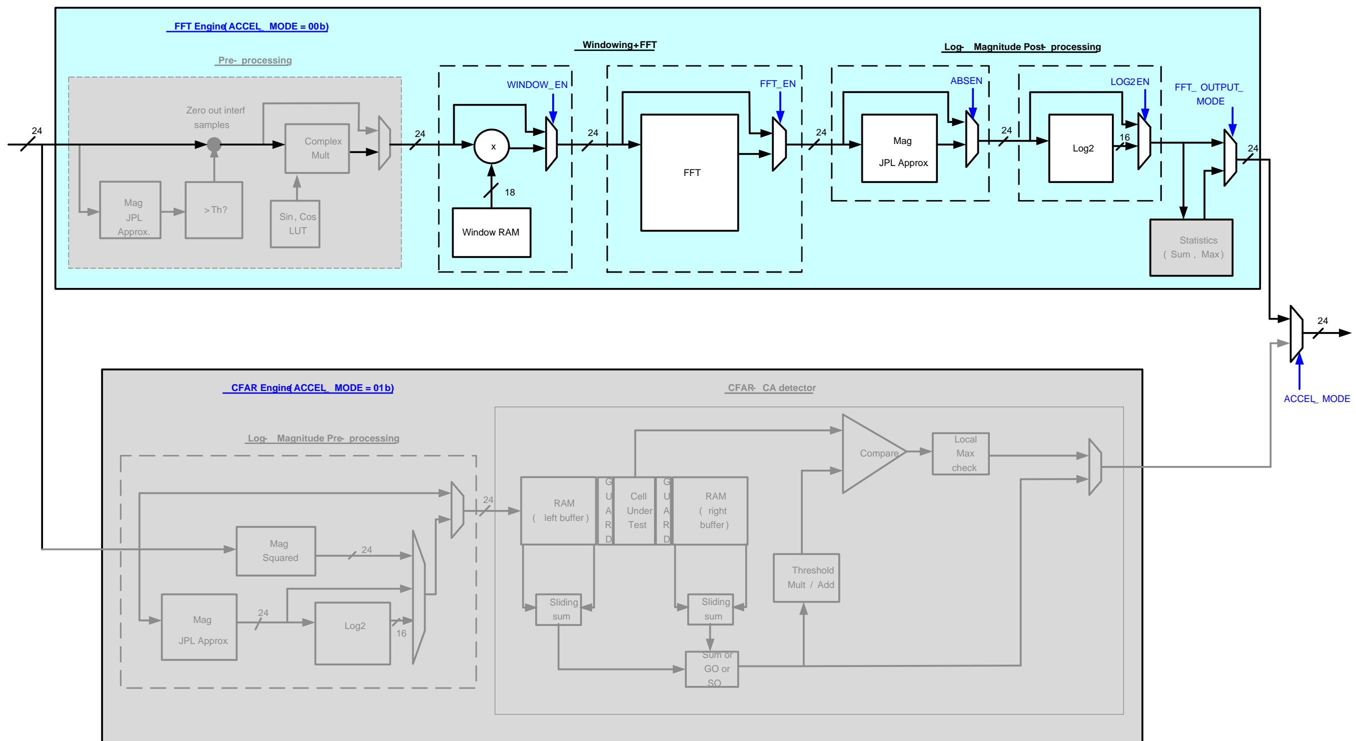


Figure 13. Core Computational Unit Block Diagram

For the purpose of part one of the user's guide, only the FFT Engine path is described. Specifically, the windowing, FFT, and log-magnitude operations are covered in this document. The greyed-out blocks in Figure 13, namely the Pre-processing, Statistics, and CFAR Engine, are covered in part two of the user's guide and can be ignored for the present purpose.

5.1.1 Core Computational Unit – Operation

The core computational unit operates on the streaming input of samples coming from the input formatter block, and in general outputs a stream of samples (after an initial latency in some cases) to the output formatter block. In general, at steady-state, one input sample is processed and one output sample is produced every 200-MHz clock.

The core computational unit has the ability to perform windowing, FFT, and log-magnitude computations. Each of these computational subblocks operate on a streaming input and produce a streaming output at the throughput of one sample per clock. These computational subblocks are stitched together one after the other in a series, as shown in Figure 13. This architecture allows multiple operations to be done in a streaming manner (for example, windowing and FFT can be done together), while at the same time, providing the user flexibility to choose one operation at a time.

The parameter-set registers WINDOW_EN, FFT_EN, ABSEN, and LOG2EN control the multiplexers (see Figure 13), which decide what operations are performed on the input samples for that parameter set.

Note that for the purpose of part one of the user's guide, the registers ACCEL_MODE and FFTOUT_MODE must be kept at zero. The purpose of these registers is covered in part two.

Copyright © 2017, Texas Instruments Incorporated

5.1.2 Core Computational Unit – Windowing

The incoming samples from the input formatter to the core computational unit are passed through the (optional) windowing operation (see Figure 13). Windowing operation is often required prior to performing FFT, to mitigate the sinc roll-off leakage from one strong FFT bin to the adjacent bins.

The implementation of the windowing operation in the accelerator is very straightforward. The window coefficients are preloaded by the Cortex-R4F processor into a dedicated Window RAM. The purpose of this RAM is to provide a fully programmable window (for example, Hann, Kaiser, or any proprietary window) to the user.

As the incoming samples from the input formatter stream in, each sample is multiplied by the appropriate window coefficient read from the RAM. The window coefficients must be real numbers and they are stored as 18-bit, signed, twos-complement numbers in the Window RAM. Because the incoming samples are complex 24-bits wide (24-bits for each I and Q), the windowing operation involves multiplying the 24-bit I and 24-bit Q of the incoming sample with the 18-bit real window coefficient (see Figure 14). The output of this multiplication is rounded back to 24-bit I and 24-bit Q by dropping excess LSBs.

Note that windowing can be enabled or disabled by using the register bit WINDOW_EN.

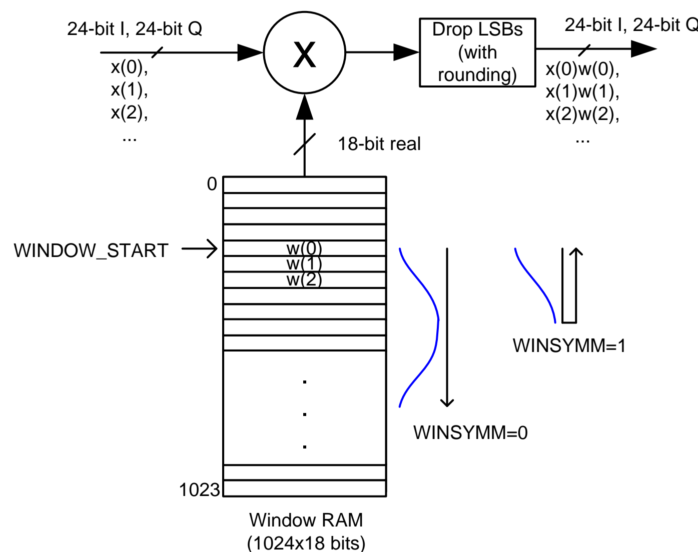


Figure 14. Windowing Computation

The Window RAM can hold a maximum of 1024 coefficients. It is possible to store more than one window function in the Window RAM. For example, two separate windows for first-dimension FFT and second-dimension FFT can be preloaded and kept in the Window RAM, as long as the total number of coefficients is 1024 or less.

The start address (for example, starting coefficient index between 0 to 1023) is programmed in a 10-bit register WINDOW_START as part of the parameter set, so that the windowing computation can pick the appropriate window coefficients starting from that index. For each incoming sample, the index keeps incrementing, so that each successive sample is multiplied by the successive window coefficient. At the end of each iteration (for example, when SRCACNT number of samples have been processed), the index resets back to the starting coefficient index programmed for the parameter set, so that the next iteration can be performed. At the end of all the iterations of the current parameter set, the next parameter set can use a different window if desired. For example, when performing second- and third-dimension FFTs one after another (in two parameter sets), the window functions for both these FFTs can be prestored in the Window RAM and appropriate start index can be provided for each of the FFT operation dimensions.

Copyright © 2017, Texas Instruments Incorporated

If the window function is symmetric, only one half of the set of window coefficients needs must be stored in the Window RAM. The register bit WINSYMM, when set, indicates that after SRCACNT / 2 samples (or, if SRCACNT is odd, (SRCACNT + 1) / 2 samples) are processed, the window coefficients read-indexing must be reversed, so that the same set of coefficients used for the first SRCACNT / 2 samples are reused in the reverse order for the next SRCACNT / 2 samples. (See Figure 14). If SRCACNT is odd, then the last window coefficient is read only once, when the direction is reversed. If SRCACNT is even, then the last window coefficients is read twice, when the direction is reversed.

The output of the windowing computation is 24-bit I and 24-bit Q, which is streamed into the FFT subblock.

5.1.3 Core Computational Unit – FFT

The FFT subblock performs FFT on the incoming 24-bit I and 24-bit Q data stream. The FFT sizes supported are all powers of 2 until 1024, for example, FFT sizes of 2, 4, 8, 16, … 512 and 1024 are supported. The lowest FFT size of 2 is mostly useful as a *complex add-subtract* feature or while using the *FFT stitching* feature. FFT sizes of 4, 8, 16, and 32 can be used for third dimension (angle estimation) FFT.

Note that FFT stitching is a feature that enables large FFT sizes, specifically, 2048 and 4096, using a two-step process (this feature is not covered here and is discussed in part two of the user's guide).

The FFT operation can be enabled or disabled by using the register bit FFT_EN. When enabled, the FFT subblock computes the FFT of the input data stream and produces a 24-bit I and 24-bit Q output stream. This output stream is initially in bit-reversed order, but the output formatter handles appropriately writing the output to the destination memory in the correct order.

The FFT implementation comprises ten butterfly stages. Depending on the FFT size needed, an appropriate number of butterfly stages are employed. The FFT size is programmed using the FFTSIZE register – for example, FFTSIZE = 5 means 32-point FFT, FFTSIZE = 7 means 128-point FFT, and so on. Note that the FFT size must be equal to or larger than SRCACNT, and the input formatter block automatically zero-pads extra samples to account for the difference between FFT size and SRCACNT. For example, if SRCACNT = 99 (for example, 100 samples) and FFTSIZE = 7 (for example, 128-point FFT), then the input formatter automatically appends 28 zero-pad samples for each iteration.

5.1.4 Core Computational Unit – FFT Quantization and Speed performance

As is well known, a butterfly stage typically consists of add-subtract and twiddle multiplication operations. At the output of each add-subtract structure, the bit-width would increase by 1 bit (for example, 24-bit input would grow to 25-bit output). To handle this one-bit growth due to add-subtract operation, there is a provision at the output of each butterfly add-subtract stage to scale the result back to 24 bits, by either dividing the output by 2 (round off one LSB) or by saturating one MSB, shown in Figure 15.

The 10-bit register BFLY_SCALING is used to control this divide-by-2 scaling operation at each stage, so that the user has full flexibility to control the signal level through the different butterfly stages. If BFLY_SCALING = 0 for a particular stage, then the 25-bit output is saturated at the MSB to get back to 24 bits. Otherwise, it is convergent-rounded at the LSB to get back to 24 bits. The user can thus control the scaling at each of the ten butterfly stages. The LSB of this 10-bit register corresponds to the last stage and the MSB of this register corresponds to the first stage. For an FFT size of 64, only the LSB 6 bits are relevant.

There is a 10-bit read-only register FFTCLIP which indicates whether there was any clipping in any of the butterfly stages. This register is a sticky register that gets set when a clipping event occurs and remains set until it is cleared using the CLR_FFTCLIP register bit. See the register description of FFTCLIP in Table 5.

Copyright © 2017, Texas Instruments Incorporated

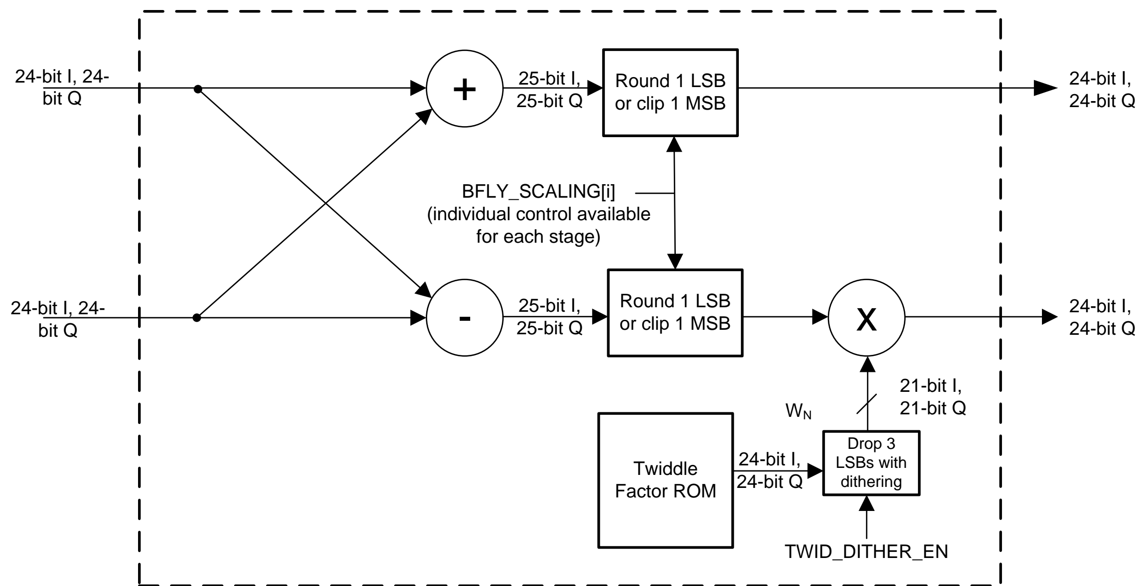


Figure 15. Butterfly Stage Fixed-Point

The twiddle factors are stored as 24-bit I and 24-bit Q coefficients. Prior to twiddle factor multiplication, the coefficients are reduced to 21-bit I and 21-bit Q by dropping three LSBs (with optional dithering). The purpose of dithering is to eliminate any repetitive quantization noise patterns from degrading the SFDR of the FFT. TI recommends that dithering be enabled (DITHERTWIDEN should be set). For dithering, an LFSR is used to generate a random pattern, for which the LFSR seed must be loaded with a non-zero value (see LFSRSEED in the register descriptions).

The SFDR performance of the FFT, with dithering enabled, is better than –140 dBc, as shown in Figure 16.

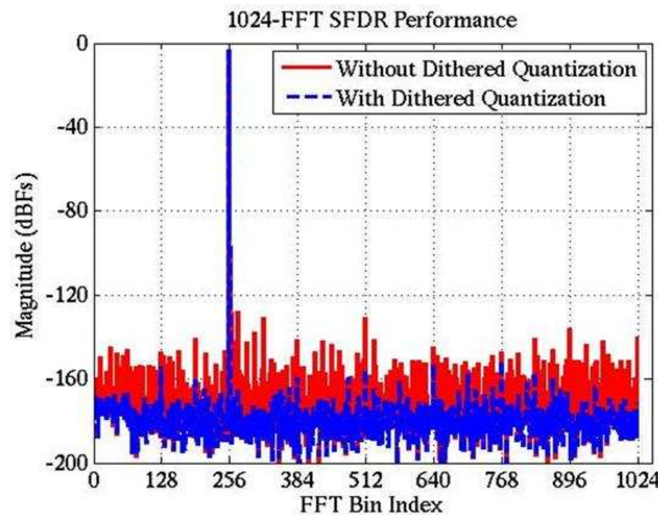


Figure 16. FFT SFDR Performance With and Without Dithering

The architecture of the FFT is such that it can take a streaming input (one sample per clock) and produce a streaming FFT output (one sample per clock), in steady-state. There is an initial latency of approximately *FFT size* number of clocks. This latency only comes into picture once for a given parameter set. Within a parameter set, multiple FFT iterations can be performed back-to-back (for example, for four RX) with no additional latency between iterations.

Copyright © 2017, Texas Instruments Incorporated

Because the implementation uses 200-MHz clock, a 256-point complex FFT for four RX chains would take 256 + 256 × 4 clock cycles to complete, which corresponds to 6.4 µs (plus a few clocks of implementation latencies, which are not accounted here). Table 4 lists the approximate computation time needed for various FFT sizes.

Table 4. FFT Computation Time

Example	FFT Size	Number of Back-to-Back Iterations	Number of Clock Cycles (Initial latency + Computation)	Total Duration
1	256	4	256 + (256 × 4)	6.4 µs
2	128	4	128 + (128 × 4)	3.2 µs
3	8	64	8 + (64 × 8)	2.6 µs

The output of the FFT can be fed to the output formatter or it can be sent to the magnitude/log-magnitude computation subblock.

NOTE: The FFT is a complex FFT implementation. If the input samples are real-only, then the SRCREAL register bit can be set, such that the imaginary part (Q-part) will be forced to zero by the input formatter block.

5.1.5 Core Computational Unit – Magnitude and Log-Magnitude Post-Processing

The magnitude and log-magnitude post-processing block computes absolute value or log2 of the absolute value of its input. Because this block is connected to the output of the FFT engine, the computation of absolute value (and log2) can be directly performed on the streaming FFT output. Alternately, the FFT block can be bypassed and only the magnitude and log-magnitude block can be employed.

The processing in this block first involves computation of magnitude (absolute value) of the input samples in the magnitude subblock (using JPL approximation). The result of the magnitude computation is fed into a Log2 computation subblock, which uses a look-up table-based approximation to compute logarithm-base-2 of the magnitude.

As shown in Figure 13, if the register-bit ABSEN is set, the magnitude computation subblock is enabled. In addition, if the register-bit LOG2EN is set, then the Log2 computation subblock is also enabled. Note that setting LOG2EN makes sense only when ABSEN is also set.

The magnitude computation uses JPL (Levitt and Morris) approximation. This approximation for magnitude of a complex number (I + jQ) is defined as follows, let $U = \max(|I|, |Q|)$ and $V = \min(|I|, |Q|)$.

Then, the magnitude can be approximated as follows in Equation 1.

$$\text{Magnitude} \approx \max(U + V / 8, 7U / 8 + V / 2) \tag{1}$$

The magnitude output is 24-bits wide (real number).

Next, the log2 computation of the magnitude value is achieved as follows. Any unsigned input number N can be written as N = 2k(1 + f) and the log2(N) can then be written as follows in Equation 2.

$$\log2(N) = k + \log2(1+f) \tag{2}$$

Copyright © 2017, Texas Instruments Incorporated

The implementation of log2 computation uses the previous formula, where a look-up table approximation is used to generate the second term, for example, log2(1 + f). The accuracy of the log2 computation is shown in Figure 17. The log2 output is 16-bits wide. The 16-bit logarithm output consists of 5 bits of integer part and 11 bits of fractional part.

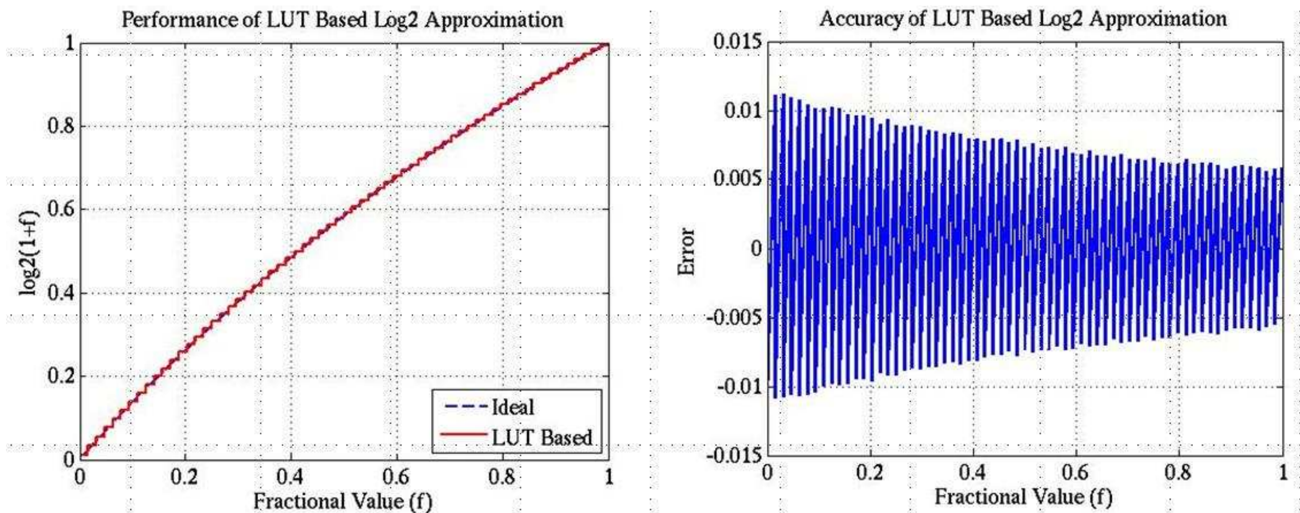


Figure 17. Accuracy of Log2 Computation

Depending on the settings of ABSEN and LOG2EN, either the magnitude or the log-magnitude is sent as the final output of the core computational unit. The final output of the core computational unit going to the output formatter is 24-bits I and 24-bits Q. Thus, if either magnitude or log-magnitude is enabled, the Q-values are just made zeros. Similarly, when log2 is enabled, because the output is 16-bits, 8 MSBs are filled as zero.

The output formatter handles writing the samples to the destination memory as per the configured destination memory access pattern described in a previous section.

5.1.6 Core Computational Unit – Register Descriptions

Table 5 lists all the registers of the core computational unit.

Table 5. Core Computational Unit Registers

Register	Width	Parameter Set	Description
WINDOW_EN	1	Yes	Windowing Enable: This register-bit enables or disables the pre-FFT windowing operation. If this register is set to 1, then the windowing is enabled, otherwise, it is disabled. The exact window function (coefficients) to be applied is specified in a dedicated Window RAM, which is 1024 × 18 bits in size.
FFT_EN	1	Yes	FFT Enable: This register-bit is used to enable the FFT computation. If FFT_EN = 1, then the FFT computation is enabled. Otherwise, it is disabled (bypassed).
ABSEN	1	Yes	Magnitude Enable: This register-bit is used to enable the magnitude calculation. If this register bit is set, then the magnitude calculation is enabled, else it is bypassed. When enabled, the magnitude (absolute value) of the input complex samples are calculated using JPL approximation and the resulting magnitude value is sent on the I-arm of the output. The Q-arm is made zeros.
LOG2EN	1	Yes	Log2 Enable: This register-bit is used to enable the Log2 computation. If this register bit is set, then the Log2 computation is enabled, else it is bypassed. Note that setting this register bit only makes sense if the inputs to the Log2 computation are unsigned real numbers, such as when the Magnitude Enable bit (ABSEN) is also set. When enabled, the Log2 of the magnitude of the input samples is calculated and sent out on the I-arm of the output. The Q-arm is made zeros.

Copyright © 2017, Texas Instruments Incorporated

Table 5. Core Computational Unit Registers (continued)

Register	Width	Parameter Set	Description
WINDOW_START	10	Yes	Windowing coefficients start index: This register specifies the starting index of the window coefficients within the Window RAM. The value of this register ranges from 0 to 1023. The purpose of this register is to allow multiple windows (for example, one window of 512 coefficients and another window of 256 coefficients) to be stored in the Window RAM and one of these windows can be used by programming this start index register appropriately in the current parameter set.
WINSYMM	1	Yes	Window symmetry: This register-bit indicates whether the complete set of window coefficients are stored in the Window RAM or whether one half of the coefficients are stored. If this register bit is set, it means that the window function is symmetric and therefore, only one half of the window function coefficients are stored in the Window RAM. See the description section related to Windowing computation for more details.
FFTSIZE	4	Yes	FFT size: This register specifies the FFT size. The mapping of the FFTSIZE register to the actual FFT size is as follows: Actual FFT size = 2 ^ (FFTSIZE). For example, a register value of 0110b specifies that the FFT size is 64. The maximum FFT size that is supported is 1024. Therefore, this register value is never expected to exceed 1010b. Note that the FFT size should be equal to or larger than SRCACNT and the Input Formatter block will automatically zero-pad extra samples to account for the difference between FFT size and SRCACNT. For large-size FFT (> 1024 point) that might be useful for industrial level-sensing applications, an FFT stitching procedure is supported, which is based on performing multiple smaller size FFTs in a first step and then stitching them in a second step (using a subsequent parameter set). This FFT stitching feature is covered in part two of the user's guide.
BFLY_SCALING	10	Yes	Butterfly scaling: This register is used to control the butterfly scaling at each stage of the FFT structure. Because the maximum FFT size is 1024, there are up to ten butterfly stages. Each butterfly stage has an add-and-subtract structure, at the output of which the bit-width would temporarily increase by 1 (from 24 to 25 bits wide). If BFLY_SCALING = 0, then the 25-bit output is saturated at the MSB to get back to 24 bits. Otherwise, it is convergent-rounded at the LSB to get back to 24 bits. The user can thus control the scaling at each of the 10 butterfly stages. The LSB of this register corresponds to the last stage and the MSB of this register corresponds to the first stage. For an FFT size of 64, only the LSB 6 bits are relevant.
DITHERTWIDEN	1	No	Twiddle factor dithering enable: This register-bit is used to enable and disable dithering of twiddle factors in the FFT. The twiddle factors are 24-bits wide (24-bits for each I and Q), but they are quantized to 21-bits before twiddle factor multiplication. This quantization is implemented with dithering on the LSB, to avoid periodic quantization pattern affecting SFDR performance of the FFT. TI recommends keeping this register bit set to 1 (for example, dithering enabled), with appropriate LSFR seed loaded (see the following).
LFSRSEED	29	No	Seed for LFSR (random pattern): For twiddle factor dithering, there is an LFSR that is used, whose seed value is loaded by writing to this 29-bit LFSRSEED register. The LFSRSEED register should be set to any non-zero value, say 0x1234567. To load the LFSR seed, a pulse signal needs to be provided, by writing a 1 followed by a 0 (i.e., by setting and clearing) the LFSRLOAD register-bit.
LFSRLOAD	1	No	
FFTCLIP	10	No	FFT Clip Status (read-only): This is a read-only status register, which indicates any saturation/clipping events that have happened in the FFT butterfly stages. Note that each of the 10 butterfly stages in the FFT can be programmed to either saturate the MSB or round the LSB. Whenever saturation of MSB is used in any stage, there is a possibility that that stage can saturate or clip samples. In that case, this saturation event is indicated in the corresponding bit in this status register, so that the Cortex-R4F processor can read it. If multiple FFTs are performed, this status register includes any saturation events happening in any of them. This status register can only be cleared by the R4F, by setting another single-bit register CLR_FFTCLIP, so that the saturation status indication gets cleared back to 0 and any subsequent saturation events can be freshly monitored.
CLR_FFTCLIP	1	No	Clear FFT Clip Status register: This register bit, when set, clears the FFTCLIP register.

Copyright © 2017, Texas Instruments Incorporated

Table 5. Core Computational Unit Registers (continued)

Register	Width	Parameter Set	Description
ACCEL_MODE	2	Yes	Select Core Computational Unit Data Path: This register selects the data-path mode of the accelerator's core computational unit – for example, it selects whether the FFT engine path or the CFAR engine path is active. This register will be covered in part two. For the purpose of part one of the user's guide, this register should be zero.
INTERFTHRESH	–	–	Described in part two of this user's guide. For the immediate purposes relevant to part one of this user's guide, all of these registers should be kept as 0.
INTERFTHRESH_EN	–	–	
WINDOW_INTERP_FRACTION	–	–	
CMULT_MODE	–	–	
TWID_INCR	–	–	
STG1LUTSELWR	–	–	
FFT_OUT_MODE	–	–	
FFTSUMDIV	–	–	
MAXn_VALUE	–	–	
ISUMn, ISUMn	–	–	
CFAR_AVG_LEFT	–	–	
CFAR_AVG_RIGHT	–	–	
CFAR_GUARD_INT	–	–	
CFAR_THRESH	–	–	
CFAR_LOG_MODE	–	–	
CFAR_INP_MODE	–	–	
CFAR_ABS_MODE	–	–	
CFAR_OUT_MODE	–	–	
CFAR_GROUPING_EN	–	–	
CFAR_NOISE_DIV	–	–	
CFAR_CA_MODE	–	–	
CFAR_CYCLIC	–	–	
FFTPEAKCNT	–	–	

Copyright © 2017, Texas Instruments Incorporated

6 Radar Hardware Accelerator – Use Case Example

This section presents a use-case example that illustrates how to configure and use the Radar Hardware Accelerator to achieve some of the frequently used computations in FMCW radar signal processing.

6.1 Ultra-Short-Range Radar Use Case

This example illustrates a typical end-to-end radar signal processing flow and how it can be accomplished using the Radar Hardware Accelerator and Cortex-R4F processor. The use case assumes a two-TX, two-RX configuration, with a chirp profile as in Table 6.

Table 6. Chirp Configuration Used for Illustration

Parameter	Value	Comments
Chirp duration	50 µs (active) + 10 µs (idle)	–
Sweep bandwidth	2 GHz	7.5-cm range resolution
Ramp slope	40 MHz/µs	–
Maximum range	15 m	–
Maximum beat frequency	4 MHz	–
ADC sampling rate	4.5 MHz	Complex I,Q sampling
Number of samples per chirp	225	–
First-dimension FFT size	256	225 samples + 31 zeros
Number of chirps per frame	64 × 2 = 128	TX1, TX2 alternating (64 chirps each)
Number of channels	Two TX, two RX	Effective four channels (assuming sparse antenna array with TX's ʎ-separated and RX's ʎ / 2-separated)
Radar cube data memory	256 KB	256 × (64 × 2) × 2 × 2 × 2 = 262144 bytes
Frame time	128 × 60 µs = 7.68 ms	–
Second-dimension FFT size	64	Every alternate chirp
Third-dimension FFT size	4 × 2 = 8	Four channels, four zero pads

Copyright © 2017, Texas Instruments Incorporated

6.1.1 Use Case Illustration – First-Dimension FFT Processing Configuration

During active chirp transmission, the digital front-end (DFE) writes ADC samples to the ADC buffer in ping-pong manner. This example assumes that the ADC data is complex – for example, the RF/Analog is configured as a complex baseband (instead of real-only) chain. The DFE is configured to write two chirps at a time into the ping ADC buffer and two chirps at a time into the pong ADC buffer. This allows effective four parallel channels (two TX and two RX) worth of ADC data to be captured in ping or pong buffer at any time. The DFE configuration details are outside the scope of this user's guide.

To achieve inline, first-dimension, FFT processing using the Radar hardware accelerator, the FFT1DEN register-bit is set, such that the ADC buffer is shared with the accelerator input memories, and the DFE output is directly available to the accelerator for processing at the end of every ping-pong switch.

The data layout in the ADC buffer and in the accelerator local memories after each processing step is shown in Figure 18, Figure 19, Figure 20, Figure 21, Figure 22, and Figure 23. The suffix *0, RX1* means the first ADC sample from the RX1 antenna, *6, RX2* means the seventh ADC sample from the RX2 antenna, and so on. In the following figures, the data corresponding to different chirps is shown in different background colors. All the pictures represent the data layout in either ping or pong memory (for example, 16KB).

$I_{0, RX1}$	$Q_{0, RX1}$	$I_{1, RX1}$	$Q_{1, RX1}$	$I_{2, RX1}$	$Q_{2, RX1}$	$I_{3, RX1}$	$Q_{3, RX1}$
$I_{4, RX1}$	$Q_{4, RX1}$	...	...	...	...	...	...
...	...	...	...	...	...	...	...
$I_{224, RX1}$	$Q_{224, RX1}$						
$I_{0, RX1}$	$Q_{0, RX1}$	$I_{1, RX1}$	$Q_{1, RX1}$	$I_{2, RX1}$	$Q_{2, RX1}$	$I_{3, RX1}$	$Q_{3, RX1}$
$I_{4, RX1}$	$Q_{4, RX1}$	...	...	...	...	...	...
...	...	...	...	...	...	...	...
$I_{224, RX1}$	$Q_{224, RX1}$						
$I_{0, RX2}$	$Q_{0, RX2}$	$I_{1, RX2}$	$Q_{1, RX2}$	$I_{2, RX2}$	$Q_{2, RX2}$	$I_{3, RX2}$	$Q_{3, RX2}$
$I_{4, RX2}$	$Q_{4, RX2}$	...	...	...	...	...	...
...	...	...	...	...	...	...	...
$I_{224, RX2}$	$Q_{224, RX2}$						
$I_{0, RX2}$	$Q_{0, RX2}$	$I_{1, RX2}$	$Q_{1, RX2}$	$I_{2, RX2}$	$Q_{2, RX2}$	$I_{3, RX2}$	$Q_{3, RX2}$
$I_{4, RX2}$	$Q_{4, RX2}$	...	...	...	...	...	...
...	...	...	...	...	...	...	...
$I_{224, RX2}$	$Q_{224, RX2}$						

Figure 18. Layout of Samples in ADC Buffer (Ping)

Copyright © 2017, Texas Instruments Incorporated

The first-dimension FFT input is directly picked up from the ADC buffer and it consists of samples from each antenna placed consecutively in memory. The key register configurations required for first-dimension FFT processing are listed in Table 7.

Table 7. Key Register Configurations for First-Dimension FFT

Register	Value	Comments
FFT_EN	1	Enable FFT computation
FFTSIZE	8	FFT size = 2 ^ 8 = 256 225 valid samples + zero padding
SRCACNT	224	225 valid samples (zero-based count)
SRCAINDX	4	Adjacent samples spaced 4 bytes apart
REG_BCNT	1	Two RX antennas processed back-to-back (zero-based count)
SRCBINDX	4096	Samples of RX1 and RX2 are spaced 4KB apart
SRCADDR	0 (parameter sets 0, 3) 1024 (parameter sets 1, 2)	Start at beginning of ACCEL_MEM0 for first chirp. Use another parameter set for second chirp, which starts 1KB away from first chirp. Two additional parameter sets are required for pong operation (this is actually required for destination memory, not for source memory which does automatic ping-pong memory selection).
DSTADDR	32KB (parameter set 0) 32KB + 8B (parameter set 1) 48KB (parameter set 2) 48KB + 8B (parameter set 3)	Destination is MEM2 or MEM3 (ping-pong). There is a separation of 8 bytes between RX1 and RX2 *interlaced* samples. See layout picture for first-dimension FFT output.
DSTAINDX	16	See layout picture for first-dimension FFT output
DSTBINDX	4	See layout picture for first-dimension FFT output
SRC16b32b	0	FFT input samples are 16-bit word aligned
DST16b32b	0	FFT output samples are 16-bit word aligned
TRIGMODE	010b (parameter set 0) 000b (parameter set 1) 010b (parameter set 2) 000b (parameter set 3)	Trigger is based on ping → pong or pong → ping switch. For chained parameter sets to process the two chirps, immediate trigger is used.

Note from Table 7 that the two chirps are processed here using two parameter sets. This is because the separation of ADC samples for the two chirps is different (intentionally kept different in this example, just for illustration) from the separation of the ADC samples for the two RX antennas. Becaue the REG_BCNT and BINDX mechanisms inherently assume uniform spacing, it is not possible to perform the first-dimension FFT processing for both RX antennas (two RX) and both chirps (two TX) using a single parameter set. To avoid this problem, it is possible to place the ADC samples for the two RX antennas 2KB apart (instead off 4KB apart) so that the spacing is uniform, or alternately, just use two parameter sets as done in this example.

The first-dimension FFT processing using ping-pong mechanism is continued for all chirps inline, during active transmission and reception of data. Due to the ping-pong mechanism, another two parameter sets would be required, so that the FFT output memories are also ping-ponged for efficient DMA transfer. In other words, a total of four parameter sets are used for first-dimension processing.

Copyright © 2017, Texas Instruments Incorporated

The layout of the first-dimension FFT output samples in the destination memory (accelerator local memories ACCEL_MEM2 and ACCEL_MEM3) are shown in Figure 19. Note that the alignment here is chosen to be *interlaced*, such that the two RX antennas and the two chirps are all interlaced with each other. This allows efficient 128-bit data transfer of each range-bin (FFT bin) to the Radar Cube memory.

$I_{0,RX1}$	$Q_{0,RX1}$	$I_{0,RX2}$	$Q_{0,RX2}$	$I_{0,RX1}$	$Q_{0,RX1}$	$I_{0,RX2}$	$Q_{0,RX2}$
$I_{1,RX1}$	$Q_{1,RX1}$	$I_{1,RX2}$	$Q_{1,RX2}$	$I_{1,RX1}$	$Q_{1,RX1}$	$I_{1,RX2}$	$Q_{1,RX2}$
$I_{2,RX1}$	$Q_{2,RX1}$	$I_{2,RX2}$	$Q_{2,RX2}$	$I_{2,RX1}$	$Q_{2,RX1}$	$I_{2,RX2}$	$Q_{2,RX2}$
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
$I_{255,RX1}$	$Q_{255,RX1}$	$I_{255,RX2}$	$Q_{255,RX2}$	$I_{255,RX1}$	$Q_{255,RX1}$	$I_{255,RX2}$	$Q_{255,RX2}$

Figure 19. Layout of First-Dimension FFT Output Samples in Accelerator Local Memory

Copyright © 2017, Texas Instruments Incorporated

6.1.2 Use Case Illustration – Radar Data Cube Storage

The first-dimension FFT outputs are stored as 16-bit complex samples in the Radar Data Cube memory, as per the layout shown in Figure 20. Different chirps are shown with different background color.

Because DMA-writes are less efficient for noncontiguous access, the transpose is not yet done, while DMA'ing data into the Radar Data Cube memory. The transpose is done only when reading data back for second and third dimension processing.

$I_{0,RX1}$	$Q_{0,RX1}$	$I_{0,RX2}$	$Q_{0,RX2}$	$I_{0,RX1}$	$Q_{0,RX1}$	$I_{0,RX2}$	$Q_{0,RX2}$
$I_{1,RX1}$	$Q_{1,RX1}$	$I_{1,RX2}$	$Q_{1,RX2}$	$I_{1,RX1}$	$Q_{1,RX1}$	$I_{1,RX2}$	$Q_{1,RX2}$
$I_{2,RX1}$	$Q_{2,RX1}$	$I_{2,RX2}$	$Q_{2,RX2}$	$I_{2,RX1}$	$Q_{2,RX1}$	$I_{2,RX2}$	$Q_{2,RX2}$
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
$I_{255,RX1}$	$Q_{255,RX1}$	$I_{255,RX2}$	$Q_{255,RX2}$	$I_{255,RX1}$	$Q_{255,RX1}$	$I_{255,RX2}$	$Q_{255,RX2}$
$I_{0,RX1}$	$Q_{0,RX1}$	$I_{0,RX2}$	$Q_{0,RX2}$	$I_{0,RX1}$	$Q_{0,RX1}$	$I_{0,RX2}$	$Q_{0,RX2}$
$I_{1,RX1}$	$Q_{1,RX1}$	$I_{1,RX2}$	$Q_{1,RX2}$	$I_{1,RX1}$	$Q_{1,RX1}$	$I_{1,RX2}$	$Q_{1,RX2}$
$I_{2,RX1}$	$Q_{2,RX1}$	$I_{2,RX2}$	$Q_{2,RX2}$	$I_{2,RX1}$	$Q_{2,RX1}$	$I_{2,RX2}$	$Q_{2,RX2}$
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
$I_{255,RX1}$	$Q_{255,RX1}$	$I_{255,RX2}$	$Q_{255,RX2}$	$I_{255,RX1}$	$Q_{255,RX1}$	$I_{255,RX2}$	$Q_{255,RX2}$
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
$I_{0,RX1}$	$Q_{0,RX1}$	$I_{0,RX2}$	$Q_{0,RX2}$	$I_{0,RX1}$	$Q_{0,RX1}$	$I_{0,RX2}$	$Q_{0,RX2}$
$I_{1,RX1}$	$Q_{1,RX1}$	$I_{1,RX2}$	$Q_{1,RX2}$	$I_{1,RX1}$	$Q_{1,RX1}$	$I_{1,RX2}$	$Q_{1,RX2}$
$I_{2,RX1}$	$Q_{2,RX1}$	$I_{2,RX2}$	$Q_{2,RX2}$	$I_{2,RX1}$	$Q_{2,RX1}$	$I_{2,RX2}$	$Q_{2,RX2}$
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
$I_{255,RX1}$	$Q_{255,RX1}$	$I_{255,RX2}$	$Q_{255,RX2}$	$I_{255,RX1}$	$Q_{255,RX1}$	$I_{255,RX2}$	$Q_{255,RX2}$

Figure 20. Layout of First-Dimension FFT Output Samples in Radar Data Cube Memory

Copyright © 2017, Texas Instruments Incorporated

6.1.3 Use Case Illustration – Second (and Third-Dimension) FFT Processing

At the end of active chirp transmission, the ADC buffer is switched over to be directly under the control of the accelerator (FFT1DEN register-bit is cleared). The first-dimension FFT output samples from the Radar Data Cube memory are brought in through DMA to the accelerator local memories (ACCEL_MEM0 and ACCEL_MEM1). This can also be done with a ping-pong mechanism to achieve best overall throughput. The accelerator must be triggered based on (additional) linked DMA channels writing the appropriate register bit into DMA2ACCTRIG to signal completion of the DMA and trigger the appropriate parameter set of the accelerator.

In the example use case shown here, three range-bins are processed at a time for second and third-dimension FFT. This is not really necessary (one range bin can be done at a time instead), but is shown here for the purpose of illustration. Note also that in some implementations, instead of third-dimension FFT, a noncoherent accumulation of the second-dimension FFT output is done across antennas and this result is used for detection. Subsequently, the third-dimension FFT is done only for detected objects. However, the example shown here assumes third-dimension FFT processing is done for all range-velocity cells and the detection is performed only after third-dimension FFT. Another thing to note is that prior to third-dimension FFT, some complex derotations for phase (angle-of-arrival) calibration may be needed. For most practical purposes, this derotation can be achieved inside the DFE and is therefore not needed in the accelerator. However, if desired, this derotation can be achieved through the Complex Vector Multiplication feature in the accelerator (described in part two of the user's guide).

Continuing with the present example, because there are 64 × 2 × 2 data samples for each range bin, the number of bytes per range bin is 1KB. This means that the number of bytes for three range bins is 3KB. Note that as part of the second and third-dimension FFT processing, this 3KB number will grow four times bigger to 12 KB. This is because the second and third-dimension FFT outputs are stored as 32-bit wide samples (2× increase) and also because the third-dimension FFT is zero-padded from 4 to 8 (another 2× increase).

The layout of the data samples at the input to the second-dimension FFT processing is shown in Figure 21. Note that the gaps left in the data sample arrangement in Figure 21 are not really required in the source memory and are shown just for illustration. The three range bins are separated by 5KB each – for example, range 1 starts at 0 KB, range 2 starts at 5KB, and range 3 starts at 10KB.

The parameter sets 0, 1, 2, and 3 used for first-dimension FFT processing are overwritten for second and third-dimension processing, which uses 12 parameter sets of its own (for illustration). The reason for using 12 parameter sets here is because of the fact that three parameter sets are required for processing the three range bins, multiplied by a factor of two for second and third dimensions, and another factor of 2 for ping-pong mechanism.

- Parameter sets 0 – 2 → second-dimension FFT processing for three range bins, ping buffer
- Parameter sets 3 – 5 → third-dimension FFT processing for three range bins, ping buffer
- Parameter sets 6 – 8 → second-dimension FFT processing for three range bins, pong buffer
- Parameter sets 9 – 11 → third-dimension FFT processing for three range bins, pong buffer

Copyright © 2017, Texas Instruments Incorporated

Table 8. Key Register Configurations for Second-Dimension FFT

Parameter	Value	Comments
FFT_EN	1	Enable FFT computation
FFTSIZE	6	FFT size = 64 64 chirps for each TX and RX combination
SRCACNT	63	64 samples (no zero padding)
SRCAINDX	16	Adjacent samples spaced 16 bytes apart
REG_BCNT	3	Two TX, two RX processed back-to-back
SRCBINDX	4	See layout picture
SRCADDR	0 (parameter set 0) 5120 (parameter set 1) 10240 (parameter set 2)	Three range bins are processed using three parameter sets. Ping-pong (not shown here) means additional three parameter sets. 5KB separation between range bins is not really required, they can be packed closer at the input memory if required.
DSTADDR	32KB (parameter set 0) 32KB+5KB (parameter set 1) 32KB+10KB (parameter set 2)	Destination is ACCEL_MEM2 (or ACCEL_MEM3 – ping-pong not shown here explicitly). Three range bins separated by 5KB in output memory.
DSTAINDX	64	Refer layout picture for second dim FFT output – (Note that output samples are 32-bits wide.)
DSTBINDX	16	Refer layout picture for second dim FFT output
SRC16b32b	0	FFT input samples are 16-bit word aligned
DST16b32b	1	FFT output samples are 32-bit word aligned
TRIGMODE	011b (parameter set 0, 6) 000b (all others)	DMA-based trigger is used to start the accelerator second-dimension FFT operations
DMA2ACC_CHANNEL_TRIGSRC	0 (parameter set 0) 1 (parameter set 6)	Bit 0 (LSB) of DMA2ACCTRIG is monitored for triggering parameter set 0. Note that a linked DMA channel can be used to set the Bit0 of DMA2ACCTRIG when the DMA transfer into ping memory is complete. Bit 1 of DMA2ACCTRIG is monitored for triggering parameter set 6, which can also be set using a linked DMA channel when the DMA transfer into pong memory is complete.

Copyright © 2017, Texas Instruments Incorporated

Note that even though it is not explicitly listed in Table 8, the second-dimension FFT processing uses parameter sets 0, 1, and 2, as well as parameter sets 6, 7, and 8.

$I_{0, RX1}$	$Q_{0, RX1}$	$I_{0, RX2}$	$Q_{0, RX2}$	$I_{0, RX1}$	$Q_{0, RX1}$	$I_{0, RX2}$	$Q_{0, RX2}$
$I_{0, RX1}$	$Q_{0, RX1}$	$I_{0, RX2}$	$Q_{0, RX2}$	$I_{0, RX1}$	$Q_{0, RX1}$	$I_{0, RX2}$	$Q_{0, RX2}$
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
$I_{0, RX1}$	$Q_{0, RX1}$	$I_{0, RX2}$	$Q_{0, RX2}$	$I_{0, RX1}$	$Q_{0, RX1}$	$I_{0, RX2}$	$Q_{0, RX2}$
$I_{1, RX1}$	$Q_{1, RX1}$	$I_{1, RX2}$	$Q_{1, RX2}$	$I_{1, RX1}$	$Q_{1, RX1}$	$I_{1, RX2}$	$Q_{1, RX2}$
$I_{1, RX1}$	$Q_{1, RX1}$	$I_{1, RX2}$	$Q_{1, RX2}$	$I_{1, RX1}$	$Q_{1, RX1}$	$I_{1, RX2}$	$Q_{1, RX2}$
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
$I_{1, RX1}$	$Q_{1, RX1}$	$I_{1, RX2}$	$Q_{1, RX2}$	$I_{1, RX1}$	$Q_{1, RX1}$	$I_{1, RX2}$	$Q_{1, RX2}$
$I_{2, RX1}$	$Q_{2, RX1}$	$I_{2, RX2}$	$Q_{2, RX2}$	$I_{2, RX1}$	$Q_{2, RX1}$	$I_{2, RX2}$	$Q_{2, RX2}$
$I_{2, RX1}$	$Q_{2, RX1}$	$I_{2, RX2}$	$Q_{2, RX2}$	$I_{2, RX1}$	$Q_{2, RX1}$	$I_{2, RX2}$	$Q_{2, RX2}$
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
...	...	...	...	...	...	...	...
$I_{2, RX1}$	$Q_{2, RX1}$	$I_{2, RX2}$	$Q_{2, RX2}$	$I_{2, RX1}$	$Q_{2, RX1}$	$I_{2, RX2}$	$Q_{2, RX2}$

(1) Three range bins at a time, in the accelerator local memory

Figure 21. Layout of Second-Dimension FFT Input Samples

Copyright © 2017, Texas Instruments Incorporated

$I_{0, RX1}$	=	$Q_{0, RX1}$	=
$I_{0, RX2}$	=	$Q_{0, RX2}$	=
$I_{0, RX1}$	=	$Q_{0, RX1}$	=
$I_{0, RX2}$	=	$Q_{0, RX2}$	=
$I_{0, RX1}$	=	$Q_{0, RX1}$	=
$I_{0, RX2}$	=	$Q_{0, RX2}$	=
$I_{0, RX1}$	=	$Q_{0, RX1}$	=
$I_{0, RX2}$	=	$Q_{0, RX2}$	=
...		...	
...		...	
...		...	
...		...	
...		...	
...		...	
$I_{0, RX1}$	=	$Q_{0, RX1}$	=
$I_{0, RX2}$	=	$Q_{0, RX2}$	=
$I_{0, RX1}$	=	$Q_{0, RX1}$	=
$I_{0, RX2}$	=	$Q_{0, RX2}$	=
$I_{1, RX1}$	=	$Q_{1, RX1}$	=
$I_{1, RX2}$	=	$Q_{1, RX2}$	=
$I_{1, RX1}$	=	$Q_{1, RX1}$	=
$I_{1, RX2}$	=	$Q_{1, RX2}$	=
$I_{1, RX1}$	=	$Q_{1, RX1}$	=
$I_{1, RX2}$	=	$Q_{1, RX2}$	=
$I_{1, RX1}$	=	$Q_{1, RX1}$	=
$I_{1, RX2}$	=	$Q_{1, RX2}$	=
...		...	
...		...	
...		...	
...		...	
...		...	
...		...	
$I_{1, RX1}$	=	$Q_{1, RX1}$	=
$I_{1, RX2}$	=	$Q_{1, RX2}$	=
$I_{1, RX1}$	=	$Q_{1, RX1}$	=
$I_{1, RX2}$	=	$Q_{1, RX2}$	=
$I_{2, RX1}$	=	$Q_{2, RX1}$	=
$I_{2, RX2}$	=	$Q_{2, RX2}$	=
$I_{2, RX1}$	=	$Q_{2, RX1}$	=
$I_{2, RX2}$	=	$Q_{2, RX2}$	=
$I_{2, RX1}$	=	$Q_{2, RX1}$	=
$I_{2, RX2}$	=	$Q_{2, RX2}$	=
$I_{2, RX1}$	=	$Q_{2, RX1}$	=
$I_{2, RX2}$	=	$Q_{2, RX2}$	=
...		...	
...		...	
...		...	
...		...	
...		...	
...		...	
$I_{2, RX1}$	=	$Q_{2, RX1}$	=
$I_{2, RX2}$	=	$Q_{2, RX2}$	=
$I_{2, RX1}$	=	$Q_{2, RX1}$	=
$I_{2, RX2}$	=	$Q_{2, RX2}$	=

(1) 3 range bins at a time in the accelerator local memory – these are stored as 32-bit wide I and Q samples

Figure 22. Layout of Second-Dimension FFT Output Samples

Copyright © 2017, Texas Instruments Incorporated

The second-dimension processing is immediately followed by third-dimension processing (immediate trigger). Table 9 lists the configuration for this.

Table 9. Key Register Configurations for Third Dimension FFT

Parameter	Value	Comments
FFT_EN	1	Enable FFT computation
FFTSIZE	3	FFT size = 2 ^ 3 = 8 Two TX, two RX gives four channels, add four zeros
SRCACNT	3	Four samples (zero-based count), four zero-pads
SRCAINDX	16	Adjacent input samples spaced 16 bytes apart
REG_BCNT	63	64 chirps to process
SRCBINDX	64	See layout picture
SRCADDR	32KB (parameter set 3) 32KB + 5KB (parameter set 4) 32KB + 10KB (parameter set 5)	Three range bins are processed using three parameter sets. Ping-pong (not shown here) means additional three parameter sets.
DSTADDR	0KB (parameter set 3) 5KB (parameter set 4) 10KB (parameter set 5)	Destination is MEM0 or MEM1 (ping-pong not shown here). Three range bins are separated by 5KB at input and output. Note that the original second-dimension FFT input samples get overwritten here with third-dimension FFT output.
DSTAINDX	8	See layout picture for third-dimension FFT output – (Note that output samples are 32-bits wide.)
DSTBINDX	64	See layout picture for third-dimension FFT output
SRC16b32b	1	FFT input samples are 32-bit word aligned
DST16b32b	1	FFT output samples are 32-bit word aligned
TRIGMODE	000b	Immediate trigger after second dimension.

Copyright © 2017, Texas Instruments Incorporated

Note that even though it is not explicitly listed in Table 9, the third-dimension FFT processing uses parameter sets 3, 4, and 5, as well as parameter sets 9, 10, and 11.

$I_{0, RX1}$	=	$Q_{0, RX1}$	=	$I_{0, RX2}$	=	$Q_{0, RX2}$	=
$I_{0, RX1}$	=	$Q_{0, RX1}$	=	$I_{0, RX2}$	=	$Q_{0, RX2}$	=
$I_{0, RX1}$	=	$Q_{0, RX1}$	=	$I_{0, RX2}$	=	$Q_{0, RX2}$	=
$I_{0, RX1}$	=	$Q_{0, RX1}$	=	$I_{0, RX2}$	=	$Q_{0, RX2}$	=
...		...		...		...	
...		...		...		...	
...		...		...		...	
...		...		...		...	
...		...		...		...	
...		...		...		...	
$I_{0, RX1}$	=	$Q_{0, RX1}$	=	$I_{0, RX2}$	=	$Q_{0, RX2}$	=
$I_{0, RX1}$	=	$Q_{0, RX1}$	=	$I_{0, RX2}$	=	$Q_{0, RX2}$	=
$I_{1, RX1}$	=	$Q_{1, RX1}$	=	$I_{1, RX2}$	=	$Q_{1, RX2}$	=
$I_{1, RX1}$	=	$Q_{1, RX1}$	=	$I_{1, RX2}$	=	$Q_{1, RX2}$	=
$I_{1, RX1}$	=	$Q_{1, RX1}$	=	$I_{1, RX2}$	=	$Q_{1, RX2}$	=
$I_{1, RX1}$	=	$Q_{1, RX1}$	=	$I_{1, RX2}$	=	$Q_{1, RX2}$	=
...		...		...		...	
...		...		...		...	
...		...		...		...	
...		...		...		...	
...		...		...		...	
...		...		...		...	
$I_{1, RX1}$	=	$Q_{1, RX1}$	=	$I_{1, RX2}$	=	$Q_{1, RX2}$	=
$I_{1, RX1}$	=	$Q_{1, RX1}$	=	$I_{1, RX2}$	=	$Q_{1, RX2}$	=
$I_{0, RX1}$	=	$Q_{0, RX1}$	=	$I_{0, RX2}$	=	$Q_{0, RX2}$	=
$I_{0, RX1}$	=	$Q_{0, RX1}$	=	$I_{0, RX2}$	=	$Q_{0, RX2}$	=
$I_{0, RX1}$	=	$Q_{0, RX1}$	=	$I_{0, RX2}$	=	$Q_{0, RX2}$	=
$I_{0, RX1}$	=	$Q_{0, RX1}$	=	$I_{0, RX2}$	=	$Q_{0, RX2}$	=
...		...		...		...	
...		...		...		...	
...		...		...		...	
...		...		...		...	
...		...		...		...	
...		...		...		...	
$I_{0, RX1}$	=	$Q_{0, RX1}$	=	$I_{0, RX2}$	=	$Q_{0, RX2}$	=
$I_{0, RX1}$	=	$Q_{0, RX1}$	=	$I_{0, RX2}$	=	$Q_{0, RX2}$	=

(1) 3 range bins at a time in the accelerator local memory – these are also stored as 32-bit wide I and Q samples

Figure 23. Layout of Zero-Padded, Third-Dimension, FFT Output Samples

Copyright © 2017, Texas Instruments Incorporated

6.1.4 Use Case Illustration – Log-Magnitude Processing

At the end of second and third-dimension FFT processing, the data can either be shipped back to the Radar Data Cube memory or it can be taken through log-magnitude and Cortex-R4F-based detection processing immediately. Alternately, CFAR-CA processing can be done in the accelerator using the CFAR Engine.

In this example, only the log-magnitude processing is assumed to be done in the accelerator and the CFAR (or some other proprietary) detection is assumed to be done in the Cortex-R4F processor. Since the Cortex-R4F must be involved for performing the detection processing anyway, TI recommends having the Cortex-R4F processor interrupted after the third-dimension FFT and log-magnitude processing is done (by setting CR4INTREN for the last parameter set).

The configuration for log-magnitude processing is very straightforward. The Log-Magnitude processing can simply take the input from ACCEL_MEM0 (or ACCEL_MEM1 for pong), compute the log-magnitude, and store the output as 16-bit unsigned numbers in ACCEL_MEM2 (or ACCEL_MEM3 for pong).

Table 10. Key Register Configurations for Log-Magnitude Processing

Parameter	Value	Comments
FFT_EN	0	Disable FFT computation
ABSEN	1	Enable Magnitude computation
LOG2EN	1	Enable Log2 computation
SRCACNT	511	512 samples (each 32-bit I, 32-bit Q) per range bin, where the 512 samples correspond to 64 chirps × 8 angle bins
SRCAINDX	8	Adjacent input samples spaced 8 bytes apart (see the third-dimension FFT output sample layout)
REG_BCNT	2	Three range bins to process
SRCBINDX	5120	Adjacent range bins are spaced 5KB apart. See layout picture.
SRCADDR	0	Source is ACCEL_MEM0 (or ACCEL_MEM1 – ping-pong not shown explicitly here). Three range bins separated by 5KB at input.
DSTADDR	32KB	Destination is ACCEL_MEM2 (or ACCEL_MEM3 – ping-pong not shown explicitly here). Log-magnitude results of three range bins separated by 1KB at output. Note that the original third-dimension FFT input samples get overwritten here with log-magnitude output.
DSTAINDX	2	Log-magnitude output is 16 bits (2 bytes) wide.
DSTBINDX	1024	Log-magnitude results for adjacent range bins are separated by 1KB.
SRC16b32b	1	Input samples are 32-bit word aligned.
DST16b32b	0	Log-magnitude output samples are 16-bit word aligned
DSTREAL	1	Log-magnitude output is real.
TRIGMODE	000b	Immediate trigger after third-dimension FFT.
CR4INTREN	1	Configures hardware accelerator to interrupt the Cortex-R4F processor after completion of log-magnitude calculations.

Note that in this approach, because the Cortex-R4F is being interrupted for performing detection processing at this stage, the ping-pong mechanism for the DMA transfer of second-dimension FFT input from the Radar Data Cube to the accelerator memories, and for the DMA transfer of third-dimension FFT and log-magnitude outputs to the Radar Data Memory, is not really required and therefore only six parameter sets (instead of 12) are required for second and third-dimension FFT processing for the three range bins. The seventh parameter set can be configured for log-magnitude processing, as given in Table 10.

After log-magnitude is computed, the Cortex-R4F must perform CFAR detection processing on the log-magnitude output and identify the cells which have detected objects. This may take up several microseconds for each range bin. Then, for each detected cell, the corresponding third-dimension FFT complex outputs are accessed by the Cortex-R4F from the input memories (ACCEL_MEM0 or ACCEL_MEM1) to estimate the angle of arrival.

Copyright © 2017, Texas Instruments Incorporated

A.1 Register Map

Figure 24 shows the layout of the parameter-set registers. The mmWave SDK includes header files that represent the registers of the Radar Hardware Accelerator, as well as a demo application.

Figure 24. Register Layout of Parameter-Set Registers

Copyright © 2017, Texas Instruments Incorporated

IMPORTANT NOTICE FOR TI DESIGN INFORMATION AND RESOURCES

Texas Instruments Incorporated ('TI") technical, application or other design advice, services or information, including, but not limited to, reference designs and materials relating to evaluation modules, (collectively, "TI Resources") are intended to assist designers who are developing applications that incorporate TI products; by downloading, accessing or using any particular TI Resource in any way, you (individually or, if you are acting on behalf of a company, your company) agree to use it solely for this purpose and subject to the terms of this Notice.

TI's provision of TI Resources does not expand or otherwise alter TI's applicable published warranties or warranty disclaimers for TI products, and no additional obligations or liabilities arise from TI providing such TI Resources. TI reserves the right to make corrections, enhancements, improvements and other changes to its TI Resources.

You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your applications and that you have full and exclusive responsibility to assure the safety of your applications and compliance of your applications (and of all TI products used in or for your applications) with all applicable regulations, laws and other applicable requirements. You represent that, with respect to your applications, you have all the necessary expertise to create and implement safeguards that (1) anticipate dangerous consequences of failures, (2) monitor failures and their consequences, and (3) lessen the likelihood of failures that might cause harm and take appropriate actions. You agree that prior to using or distributing any applications that include TI products, you will thoroughly test such applications and the functionality of such TI products as used in such applications. TI has not conducted any testing other than that specifically described in the published documentation for a particular TI Resource.

You are authorized to use, copy and modify any individual TI Resource only in connection with the development of applications that include the TI product(s) identified in such TI Resource. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER TI INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT OF TI OR ANY THIRD PARTY IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information regarding or referencing third-party products or services does not constitute a license to use such products or services, or a warranty or endorsement thereof. Use of TI Resources may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

TI RESOURCES ARE PROVIDED "AS IS" AND WITH ALL FAULTS. TI DISCLAIMS ALL OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING TI RESOURCES OR USE THEREOF, INCLUDING BUT NOT LIMITED TO ACCURACY OR COMPLETENESS, TITLE, ANY EPIDEMIC FAILURE WARRANTY AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

TI SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY CLAIM, INCLUDING BUT NOT LIMITED TO ANY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON ANY COMBINATION OF PRODUCTS EVEN IF DESCRIBED IN TI RESOURCES OR OTHERWISE. IN NO EVENT SHALL TI BE LIABLE FOR ANY ACTUAL, DIRECT, SPECIAL, COLLATERAL, INDIRECT, PUNITIVE, INCIDENTAL, CONSEQUENTIAL OR EXEMPLARY DAMAGES IN CONNECTION WITH OR ARISING OUT OF TI RESOURCES OR USE THEREOF, AND REGARDLESS OF WHETHER TI HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You agree to fully indemnify TI and its representatives against any damages, costs, losses, and/or liabilities arising out of your non-compliance with the terms and provisions of this Notice.

This Notice applies to TI Resources. Additional terms apply to the use and purchase of certain types of materials, TI products and services. These include; without limitation, TI's standard terms for semiconductor products http://www.ti.com/sc/docs/stdterms.htm), evaluation modules, and samples (http://www.ti.com/sc/docs/sampterms.htm).

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2017, Texas Instruments Incorporated