



# Calterah Alps Radar Baseband User Guide

*Release 1.0.2*

Calterah Semiconductor

Aug 18, 2021



# TABLE OF CONTENT

<b>1 Overview</b>	<b>11</b>
1.1 Baseband Features for Alps Family . . . . .	11
1.2 Introduction . . . . .	11
1.3 A Brief Introduction to FMCW Radar . . . . .	12
1.4 Architecture Overview . . . . .	14
1.4.1 Data Processing Flow . . . . .	14
1.4.2 Performance Benchmarks . . . . .	15
1.5 Advanced Features . . . . .	16
<b>2 Data Representation</b>	<b>17</b>
2.1 Fixed-Point Integer . . . . .	17
2.2 Fixed-Point Real Number . . . . .	18
2.3 Pseudo-Floating Real Number . . . . .	19
2.4 Complex Number . . . . .	20
2.4.1 Simple Complex Number . . . . .	20
2.4.2 Complex Number with Common Exponent . . . . .	20
2.5 Examples . . . . .	21
<b>3 Hardware Programming Interfaces</b>	<b>23</b>
3.1 Overview of Radar Baseband Process Unit (RBPU) . . . . .	23
3.1.1 Hardware Architecture . . . . .	23
3.2 Interface between Hardware and CPU . . . . .	27
3.2.1 Accessing Registers and Memories of RBPU . . . . .	27
3.2.1.1 Accessing Registers . . . . .	28
3.2.1.2 Accessing Memories . . . . .	28
3.2.1.3 Memory Refresh . . . . .	29
3.2.2 Bank Shifting . . . . .	29
3.2.2.1 Static Access . . . . .	29
3.2.2.2 Dynamic Control . . . . .	30
3.2.3 Interrupts . . . . .	30
3.2.4 Outputting Results . . . . .	31
3.3 Hardware Constraints . . . . .	32
3.4 SoC Address Mapping . . . . .	32
3.5 Software Suggestions . . . . .	32
3.6 Examples . . . . .	34
3.6.1 Configuration for Basic Radar Baseband Signal Processing . . . . .	34
3.6.1.1 Example 1 . . . . .	34
3.6.1.2 Example 2 . . . . .	35
<b>4 FMCW Waveform Generator</b>	<b>37</b>

<b>4.1</b>	<b>Overview . . . . .</b>	<b>37</b>
<b>4.2</b>	<b>Features . . . . .</b>	<b>38</b>
<b>4.3</b>	<b>Functional Description . . . . .</b>	<b>38</b>
4.3.1	Basic Chirp Cell in FMCW . . . . .	38
4.3.2	Virtual Array Mode (VAM) . . . . .	39
4.3.3	Frequency Hopping Mode (FH) . . . . .	40
4.3.4	Phase Scramble Mode (PS) . . . . .	41
4.3.5	Chirp Shifting Mode (CS) . . . . .	41
4.3.6	Anti Velocity Ambiguity with Chirp Delay (Anti-VELAMB CD) . . . . .	42
4.3.7	Auto Gain Control (AGC) . . . . .	43
4.3.8	Frame Interleaving . . . . .	43
4.3.9	Programmable Commands for Chirps . . . . .	43
<b>4.4</b>	<b>Programming Interfaces . . . . .</b>	<b>44</b>
4.4.1	How to Access FMCW Related Registers . . . . .	44
4.4.1.1	Example . . . . .	46
4.4.2	Basic Functions . . . . .	46
4.4.2.1	Generating Basic FMCW Frames . . . . .	46
4.4.2.2	Starting FMCW Frame . . . . .	47
4.4.3	Advanced Features . . . . .	50
4.4.3.1	Virtual Array Mode (VAM) . . . . .	50
4.4.3.2	Frequency Hopping Mode (FH) . . . . .	54
4.4.3.3	Phase Scramble Mode (PS) . . . . .	55
4.4.3.4	Chirp Shifting Mode (CS) . . . . .	56
4.4.3.5	Anti Velocity Ambiguity with Chirp Delay (Anti-VELAMB CD) . . . . .	56
4.4.3.6	Auto Gain Control (AGC) . . . . .	60
4.4.3.7	Frame Interleaving . . . . .	61
4.4.4	Coexistence among Features . . . . .	63
4.4.5	Limitation and Constraints . . . . .	64
<b>4.5</b>	<b>Examples . . . . .</b>	<b>65</b>
4.5.1	Example A: Generating Basic FMCW Waveform . . . . .	65
4.5.2	Example B: FMCW with VAM Enabled . . . . .	68
4.5.3	Example C: FMCW with FH and VAM Enabled . . . . .	71
4.5.4	Example D: FMCW with VAM, AGC, and CS Enabled . . . . .	74
<b>4.6</b>	<b>Software and Setting Suggestions . . . . .</b>	<b>77</b>
<b>4.7</b>	<b>Register Descriptions . . . . .</b>	<b>79</b>
4.7.1	Non-Banked Registers . . . . .	79
4.7.2	Banked Registers . . . . .	81
<b>5</b>	<b>Sample Preprocess</b>	<b>87</b>
<b>5.1</b>	<b>Overview . . . . .</b>	<b>87</b>
5.1.1	Features . . . . .	87
<b>5.2</b>	<b>Functional Description . . . . .</b>	<b>87</b>
<b>5.3</b>	<b>Programming Interfaces . . . . .</b>	<b>91</b>
5.3.1	Programming Chirp Information . . . . .	91
5.3.2	Programming Decimation Filter . . . . .	91
<b>5.4</b>	<b>Limitation and Constraints . . . . .</b>	<b>92</b>
<b>5.5</b>	<b>Software and Configuration Suggestions . . . . .</b>	<b>92</b>
5.5.1	Configuring Decimation Filter for Constant DC Gain . . . . .	92
5.5.2	Coexistence with DC Calibration . . . . .	95
<b>5.6</b>	<b>Examples . . . . .</b>	<b>95</b>
<b>6</b>	<b>Fast Fourier Transform (FFT)</b>	<b>97</b>
<b>6.1</b>	<b>Overview . . . . .</b>	<b>97</b>
6.1.1	Features . . . . .	97

6.2	Functional Description . . . . .	98
6.2.1	FFT Source Selection . . . . .	99
6.2.2	FFT Windows . . . . .	99
6.2.3	FFT Engine . . . . .	100
6.2.4	Zero Doppler Removal . . . . .	100
6.2.5	Phase Compensation . . . . .	101
6.3	Programming Interfaces . . . . .	101
6.3.1	FFT Source Selection . . . . .	101
6.3.1.1	CFG_SAM_SINKER . . . . .	101
6.3.1.2	CFG_SYS_ENABLE . . . . .	101
6.3.2	FFT Windows . . . . .	101
6.3.2.1	Window Size . . . . .	101
6.3.2.2	Window RAM . . . . .	102
6.3.2.3	No Windowing . . . . .	102
6.3.3	FFT Engine . . . . .	102
6.3.3.1	FFT Size . . . . .	102
6.3.3.2	FFT Stage Shifter . . . . .	103
6.3.4	Zero Doppler Removal . . . . .	103
6.3.5	Phase Compensation . . . . .	103
6.4	Limitation and Constraints . . . . .	103
6.4.1	FFT Source Selection . . . . .	103
6.4.2	FFT Windows . . . . .	104
6.4.3	FFT Size . . . . .	104
6.4.4	Phase Compensation . . . . .	104
6.5	Software and Config Suggestions . . . . .	104
6.5.1	Sizes of FFTs . . . . .	104
6.5.2	FFT Stage Shifters . . . . .	104
6.6	Examples . . . . .	105
6.6.1	Register Settings . . . . .	105
6.6.2	FFT Windows . . . . .	105
<b>7</b>	<b>Constant False Alarm Rate (CFAR) Detectors</b>	<b>107</b>
7.1	Overview . . . . .	107
7.1.1	Features . . . . .	107
7.2	Functional Description . . . . .	107
7.2.1	CFAR Combiner . . . . .	107
7.2.2	RDM Region Partition . . . . .	109
7.2.3	Sliding Windows . . . . .	110
7.2.3.1	Rectangular Window . . . . .	110
7.2.3.2	Cross Window . . . . .	111
7.2.3.3	Boundary Behavior . . . . .	112
7.2.4	Object Searching . . . . .	112
7.2.4.1	Peak Detector . . . . .	112
7.2.4.2	CA-CFAR . . . . .	112
7.2.4.3	OS-CFAR . . . . .	113
7.2.4.4	SOGO-CFAR . . . . .	114
7.2.4.5	NR-CFAR . . . . .	114
7.3	Programming Interfaces . . . . .	116
7.3.1	Combination Engines . . . . .	116
7.3.2	RDM Region Partition . . . . .	117
7.3.3	Sliding Window . . . . .	117
7.3.3.1	Rectangular Window . . . . .	117
7.3.3.2	Cross Window . . . . .	118
7.3.4	Object Searching . . . . .	119

7.3.4.1	Peak Detector . . . . .	119
7.3.4.2	CFAR Parameter Space Management . . . . .	119
7.3.4.3	CA-CFAR . . . . .	122
7.3.4.4	OS-CFAR . . . . .	122
7.3.4.5	SOGO-CFAR . . . . .	122
7.3.4.6	NR-CFAR . . . . .	123
7.4	Limitation and Constraints . . . . .	123
7.5	Software and Configuration Suggestions . . . . .	124
7.5.1	SISO vs MIMO Combiners . . . . .	124
7.5.2	Cross Window vs Rectangular Window . . . . .	124
7.5.3	Peak Detector . . . . .	124
7.5.4	CFAR Tuning . . . . .	124
7.5.5	Region Dependent CFAR Schemes . . . . .	125
7.6	Examples . . . . .	125
<b>8</b>	<b>Direction of Arrival (DoA) Estimators</b>	<b>127</b>
8.1	Overview . . . . .	127
8.1.1	Features . . . . .	127
8.2	Functional Description . . . . .	128
8.2.1	Digital Beamforming . . . . .	128
8.2.1.1	Overview . . . . .	128
8.2.1.2	DBF Power Spectrum Engine . . . . .	129
8.2.1.3	Object Decision Engine . . . . .	129
8.2.1.3.1	Orthogonal Match Pursuit Mode (OMP) . . . . .	130
8.2.1.3.2	Iterative Peak Mode (IPM) . . . . .	130
8.2.1.3.3	Non-Iterative Peak Mode (Non-IPM) . . . . .	131
8.2.1.3.4	Full Peak Mode (FPM) . . . . .	131
8.2.1.3.5	Raw-Fine Searching . . . . .	131
8.2.2	Deterministic Maximum Likelihood (DML) . . . . .	132
8.2.2.1	DML Algorithm . . . . .	132
8.2.2.2	Two-Step Searching . . . . .	132
8.2.2.3	Distribution of Candidate Angles . . . . .	133
8.2.2.4	Reflector Number Estimator (RNE) . . . . .	134
8.3	Programming Interfaces . . . . .	134
8.3.1	Control of Signal Vectors . . . . .	135
8.3.2	Programming Steering Vectors . . . . .	136
8.3.3	Digital Beamforming . . . . .	137
8.3.3.1	DBF Power Spectrum Engine . . . . .	137
8.3.3.2	Object Decision Engine . . . . .	137
8.3.3.2.1	Orthogonal Match Pursuit Mode (OMP) . . . . .	137
8.3.3.2.2	(Non-)Iterative Peak Mode . . . . .	138
8.3.3.2.3	Full Peak Mode (FPM) . . . . .	138
8.3.3.2.4	Raw-Fine Searching . . . . .	138
8.3.4	Deterministic Maximum Likelihood . . . . .	139
8.3.4.1	Two-Step Searching . . . . .	139
8.3.4.2	Distribution of Candidate Angles . . . . .	139
8.3.4.3	Reflector Number Estimator (RNE) . . . . .	140
8.3.5	Number of CFAR Outputs for DoA . . . . .	141
8.4	Limitation and Constraints . . . . .	142
8.4.1	Digital Beamforming . . . . .	142
8.4.2	Deterministic Maximum Likelihood . . . . .	142
8.5	Software and Configuration Suggestions . . . . .	143
8.5.1	DBF vs DML . . . . .	143
8.5.2	Steering Vectors . . . . .	143

8.5.3	Digital Beamforming . . . . .	144
8.5.3.1	Modes of Object Decision Engine . . . . .	144
8.5.3.2	Tuning Knobs . . . . .	145
8.5.4	Deterministic Maximum Likelihood . . . . .	145
8.5.4.1	Steering Vectors . . . . .	145
8.5.4.2	Configuration of $d$ . . . . .	145
8.5.4.3	Configuration of $k$ . . . . .	145
8.5.4.4	Configuration of <i>start</i> , <i>step</i> and <i>end</i> . . . . .	146
8.5.4.5	Reflector Number Estimator (RNE) . . . . .	147
8.6	Examples . . . . .	147
8.6.1	Digital Beamforming . . . . .	147
8.6.2	Deterministic Maximum Likelihood . . . . .	147
<b>9</b>	<b>Virtual Array (MIMO)</b>	<b>151</b>
9.1	Overview . . . . .	151
9.1.1	Background . . . . .	151
9.1.2	Features . . . . .	152
9.2	Functional Description . . . . .	153
9.2.1	TDM (Time-Division Modulation) MIMO . . . . .	153
9.2.2	BPM (Binary-Phase Modulation) MIMO . . . . .	154
9.2.3	Phase Compensation in Virtual Array . . . . .	155
9.2.4	2D DoA in Virtual Array . . . . .	157
9.2.4.1	Background . . . . .	157
9.2.4.2	Normal Mode . . . . .	158
9.2.4.3	Single Shot Mode . . . . .	159
9.2.4.4	Combined Mode . . . . .	160
9.3	Programming Interfaces . . . . .	162
9.3.1	Virtual Array . . . . .	162
9.3.2	2D DoA . . . . .	163
9.3.2.1	Digital Beamforming . . . . .	166
9.3.2.1.1	Normal Mode . . . . .	166
9.3.2.1.2	Single Shot Mode . . . . .	168
9.3.2.1.3	Combined Mode . . . . .	168
9.3.2.2	Deterministic Maximal Likelihood . . . . .	168
9.3.2.2.1	Normal Mode . . . . .	168
9.3.2.2.2	Single Shot Mode . . . . .	168
9.3.2.2.3	Combined Mode . . . . .	169
9.4	Limitation and Constraints . . . . .	169
9.4.1	Functional Constraints . . . . .	169
9.4.2	Memory Constraints . . . . .	170
9.4.3	Constraints for 2D DoA . . . . .	170
9.4.3.1	Using Digital Beamforming . . . . .	170
9.4.3.2	Using Deterministic Maximum Likelihood . . . . .	171
9.4.3.2.1	Memory Limitation . . . . .	171
9.4.3.2.2	Two-Step Searching . . . . .	172
9.5	Software and Configuration Suggestions . . . . .	172
9.5.1	Software and Configuration Flow . . . . .	172
9.5.2	Software Suggestions for DoA Related Registers . . . . .	173
9.5.3	Digital Beamforming . . . . .	173
9.5.4	Deterministic Maximal Likelihood . . . . .	173
9.6	Examples . . . . .	173
9.6.1	Configuring Radio Part . . . . .	175
9.6.2	Configuring Baseband for Virtual Array . . . . .	176
9.6.3	Configuring Baseband for 2D DoA . . . . .	176

9.6.4	Configuring Signal Vectors . . . . .	177
9.6.5	Configuring Steering Vectors . . . . .	177
9.6.6	Configuring for Digital Beamforming . . . . .	178
9.6.7	Deterministic Maximal Likelihood . . . . .	179
<b>10</b>	<b>Auto Gain Control (AGC)</b>	<b>185</b>
10.1	Overview . . . . .	185
10.1.1	Features . . . . .	185
10.2	Functional Description . . . . .	185
10.2.1	Basic AGC . . . . .	185
10.2.2	AGC Alignment . . . . .	188
10.2.3	ADC Compensation . . . . .	188
10.2.4	AGC IRQs . . . . .	188
10.3	Programming Interfaces . . . . .	189
10.3.1	Basic AGC . . . . .	189
10.3.1.1	Gain Code and Gain Value . . . . .	189
10.3.1.2	Glitch Protection . . . . .	190
10.3.1.3	Clip Tables . . . . .	190
10.3.1.4	Monitoring-Related AGC Registers . . . . .	191
10.3.1.5	IRQ-Related AGC Registers . . . . .	192
10.3.2	AGC Alignment . . . . .	193
10.3.3	ADC Compensation . . . . .	193
10.3.4	AGC IRQs . . . . .	193
10.3.5	AGC Coexistence with Virtual Array . . . . .	194
10.4	Limitation and Constraints . . . . .	194
10.5	Software and Configuration Suggestions . . . . .	194
10.6	Examples . . . . .	195
10.6.1	Configuring Radio Part . . . . .	195
10.6.2	Configuring Baseband Part . . . . .	196
10.6.2.1	Configuring Gain Code . . . . .	196
10.6.2.2	Configuring Glitch Protection . . . . .	197
10.6.2.3	Configure Initial Gain and First Two Clip Tables . . . . .	197
10.6.2.4	Setting Final Gain Table . . . . .	198
10.6.2.5	Setting IRQ-Related AGC . . . . .	199
10.6.2.6	Configuring AGC Alignment . . . . .	199
10.6.2.7	Configuring ADC Compensation . . . . .	200
10.6.3	Configuring Control Bit to Enable AGC . . . . .	200
<b>11</b>	<b>Interference Avoidance and Mitigation</b>	<b>201</b>
11.1	Overview . . . . .	201
11.2	Functional Description . . . . .	201
11.2.1	Interference Avoidance Modes . . . . .	201
11.2.1.1	XOR Chain . . . . .	201
11.2.1.2	Phase Scrambling . . . . .	202
11.2.1.3	Frequency Hopping . . . . .	203
11.2.1.4	Chirp Shifting . . . . .	204
11.2.1.5	Coexistence of Frequency Hopping and Chirp Shifting . . . . .	204
11.2.2	Interference Mitigation . . . . .	205
11.2.2.1	Interference Generation Analysis . . . . .	205
11.2.2.2	Mitigation of High Frequency Part in Interference Waveform . . . . .	207
11.2.2.2.1	Detecting HF Part of Interference . . . . .	207
11.2.2.2.2	Suppressing HF Part of Interference . . . . .	207
11.2.2.3	Mitigation of Low Frequency Part in Interference Waveform . . . . .	208
11.2.2.3.1	Detecting LF Part of Interference . . . . .	208

11.2.2.3.2	Suppressing LF Part of Interference . . . . .	208
11.3	Programming Interfaces . . . . .	208
11.3.1	Interference Avoidance Modes . . . . .	208
11.3.1.1	Enabling Interference Avoidance Modes . . . . .	208
11.3.1.2	Programming XOR Chain . . . . .	209
11.3.1.3	Programming Frequency Hopping and Chirp Shifting . . . . .	209
11.3.1.4	Programming Phase Scrambling . . . . .	209
11.3.2	Interference Mitigation . . . . .	210
11.4	Limitation and Constraints . . . . .	211
11.4.1	Interference Avoidance Modes . . . . .	211
11.4.2	Interference Mitigation . . . . .	211
11.5	Software and Configuration Suggestions . . . . .	211
11.5.1	Interference Avoidance Modes . . . . .	211
11.5.1.1	XOR chain . . . . .	211
11.5.1.2	Frequency Hopping . . . . .	211
11.5.1.3	Chirp Shifting . . . . .	212
11.5.2	Interference Mitigation . . . . .	212
11.6	Examples . . . . .	213
11.6.1	Interference Avoidance Modes . . . . .	213
11.6.2	Interference Mitigation . . . . .	215
<b>12</b>	<b>Frame Interleaving</b> . . . . .	<b>217</b>
12.1	Overview . . . . .	217
12.1.1	Features . . . . .	217
12.2	Functional Description . . . . .	217
12.2.1	Single Mode . . . . .	218
12.2.2	Rotate Mode . . . . .	218
12.3	Programming Interfaces . . . . .	218
12.3.1	Single Mode . . . . .	219
12.3.1.1	Baseband Configuration . . . . .	219
12.3.2	Rotate Mode . . . . .	219
12.3.2.1	Baseband Configuration . . . . .	219
12.3.3	Interaction Between CPU and Baseband . . . . .	219
12.4	Limitation and Constraints . . . . .	220
12.4.1	Registers for Rotate Mode . . . . .	220
12.4.2	RAMs for Rotate Mode . . . . .	220
12.4.2.1	Non-Banked RAMs . . . . .	220
12.4.2.2	Banked RAMs . . . . .	220
12.4.2.3	MEM_RLT . . . . .	221
12.4.2.4	RAMs with Address Pointers . . . . .	221
12.4.3	Aligning Baseband and Radio . . . . .	224
12.5	Software and Configuration Suggestions . . . . .	224
12.5.1	Initialization . . . . .	224
12.5.1.1	Initialization Flow . . . . .	224
12.5.1.2	Example . . . . .	225
12.5.1.3	Pseudo-Code for Rotate Mode Initialization . . . . .	226
12.5.1.4	Pseudo-Code for Single Mode Initialization . . . . .	226
12.5.2	Configuration Suggestions . . . . .	227
12.6	Examples . . . . .	227
12.6.1	MRR and SRR Settings . . . . .	227
12.6.1.1	MRR Settings in Bank 0 . . . . .	227
12.6.1.2	SRR Settings in Bank 1 . . . . .	228
12.6.2	Frame Interleaving Settings . . . . .	229
12.6.2.1	Baseband Switch . . . . .	229

12.6.2.2 Radio Switch . . . . .	229
<b>13 Anti Velocity Ambiguity (Anti-VELAMB) . . . . .</b>	<b>231</b>
13.1 Overview . . . . .	231
13.1.1 Features . . . . .	232
13.2 Functional Description . . . . .	232
13.2.1 Anti Velocity Ambiguity with Chirp Delay . . . . .	232
13.2.1.1 Number Of $q$ and Unambiguous Range of Doppler Frequency . . . . .	233
13.2.1.2 Estimating Ambiguity Factor $q$ . . . . .	234
13.2.1.3 Phase Compensation for Virtual Array . . . . .	234
13.2.1.4 High Speed Compensation for $p_{peak}$ . . . . .	235
13.2.1.5 Coexistence with Interference Avoidance and AGC . . . . .	235
13.2.2 Anti Velocity Ambiguity with Multi-Frame . . . . .	235
13.2.2.1 Mechanism . . . . .	235
13.3 Programming Interfaces . . . . .	237
13.3.1 Anti Velocity Ambiguity with Chirp Delay . . . . .	237
13.3.1.1 Enabling Anti-VELAMB CD . . . . .	237
13.3.1.2 Configuring Unambiguous Range of Doppler frequency . . . . .	237
13.3.1.3 Phase Compensation for Virtual Array . . . . .	238
13.3.1.4 Estimating Ambiguity Factor $q$ . . . . .	239
13.3.1.5 High Speed Compensation for $p_{peak}$ . . . . .	239
13.3.2 Anti Velocity Ambiguity with Multi-Frame . . . . .	239
13.4 Limitation and Constraints . . . . .	240
13.5 Software and Configuration Suggestions . . . . .	240
13.5.1 Anti Velocity Ambiguity with Chirp Delay . . . . .	240
13.5.2 Anti Velocity Ambiguity with Multi-Frame . . . . .	243
13.6 Examples . . . . .	244
13.6.1 Anti Velocity Ambiguity with Chirp Delay . . . . .	244
13.6.2 Anti Velocity Ambiguity with Multi-Frame . . . . .	246
13.6.2.1 Multi-Frame Interleaving Settings . . . . .	246
13.6.2.2 Example Matching Algorithm . . . . .	246
13.6.2.3 Solving Velocity Ambiguity . . . . .	251
<b>14 Noise Estimator . . . . .</b>	<b>253</b>
14.1 Features . . . . .	253
14.1.1 Functional Description . . . . .	253
14.1.2 Programming Interfaces . . . . .	254
14.1.3 Limitation and Constraints . . . . .	254
14.1.4 Software and Configuration Suggestions . . . . .	254
14.1.5 Examples . . . . .	255
<b>15 Data Collection and Hardware-in-the-Loop (HIL) . . . . .</b>	<b>257</b>
15.1 Overview . . . . .	257
15.1.1 Features . . . . .	257
15.2 Functional Description . . . . .	257
15.2.1 Standard Output Data Collection . . . . .	257
15.2.1.1 Data Collection Block Diagram . . . . .	258
15.2.1.2 SAMPLE Output Data . . . . .	258
15.2.1.2.1 SAMPLE Output Data Format . . . . .	258
15.2.1.2.2 SAMPLE Output Data Size . . . . .	259
15.2.1.2.3 SAMPLE Output Data Order . . . . .	260
15.2.1.3 1D-FFT and 2D-FFT Output Data . . . . .	262
15.2.1.3.1 FFT Output Data Format . . . . .	262
15.2.1.3.2 FFT Output Data Size . . . . .	262

15.2.1.3.3	FFT Output Data Order . . . . .	263
15.2.1.4	Clock Sequence . . . . .	263
15.2.2	Debug Data Collection . . . . .	264
15.2.2.1	Data Collection Block Diagram . . . . .	265
15.2.2.2	SAMPLE Debug Data . . . . .	265
15.2.2.2.1	Debug Data Format . . . . .	265
15.2.2.2.2	SAMPLE Debug Data Size . . . . .	265
15.2.2.2.3	SAMPLE Debug Data Order . . . . .	266
15.2.2.3	CFAR Debug Data . . . . .	267
15.2.2.3.1	CFAR Debug Data Format . . . . .	267
15.2.2.3.2	CFAR Debug Data Size . . . . .	267
15.2.2.3.3	CFAR Debug Data Order . . . . .	267
15.2.2.4	DoA Debug Data . . . . .	267
15.2.2.4.1	DoA Debug Data Format . . . . .	268
15.2.2.4.2	DoA Debug Data Size . . . . .	268
15.2.2.4.3	DoA Debug Data Order . . . . .	269
15.2.2.5	Clock Sequence . . . . .	269
15.2.3	HIL . . . . .	269
15.2.3.1	HIL Block Diagram . . . . .	269
15.2.3.2	HIL Input Data . . . . .	270
15.2.3.2.1	HIL Input Data Format . . . . .	270
15.2.3.2.2	HIL Input Data Size . . . . .	270
15.2.3.2.3	HIL Input Data Order . . . . .	270
15.2.3.3	Clock Sequence . . . . .	271
15.3	Programming Interfaces . . . . .	271
15.3.1	Baseband Registers . . . . .	271
15.3.1.1	Data Source Selection Settings . . . . .	272
15.3.1.2	Data Size Settings . . . . .	272
15.3.2	DMU Registers . . . . .	273
15.4	Limitation and Constraints . . . . .	273
15.4.1	Standard Output Data Size . . . . .	273
15.4.1.1	2MB Limited . . . . .	273
15.4.1.2	Memory Sharing . . . . .	274
15.4.1.3	Anti Velocity Ambiguity . . . . .	275
15.4.2	SAMPLE Debug Data Size . . . . .	275
15.4.3	CFG_SYS_ENABLE . . . . .	275
15.4.4	CFG_SAM_FORCE . . . . .	275
15.4.5	Streaming out . . . . .	276
15.5	Software and Configuration Suggestions . . . . .	276
15.5.1	Register Suggestions . . . . .	276
15.5.2	Standard Output Data Collection . . . . .	277
15.5.2.1	Baseband Configuration . . . . .	277
15.5.2.1.1	SAMPLE Output Data Collection . . . . .	277
15.5.2.1.2	1D-FFT Output Data Collection . . . . .	279
15.5.2.1.3	2D-FFT Output Data Collection . . . . .	281
15.5.2.2	DMU Configuration . . . . .	281
15.5.3	Debug Data Collection . . . . .	281
15.5.3.1	Baseband Configuration . . . . .	281
15.5.3.2	DMU Configuration . . . . .	281
15.5.4	HIL . . . . .	282
15.5.4.1	Baseband Configuration . . . . .	282
15.5.4.2	DMU Configuration . . . . .	284
15.6	Examples . . . . .	284

<b>16 Functional Safety</b>	<b>285</b>
16.1 Overview . . . . .	285
16.1.1 Features . . . . .	285
16.2 Functional Description . . . . .	286
16.2.1 Logic BIST . . . . .	286
16.2.2 Analog BIST with Shadow Bank . . . . .	286
16.2.3 ECC . . . . .	287
16.3 Programming Interfaces . . . . .	288
16.3.1 Logic BIST . . . . .	288
16.3.2 Analog BIST with Shadow Bank . . . . .	288
16.3.2.1 Baseband Registers . . . . .	289
16.3.2.2 Analog Registers . . . . .	289
16.3.3 ECC . . . . .	291
16.4 Limitation and Constraints . . . . .	291
16.4.1 Logic BIST . . . . .	291
16.4.2 Analog BIST with Shadow Bank . . . . .	291
16.4.2.1 CFG_SYS_SIZE_VEL_FFT . . . . .	291
16.4.2.2 CFG_SAM_FORCE . . . . .	292
16.4.3 ECC . . . . .	292
16.4.3.1 Single-Bit Error to IRQ . . . . .	292
16.4.3.2 Double-Bit Error to EMU . . . . .	292
16.5 Examples . . . . .	292
16.5.1 Baseband Settings . . . . .	293
16.5.2 Firmware Flow . . . . .	293
<b>17 Cascade Mode</b>	<b>295</b>
17.1 Overview . . . . .	295
17.1.1 Features . . . . .	296
17.2 Functional Description . . . . .	296
17.2.1 Synchronization . . . . .	297
17.2.2 Data Exchange and Process Flow . . . . .	298
17.3 Limitation and Constraints . . . . .	298
17.4 Software and Config Suggestions . . . . .	299
<b>A Memory Mapping Table</b>	<b>301</b>
<b>B Baseband Register Tables</b>	<b>303</b>
<b>C Baseband Shadow Bank (Bank 5) Registers</b>	<b>333</b>
<b>D Revision History</b>	<b>335</b>
<b>E Important Notice and Disclaimer</b>	<b>337</b>
<b>Bibliography</b>	<b>339</b>
<b>Index</b>	<b>341</b>

## LIST OF FIGURES

1.1	View of Linear Chirp Signal over Time . . . . .	12
1.2	Multiple FMCW Chirps Form an FMCW Frame . . . . .	13
1.3	FMCW Chirps with Their Reflection . . . . .	13
1.4	Direction of Arrival Estimation with Antenna Array . . . . .	14
1.5	Signal/Data Processing Flow of Alps Radar Baseband Processing Unit . . . . .	15
2.1	Bit Arrangement of Pseudo-Floating Real Number . . . . .	19
2.2	Underlying Structure of Complex Number . . . . .	20
2.3	Underlying Structure of Complex Number with Common Exponent . . . . .	20
3.1	Baseband Block Diagram and Data Flow . . . . .	23
3.2	Finite State Machine of Baseband Core . . . . .	24
3.3	Signal and Control Flow between FMCW Generator and Baseband Core . . . . .	27
3.4	Memory Refresh . . . . .	29
3.5	Baseband Register and Memory Bank Shift . . . . .	30
3.6	Memory Layout of Result Output . . . . .	31
3.7	Baseband Process Flow When CFG_SAM_SINKER is Set to 0 . . . . .	34
3.8	Baseband Process Flow When CFG_SAM_SINKER is Set to 1 . . . . .	35
4.1	Structure of FMCW Waveform Generator . . . . .	37
4.2	A Basic FMCW Chirp Cell . . . . .	39
4.3	Time Division Modulation . . . . .	39
4.4	Binary Phase Modulation . . . . .	40
4.5	Frequency Hopping Mode . . . . .	40
4.6	Phase Scramble Mode . . . . .	41
4.7	Chirp Shifting Mode . . . . .	42
4.8	Anti Velocity Ambiguity with Chirp Delay Mode . . . . .	42
4.9	AGC Mode . . . . .	43
4.10	Frame Interleaving Mode . . . . .	43
4.11	Fast Settling during FMCW . . . . .	44
4.12	Topology between radio registers and CPU . . . . .	45
4.13	Layout for Radio Registers . . . . .	45
4.14	Data Format for Accessing Radio Registers . . . . .	45
4.15	Data Format for Returned Values of Radio Registers . . . . .	46
4.16	A Basic FMCW Frame . . . . .	47
4.17	FMCW Start Source . . . . .	47
4.18	FMCW Continuous Waveform . . . . .	48
4.19	FMCW Waveform Generated by Frame . . . . .	49
4.20	Single Tone Waveform . . . . .	50
4.21	TX Status Controlling Groups in VAM . . . . .	51

## List of Figures

---

4.22 Controlling Group Sequence When <i>period</i> = 1 in VAM . . . . .	52
4.23 Controlling Group Sequence When <i>period</i> = 2 in VAM . . . . .	52
4.24 Controlling Group Sequence When <i>period</i> = 3 in VAM . . . . .	53
4.25 Controlling Group Sequence <i>period</i> = 4 in VAM . . . . .	53
4.26 32-Bit XOR Chain . . . . .	54
4.27 Both Anti-VELAMB CD and VAM Are Enabled, When VAM Period <i>P</i> = 1 . . . . .	57
4.28 Differences between VAM on and off Under Anti-VELAMB CD . . . . .	57
4.29 Both Anti-VELAMB CD and VAM Are Enabled, When VAM Period <i>P</i> = 2 . . . . .	58
4.30 Both Anti-VELAMB CD and VAM Are Enabled, When VAM Period <i>P</i> = 3 . . . . .	58
4.31 Both Anti-VELAMB CD and VAM Are Enabled, When VAM Period <i>P</i> = 4 . . . . .	59
4.32 RXBB Gain Control in AGC . . . . .	61
4.33 Frame Interleaving Controlling Vector . . . . .	62
4.34 Frame Limitation for CS . . . . .	64
4.35 FMCW Virtual Array Mode with Period = 3 . . . . .	69
4.36 FMCW Frequency Hopping Mode . . . . .	73
4.37 FMCW with Frequency Hopping and VAM Enabled . . . . .	74
4.38 FMCW Chirp Shifting Mode . . . . .	75
4.39 FMCW Chirp Shifting Mode with Delay of 5 us . . . . .	76
4.40 FMCW with VAM, AGC, and CS Enabled . . . . .	77
4.41 Structure between ADC and Dual Port Memory . . . . .	77
4.42 Relationship between Counter Length and Chirp Length . . . . .	78
 5.1 Data Storage Format in MEM_SAM . . . . .	88
5.2 Functional Diagram of the Sampling Module . . . . .	88
5.3 Timing of 4-Channel ADC Data Transmission and Chirp Synchronization . . . . .	89
5.4 Dumping Sampling Output to GPIO before Fed into 2D-FFT Module . . . . .	89
5.5 Parameters of Chirp and Its Effective Sampling Positions . . . . .	89
5.6 Down-Sampling Structure . . . . .	90
5.7 Second-Order Section of Chebyshev Type II Filter . . . . .	90
5.8 Post-Process at Output of Last (Fourth) Second-Order Section . . . . .	90
5.9 Data Flow in Decimation Filter . . . . .	92
5.10 IIR Filter Amplitude and Phase Response for Decimation Factor = 2 . . . . .	93
5.11 IIR Filter Amplitude and Phase Response for Decimation Factor = 4 . . . . .	93
5.12 IIR Filter Amplitude and Phase Response for Decimation Factor = 8 . . . . .	94
 6.1 Diagram of FFT Computing Unit . . . . .	98
6.2 A Diagram of Radix-2 Butterfly Used in FFT Engine . . . . .	100
6.3 Memory Table Layout of FFT Windows . . . . .	102
 7.1 Diagram of SISO Combination . . . . .	108
7.2 MIMO Combination . . . . .	108
7.3 Region Partition in RDM . . . . .	109
7.4 Decimation of Rectangular Window Mask . . . . .	111
7.5 Example Cross Window . . . . .	111
7.6 Diagram of CA-CFAR . . . . .	113
7.7 Diagram of OS-CFAR . . . . .	113
7.8 Diagram of SOGO-CFAR . . . . .	114
7.9 Diagram of NR-CFAR . . . . .	115
7.10 SISO Combination Engine Diagram . . . . .	116
7.11 MIMO Combination Engine Diagram . . . . .	116
7.12 Bit Mask Packing in Register CFG_CFR_MSK_DS_i [0–3] . . . . .	118
7.13 Bit Map of CFG_CFR_MSK_PK_0i . . . . .	119
7.14 Parameter 0: Bit Map of CFG_CFR_PRM_0_0 and CFG_CFR_PRM_0_1 . . . . .	120
7.15 Parameter 1: Bit Map of CFG_CFR_PRM_1_0, CFG_CFR_PRM_1_1, and CFG_CFR_PRM_1_2 . . . . .	120



7.16 Parameter 2: Bit Map of CFG_CFR_PRM_2_0, CFG_CFR_PRM_2_1, and CFG_CFR_PRM_2_2 . . . . .	120
7.17 Parameter 3: Bit Map of CFG_CFR_PRM_3_0, CFG_CFR_PRM_3_1, CFG_CFR_PRM_3_2, and CFG_CFR_PRM_3_3 . . . . .	121
7.18 Parameter 4: Bit Map of CFG_CFR_PRM_4_0 and CFG_CFR_PRM_4_1 . . . . .	121
7.19 Parameter 5: Bit Map of CFG_CFR_PRM_5_0 and CFG_CFR_PRM_5_1 . . . . .	121
7.20 Parameter 6: Bit Map of CFG_CFR_PRM_6_0, CFG_CFR_PRM_6_1, CFG_CFR_PRM_6_2, and CFG_CFR_PRM_6_3 . . . . .	121
8.1 Diagram of DBF Engine for Estimation of DoA . . . . .	128
8.2 Diagram of DBF Power Spectrum Engine for Estimation of DoA . . . . .	129
8.3 DML Engine Diagram . . . . .	134
8.4 Signal Vector Mapping for Rx channels with Virtual Array. . . . .	135
8.5 Memory Layout for Steering Vector Programming . . . . .	136
8.6 Memory Layout of $d$ and $k$ . . . . .	140
8.7 Illustration of Reflected Wave and Its Position Related to Rx Channels . . . . .	143
8.8 DBF Power Spectrum with Two Objects Closed to Each Other . . . . .	144
8.9 K Factors $g(i)$ . . . . .	146
9.1 1T4R Radar . . . . .	151
9.2 2T4R MIMO Radar . . . . .	152
9.3 1T8R Radar . . . . .	152
9.4 TDM Virtual Array Scheme with 2 TX Channels . . . . .	153
9.5 TDM Virtual Array Scheme with 3 TX Channels . . . . .	153
9.6 TDM Virtual Array Scheme with 4 TX Channels . . . . .	154
9.7 BPM Virtual Array Scheme with 2 TX Channels . . . . .	154
9.8 BPM Virtual Array Scheme with 4 TX Channels . . . . .	155
9.9 Moving Target Causing Extra Phase Differences in MIMO . . . . .	156
9.10 Example of Coordinate System . . . . .	157
9.11 Planar Array with Virtual Array . . . . .	158
9.12 Example of Selecting 3 Different Groups for 2D-DoA with Normal Mode . . . . .	158
9.13 Flow Chart of 2D DoA with Normal Mode . . . . .	159
9.14 Flow Chart of 2D DoA with Single Shot Mode . . . . .	160
9.15 Main Processing Flow of 2D DoA with Combined Mode . . . . .	161
9.16 Example of Selecting Groups of Channels for 2D-DoA with Combined Mode . . . . .	162
9.17 Example of Planar Array with Virtual Array . . . . .	174
9.18 Example of Memory Layout of Coefficient Memory . . . . .	175
9.19 Example of Memory Layout of DBPM Coefficients . . . . .	176
10.1 FMCW Frequency with AGC Mode On . . . . .	185
10.2 RX Chain Block and Its Saturation Detectors . . . . .	186
10.3 Detailed Mechanisms of AGC . . . . .	186
10.4 HW Structure of Final RX Gain . . . . .	187
10.5 Hardware Structure for ADC Compensation Mode . . . . .	188
11.1 XOR Chain Used to Generate Random Sequence . . . . .	202
11.2 Phase Scrambling Frame . . . . .	202
11.3 Frequency Hopping & Chirp Shifting . . . . .	203
11.4 Frequency Hopping & Chirp Shifting Compensation Procedure . . . . .	205
11.5 Mechanism of Mutual Interference . . . . .	205
11.6 Waveform of Chirp Samples with Interference . . . . .	206
11.7 Waveform of Spike Interference . . . . .	206
11.8 Interference Compositions and On-Line Suppression Process . . . . .	206
11.9 Enabling Interference Mitigation . . . . .	210
11.10 An Example of Interference Mitigation . . . . .	216



## List of Figures

---

12.1 Frame Interleaving Example of Two Banks . . . . .	217
12.2 Single Mode Using Bank0 . . . . .	218
12.3 Rotate Mode Using Bank0 and Bank1 . . . . .	218
12.4 Memory Space in MEM_COE . . . . .	221
12.5 Baseband Initialization Flow for Frame Interleaving . . . . .	225
13.1 Conventional Chirp Sequence w/o Virtual Array . . . . .	232
13.2 Anti Velocity Ambiguity Chirp Sequence w/o Virtual Array . . . . .	232
13.3 Anti Velocity Ambiguity Chirp Sequence With Virtual Array . . . . .	233
13.4 Mechanism of Solving Velocity Ambiguity by Two frames with Different Chirp Periods $T_c$ . . . . .	236
13.5 Typical Processing Flow of Alps Chip for Solving Velocity Ambiguity in Frame Interleaving Mode . . . . .	237
13.6 Memory Layout of Anchor Phases . . . . .	239
13.7 Coherent Configuration: $va=2 / Tr=30 / a=18 / N_q = 9$ . . . . .	241
13.8 Incoherent Configuration: $va=2 / Tr=30 / a=11 / N_q = 9$ . . . . .	242
13.9 Candidate Target Chain for a Target in Frame A . . . . .	247
13.10 Candidate Target Chains for All Targets in Both Frames . . . . .	248
13.11 Program Flow Diagram of Function pairA() . . . . .	249
13.12 Program Flow Diagram of Function pairB() . . . . .	250
13.13 Remove One Candidate Target from $A_m$ 's Candidate Targets Chain . . . . .	250
13.14 Program Flow Diagram of Matching Process . . . . .	251
14.1 Noise Estimator Diagram . . . . .	253
14.2 Overshooting of Noise Estimator . . . . .	255
15.1 Data Collection of Standard Output Flow . . . . .	258
15.2 SAMPLE Output Data Collection . . . . .	260
15.3 MEM_BUFS Output Data Order . . . . .	261
15.4 Clock Sequence of GPIO Data . . . . .	263
15.5 Clock Sequence of LVDS Data . . . . .	264
15.6 Flow of Debug Data Collection . . . . .	265
15.7 SAMPLE Debug Data Order . . . . .	266
15.8 HIL Data Flow . . . . .	269
15.9 Clock Sequence of HIL from CPU AHB . . . . .	271
15.10 Clock Sequence of HIL from GPIO . . . . .	271
15.11 Data Collection of Standard Output Data when Buffering SAMPLE . . . . .	278
15.12 Data Collection of Standard Output Data Without Buffering SAMPLE . . . . .	280
15.13 HIL State Flow . . . . .	283
16.1 Diagram of Logic BIST . . . . .	286
16.2 Analog BIST Process Flow . . . . .	287
16.3 Baseband Logic BIST Running Flow . . . . .	291
17.1 Cascade System with Two Alps CAL77S244-AE ICs . . . . .	295
17.2 High-level Block Diagram of a Cascade System . . . . .	296
17.3 Hardware Synchronization Diagram . . . . .	297
17.4 Data Exchange and Process Diagram . . . . .	298



## LIST OF TABLES

1.1	Summary of Baseband Features for Alps Family . . . . .	11
1.2	Performance Benchmark of a Typical Scenario . . . . .	15
1.3	Performance Benchmark of a Tough Scenario . . . . .	16
2.1	Example of FXR( $W, I, SU$ ) . . . . .	18
2.2	Interpretation of FLR( $W_m, I, SU_m, E, SU_e$ ) . . . . .	19
2.3	Examples of Interpretation of Data Type . . . . .	21
3.1	Descriptions of Macros for FSM . . . . .	25
3.2	Bit mask of CFG_SYS_ENABLE . . . . .	26
3.3	Memory Mapping . . . . .	28
3.4	Banked Memories . . . . .	30
3.5	Registers Settings of Example 1 . . . . .	35
3.6	Registers Settings of Example 2 . . . . .	36
4.1	Chirp Parameters . . . . .	38
4.2	Parameters for a Basic FMCW Frame . . . . .	46
4.3	Configuration for Generating a Continuous Waveform . . . . .	48
4.4	Configuration for Generating Waveform by Frame . . . . .	49
4.5	Configuration for Generating a Single-Tone Waveform . . . . .	50
4.6	FMCW Related Registers for VAM . . . . .	51
4.7	Registers for FH . . . . .	55
4.8	Registers for Phase Scramble Mode . . . . .	55
4.9	Registers for Chirp Shifting Mode . . . . .	56
4.10	Registers for Anti-VELAMB CD . . . . .	60
4.11	Register for AGC . . . . .	61
4.12	Frame Interleaving Controlling Register . . . . .	62
4.13	Registers for Frame Interleaving . . . . .	62
4.14	Register Settings for Frame Interleaving . . . . .	63
4.15	Final Transmitted Phased . . . . .	63
4.16	Example Values of FMCW Parameters . . . . .	65
4.17	FMCW Decimal Registers . . . . .	65
4.18	Register Setting for FMCW Start Frequency . . . . .	66
4.19	Register Setting for FMCW Stop Frequency . . . . .	66
4.20	Register Setting for FMCW Step up Frequency . . . . .	66
4.21	Register Setting for FMCW Step down Frequency . . . . .	67
4.22	Register Setting for FMCW Idle Time . . . . .	67
4.23	Register Setting for FMCW Chirp Number . . . . .	67
4.24	Register Setting for FMCW Basic Mode Selection . . . . .	68
4.25	Register Settings for FMCW Start . . . . .	68

## List of Tables

---

4.26 Register Settings for FMCW Basic Parameters . . . . .	70
4.27 Register Settings of VAM for TX0 . . . . .	70
4.28 Register Settings of VAM for TX1 . . . . .	70
4.29 Register Settings of VAM for TX2 . . . . .	71
4.30 Register Settings of VAM for TX3 . . . . .	71
4.31 Register Settings for FMCW Basic Parameters and VAM . . . . .	71
4.32 The Second Set of FMCW Parameters . . . . .	72
4.33 Register Settings for The Second Set of FMCW Parameters . . . . .	72
4.34 Register Settings for Frequency Hopping Mode . . . . .	73
4.35 FMCW Chirp Shifting Parameter . . . . .	75
4.36 FMCW Chirp Shifting Parameter Definition . . . . .	75
4.37 Register Settings for FMCW Chirp Shifting Delay . . . . .	75
4.38 Register Settings for CS Mode . . . . .	76
4.39 Register Setting for AGC Enable . . . . .	76
4.40 Parameters Definition . . . . .	78
4.41 FMCW Non-Banked Registers . . . . .	79
4.42 FMCW Banked Registers . . . . .	81
5.1 Example of Chirp Parameters and Baseband Setup . . . . .	91
5.2 Example Settings of Decimation Filter Coefficients with the Same DC Gain . . . . .	94
5.3 Example Settings of Sampling Related Registers . . . . .	96
6.1 Supported FFT Sizes . . . . .	100
6.2 FFT Configuration Suggestions . . . . .	105
7.1 Region Decision Table of Bin with Index $(k, p)$ . . . . .	110
7.2 Cross Window Parameters and Its Corresponding Registers . . . . .	118
7.3 Parameters for CA-CFAR . . . . .	122
7.4 Parameters for OS-CFAR . . . . .	122
7.5 Parameters for SOGO-CFAR . . . . .	122
7.6 Parameters for NR-CFAR . . . . .	123
7.7 Example Settings for CA-CFAR . . . . .	125
7.8 Example Settings for SOGO-CFAR . . . . .	125
8.1 Metric Function Description . . . . .	133
8.2 DBF Mode Control . . . . .	137
8.3 Variables in Inequality (8.7) . . . . .	141
8.4 Coefficient Configuration of RNE . . . . .	147
8.5 Table of Example with DBF . . . . .	147
8.6 Baseband Settings to Enable MIMO . . . . .	148
8.7 Baseband Settings to Activate Normal Mode . . . . .	148
8.8 Signal Vector Settings . . . . .	148
8.9 Steering Vectors of Settings for Combined Mode . . . . .	148
8.10 DML Register Settings for Combined Mode . . . . .	149
9.1 Example of Selecting 3 Different Groups for 2D-DoA with Normal Mode . . . . .	159
9.2 Settings for 2D-DoA with Combined Mode . . . . .	162
9.3 Programming Interfaces for virtual array . . . . .	163
9.4 Programming Interfaces for 2D DoA . . . . .	163
9.5 Programming Interfaces for 2D DoA . . . . .	164
9.6 Programming Interfaces for 2D DoA Signal Vectors . . . . .	164
9.7 Bit Mapping of CFG_DoA_GRP0_DAT_IDX_t . . . . .	164
9.8 Programming Interfaces for Steering Vectors of 2D DoA . . . . .	165
9.9 Programming Interfaces for 2D DoA . . . . .	166
9.10 Programming Interfaces for 2D DoA Using DBF in Normal Mode . . . . .	167

---



9.11 Programming Interfaces for Normal Mode Using DML . . . . .	168
9.12 Programming Interfaces for Combined Mode Using DML . . . . .	169
9.13 Descriptions of Parameters . . . . .	170
9.14 Nick Name for Values in Registers . . . . .	171
9.15 Example of Channel Selection for 2D-DoA Using Combined mode . . . . .	174
9.16 Radio Settings to Enable BPM MIMO . . . . .	175
9.17 Baseband Settings to Enable BPM MIMO . . . . .	176
9.18 Example for 2D DoA in Normal Mode . . . . .	176
9.19 Example for 2D DoA in Combined Mode . . . . .	176
9.20 Example Settings for Signal Vectors of 2D DoA in Combined Mode . . . . .	177
9.21 Example Settings for Steering Vectors of 2D DoA in combined mode . . . . .	178
9.22 Example Settings for Combined Mode Using DBF . . . . .	179
9.23 Baseband Settings to Enable MIMO . . . . .	180
9.24 Baseband Settings for 2D DoA in Combined Mode . . . . .	180
9.25 Signal Vector Settings . . . . .	180
9.26 Steering Vector Settings for Combined Mode . . . . .	181
9.27 DML Register Settings for Combined Mode . . . . .	181
10.1 Gain Settings for First Chirp . . . . .	186
10.2 Gain Settings for Second Chirp . . . . .	187
10.3 Gain Code Mapping . . . . .	189
10.4 Programming Interfaces for Gain Code of LNA and TIA . . . . .	189
10.5 Programming Interfaces for Saturation Detection . . . . .	190
10.6 Bit Mapping of 24 Bits of Saturation Registers . . . . .	190
10.7 Programming Interfaces for Init gain and First Two Clip Tables . . . . .	190
10.8 Bit Mapping of 17 Bits for LUTs and Initial Gain Code . . . . .	190
10.9 Programming Interfaces for Choosing Final Gain . . . . .	191
10.10 Interfaces for Storing Final AGC Status . . . . .	191
10.11 Programming Interfaces for Setting AGC IRQ . . . . .	192
10.12 Bit mapping rule of register CFG_AGC_IRQ_ENA . . . . .	192
10.13 Bit mapping rule of register CTL_AGC_IRQ_CLR and FDB_AGC_IRQ_STATUS . . . . .	193
10.14 Programming Interface for AGC Alignment . . . . .	193
10.15 Programming Interfaces for ADC Compensation . . . . .	193
10.16 Radio Gain Setting . . . . .	195
10.17 Radio & RF BB Settings to Enable AGC . . . . .	196
10.18 AGC LNA Code and TIA Code . . . . .	196
10.19 Thresholds to Indicate Saturation and Maximal ADC Data Selection . . . . .	197
10.20 AGC Initial Gain Code and Clip1/Clip2 Code . . . . .	198
10.21 Set Minimum Input Power and Target dB . . . . .	199
10.22 Setting IRQ-Related AGC . . . . .	199
10.23 Enabling AGC Alignment . . . . .	199
10.24 Enabling ADC Compensation . . . . .	200
10.25 Enabling AGC Mode . . . . .	200
11.1 Parameter Descriptions . . . . .	204
11.2 CFG_FFT_DINT_ENA description . . . . .	208
11.3 Registers of XOR chain . . . . .	209
11.4 Register Settings for Phase Compensation in FH & CS . . . . .	209
11.5 Phase Compensation for PS . . . . .	210
11.6 Interference Detection Sensitivity Adjusting Principle . . . . .	212
11.7 Mitigation Performances for Different Types of Interference . . . . .	212
11.8 Initial System Configuration . . . . .	213
11.9 Register Settings for XOR Chains . . . . .	213
11.10 Phase Compensation for FH & CS . . . . .	214



## List of Tables

---

11.11 Phase Compensation for PS . . . . .	214
11.12 Example Settings for Interference Mitigation . . . . .	215
12.1 MEM_COE Size . . . . .	223
12.2 MEM_COE Address Pointers . . . . .	224
12.3 MRR Settings . . . . .	228
12.4 SRR Settings . . . . .	228
12.5 baseband switch in frame interleaving . . . . .	229
12.6 FMCW Frame Interleaving Registers Settings . . . . .	229
13.1 Parameter Descriptions . . . . .	235
13.2 CFG_DoA_DAMB_TYP Configuration . . . . .	238
13.3 Anti-VELAMB CD Configuration for $va = 2$ . . . . .	241
13.4 Pre-Configuration for Anti-VELAMB CD . . . . .	244
13.5 Impact of $a$ and Other Parameters . . . . .	244
14.1 Example Settings for DBF . . . . .	255
15.1 Standard Output Data Format . . . . .	258
15.2 Registers for Data Collection of SAMPLE Output Data . . . . .	259
15.3 Data Size of SAMPLE Output Data . . . . .	259
15.4 SAMPLE Standard Output Data Size Example . . . . .	259
15.5 SAMPLE Standard Output Data in MEM_BUF . . . . .	260
15.6 FFT Standard Output Data Format . . . . .	262
15.7 Registers for Data Collection of FFT Output Data . . . . .	262
15.8 Data Size of FFT Output Data . . . . .	262
15.9 FFT Data Size Example . . . . .	262
15.10 Debug Data Formats . . . . .	265
15.11 Registers for Data Collection of SAMPLE Debug Data . . . . .	265
15.12 Data Size of SAMPLE Debug Data . . . . .	266
15.13 SAMPLE Debug Data Size Example . . . . .	266
15.14 CFAR Debug Data Format . . . . .	267
15.15 Registers for Data Collection of CFAR Debug Data . . . . .	267
15.16 Data Size of CFAR Debug Data . . . . .	267
15.17 CFAR Debug Data Size Example . . . . .	267
15.18 DoA Debug Data Format . . . . .	268
15.19 Registers for Data Collection of DoA Debug Data . . . . .	268
15.20 Data Size of DoA Debug Data . . . . .	268
15.21 DoA Debug Data Size Example . . . . .	268
15.22 HIL Input Data Format . . . . .	270
15.23 Registers for HIL Input Data . . . . .	270
15.24 HIL Input Data Size Example . . . . .	270
15.25 Data Size Settings . . . . .	273
15.26 Data Size Example . . . . .	274
15.27 1MB Memory Sharing . . . . .	274
15.28 Anti Velocity Ambiguity . . . . .	275
15.29 Data Size Settings . . . . .	275
15.30 Sample Force Start . . . . .	275
15.31 Configuration Suggestions of Data Collection and HIL . . . . .	276
15.32 SAMPLE Standard Output Data Collection Example . . . . .	284
16.1 ECC and Memory . . . . .	288
16.2 Register Settings for Analog BIST . . . . .	290
16.3 Example Settings for Analog BIST . . . . .	293



A.1	Memory Mapping . . . . .	301
B.1	Baseband Register Table . . . . .	303
C.1	Registers in Shadow Bank (Bank 5) . . . . .	333
D.1	Revision History . . . . .	335



## **List of Tables**

---



---

**CHAPTER  
ONE**

---

**OVERVIEW**

Calterah Alps radar sensor chip integrates not only a 77/79GHz FMCW transceiver but also a radar baseband processing unit (RBPU), which enables all kinds of radar applications. This book is a user guide specially developed for the usage of the baseband processor. This chapter gives a summary of baseband features for the products in Alps Family and an overview of the processing unit, which, in particular, includes a general introduction, a brief review of FMCW radar system, and special built-in features designed for the processor.

## 1.1 Baseband Features for Alps Family

Table 1.1: Summary of Baseband Features for Alps Family

Feature	<b>CAL77S224-AE</b>	<b>CAL77S244-AE</b>	<b>CAL77S244-AB</b>	<b>CAL77S244-IB</b>
Baseband	Virtual array frame: time-division modulation (TDM) and binary-phase modulation (BPM)			
	Digital decimation: up to factor of 16			
	Configurable FFT sizes: up to 2048			
	Configurable CFAR type: cell averaging (CA)/ordered statistics (OS)/smallest order and greatest order (SOGO)/noise reference (NR)			
	Configurable CFAR window size: up to 21×21 (rectangular window)			
	Output objects: up to 1024 objects			
	Multi-object DoA: up to 4 objects per bin			
	Super resolution support: deterministic maximum likelihood (DML)			
	Frame interleaving			
	Memory: 2 MB			
	Support for functional safety (ASIL-B capable)			
	NA	Support for cascade mode	NA	NA

## 1.2 Introduction

Usually, a radar microcontroller unit (MCU) integrates a customized digital signal processor (DSP) to accelerate several common algorithms, such as FFT, CFAR, and so on. This approach shortens the processing time while maintaining high-level flexibility to design and customize special radar signal processing algorithms for special applications. However, its disadvantages are also obvious:

- It usually takes developers great effort to develop radar algorithms, since they need to fit their algorithms into particular designs of the DSP and other hardware implementation. This includes familiarizing themselves with special calling rules for a particular DSP, understanding the interactions between the DSP and the CPU, and going back-and-forth in a lengthy reference manual (usually over 2,000 pages).

### 1.3. A Brief Introduction to FMCW Radar

---

- The given flexibilities usually trade off some other system performances, such as power consumption, underlying hardware capabilities, and so on. For example, to keep both computation performance and flexibility at a certain level, a DSP is usually running at a very high frequency, which consumes greater power. Also, developers often spend great effort tuning their code in order to have an “efficient” way to access certain memory. In other words, A DSP might have all kinds of good accelerators, but how to make hardware, software, and customized algorithms work nicely together is a different story.
- In fact, most FMCW radar modules have similar data processing flow, which means that repeated efforts are made on different MCUs and among different groups. It is not only a waste of community resources, but also limits higher-level applications such as data fusion among different sensors.

The radar baseband processor unit (RBPU) included in Calterah Alps takes a different approach. In particular, RBPU is built along with a general radar signal processing flow. It means that the data flow is fixed but performance tuning knobs are provided for each module. Although it reduces the level of flexibility compared against the DSP approach, this approach achieves the following advantages:

- It shortens the development cycle greatly. The RBPU of Alps essentially handles the whole process in an automatic way. Developers only need to configure the unit in a proper way to make it work smoothly.
- The tuning knobs of the RBPU are well designed so that the final radar module can meet most application scenarios. Therefore, in terms of performance and application, end-users do not have many constraints.
- The whole baseband unit is running at 200 MHz, which has much lower power consumptions compared against DSP.

This user guide is all about the RBPU. But before delving into more details, we will have a brief introduction to frequency-modulation continuous wave (FMCW) radar in the next section. Experienced readers can safely skip it.

## 1.3 A Brief Introduction to FMCW Radar

Generally speaking, any radar system senses environment by sending a certain type of waveform and examining its reflection. In particular, FMCW radar waveform has the property that its frequency varies continuously during the sensing period. A very common one is shown in Fig. 1.1, which is a sine wave with the frequency increasing over time and called a chirp signal.

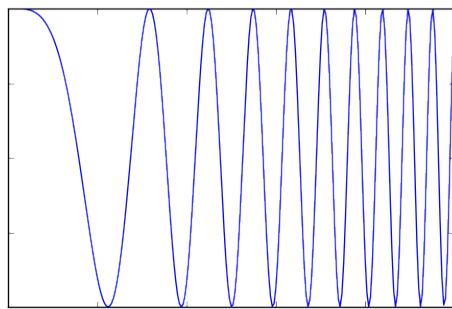


Fig. 1.1: View of Linear Chirp Signal over Time

Usually multiple back-to-back chirps form a frame as shown in Fig. 1.2, where the horizontal direction indicates the time evolution and the vertical direction indicates the frequency changes of the transmitter (TX) waveform.



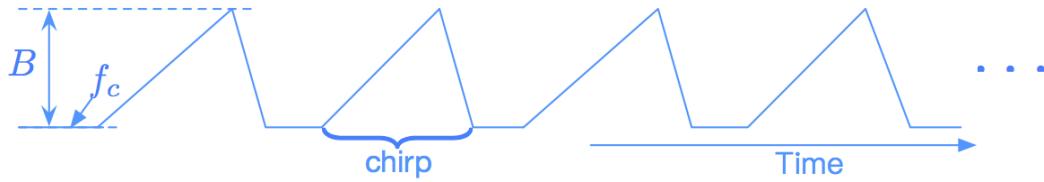


Fig. 1.2: Multiple FMCW Chirps Form an FMCW Frame

The maximum frequency difference is called bandwidth and denoted by  $B$ . And the lowest frequency is denoted by  $f_c$ . The reflected waveform preserves the change of frequency and after being mixed down, the signal at receiver (RX) output bears the position information about the objects or reflectors in the field of view (FoV).

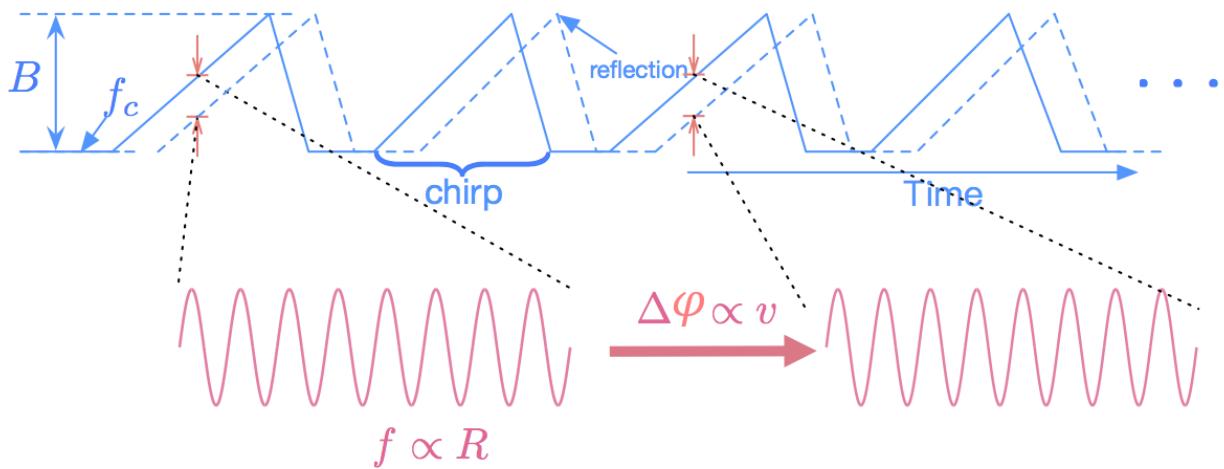


Fig. 1.3: FMCW Chirps with Their Reflection

Roughly speaking, within each chirp period, each reflected wave generates a sine wave and its frequency is proportional to its distance. For a moving object, across multiple chirps, the phase of reflected sine wave also changes due to the change of distance, or radical velocity. (See Fig. 1.3.) The phase difference between adjacent chirps is proportional to the radical velocity of the object. In other words, one can estimate the distance and radical velocity of a reflector by estimating the frequencies of samples points in a chirp and across chirps respectively. A popular approach is to use 2 dimensional fast Fourier transform (2D-FFT), which is adopted in Calterah Alps and discussed in Section 1.4.1.

To obtain the direction or angle information of reflectors, also known as the direction of arrival (DoA), one needs to look into RX signals across different RX antennas or channels.

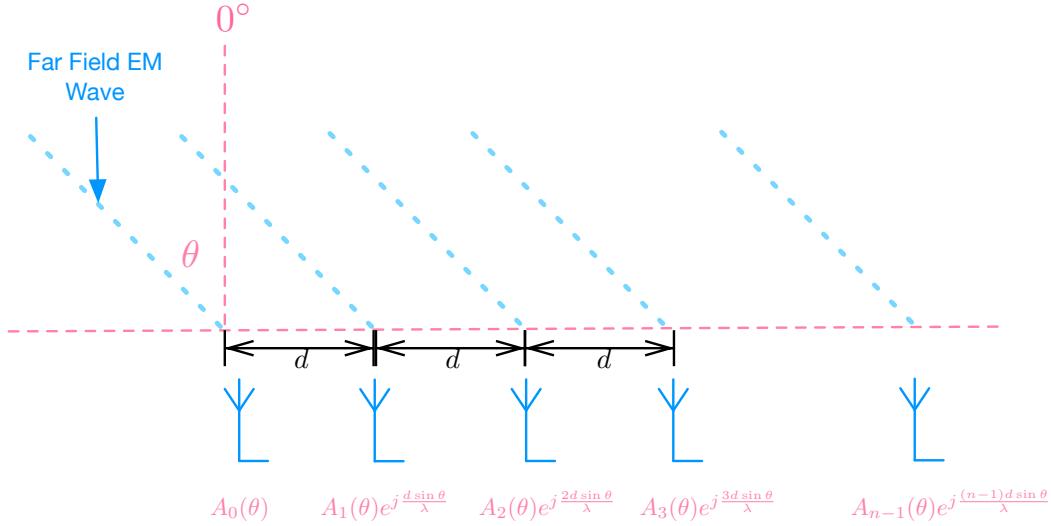


Fig. 1.4: Direction of Arrival Estimation with Antenna Array

As shown in the above Fig. 1.4, the output of each antenna has a different phase due to its relative space location. Therefore, one can use this property to estimate the direction of arrival. There are various methods to estimate the DoA. In general, the antenna locations need to be passed to the system, which is recommended to obtain through calibration instead of deriving from physical antenna size (see discussion in Section 8).

## 1.4 Architecture Overview

The RBPU of Calterah Alps is designed for FMCW waveforms together with 2D-FFT based signal processing. Here we have a brief introduction to its fundamental principles. For more details, please refer to standard text books or papers, for example, [Barrick1973].

### 1.4.1 Data Processing Flow

The fundamental radar waveform supported by Alps is frame-based. Each frame consists of a train of linear FMCW chirps as shown in Section 1.3. Fig. 1.5 shows the data processing flow adopted by Alps<sup>2</sup>.

---

<sup>2</sup> Some aspects of Fig. 1.5 do not exactly reflect the underlying hardware. For example, hardware pipeline structure is not shown in this figure.

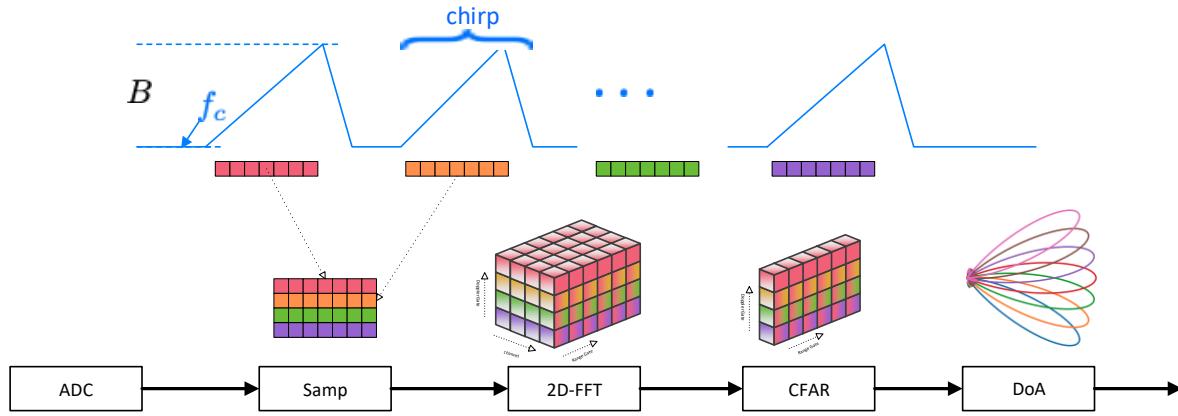


Fig. 1.5: Signal/Data Processing Flow of Alps Radar Baseband Processing Unit

Within each chirp, the processing unit collects samples from ADC within each ramp-up time period. Before it is fed into the 2D-FFT module, for each RX channel, the data is re-organized as a 2D array whose row is a collection of data from the same chirp. The 2D-FFT module adds windows and performs standard 2D-FFT on its input. Since the processing unit collects data and does 2D-FFT for each RX channel, the 2D-FFT output data has an additional dimension in the *channel* domain. At the constant-false-alarm (CFAR) module, the 3D data is squeezed or combined into a 2D array for object searching. For each output object from CFAR, the Direction of Arrival (DoA) unit performs the classical digital beamforming (DBF) algorithm or deterministic maximum likelihood (DML) algorithm to figure out the angle information of the object. Once the DoA unit finishes its job, it raises an *IRQ* to CPU for further processing, such as tracking and other application-specific data process.

#### 1.4.2 Performance Benchmarks

Table 1.2 shows a performance benchmark with typical radar system parameters<sup>1</sup>. Not only the power consumption is kept at a low level, but also the process time is very short. In particular, for such a system, FMCW frame length (6.4 ms) is a dominant factor if one would like to achieve a high frame-per-second (FPS) value. Even in a quite tough situation as shown in Table 1.3, baseband total process time is still much less than FMCW frame length.

Table 1.2: Performance Benchmark of a Typical Scenario

System Parameters		Baseband Performance		
chirp period	50 $\mu$ s	Process time	FFT2D	0.33ms
No. of chirps	128		CFAR	0.11ms
FFT size	$256 \times 128$		DBF/DML	0.12/0.52ms
No. of CFAR output	128		Total	0.56/0.96ms
No. of angle in DoA	180		Power consumption	Baseband < 100mW

<sup>1</sup> Process time of 2D-FFT is taken into the pipeline of hardware. In particular, 0.33 ms is essentially the time for computing FFT in the second dimension, because most of the first dimension of FFT is done as the system is transmitting FMCW waveform.

Table 1.3: Performance Benchmark of a Tough Scenario

System Parameters		Baseband Performance		
chirp period	30 $\mu$	Process time	FFT2D	1.31ms
No. of chirps	256		CFAR	0.38ms
FFT size	512 $\times$ 256		DBF/DML	0.46/1.54ms
No. of CFAR output	256		Total	2.15/3.23ms
No. of angle in DoA	360	Power consumption	Baseband	< 100mW

As we mentioned before, each module has sufficient tuning knobs to tune the performance so that the final radar module can fit into various applications. The remaining of the section gives an overview of programming interfaces. From the next chapter, each sub-unit will be discussed in detail.

## 1.5 Advanced Features

Besides the basic functions mentioned in Section 1.4, Alps also provides quite a few advanced features:

- **Virtual Array:** This feature provides the capability to increase the resolution of DoA estimation by virtualizing more RX channels ([Section 9](#));
- **2D DoA:** This feature supports both horizontal and elevation DoA estimation ([Section 9](#));
- **Auto Gain Control:** This feature prevents an RX channel from being saturated in an environment with large reflectors by ensuring that the system always uses the best RX Gain ([Section 10](#));
- **Interference Avoidance Mode:** This feature avoids interference during field usage by avoiding frequency and time collisions ([Section 11](#));
- **Frame Interleaving:** This feature allows users to program up to four different types of frames at a time and use them for different application scenarios ([Section 12](#));
- **Anti Velocity Ambiguity:** This feature provides different solutions to solve the velocity ambiguity issue arising from sub-sampling at Doppler domain ([Section 13](#));
- **Noise Estimator:** This feature provides noise floor estimation ([Section 14](#));
- **Hardware-in-the-Loop:** This feature supports hardware-in-the-loop tests ([Section 15](#));
- **Functional Safety:** This feature provides easy-to-use functional safety mechanisms ([Section 16](#));
- **System Level Cascade:** Alps is not only able to work independently but also can be cascaded at the system level to form a larger radar system ([Section 17](#)).

## DATA REPRESENTATION

Basic programming of Alps's RBPU is nothing but accessing different registers and memories. Both baseband input data and configuration are packed in a certain format to fit into the underlying hardware. Therefore, having a good understanding of these data formats is key to understand how to configure the baseband properly. This chapter is devoted to an in-depth discussion on this topic.

From the viewpoint of hardware implementation, all data is stored in certain space with certain bit-width. To facilitate the discussion, we distinguish the following two terms:

- *HW Value*: The integer value of underlying stored bits. It is the value you are going to write to registers or memories.
- *Numeric Value*: The value that it represents. In other words, it is the physical meaning of the underlying bits.

The following list includes all fundamental data types or formats used in baseband hardware. Please make sure that you have a good understanding of them before moving forward.

- Fixed-point integer;
- Fixed-point real number;
- Pseudo-floating real number;
- Complex number.

### 2.1 Fixed-Point Integer

A fixed-point integer is characterized by two parameters and is denoted as FXI( $W, SU$ ), where

- $W$ : bitwidth of the integer;
- $SU$ : Take value  $S$  (when its numeric value is signed) or  $U$  (when its numeric value is unsigned)

The underlying bits are encoded in standard two's complement format.

For example, data of type FXI(5,  $S$ ) can represent values from  $-16$  to  $15$ . If the underlying bit is 10000b (*HW value*), it represents the value of  $-16$  (*numeric value*).

Here is another example. Data of type FXI(5,  $U$ ) can take values from  $0$  to  $31$  and its *numeric value* is the same with its *HW value*.

More generally, data of type FXI( $W, S$ ) ranges from  $-2^{W-1}$  to  $2^{W-1} - 1$ ; data of type FXI( $W, U$ ) ranges from  $0$  to  $2^W - 1$ .

## 2.2 Fixed-Point Real Number

A fixed-point real number is to represent a real number with an integer together with its position of decimal point. Therefore, it has three parameters:

1.  $W$ : Total bitwidth, which must be a **positive** integer;
2.  $I$ : Bitwidth of the integer part of the real number, which is a **signed** integer;
3.  $SU$ : Take value  $S$  (when its numeric value is signed) or  $U$  (when its numeric value is unsigned)

Fixed-point real numbers can be denoted as  $\text{FXR}(W, I, SU)$ . A list of examples are:

Table 2.1: Example of  $\text{FXR}(W, I, SU)$

Data Type	Numeric Range	Numeric Quantum	HW Value Range
$\text{FXR}(5, 7, U)$	$0 \sim 124$	4	$0 \sim 31$
$\text{FXR}(5, 7, S)$	$-64 \sim 60$	4	$0 \sim 31$
$\text{FXR}(5, 5, U)$	$0 \sim 31$	1	$0 \sim 31$
$\text{FXR}(5, 5, S)$	$-16 \sim 15$	1	$0 \sim 31$
$\text{FXR}(5, 1, U)$	$0 \sim 31/16$	$1/16$	$0 \sim 31$
$\text{FXR}(5, 1, S)$	$-1 \sim 15/16$	$1/16$	$0 \sim 31$
$\text{FXR}(5, -1, U)$	$0 \sim 31/64$	$1/64$	$0 \sim 31$
$\text{FXR}(5, -1, S)$	$-1/4 \sim 15/64$	$1/64$	$0 \sim 31$

Generally, numeric range of  $\text{FXR}(W, I, U)$  is from 0 to  $2^I (1 - 2^{-W})$  with quantum of  $2^{I-W}$ ; numeric range of  $\text{FXR}(W, I, S)$  is from  $-2^{I-1}$  to  $2^I (1/2 - 2^{-W})$  with quantum of  $2^{I-W}$ . In both cases, the HW value range is from 0 to  $2^W - 1$ . The underlying bits are encoded in standard two's complement format.

Suppose that the HW value of a real number with the format of  $\text{FXR}(W, I, SU)$  equals  $d$ . Then its numeric value can be obtained by  $\tilde{d} \times 2^{I-W}$ , where

$$\tilde{d} = \begin{cases} d & \text{if } SU = U \\ d & \text{if } SU = S \text{ and } d < 2^{W-1} \\ d - 2^W & \text{if } SU = S \text{ and } d \geq 2^{W-1} \end{cases}$$

Conversely, given a numeric value  $v$ , the corresponding HW value of  $\text{FXR}(W, I, SU)$  can be obtained with following code

```

1 uint32_t float_to_fx(float val, uint32_t W, int32_t I, bool sign)
{
2     assert(W < 32); // prevent large bit width overflow
3     uint32_t retval = 0;
4     int32_t pow = W-I; // compute the power
5     uint32_t mask = -1L << (sign ? W-1 : W);
6     int32_t d_min = sign ? ((int32_t) mask) : 0; // Find out the HW value for_
7     ↪minimum numeric value
8     uint32_t d_max = ~mask; // Find out the HW value for_
9     ↪maximum numeric value
10    double tmp = pow > 0 ? ((double)val) * (1 << pow) : ((double)val) / (1 << (-
11    ↪pow));
12    tmp = round(tmp);
13    if (d_min > tmp) // prevent underflow
14        retval = d_min;
15    else if (d_max < tmp) // prevent overflow
16        retval = d_max;
17    else

```

(continues on next page)



(continued from previous page)

```

16     /* It is very trick: we need to convert it to int32_t instead of 
17     ↵uint32_t! */
18     retval = (int32_t) tmp;
19     return retval & (~(-1L << W)); // zero out unused leading bit in retval
}

```

Here are some highlights of the above code snippet

1. Lines 11-14 are dealing with the cases of underflow and overflow, since numeric  $v$  may be out of range of  $\text{FXR}(W, I, SU)$ .
2. The routine utilizes the fact that 2's complement representation is used for int types.

## 2.3 Pseudo-Floating Real Number

A pseudo-floating real number has two parts: mantissa ( $M$ ) and exponent ( $E$ ). It is denoted as  $\text{FLR}(W_m, I, SU_m, E, SU_e)$ , where its mantissa is of type  $\text{FXR}(W_m, I, SU_m)$  and the exponent is of type  $\text{FXI}(E, SU_e)$ . The underlying bit arrangement looks like below:

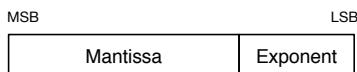


Fig. 2.1: Bit Arrangement of Pseudo-Floating Real Number

Suppose that the numeric values of mantissa and exponent are  $v_{man}$  and  $v_{exp}$ . Then the numeric value of the pseudo-floating real number is given by

Table 2.2: Interpretation of  $\text{FLR}(W_m, I, SU_m, E, SU_e)$ .

$SU_e$	Numeric Value
U	$v_{man} \times 2^{-v_{exp}}$
S	$v_{man} \times 2^{v_{exp}}$

Thus, given HW value  $d$  with the type  $\text{FLR}(W_m, I, SU_m, E, SU_e)$ , one can obtain its numeric value by

1. Parsing HW values of mantissa ( $d_{man}$ ) and exponent ( $d_{exp}$ ) according to their bit-widths and Fig. 2.1.
2. Converting  $d_{man}$  and  $d_{exp}$  to  $v_{man}$  and  $v_{exp}$ , respectively according to Section 2.2 and Section 2.1, respectively.
3. Getting the numeric value according to Table 2.2.

Conversion from a numeric value to a HW value is more involved. The underlying reason is that given numeric value  $v$ , there might be multiple ways to split it into  $v_{man}$  and  $v_{exp}$ . So different ways of splitting lead to different HW values. To overcome this, we need to follow a convention on HW values:

- $d_{man}$  needs to have the smallest possible leading bits (number of sign bits minus 1).

This means that  $d_{exp}$  needs to accommodate as much bit shift as possible. For example, consider a data type of  $\text{FLR}(4, 0, S, 2, U)$ . Consider two bit representations 0110\_01b and 0011\_00b. They have the same numeric value according to our definition. However, by our convention, we choose 0110\_01b as the HW value instead of 0011\_00b. Because the latter mantissa has 1 leading bit but the former one has 0 leading bit. As another example, we choose 1000\_01b over 1100\_00b for the HW value. In particular, the numeric value of 0 will be represented as 0000\_11b.



## 2.4. Complex Number

---

With the convention we use here, it is easy to see that the conversion from a numeric value to a HW value is not only unique but also has the least precision loss.

## 2.4 Complex Number

There are two representations of complex numbers: One is simply packing real and imaginary parts together, while the other has a common exponent shared by the real and imaginary parts.

### 2.4.1 Simple Complex Number

As shown in Fig. 2.2, complex representation packs the real part in its most significant bit (MSB) and the imaginary part in its least significant bit (LSB). Both real and imaginary parts are in the same format as FXR( $W, I, SU$ ). Therefore, conversion between a numeric value and a HW value can be done in a similar way as in Section 2.2 except that the bit packing of Fig. 2.2 needs to be taken care accordingly. This user guide uses CFX( $W, I, SU$ ) to denote this data representation.

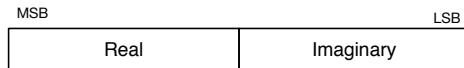


Fig. 2.2: Underlying Structure of Complex Number

### 2.4.2 Complex Number with Common Exponent

By sharing a common exponent, a complex number can be stored more efficiently. Fig. 2.3 shows the underlying bit storage structure. This user guide uses CFL( $W_m, I_m, SU_m, W_e, SU_e$ ) to denote this data representation. In particular, real and imaginary parts are both in the format of FXR( $W_m, I_m, SU_m$ ) and the exponent part is in the format of FXI( $W_e, SU_e$ ).

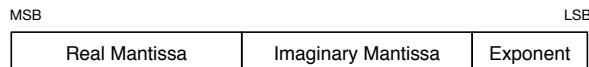


Fig. 2.3: Underlying Structure of Complex Number with Common Exponent

Conversion from a HW value to a numeric value is the same as the way described in Section 2.2 except that the bit arrangement of Fig. 2.3 needs to be taken care accordingly. For conversion from a numeric value to a HW value, the convention in Section 2.2 should be modified to:

- $d_{r,man}$  and  $d_{i,man}$  need to have the smallest possible leading bits with a common exponent.



## 2.5 Examples

This section gives some examples of data types mentioned in previous sections.

Table 2.3: Examples of Interpretation of Data Type

Data Type	Value	HW Value	Numeric Value
$\text{FXI}(5, U)$	7	00111b	7
$\text{FXI}(7, S)$	-6	1111010b	-6
$\text{FXR}(5, 7, S)$	-20	11011b	-20
$\text{FXR}(5, 7, S)$	-21	11011b	<p>-20            (The granularity of the type is 4. -21 is not presentable. The closest presentable value for -21 is -20.)</p>
$\text{FXR}(5, 1, U)$	1.2	0x13	<p>1.1875            (The granularity of the type is <math>2^{-4} = 0.0625</math>. The closest presentable value for 1.2 is 1.1875.)</p>
$\text{FLR}(8, 2, S, 4, S)$	0	0x8	<p>0            (The mantissa is 0 and exponent is -8.)</p>
$\text{FLR}(14, 2, S, 4, S)$	18.12	0x121e4	18.1171875
$\text{CFX}(5, 1, S)$	0.1+0.2i	0x43	0.125 + 0.1875i
$\text{CFX}(7, -2, S)$	-0.02+0.03i	0x3b0f	-0.0195312+0.0292969i
$\text{CFL}(8, 1, S, 4, U)$	0.02-0.1i	0x14993	0.0195312-0.100586i
$\text{CFL}(5, -1, S, 4, S)$	0.02-0.1i	0x53f	0.015625-0.101562i



## **2.5. Examples**

---



## HARDWARE PROGRAMMING INTERFACES

### 3.1 Overview of Radar Baseband Process Unit (RBPU)

Radar Baseband Process Unit (RBPU) is a digital signal processor in Calterah Alps radar SoC. The function of RBPU is receiving raw ADC data, detecting targets, and obtaining the distance, velocity and direction information of targets by radar baseband algorithms. Compared to the traditional DSP solution, RBPU hardware acceleration scheme can improve digital process efficiency by more than 50%. It provides a low-power, cost-effective, and high-performance solution for radar SoC systems.

#### 3.1.1 Hardware Architecture

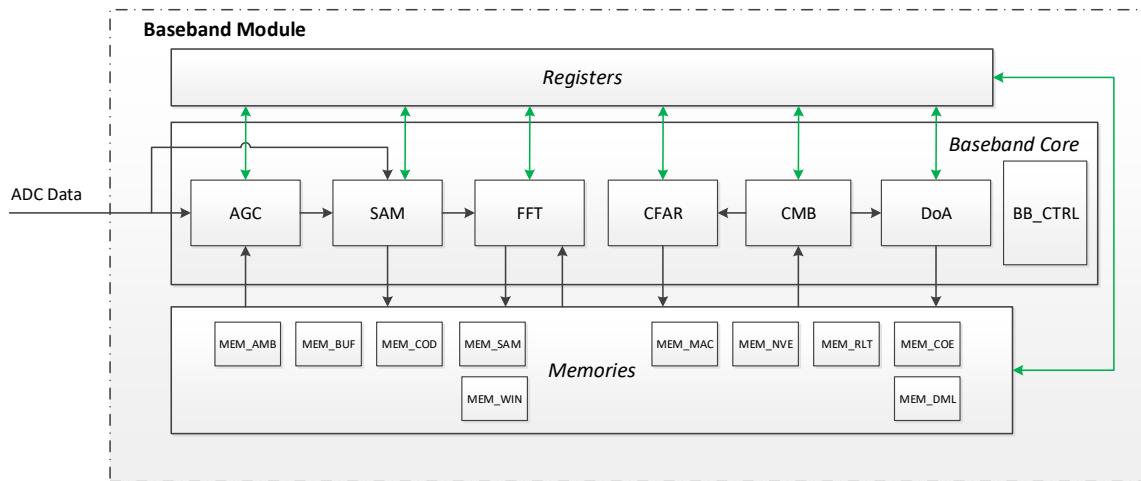


Fig. 3.1: Baseband Block Diagram and Data Flow<sup>1</sup>

As illustrated by Fig. 3.1, RBPU hardware module can be largely divided into 3 parts, with each playing a different role for the digital data process.

- *Registers*: Programmable registers for each sub-module.

<sup>1</sup> Black arrows indicate data flow, and green arrows indicate static configuration flow. Here *static* means that the value cannot be changed within a frame. The data interaction between FFT and CFAR/DoA is realized through memory.

### 3.1. Overview of Radar Baseband Process Unit (RBPU)

- *Baseband core*: Digital data process module that involves major baseband computation. This part consists of different sub-modules that correspond to different radar signal processing stages.
- *Memories*: Various memories are used for different purposes, such as buffering pre/post processing data, storing predetermined parameters used by *baseband core*. Most of them can be accessed (read/write) through CPU.

The majority of this guide is focused on discussing how to program these registers and memories and trigger the baseband core to run properly.

When radar SoC initialization is done, the baseband starts work. Then registers and memory can be set. After RBPU is properly configured, baseband core is ready to be triggered. Once it is triggered, baseband core runs through several (programmable) stages for every frame. Fig. 3.2 illustrates all stages/status of baseband core. The IDLE and AGC stages are setup stages, which can be viewed as preparation for upcoming stages. Other stages are data processing stages. The stages marked in green can be skipped through register configuration. The conditions for baseband core to move from one stage to the next are described in Fig. 3.2. Table 3.1 give detailed descriptions of each jump condition.

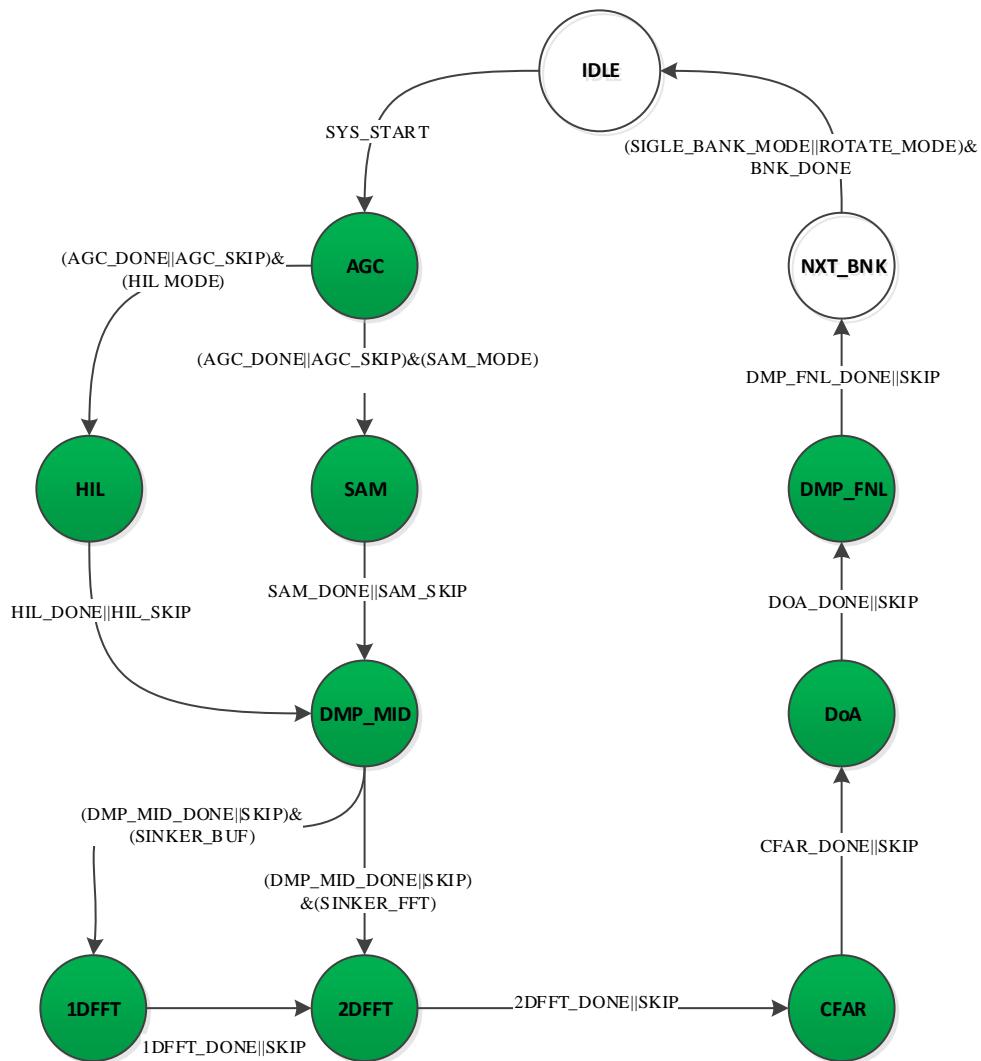


Fig. 3.2: Finite State Machine of Baseband Core

Table 3.1: Descriptions of Macros for FSM

Macro Name	Description	Trigger	Register Address
SYS_START	Baseband start signal	firmware	0x0000 High: Start
AGC_DONE	AGC process done	baseband hardware	NA
AGC_SKIP	AGC status skip signal	firmware	0x0018 Bit0 Low: Skip
HIL_MODE	Hardware in the loop mode enable signal	firmware	0x0018 Bit1 High: Enable
SAM_MODE	Sample mode enable signal	firmware	0x0018 Bit2 High: Enable
HIL_DONE	Hardware in the loop process done	baseband hardware	NA
SAM_DONE	Sample mode process done	baseband hardware	NA
SAM_SKIP	Sample mode disable signal	firmware	0x0018 Bit2 Low: Skip
HIL_SKIP	Hardware in the loop disable signal	firmware	0x0018 Bit1 Low: Skip
DMP_MID_DONE	1DFFT data dump done	baseband hardware	NA
DMP_MID_SKIP	1DFFT data dump mode disable signal	firmware	0x0018 Bit3 Low: Skip
1DFFT_DONE	1DFFT process done	baseband hardware	NA
1DFFT_SKIP	1DFFT status skip signal	firmware	0x0018 Bit4 Low: Skip
2DFFT_DONE	2DFFT process done	baseband hardware	NA
2DFFT_SKIP	2DFFT status skip signal	firmware	0x0018 Bit5 Low: Skip
CFAR_DONE	CFAR process done	baseband hardware	NA
CFAR_SKIP	CFAR status skip signal	firmware	0x0018 Bit6 Low: Skip
DoA_DONE	DoA process done	baseband hardware	NA
DoA_SKIP	DoA status skip signal	firmware	0x0018 Bit7 Low: Skip
DMP_FNL_DONE	Dump final results process done	baseband hardware	NA
DMP_FNL_SKIP	Dump final results status skip signal	firmware	0x0018 Bit8 Low: Skip
SINGLE_BANK_MODE	Single bank mode enable signal	firmware	0x0004 Set to 2'b00
ROTATE_MODE	Rotate bank mode enable signal	firmware	0x0004 Set to 2'b10
BNK_DONE	Bank process done	baseband hardware	NA
MULTIPLE_MODE	Multiple bank mode enable signal	firmware	0x0004 Set to 2'b01

The following describes all the stages in detail:

- **IDLE**

System idle status. It is designed for CPU to program the registers and memories properly. In this stage, baseband core waits for a synchronization signal from FMCW generator (or CPU) to trigger the remaining processes. For baseband core triggering mechanism, refer to discussion in [Section 3.5](#).

- **AGC**

Stage for auto gain control. In this stage, the AGC sub-module adjusts gains of RX components to achieve good signal reception. This function is covered in [Section 10](#).

- **HIL**



### 3.1. Overview of Radar Baseband Process Unit (RBPU)

---

Stage of hardware-in-the-loop. In this stage, input data of baseband core comes from memory instead of ADC. This is designed for debugging purposes, for example, for verifying the integrity of baseband core itself, or for performance tuning by playing back pre-captured ADC raw data. For more details about HIL, refer to [Section 15](#).

- **SAM**

In this stage, baseband core goes through some pre-FFT processes, including decimation filtering, data reorganization, and so on. These are covered in [Section 5](#).

- **DMP\_MID**

This stage is designed for debugging purposes, indicating a data dump from MEM\_BUF to the interface of the Alps device.

- **1DFFT**

Stage of first-dimension Fast Fourier Transform (FFT), or in other words, FFT along the range gate. By default, this stage is shared with SAM, which means that SAM is enabled and at the same time 1DFFT is disabled. See [Section 6.4.1](#) to get more details.

- **2DFFT**

Stage of second-dimension FFT, or in other words, FFT along the Doppler gate.

- **CFAR**

Stage of constant false alarm rate (CFAR) detection. In this stage, object searching is done in the range-Doppler plane. [Section 7](#) has a full discussion on this topic.

- **DoA**

Stage of direction of arrival (DoA) estimation. In this stage, for the CFAR output, DoA engine estimates the reflector's angle related to the radar system. Details are covered in [Section 8](#).

- **DMP\_FNL**

This stage is designed for debugging purposes, indicating a data dump from MEM\_BUF to the interface of the Alps device. See [Section 15.2.1.2.2](#) for more details about DMP\_MID and DMP\_FNL.

- **NXT\_BNK**

Bank shift stage for frame interleaving mode. Baseband engine can support maximum 5 banks of registers and memory shifting through the BANK\_QUE register, which can reduce the CPU workload. For details, refer to [Section 12](#).

In terms of programmability, the stages marked in green can be skipped and the other stages cannot be skipped. Register CFG\_SYS\_ENABLE controls which stages to be run in the format of bit mask shown in [Table 3.2](#).

Table 3.2: Bit mask of CFG\_SYS\_ENABLE

Bit	8	7	6	5	4	3	2	1	0
Definition	DMP_FNL	DoA	CFAR	2DFFT	1DFFT	DMP_MID	SAM	HIL	AGC



## 3.2 Interface between Hardware and CPU

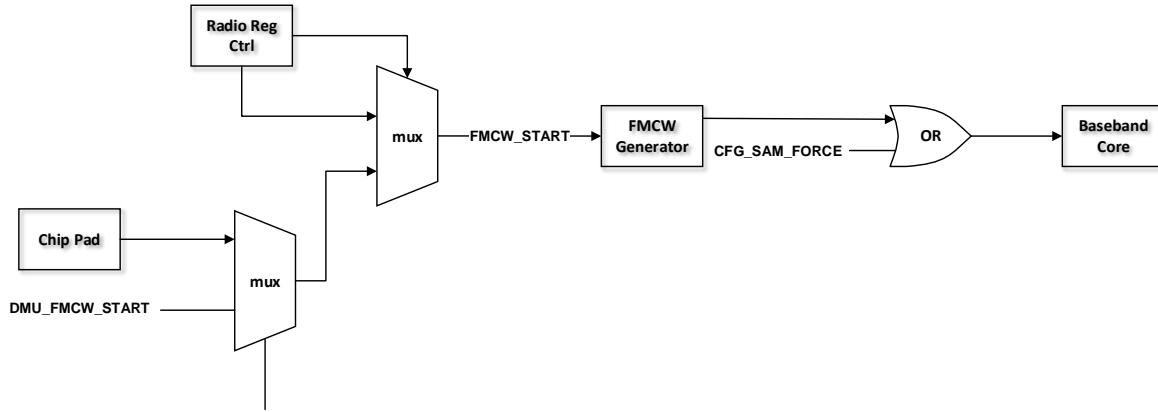


Fig. 3.3: Signal and Control Flow between FMCW Generator and Baseband Core

Fig. 3.3 illustrates the control flow between FMCW generator and baseband core:

- *Baseband core* is triggered by the signal BB\_START, which is an OR result between FMCW\_SYNC and CFG\_SAM\_FORCE. The former is generated by *FMCW generator* and it indicates that FMCW is started to transmit. The latter is generated by CPU, which is useful when baseband is put in a special mode, and one wants to force *baseband core* to run without FMCW's synchronization signal. Once triggered, *baseband core* runs according to its internal state machine as illustrated by Fig. 3.2, depending on the setting of CFG\_SYS\_ENABLE. When it finishes its job, *baseband core* raises an IRQ to inform CPU.
- *FMCW generator* is triggered by the signal FMCW\_START, which is chosen from two sources: one is from *Radio Reg Ctrl*, and the other is from an OR result between a PAD input and the signal DMU\_FMCW\_START (0xBA\_0018). The latter source gives flexibility to trigger FMCW generator from chip input or CPU.

CPU can interact with RBPU through register access, memory access and interrupts.

### 3.2.1 Accessing Registers and Memories of RBPU

Registers and memories can be programmed to fit into different applications. But all writing options should be done when baseband core is in the stage of IDLE. There is no guarantee of the results during other stages. For a quick reference:

- *Registers*: See Section B.
- *Memories*: See Table 3.3.

There are also some constraints in hardware. Refer to Section 3.3 and the corresponding discussion in each following chapter.

## 3.2. Interface between Hardware and CPU

---

### 3.2.1.1 Accessing Registers

There are altogether 400 registers in Alps baseband. They can be roughly divided into three types, each with a different prefix in register names, as shown in the table of [Section B](#):

- **CFG\_**: Registers with the CFG\_ prefix are configuration registers. They are usually for mode selection or parameter settings. This type of registers can *only* be written when the system is in the idle status (IDLE in [Fig. 3.2](#)) and can be read at any time.
- **CTL\_**: Registers with the CTL\_ prefix are usually designed for baseband control purpose and implemented in self-clean fashion. For example, the register CTL\_SYS\_START is a trigger to start baseband core. This type of registers can *only* be written when the system is in the idle status. Reading is not meaningful for this type of registers since they are self-clean.
- **FDB\_**: Registers with the FDB\_ prefix are designed for users to query baseband results and status. This type of registers are usually read-only and can be read at any time.

All baseband registers are mapped into the memory space between 0xC0\_0000 and 0xC0\_FFFF ([Section A](#)). Addresses shown in the *Addr* column of the table in [Section B](#) are offset addresses with the base address being 0xC0\_0000.

### 3.2.1.2 Accessing Memories

In Alps, there are altogether 10 memories for CPU access. Some of the memories are used for buffering intermediate data generated by RBPU. Others are for storing parameters and the final results. All these memories can be accessed directly in the memory space between 0xE0\_0000 and 0xFF\_FFFF. Register CFG\_SYS\_MEM\_ACT (0xC0\_0014) is a bus selection register. By changing its value ([Table 3.3](#)), the corresponding memory can be accessed accordingly.

Table 3.3: Memory Mapping

CFG_SYS_MEM_ACT	Memory	Description	Memory Type	Bank Support	Size (N × D × W)	Internal Users
0	MEM_COD	AGC code memory	Dual port asynchronous	Single bank	1 × 64 × 17	AGC
1	MEM_SAM	ADC sample buffer	Dual port asynchronous	Single bank	1 × 8192 × 32	SAMPLE
2	MEM_WIN	FFT windowing LUT	Single port synchronous	4 banks	(4096+1024) × 28	1DFFT and 2DFFT, respectively
3	MEM_NVE	NVE results	Single port synchronous	4 banks	1 × 4096 × 20	FFT
4	MEM_BUF	FFT output memory	Single port synchronous	Single bank	32 × 16384 × 32	SAMPLE, FFT, CFAR, DoA
5	MEM_COE	CFAR and DoA coefficients LUT	Single port synchronous	Single bank	4 × 3872 × 28	CFAR, DoA
6	MEM_MAC	Cascade memory	Single port synchronous	4 banks <sup>2</sup>	4 × 4096 × 32	FFT, CFAR, DoA
7	MEM_RLT	CFAR or DoA results	Single port synchronous	4 banks	1 × 1024 × 53	CFAR
7	MEM_RLT	CFAR or DoA results	Single port synchronous	4 banks	1 × 12288 × 34	DoA
8	MEM_AMB	Anti velocity ambiguity parameters	Single port synchronous	4 banks	1 × 128 × 15	DoA
9	MEM_DML	DML setting parameters	Single port synchronous	4 banks	1 × 4096 × 64	DoA

<sup>2</sup> The memory MEM\_MAC supports many functions. When it is used for cascade mode to store FFT data from the slave chip (see [Section 17](#)), MEM\_MAC is *NOT* divided into 4 banks, that is, the whole space of MEM\_MAC is all used for the current frame.

---

**Note:** When CFG\_SYS\_MEM\_ACT is set to 7, 4 banks can be read and written independently when the target number is less than 257.

---

### 3.2.1.3 Memory Refresh

In hardware, memory refresh operations are supported in MEM\_BUF. MEM\_BUF is split to 32 slices, and each slice can be refreshed autonomously. 3 registers are used for the refresh operation. Each bit of these 3 registers indicates one specific memory slice. Bit 0 indicates memory slice 0, bit 1 indicates memory slice 1, and so on.

- CTL\_DBG\_RFRSH\_START

Setting this register to 1 starts the memory refresh operation on the specific memory slice corresponding to the bit index.

- FDB\_DBG\_RFRSH\_STATUS

This register indicates the feedback of refresh status. Each bit represents the corresponding memory slice. Bit value 1 means the refresh operation is done and bit value 0 means the refresh operation is not yet done or not started.

- CTL\_DBG\_RFRSH\_CLR

Setting this register to 1 clears the refresh status to 0 on the specific memory slice corresponding to the bit index.

The timing sequence of a memory refresh operation is shown in Fig. 3.4. A memory refresh operation consumes about 80  $\mu$ s (baseband runs at 200 MHz), no matter how many slices are refreshed.

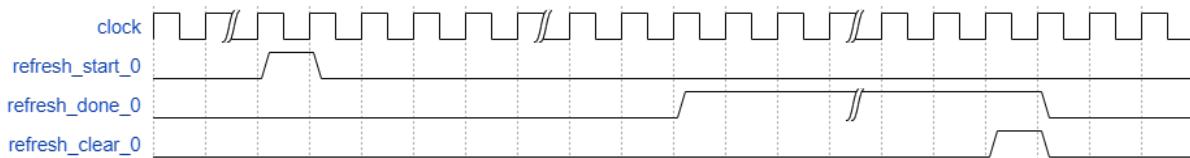


Fig. 3.4: Memory Refresh

### 3.2.2 Bank Shifting

To support the feature of *Frame Interleaving*, a large portion of the registers and some memories are banked.

#### 3.2.2.1 Static Access

Most of the banked registers are with the prefix *CFG\_* and the bank access is controlled by CFG\_SYS\_BNK\_ACT (0xC0\_0008). There are altogether 5 banks. Bank0, Bank1, Bank2, and Bank3 are working banks that support all *CFG\_* registers so that the normal baseband engine can work in completely different modes. Bank4 is a shadow bank for testing purpose, which only contains AGC/SAMPLE/FFT related registers. Section C lists all the registers in the fifth bank. Registers in the first four banks can be found in Section B.

Memories listed in Table 3.4 are also banked but no memory supports the fifth bank.

## 3.2. Interface between Hardware and CPU

---

Table 3.4: Banked Memories

CFG_SYS_MEM_ACT	Memory/LUT	Bank Offsets (Bytes)
2	FFT Windowing LUT	0x000, 0x2000, 0x4000, and 0x6000
3	NVE Results	0x000, 0x1000, 0x2000, and 0x3000
6	When not reused for cascade mode	0x000, 0x4000, 0x8000, and 0xc000
7	CFAR and DoA Results	0x000, 0x8000, 0x10000, and 0x18000
8	Anti velocity ambiguity Parameters	0x000, 0x0080, 0x0100, and 0x0180
9	DML Setting Parameters	0x000, 0x2000, 0x4000, and 0x6000

There is no overall control for memory bank access as for register bank access. Different banks of the same type of memory have different address offsets, as listed in [Table 3.4](#).

### 3.2.2.2 Dynamic Control

[Fig. 3.5](#) shows the block diagram of baseband registers and memory bank shifting. If registers and memories are maintained in multiple banks, bank execution sequence is controlled by *CFG\_SYS\_BANK\_QUE* and *CFG\_SYS\_BANK\_MODE*.

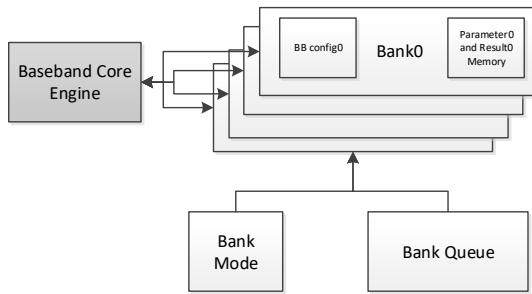


Fig. 3.5: Baseband Register and Memory Bank Shift

Detailed usage of these banked registers and memories can be found in [Section 12](#). For usage of the fifth bank, refer to [Section 16](#).

### 3.2.3 Interrupts

There are 3 interrupt (IRQ) signals from baseband, which can be disabled or enabled through the register *CFG\_SYS\_IRQ\_ENA* (0xC0\_0044).

- Bit 0: All processes of baseband are done.
- Bit 1: Reserved.
- Bit 2: SAM is done.

Note that in multiple bank mode, bit 1 of *CFG\_SYS\_IRQ\_ENA* will raise when each bank finishes.

IRQ status can be accessed from the register *FDB\_SYS\_IRQ\_STATUS* (0xC0\_0064) and can be cleared through *CTL\_SYS\_IRQ\_CLR* (0xC0\_0048). When the system is reset, all the IRQ signals are low before every frame starts. Only after RBPU is triggered to start, CPU can access the register *FDB\_SYS\_IRQ\_STATUS* to monitor the corresponding IRQ status.

### 3.2.4 Outputting Results

0x0	RSV[31:28]	rng_idx[27:18]	rng_acc[17:14]	vel_idx[13:04]	vel_acc[03:00]
0x4	RSV[31:25]	amb_idx[24:20]		Noi[19:00]	
0x8	RSV[31:14]		ang_val0[13]	ang_idx0[12:04]	ang_acc0[03:00]
0xc	RSV[31:20]			Pow0[19:00]	
...	RSV[31:14]		ang_val1[13]	ang_idx1[12:04]	ang_acc1[03:00]
0x24	RSV[31:20]			Pow1[19:00]	
...	RSV[31:14]		ang_val2[13]	ang_idx2[12:04]	ang_acc2[03:00]
0x48	RSV[31:20]			Pow2[19:00]	
...	RSV[31:14]		ang_val3[13]	ang_idx3[12:04]	ang_acc3[03:00]
0x64	RSV[31:20]			Pow3[19:00]	
	RSV[31:14]		ang_val4[13]	ang_idx4[12:04]	ang_acc4[03:00]
	RSV[31:20]			Pow4[19:00]	
	RSV[31:14]		ang_val5[13]	ang_idx5[12:04]	ang_acc5[03:00]
	RSV[31:20]			Pow5[19:00]	
	RSV[31:14]		ang_val6[13]	ang_idx6[12:04]	ang_acc6[03:00]
	RSV[31:20]			Pow6[19:00]	
	RSV[31:14]		ang_val7[13]	ang_idx7[12:04]	ang_acc7[03:00]
	RSV[31:20]			Pow7[19:00]	
	RSV[31:14]		ang_val8[13]	ang_idx8[12:04]	ang_acc8[03:00]
	RSV[31:20]			Pow8[19:00]	
	RSV[31:14]		ang_val9[13]	ang_idx9[12:04]	ang_acc9[03:00]
	RSV[31:20]			Pow9[19:00]	
	RSV[31:14]		ang_val10[13]	ang_idx10[12:04]	ang_acc10[03:00]
	RSV[31:20]			Pow10[19:00]	
	RSV[31:14]		ang_val11[13]	ang_idx11[12:04]	ang_acc11[03:00]
	RSV[31:20]			Pow11[19:00]	

Fig. 3.6: Memory Layout of Result Output

When DoA is done, the CPU can fetch the results from hardware. The following information is recommended to be read:

- Target number: The total target number is stored in FDB\_CFR\_NUMB\_OBJ. Note that it is possible that the number equals 0, which means no target is detected in the current frame.
- Detailed target information: Each target's detailed information, such as range and velocity indices and DoA indices, is stored in CFAR and DoA result memory (CFG\_SYS\_MEM\_ACT = 7).

For each CFAR output, DoA may recognize multiple targets. Therefore, target information is packed for each CFAR output instead of for each target. For multiple CFAR outputs, target information is packed continuously in memory with the same format.

Fig. 3.6 shows the memory layout of one CFAR output.

- rng\_idx and vel\_idx are range-gate and Doppler-gate indices, respectively.
- rng\_acc and vel\_acc are hardware range-gate and Doppler-gate interpolation results, respectively. Both are in the format of FXR(4, 0, S).
- amb\_idx is velocity ambiguity index in the format of FXI(5, 0, S).
- noi is noise level estimation from CFAR in the format of FLR(15, 1, U, 5, U).
- There are 12 groups of data related to directions:
  - ang\_val indicates whether it is a valid DoA result.
  - ang\_idx is ~~DoA result in index~~



### 3.3. Hardware Constraints

---

- `ang_acc` is interpolation results of DoA index, which is in the format of  $\text{FXR}(4, 0, S)$ .
- `pow` is the power of the corresponding target in the format of  $\text{FLR}(15, 1, U, 5, U)$ .

For each CFAR output, the 12 groups of DoA results are partitioned in different sub-groups for different 2D DoA modes.

- Single shot mode: 12 groups of DoA results come from the same digital beamforming result.
- Normal mode: 12 groups of DoA results are split into three sub-groups. Each sub-group has up to 4 outputs, which come from DoA of the same group of Rx channels.
- Combined mode: The first 4 groups of DoA results (Groups 0-3) are the DoA results of the primary dimension (usually, it is the horizontal one). Groups 4-5 are the DoA results of the secondary dimension (usually, it is the vertical one), corresponding to Group 0. Similarly, Groups 6-7 are the DoA results of the secondary dimension corresponding to Group 1, ....

For details about 2D DoA modes, refer to [Section 9.2.4](#).

## 3.3 Hardware Constraints

There are some setting constraints for RBPU:

- All register and memory settings must be done in the `IDLE` stage.
- After receiving IRQ signals, CPU must read results.
- Some register settings such as chirp and frame length must be aligned with FMCW.
- In multiple bank mode, each bank can only report up to 256 Range-Doppler matrix (RDM) indices, while in single bank mode, it can report up to 1024 RDM indices.

## 3.4 SoC Address Mapping

RBPU's registers are mapped to addresses between `0xC0_0000` and `0xC0_FFFF` and memory/LUT is mapped to addresses from `0xE0_0000` to `0xFF_FFFF`. Read/write for baseband registers is the same as memory read/write from CPU's viewpoint. Data at each address is in 8 bits and longer data is stored in little-endian.

For further reference, the complete memory mapping of Alps is listed in [Section A](#).

## 3.5 Software Suggestions

To accommodate the hardware design shown in [Fig. 3.3](#), it is suggested that a general firmware flow should be as follows:

1. Configure *FMCW Generator* according to the application (which is covered in [Section 4](#));
2. Configure *baseband core* accordingly (which is the major content of this user guide);
3. Enable *FMCW Generator*, *baseband core*, and *baseband core*'s IRQs;
4. Trigger system by writing 0 to `DMU_FMCW_START` (`0xBA_0018`);
5. After getting IRQs, CPU will:
  - a. Clear corresponding IRQs;
  - b. Move hardware results out for further processing, such as tracking;

- c. Trigger the next FMCW frame<sup>3</sup>;
- d. Continue its further process.

A general framework is shown below:

```

void write_reg(uint32_t *address, uint32_t val)
{
    *address = val;
}

int irq_bb()
{
    release_hw_token();
}

int timer_bb()
{
    release_timer_token();
}

void main_task()
{
    init_fmcw_radio();
    init_baseband_core();
    enable_fmcw_radio();
    enable_baseband_core();
    set_timer(50); // release token every 50ms
    write_reg(0xba0018, 1); // kick off FMCW
    while(true) {
        if (wait_for_timer_token()) {
            if (wait_for_hw_token()) {
                read_hw_result_out();
                write_reg(0xba0018, 1); // kick off FMCW
                do_remaining_process();
            }
        }
    }
}

```

There are some comments on the above code:

- Function `write_reg` is nothing but access baseband register;
- Functions `irq_bb` and `timer_bb` are handlers of baseband-done IRQ and baseband timer, respectively. Note that *baseband core* does not provide hardware timer. One can use hardware timers inside the chip or software timers provided by the real-time operating system (RTOS);
- Functions `release_hw_token` and `wait_for_hw_token` are a pair that control the access to *baseband core*. If `wait_for_hw_token` is called before `release_hw_token`, the corresponding task will be blocked until the `release_hw_token` is called. Meanwhile, the task yields its CPU occupation. Functions `wait_for_hw_token` and `wait_for_timer_token` have the similar meaning except that `wait_for_timer_token` controls the update rate.
- The implementation depends on the underlying RTOS. If you do not have an RTOS in your design, you need to follow similar principles to make *baseband core* desired.

<sup>3</sup> This can be done by writing 1 to DMU\_FMCW\_START (0xBA\_0018). Or more elegantly, first, grant the permission by setting a certain flag and write 1 to DMU\_FMCW\_START (0xBA\_0018) only when a pre-programmed timer is expired and the flag is set.

## 3.6 Examples

### 3.6.1 Configuration for Basic Radar Baseband Signal Processing

The following gives two typical baseband configuration examples that involve basic radar signal processing.

#### 3.6.1.1 Example 1

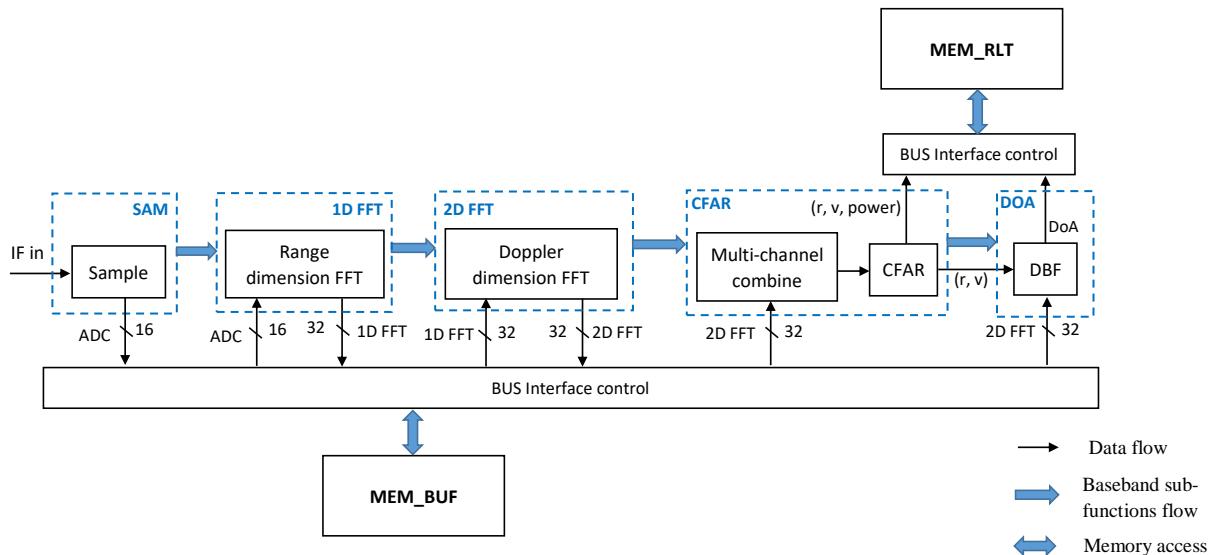


Fig. 3.7: Baseband Process Flow When CFG\_SAM\_SINKER is Set to 0

In Example 1, the baseband workflow is illustrated in Fig. 3.7. By setting the bit register CFG\_SAM\_SINKER to 0, ADC data in one frame are first stored in MEM\_BUF, which is a 2M RAM. Then 1D-FFT, 2D-FFT, CFAR, and DoA processes take place in sequence. During these processes, the generated ADC data, 1D-FFT data, 2D-FFT data are all stored in MEM\_BUF. Due to the limited size of MEM\_BUF, newly generated data in the current stage will overwrite data stored in MEM\_BUF in the previous stage. The object information generated in the CFAR and DoA stages is stored in MEM\_RLT as shown in Fig. 3.7. One thing to note is that the range-dimension FFT and Doppler-dimension FFT processes are conducted in sequence by the only one FFT hardware engine.

So in this example:

- The bit register CFG\_SAM\_SINKER is set to 0.
- The resulting value for CFG\_SYS\_ENABLE is 0xF4.

When CFG\_SAM\_SINKER is set to 0, 1D-FFT must be enabled.

Table 3.5 lists the register settings of Example 1.

Table 3.5: Registers Settings of Example 1

Register Name	Address	Register Value	Corresponding Parameters
CFG_SAM_SINKER	0x400	0	<p>ADC sampled data in one frame is all fed into MEM_BUF in SAM subfunction.</p> <p>This setting is only used under debugging mode, for collecting ADC data in MEM_BUF and then dumping ADC data outside the device through peripheral interfaces.</p>
CFG_SYS_ENABLE	0x18	0xF4	Baseband enabled subfunctions: SAM, 1D-FFT, 2D-FFT, CFAR, and DoA.
CTL_SYS_START	0x0	1	Start baseband subfunctions in the order of their corresponding enable bits in CFG_SYS_ENABLE from low to high

### 3.6.1.2 Example 2

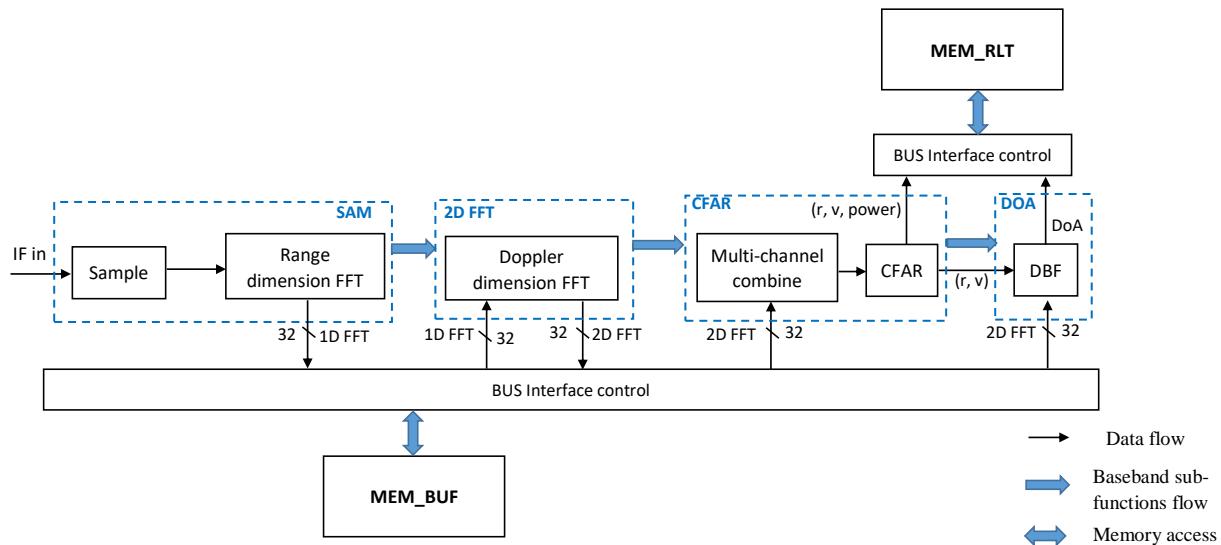


Fig. 3.8: Baseband Process Flow When CFG\_SAM\_SINKER is Set to 1

In Example 2, the baseband workflow is illustrated in Fig. 3.8. By setting the bit register CFG\_SAM\_SINKER to 1, ADC sampled data in one chirp is directly sent to FFT engine for range-dimension FFT while ADC starts to sample the next chirp. This means the **SAM stage** includes the sample and 1D-FFT processes, as shown in Fig. 3.8. So when the SAM stage is over, 1D-FFT data of the current frame is already stored in MEM\_BUF for Doppler-dimension FFT, which takes place in the 2D-FFT stage. This pipeline scheme in the SAM stage can greatly reduce the baseband processing time.

In this example:

- The bit register CFG\_SAM\_SINKER is set to 1.
  - CFG\_SYS\_ENABLE is set to 0xE4.

### 3.6. Examples

---

Since range-dimension FFT is operated in the SAM stage, the 1D-FFT stage should **not** be enabled to avoid duplicated operation. The enabled stages should be SAM, 2D-FFT, CFAR, and DoA.

Table 3.6 lists the register settings of Example 2.

Table 3.6: Registers Settings of Example 2

Register Name	Address	Register Value	Corresponding Parameters
CFG_SAM_SINKER	0x400	1	ADC sampled data are directly fed into FFT engine for range-dimension FFT computation in SAM subfunction.
CFG_SYS_ENABLE	0x18	0xE4	Baseband enabled subfunctions: SAM, 2D-FFT, CFAR, DoA.
CTL_SYS_START	0x0	1	Start baseband subfunctions in the order of their corresponding enable bits in CFG_SYS_ENABLE from low to high



## FMCW WAVEFORM GENERATOR

### 4.1 Overview

The FMCW Waveform Generator is based on a fractional-N phase-locked loop (PLL) that adopts a frequency synthesizer-N structure with a delta-sigma modulator. It is a common practice that the frequency synthesizer-N is implemented with analog circuits, while the delta-sigma modulator is implemented with digital circuits. The output of the delta-sigma modulator contains the necessary information about the frequency sweep bandwidth and duration of an FMCW waveform. The output is then passed to the frequency synthesizer-N. After being synthesized by the frequency synthesizer-N, an FMCW waveform is generated, as shown in Fig. 4.1.

A frequency synthesizer-N usually consists of a phase frequency detector (PFD), a charge pump (CP), a loop filter (LPF), a voltage-controlled oscillator (VCO), and a multi-modulus divider (MMD).

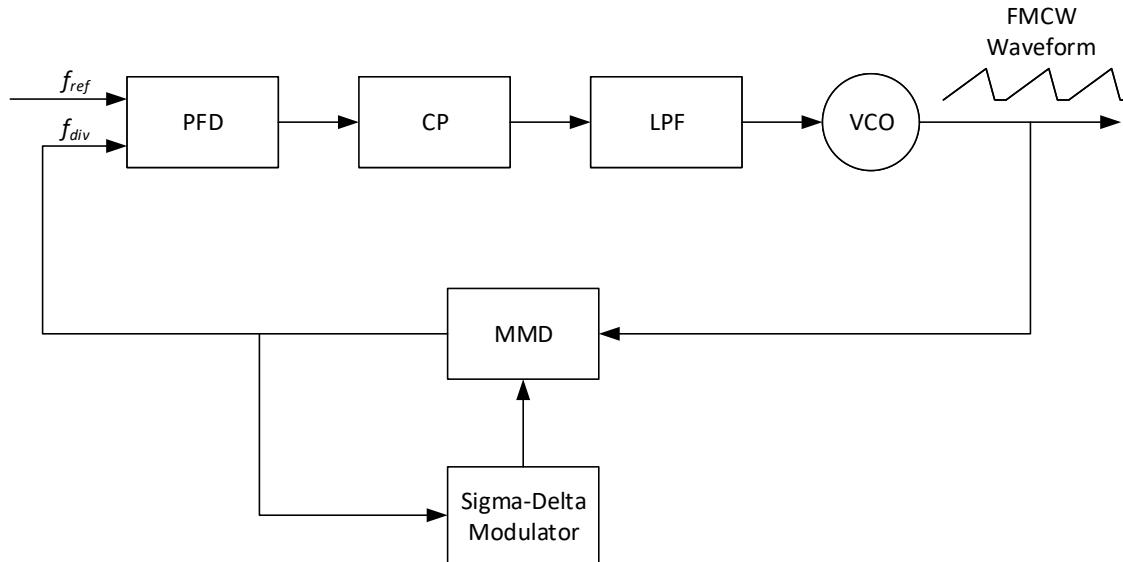


Fig. 4.1: Structure of FMCW Waveform Generator

### 4.2 Features

FMCW waveform generator in Calterah Alps radar chip is embedded with the following features for better radar ranging and anti-interference. All these features can be programmed and enabled independently.

- Virtual Array Mode (VAM)

VAM controls the on/off status and phase status of each TX channel, according to the user's configuration.

- Frequency Hopping Mode (FH)

FH enables the user to get different carrier frequencies and bandwidths for two adjacent chirps in the same frame.

- Phase Scramble Mode (PS)

PS provides the capability to rotate the phase by a degree of 180 in TX for each rotate.

- Chirp Shifting Mode (CS)

CS shifts both the up ramp and down ramp by a specific time inside a chirp, while the total duration of one chirp keeps unchanged.

- Anti Velocity Ambiguity with Chirp Delay (Anti-VELAMB CD)

Anti-VELAMB CD extends some chirps' duration by a specific time, which is used for resolving velocity ambiguity.

- Auto Gain Control (AGC)

AGC adds 3 chirps at the beginning of each frame, and at the same time, hands over the gain control over RXBB in the radio part from radio registers to CPU.

- Frame Interleaving

FMCW waveform generator supports different frame types. Interleaving different types of frames enables achieving different applications simultaneously.

### 4.3 Functional Description

#### 4.3.1 Basic Chirp Cell in FMCW

A basic FMCW chirp cell is defined by the parameters listed in Table 4.1.

Table 4.1: Chirp Parameters

Parameters	Denotation
Start frequency	$F_{start}$
Stop frequency	$F_{stop}$
Step up frequency	$F_{stepup}$
Step down frequency	$F_{stepdown}$
Time to cover ramp down and idle state	$T_{idle}$

A more intuitive illustration is shown in Fig. 4.2 (where  $T_{ref}$  is the period of the clock for FMCW generator, meaning the time each up step or down step takes).



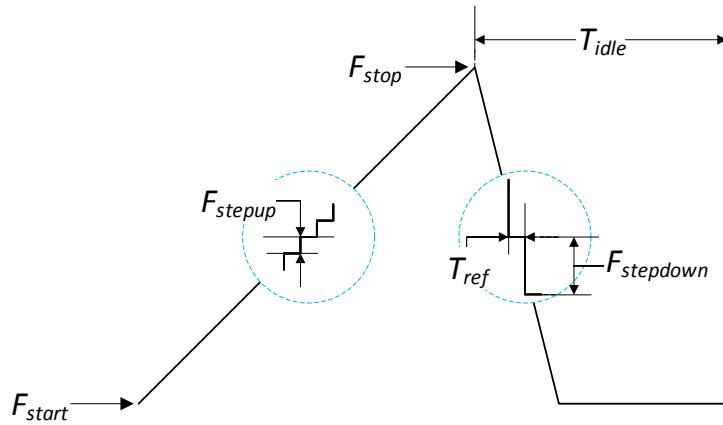


Fig. 4.2: A Basic FMCW Chirp Cell

### 4.3.2 Virtual Array Mode (VAM)

VAM supports time division modulation (TDM) and binary phase modulation (BPM).

- TDM is achieved by automatically turning on or off TX channels every a fixed period. Each TX can be auto-controlled individually, as shown in Fig. 4.3, where 1 stands for on, and 0 stands for off.

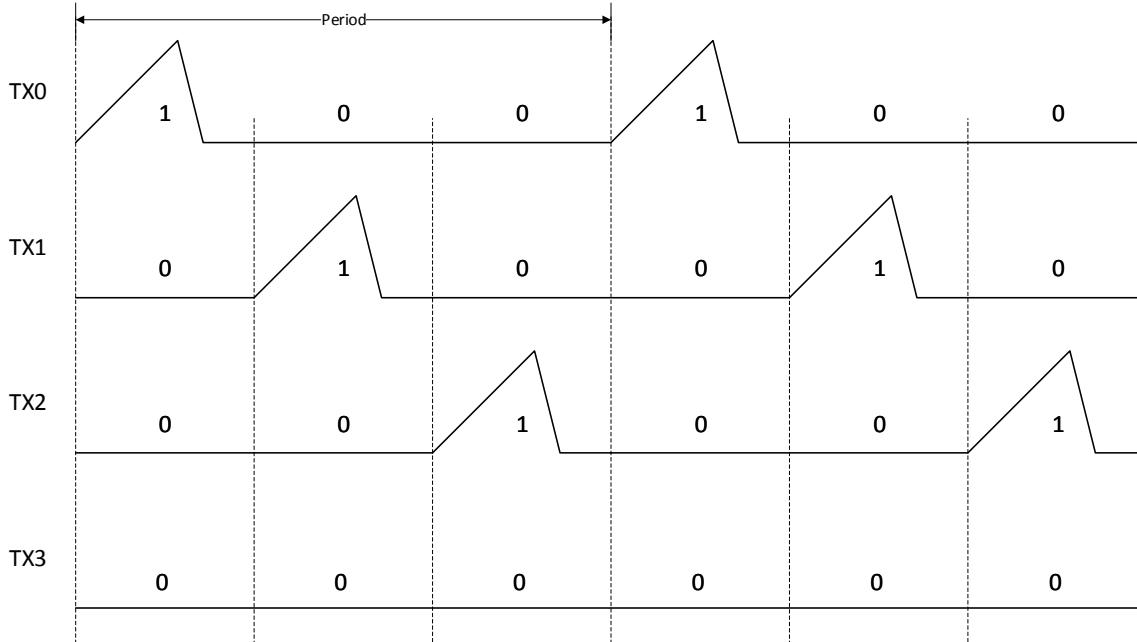


Fig. 4.3: Time Division Modulation

- BPM is achieved by rotating the phase by  $\pi$ -phase, as shown in Fig. 4.4, where 1 stands for without phase

### 4.3. Functional Description

---

rotate, and -1 stands for rotation by  $\pi$ .

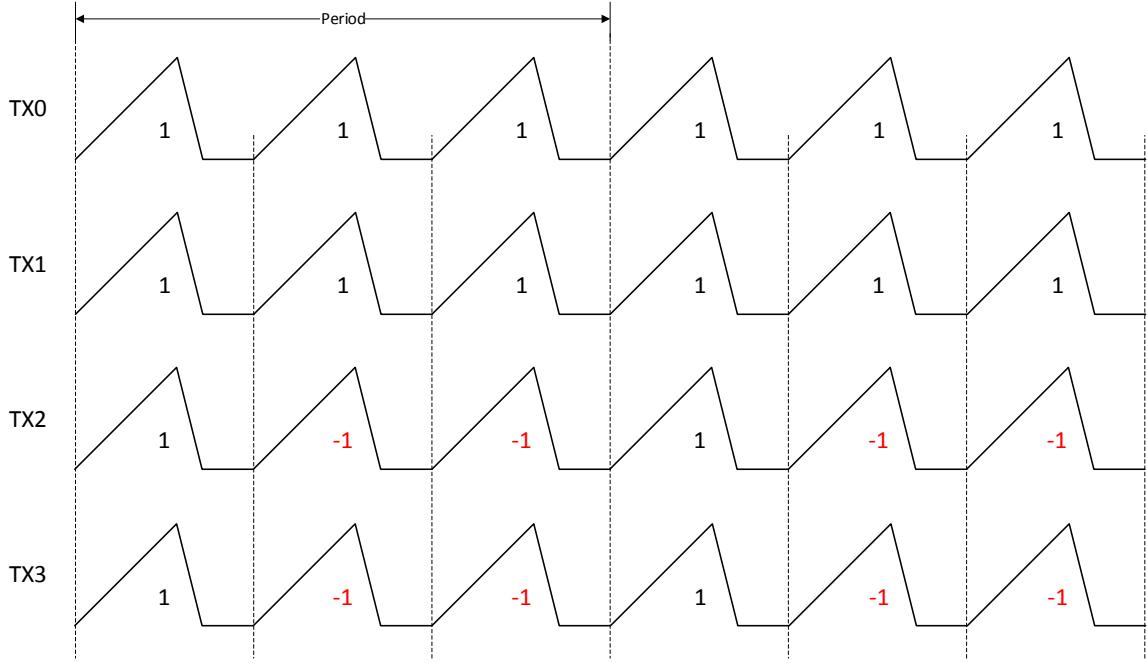


Fig. 4.4: Binary Phase Modulation

#### 4.3.3 Frequency Hopping Mode (FH)

In FH, two adjacent chirps switch smoothly from one chirp type to another chirp type, or keep the same chirp type. Chirp type is controlled by the output of a 32-bit XOR chain. If the output is 0, chirp type will be type-0. If the output is 1, chirp type will be type-1, as shown in Fig. 4.5.

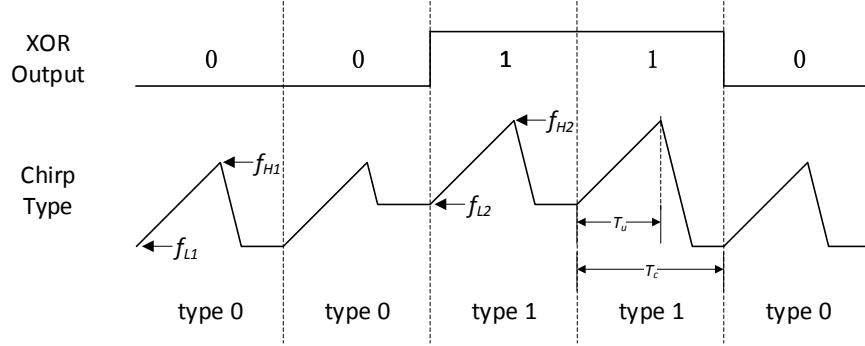


Fig. 4.5: Frequency Hopping Mode

Lower side start frequency ( $f_L$ ), higher side stop frequency ( $f_H$ ), up ramp time ( $T_u$ ), and chirp time ( $T_c$ ) are fully programmable, but for better usage, those parameters should meet all of the following requirements:

1.  $f_{H1} - f_{L1} = f_{H2} - f_{L2}$
2.  $T_c$  is constant within the same frame.
3.  $T_u$  is constant within the same frame.

#### 4.3.4 Phase Scramble Mode (PS)

In PS mode, random phase rotation is applied across all TX channels for interference mitigation. In mathematics, the TX signal is multiplied by 1 or -1 randomly, which is controlled by another 32-bit XOR chain. 1 means in-phase, and -1 means opposite phase. If the XOR chain outputs 0, TX will be in-phase. If the XOR chain outputs 1, TX will be opposite-phase, as shown in Fig. 4.6.

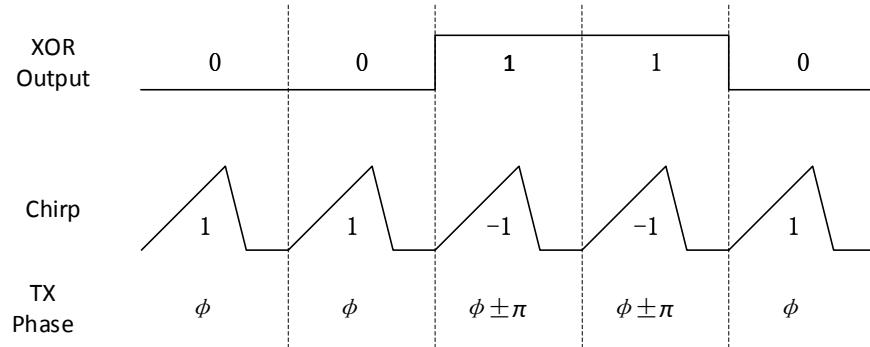


Fig. 4.6: Phase Scramble Mode

#### 4.3.5 Chirp Shifting Mode (CS)

In CS, a 32-bit XOR chain determines whether to shift the up and down ramps or not. Fig. 4.7 illustrates the difference between normal chirps and chirps in chirp shifting mode. If the XOR chain outputs 1, the ramps will be shifted by  $\delta$ , a delay period in 32 bits and programmable. But note that the shape of shifted chirps is the same with that of normal chirps, which means that  $F_L$ ,  $F_H$ ,  $T_c$ , and  $T_u$  remain.

### 4.3. Functional Description

---

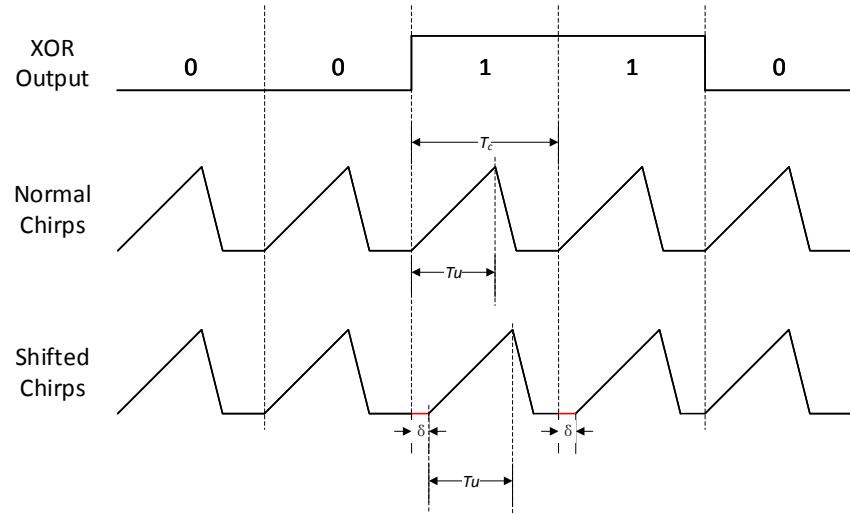


Fig. 4.7: Chirp Shifting Mode

#### 4.3.6 Anti Velocity Ambiguity with Chirp Delay (Anti-VELAMB CD)

Anti-VELAMB CD updates FMCW waveform by extending some chirps' duration by a specific time.

Anti-VELAMB CD has some interactions with VAM. When VAM is off, the transmitted signal is as shown in Fig. 4.8. The odd chirp is always extended by an additional delay  $a$ , which is 32-bit and programable. To simplify the description, let  $P_{CD}$  denote a period of this mode. For the waveform in Fig. 4.8,  $P_{CD} = 2$ .

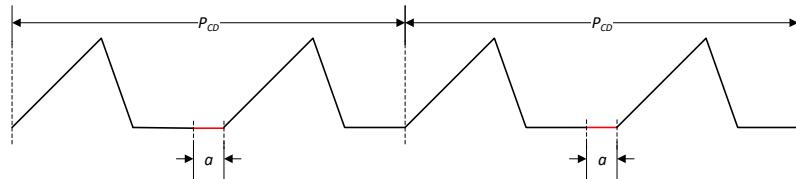


Fig. 4.8: Anti Velocity Ambiguity with Chirp Delay Mode

### 4.3.7 Auto Gain Control (AGC)

In AGC mode, 3 additional chirps are added before each traditional FMCW frame, as shown in Fig. 4.9.

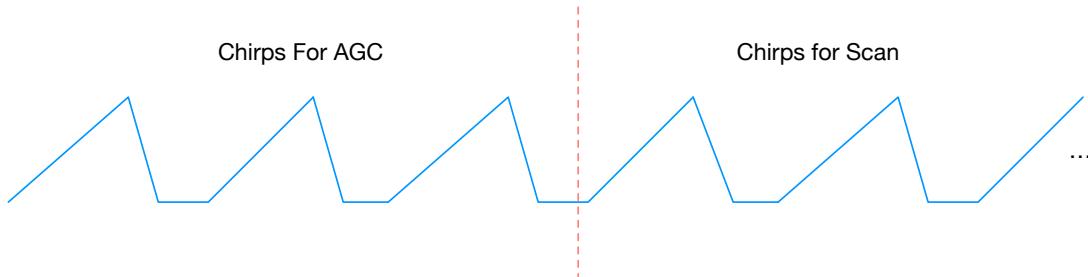


Fig. 4.9: AGC Mode

### 4.3.8 Frame Interleaving

Frame interleaving enables the radar chip to generate different frame types without having to configure the FMCW generator multiple times. When the feature is enabled, different frame types are automatically interleaved based on the start signal, as shown in Fig. 4.10.

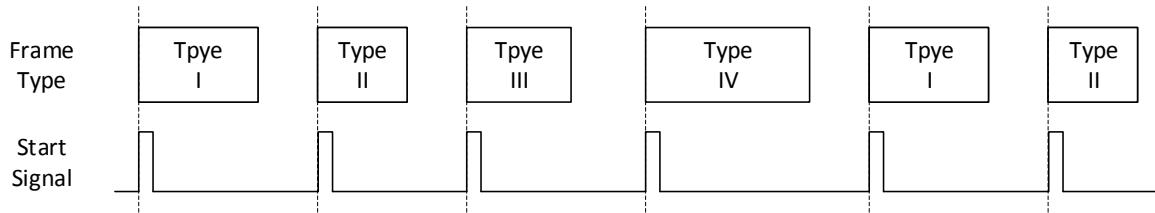


Fig. 4.10: Frame Interleaving Mode

### 4.3.9 Programmable Commands for Chirps

During the generation of FMCW waveforms, a convenient way to control radio frequency and analog registers is provided through a LUT (Look Up Table). This LUT refers to Register Bank 4. The corresponding commands will be applied at the specified time to the specified radio frequency and analog register address.

Three groups of parameters are provided for command reading and register writing.

- cmd\_num\_1, timer\_x
- cmd\_num\_2, timer\_y
- cmd\_num\_3, timer\_z

cmd\_num\_1, cmd\_num\_2, cmd\_num\_3 are for the users to define the number of commands to send each time. And timer\_x, timer\_y, timer\_z are used to define the time when to send the commands, relative to the starting point of the up ramp.

Once the parameters are configured, the chip takes these three groups of commands from the register bank for command reading and register writing. Commands are repeated with the cycle of cmd\_period.

## 4.4. Programming Interfaces

For the fast settling of HP1 (the first stage of high pass filter in RXBB), HP2 (the second stage of high pass filter in RXBB), and the bandwidth of PLL, a single enable register bit for each of them is available, instead of controlling them with the LUT. The time when to launch the fast settling can be configured, as shown in Fig. 4.11.

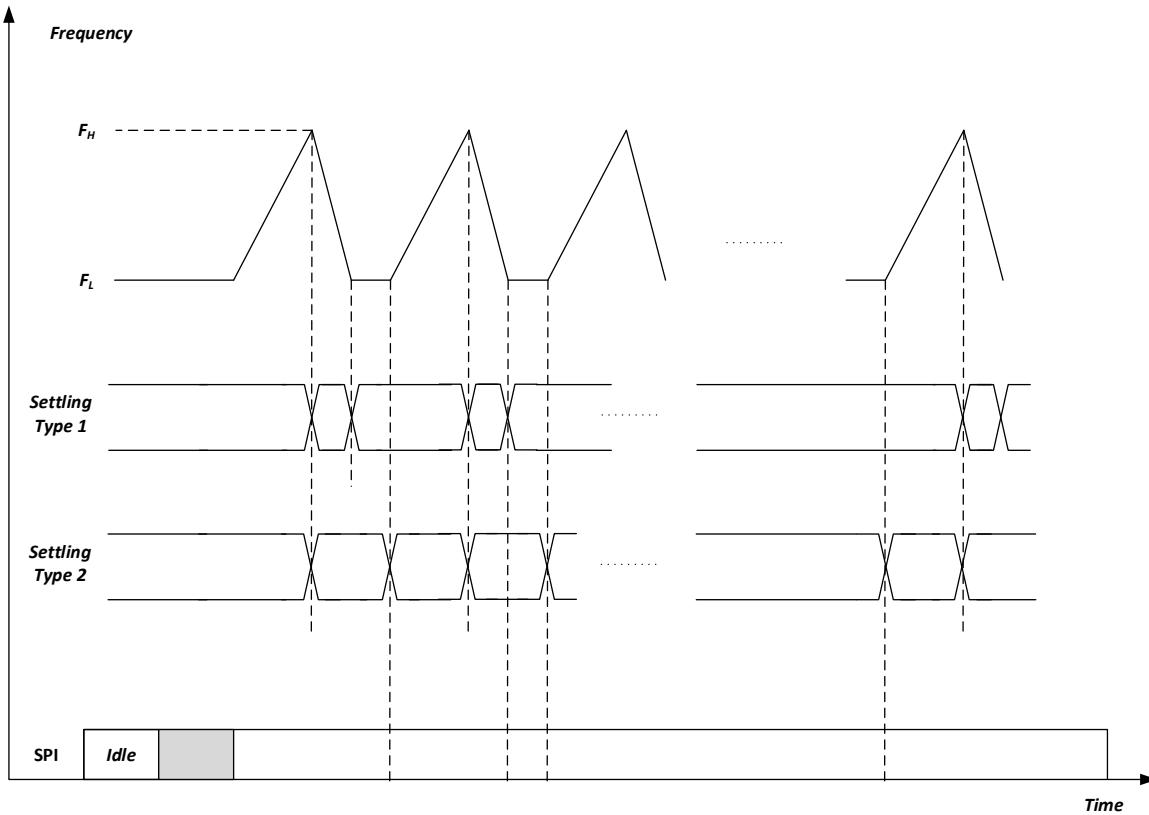


Fig. 4.11: Fast Settling during FMCW

## 4.4 Programming Interfaces

### 4.4.1 How to Access FMCW Related Registers

FMCW related registers are in the radio part, and they can be accessed by CPU.

To access FMCW related registers:

1. Switch the control source of radio registers to CPU.

The topology between radio registers and CPU are shown in Fig. 4.12. CPU not only controls the multiplexer selection, but also controls radio registers.

- If CPU writes 0x0000\_0002 (data) to 0xba0000 (address), CPU is selected to control radio registers.
- If CPU writes 0x0000\_0000 (data) to 0xba0000 (address), external pads are selected to control radio registers.

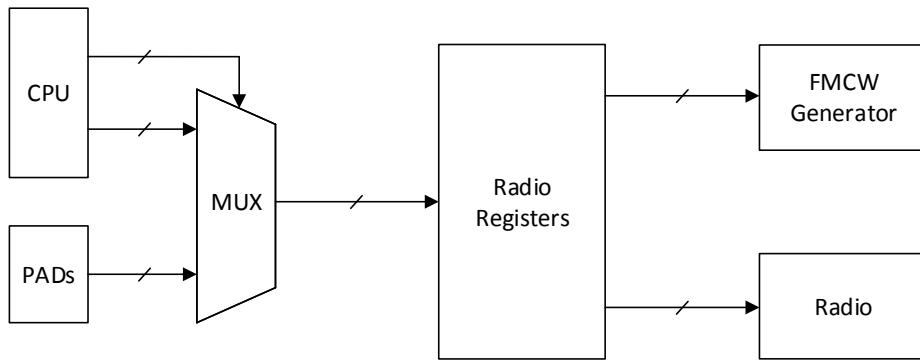


Fig. 4.12: Topology between radio registers and CPU

2. Select the bank of FMCW related registers.

Radio registers are sorted by bank. Each bank has 128 addresses, with each address being an 8-bit register. The first address is used for bank selection, as shown in Fig. 4.13. Write and read operations should firstly switch to the correct bank, and FMCW related registers are located in Bank 3, 5, 6, 7 and 8.

Address	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0					bk_sel[3]	bk_sel[2]	bk_sel[1]	bk_sel[0]
1	reg_1[7]	reg_1[6]	reg_1[5]	reg_1[4]	reg_1[3]	reg_1[2]	reg_1[1]	reg_1[0]
2	reg_2[7]	reg_2[6]	reg_2[5]	reg_2[4]	reg_2[3]	reg_2[2]	reg_2[1]	reg_2[0]
...	...	...	...	...	...	...	...	...
127	reg_127[7]	reg_127[6]	reg_127[5]	reg_127[4]	reg_127[3]	reg_127[2]	reg_127[1]	reg_127[0]

Fig. 4.13: Layout for Radio Registers

3. Write the desired value to the corresponding address of a radio register, by sending the combined information to the address 0xba0004. The combined data format is shown in Fig. 4.14

0xa0004				
Bit	31:16	15	14:8	7:0
Description	0 Unused	1 for writing 0 for reading	Address	Data

Fig. 4.14: Data Format for Accessing Radio Registers

4. Read the desired value from the corresponding address of the radio register, by sending the combined information to address 0xa0004.

## 4.4. Programming Interfaces

---

After that, the register value is stored at address 0xba0008, and the returned data format is shown in Fig. 4.15

0xba0008		
Bit	31:8	7:0
Description	0 Unused	Data

Fig. 4.15: Data Format for Returned Values of Radio Registers

### 4.4.1.1 Example

If you want to write data 0xAA to the address 0x05 at Bank 3 in radio registers, follow these steps:

1. Change the control source of radio registers to CPU, by writing 0x0000\_0002 to the address 0xba\_0000.
2. Select Bank 3 of radio registers, by writing 0x0000\_8003 to the address 0xba\_0004.
3. Write the desired value of 0x0000\_85AA to address 0xba\_0004.
4. Read back the register value, by writing 0x0000\_0500 to the address 0xba\_0004.
5. Confirm the value, by reading 0xba\_0008.

## 4.4.2 Basic Functions

### 4.4.2.1 Generating Basic FMCW Frames

To generate an FMCW frame, one should configure the chirp number and parameters that define a basic chirp, as shown in Table 4.2 .

Table 4.2: Parameters for a Basic FMCW Frame

Parameters	Denotation	Bank Number	Address	Width (Bits)
Start frequency	$F_{start}$	5	5 - 8	32
Stop frequency	$F_{stop}$	5	9 - 12	32
Step up frequency	$F_{stepup}$	5	13 - 16	28
Step down frequency	$F_{stepdown}$	5	17 - 20	28
Time to cover ramp down and idle state	$T_{idle}$	5	21 - 24	32
Chirp number	$N_{chirp}$	5	25 - 26	16
Mode selection	Mode_sel	3	44	3

A basic FMCW frame with the parameters  $F_{start}$ ,  $F_{stop}$ ,  $F_{stepup}$ ,  $F_{stepdown}$ ,  $T_{idle}$ ,  $N_{chirp}$  is shown in Fig. 4.16.



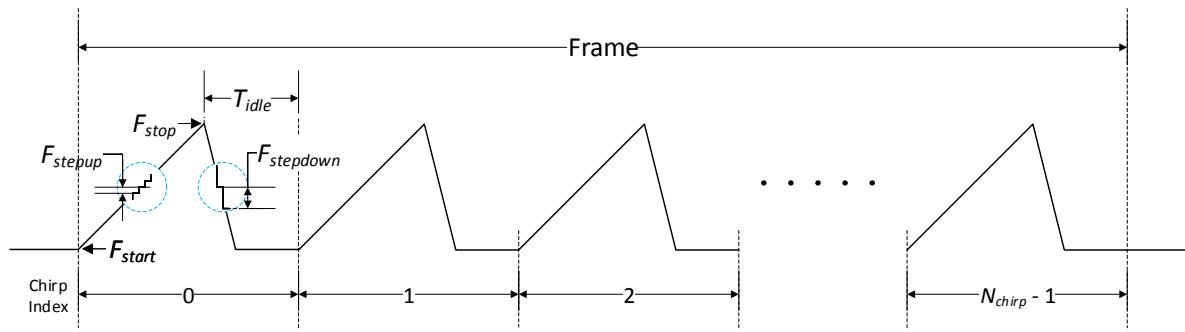


Fig. 4.16: A Basic FMCW Frame

#### 4.4.2.2 Starting FMCW Frame

FMCW generator can be triggered by a start signal from different sources, PAD, CPU, or radio registers, shown in Fig. 4.17.

The start signals from the CPU and the start PAD are first multiplexed, controlled by the CPU. And then the output is further get multiplexed with the start signal from radio registers, controlled by radio registers.

The start signal from either the CPU or the start PAD triggers FMCW generator with the rising edge. The start signal from radio registers triggers FMCW generator with logic high level.

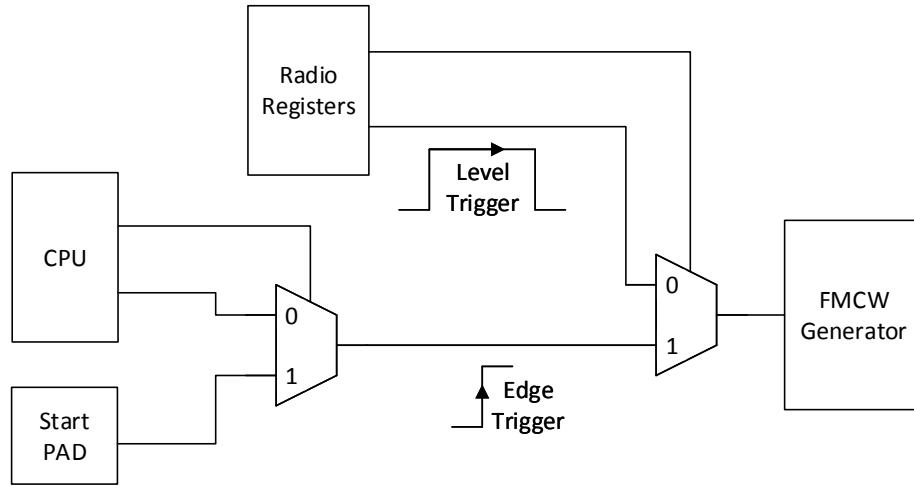


Fig. 4.17: FMCW Start Source

With the basic frame, three kinds of FMCW waveforms could be obtained:

- Continuous waveform

In this situation, FMCW generator will not stop generating waveform until the start signal is invalid.

## 4.4. Programming Interfaces

---

To obtain full control over the waveform, the start signal should come from a radio register that is level triggered, as shown in Fig. 4.18.

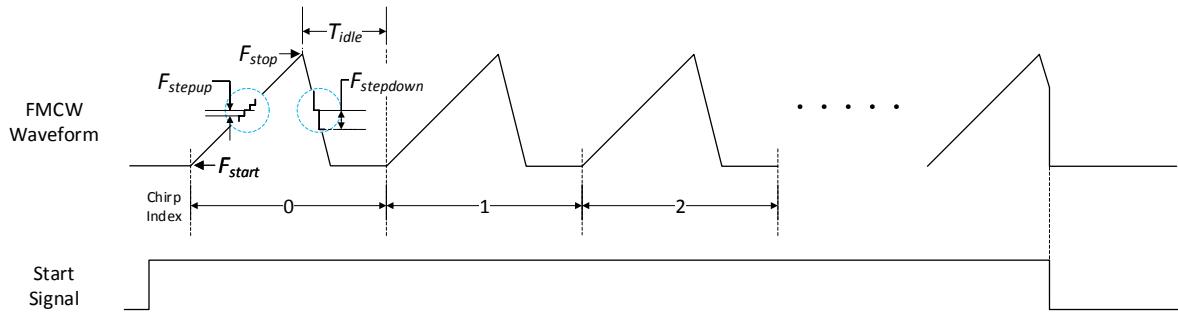


Fig. 4.18: FMCW Continuous Waveform

Detailed configuration to obtain a continuous waveform is listed in Table 4.3.

Table 4.3: Configuration for Generating a Continuous Waveform

Step	Parameters	Bank Number	Address	Width (Bits)	Value
1	Start frequency	5	5 - 8	32	$F_{start}$
2	Stop frequency	5	9 - 12	32	$F_{stop}$
3	Step up frequency	5	13 - 16	28	$F_{stepup}$
4	Step down frequency	5	17 - 20	28	$F_{stepdown}$
5	Time to cover ramp down and idle state	5	21 - 24	32	$T_{idle}$
6	Chirp number	5	25 - 26	16	0
7	Mode selection	3	44	3	0x2
8	Start generation	3	45	5	0x15
9	Stop generation	3	45	5	0

- Waveform by frame

Compared to continuous waveform, to generate a waveform by frame, the parameter  $N_{chirp}$  should be set to the exact number of chirps that a frame contains, and the start signal should come from PAD or CPU that is rising edge triggered, as shown in Fig. 4.19.

Each trigger will get a frame of FMCW waveform, containing  $N_{chirp}$  chirps. As soon as one frame is transmitted, the FMCW generator stops and waits for another trigger.

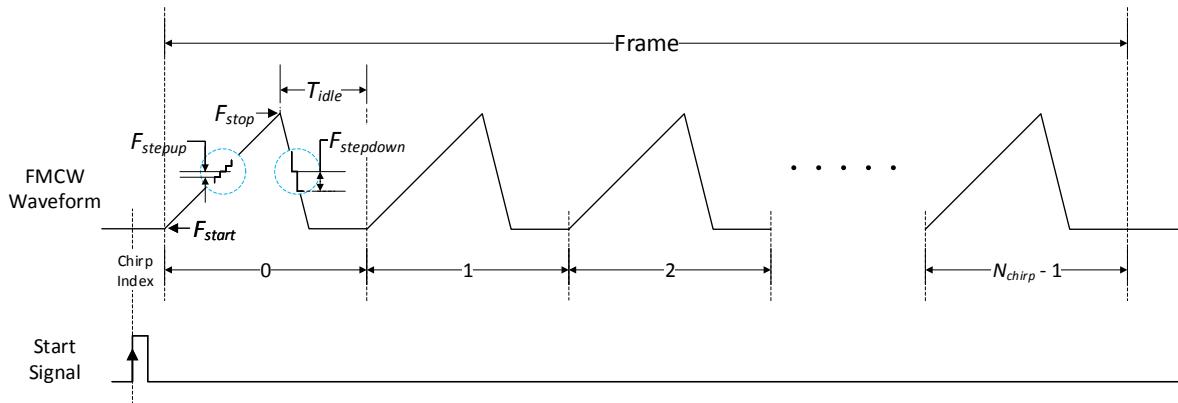


Fig. 4.19: FMCW Waveform Generated by Frame

Detailed configuration to generate waveform by frame with  $N_{chirp}$  chirps in a frame is listed in Table 4.4,

Table 4.4: Configuration for Generating Waveform by Frame

Step	Parameters	Bank Number	Address	Width (Bits)	Value
1	Start frequency	5	5 - 8	32	$F_{start}$
2	Stop frequency	5	9 - 12	32	$F_{stop}$
3	Step up frequency	5	13 - 16	28	$F_{stepup}$
4	Step down frequency	5	17 - 20	28	$F_{stepdown}$
5	Time to cover ramp down and idle state	5	21 - 24	32	$T_{idle}$
6	Chirp number	5	25 - 26	16	$N_{chirp}$
7	Mode selection	3	44	3	0x2
8	Start selection from CPU or PAD	3	45	5	0x16
9	Start generation	1). From CPU a). Write 0x0 to 0xba_0014 to select CPU b). Write 0x1 to 0xba_0018 to start generating or, 2). From PAD a). Write 0x1 to 0xba_0014 to select PAD b). Write 0x2 to 0xba_023c to MUX FMCW start signal c). Stimulate a pulse with the rising edge to start generation			
10	Stop generation	Automatically stop generating after transmitted $N_{chirp}$ number of chirps			

- Single tone

Single tone means generating a waveform with a fixed frequency, as shown in Fig. 4.20. Only  $F_{start}$  is needed for a chirp. Since it can be seen as a special kind of continuous waveform, the start signal source should be switched to *radio registers*. FMCW generator stops the transmission when the start signal becomes invalid.

$T_{settling}$  in Fig. 4.20 denotes the settling time that the radio part takes to settle to the correct state after each start

## 4.4. Programming Interfaces

---

signal, no matter where the start signal comes from.

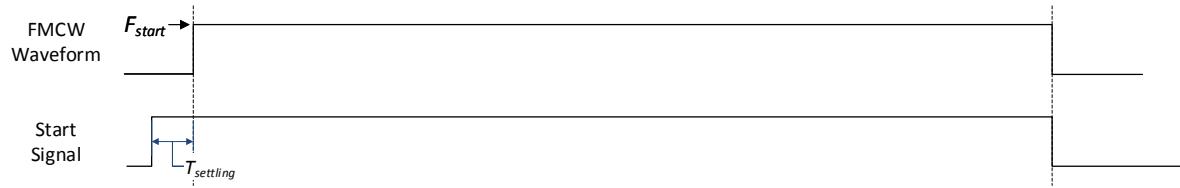


Fig. 4.20: Single Tone Waveform

Detailed configuration to obtain a single-tone waveform with  $F_{start}$  is listed in [Table 4.5](#),

Table 4.5: Configuration for Generating a Single-Tone Waveform

Step	Parameters	Bank Number	Address	Width (Bits)	Value
1	Start frequency	5	5 - 8	32	$F_{start}$
2	Mode selection	3	44	3	0x0
3	Start generation	3	45	5	0x15
4	Stop generation	3	45	5	0

### 4.4.3 Advanced Features

#### 4.4.3.1 Virtual Array Mode (VAM)

In VAM, the status of a TX channel at a particular chirp can be any of the following:

- in-phase
- opposite-phase
- off

Along with the generation of FMCW waveform, the status of a TX channel is updated periodically. It is controlled by the configuration of registers shown in [Table 4.6](#) for that TX channel.

Table 4.6: FMCW Related Registers for VAM

Register	Width	Description
FMCW_VAM_CTL	8	divided into 4 groups 2'b00 -> off 2'b01 -> in phase 2'b10 -> opposite phase
FMCW_VAM_PRD	2	Period for VAM $FMCW\_VAM\_PRD = \text{Period} - 1$
FMCW_VAM_ENA	1	Enable for VAM 0 disable 1 enable

As shown in Fig. 4.21, FMCW\_VAM\_CTL is an 8-bit register that is divided into 4 groups, with each 2-bit group indicating the TX channel's status at a particular chirp. ( 2'b00 means off, 2'b01 means in-phase, 2'b10 means opposite-phase, and 2'b11 is unused.)

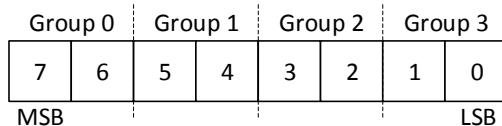


Fig. 4.21: TX Status Controlling Groups in VAM

FMCW\_VAM\_PRD is a 2-bit register that specifies the period  $P$  of repeating the groups of FMCW\_VAM\_CTL. Valid values of  $P$  are 1, 2, 3, and 4. The value of  $P$  indicates the number of groups within a period. The groups are repeated with the generation of chirps by period.

$P$  also determines the particular groups and their sequence within a period. Within a frame, the  $n$  th chirp at the TX channel is controlled by Group [ $n$  mode  $P$ ].

- If  $P = 1$ , the controlling group sequence for the TX channel is shown in Fig. 4.22.

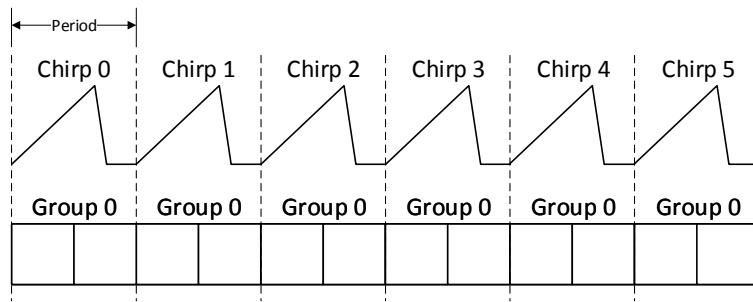


Fig. 4.22: Controlling Group Sequence When  $period = 1$  in VAM

- If  $P = 2$ , the controlling group sequence for the TX channel is shown in Fig. 4.23.

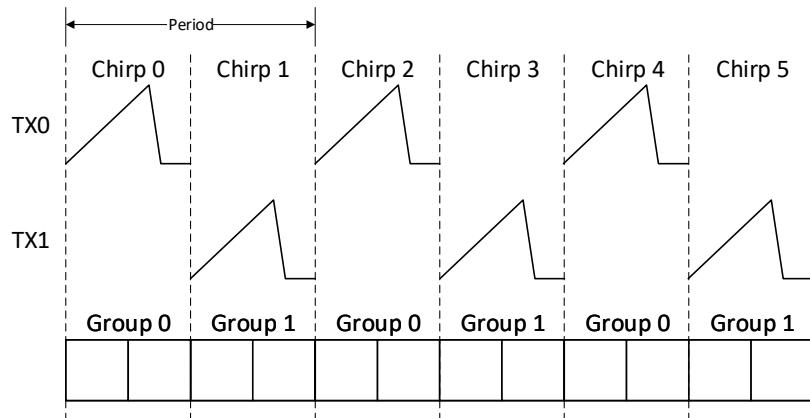
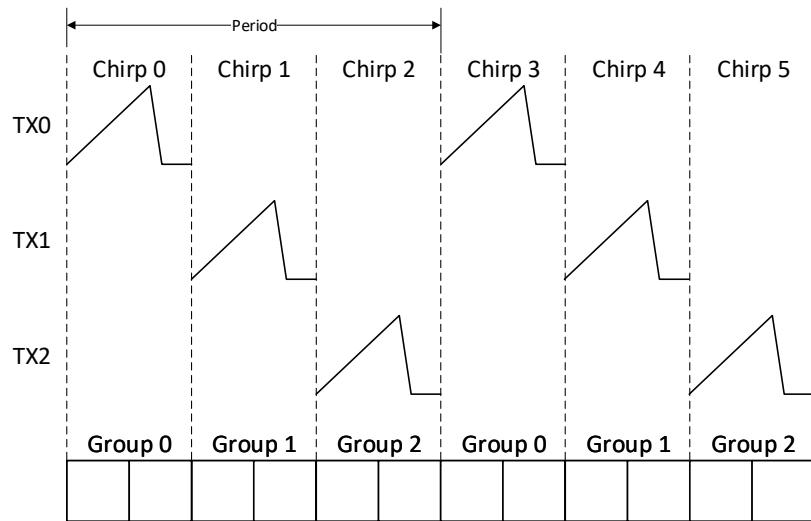
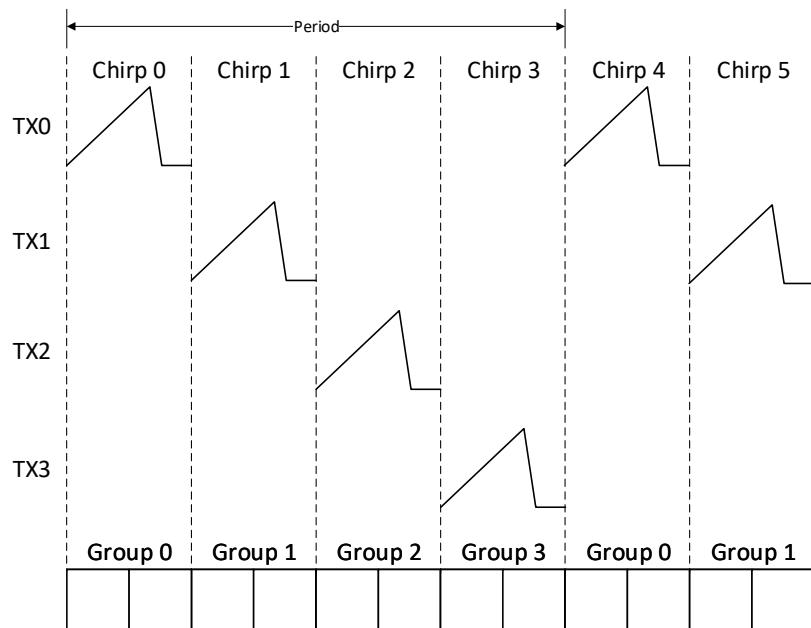


Fig. 4.23: Controlling Group Sequence When  $period = 2$  in VAM

- If  $P = 3$ , the controlling group sequence for the TX channel is shown in Fig. 4.24.

Fig. 4.24: Controlling Group Sequence When  $period = 3$  in VAM

- If  $P = 4$ , the controlling group sequence for the TX channel is shown in Fig. 4.25

Fig. 4.25: Controlling Group Sequence  $period = 4$  in VAM

### 4.4.3.2 Frequency Hopping Mode (FH)

As shown in Fig. 4.26, an XOR chain controls the chirp type.

1. A 32-bit initial state is first loaded into a 32-bit shift register.
2. Then both the shift register and a 32-bit tap are passed to an AND gate, which produces a 32-bit shadow register.
3. The shadow register is then processed by a reduction XOR logic, which outputs a 1-bit result.
4. The result is fed back to the MSB of the shift register, which controls the chirp type with its LSB.

Both the 32-bit state and 32-bit tap are in the radio register domain, and the output of the XOR chain will be updated at the end of each chirp. If the XOR chain outputs 0, the chirp type would be type-0. If the XOR chain outputs 1, the chirp type would be type-1.

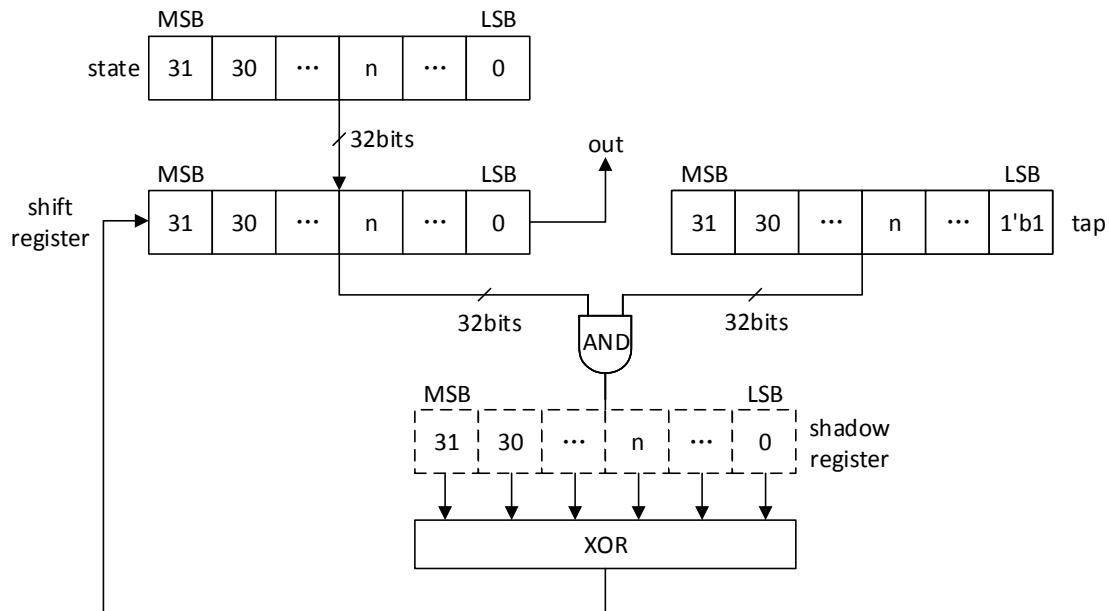


Fig. 4.26: 32-Bit XOR Chain

There is another 1-bit register indicating whether to reset the XOR chain to the initial state at the beginning of each frame. 1 means to reset the XOR chain to the initial state.

Registers for FH are summarized in Table 4.7.

Table 4.7: Registers for FH

Register	Width	Description
FMCW_FH_ENA	1	0 disable 1 enable
FMCW_FH_TAP	32	XOR-chain tap. e.g. set it to 0x00000407
FMCW_FH_STA	32	XOR-chain initial state after reset is released. e.g. set it to 0x12348765
FMCW_FH_RNM	1	XOR-chain running mode 1: reset at the beginning of each frame 0: continuous running

#### 4.4.3.3 Phase Scramble Mode (PS)

While VAM not only controls the status of each TX channel but also the phase rotator of each channel, PS only controls the phase rotator and the status of all the phase rotators are based on one XOR chain. The structure of the XOR chain is exactly the same as in frequency hopping mode, illustrated by Fig. 4.26, but this XOR chain is controlled by another group of registers described in Table 4.8.

Table 4.8: Registers for Phase Scramble Mode

Register	Width	Description
FMCW_PS_ENA	1	0 disable 1 enable
FMCW_PS_TAP	32	XOR-chain tap. e.g. set it to 0x00000401
FMCW_PS_STA	32	XOR-chain initial state after reset is released. e.g. set it to 0xdeadbeaf
FMCW_PS_RNM	1	XOR-chain running mode 1: reset at the beginning of each frame 0: continuous running



### 4.4.3.4 Chirp Shifting Mode (CS)

Another 32-bit XOR chain controls whether to shift the up ramp and down ramp inside a chirp or not. The structure of the XOR chain is also the same as in frequency hopping mode and phase scramble mode.

Table 4.9: Registers for Chirp Shifting Mode

Register	Width	Description
FMCW_CS_ENA	1	0 disable 1 enable
FMCW_CS_TAP	32	XOR-chain tap, e.g. set it to 0x00000409
FMCW_CS_STA	32	XOR-chain initial state after reset is released, e.g. set it to 0x87654321
FMCW_CS_RNM	1	XOR-chain running mode 1: reset at the beginning of each frame 0: continuous running
FMCW_CS_DLY	32	Chirp delay value, based on reference clock cycle

### 4.4.3.5 Anti Velocity Ambiguity with Chirp Delay (Anti-VELAMB CD)

When both VAM and Anti-VELAMB CD are enabled,  $P_{CD}$  is updated along with the period of VAM. In this case:

- One extended chirp is always followed by one or more continuous normal chirps.
- The maximum value of  $P_{CD}$  is 5.
- The extended chirp is always at the beginning of one period.
- The first two chirps inside an Anti-VELAMB CD period always have the same TX controlling group, which is Group 0 in VAM, as shown in Fig. 4.27 (where  $P$  denotes the period of VAM).
  - If  $P = 1$ , the controlling group sequence for one TX channel is shown in Fig. 4.27.



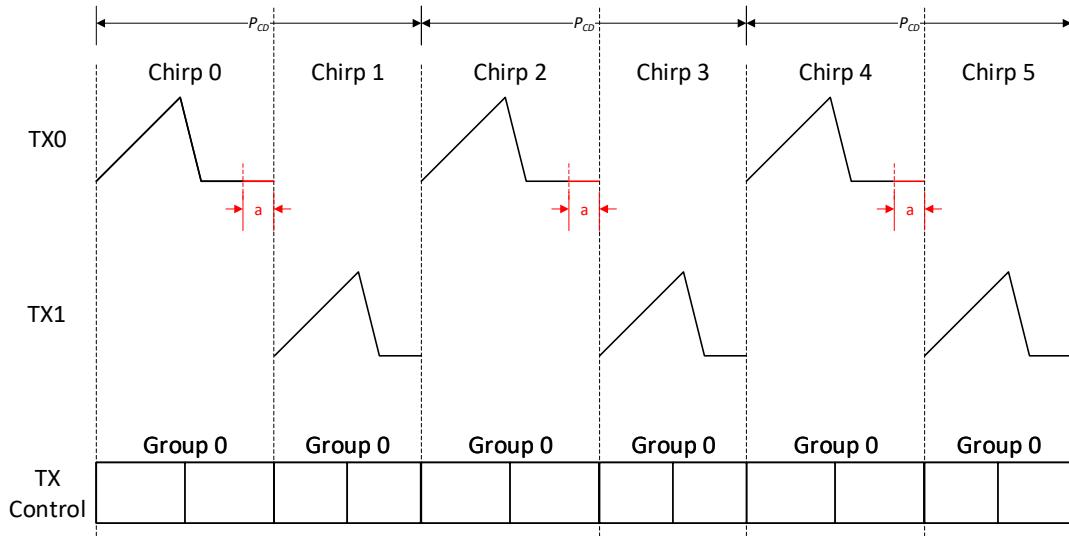


Fig. 4.27: Both Anti-VELAMB CD and VAM Are Enabled, When VAM Period  $P = 1$

Under Anti-VELAMB CD, if VAM is enabled, TX status is controlled by VAM; if VAM is disabled, TX status is controlled by radio registers, as shown in Fig. 4.28.

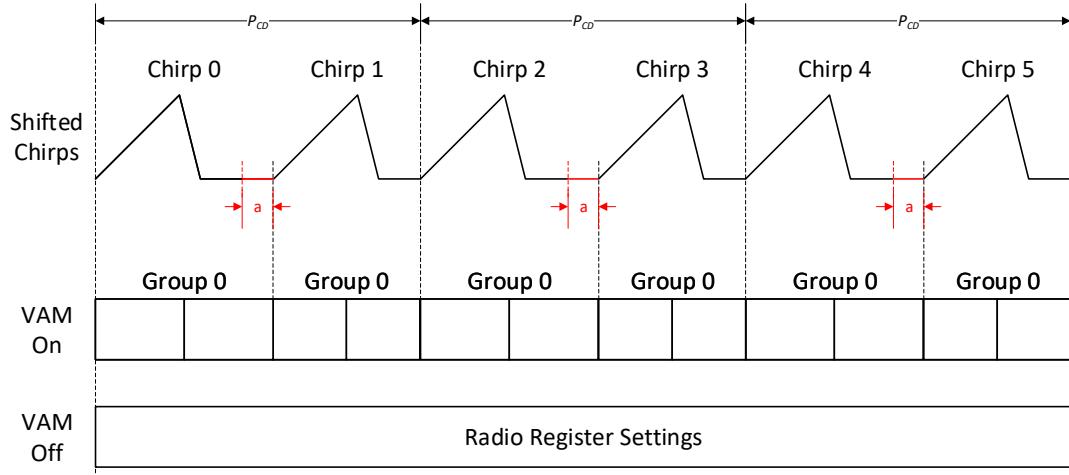


Fig. 4.28: Differences between VAM on and off Under Anti-VELAMB CD

- If  $P = 2$ , the controlling group sequence for one TX channel is shown in Fig. 4.29.

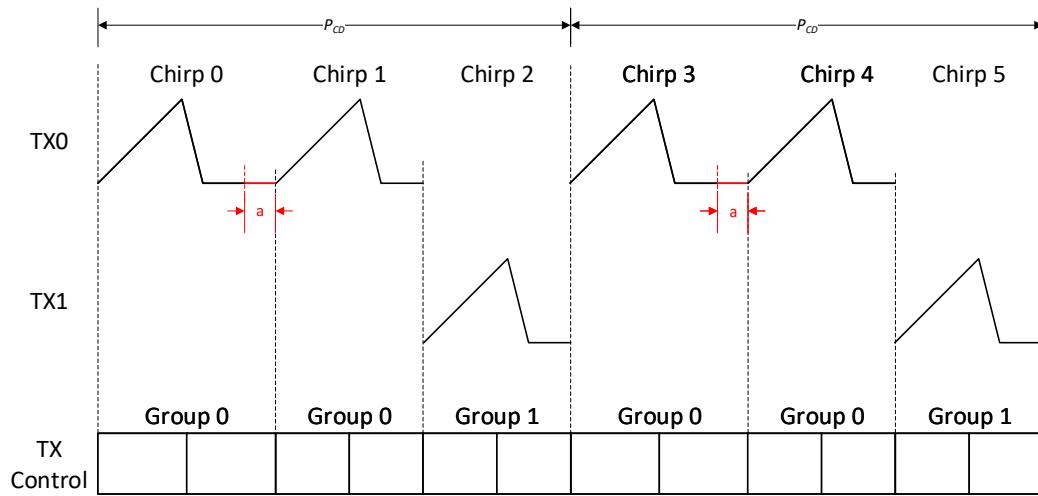


Fig. 4.29: Both Anti-VELAMB CD and VAM Are Enabled, When VAM Period  $P = 2$

- If  $P = 3$ , the controlling group sequence for one TX channel is shown in Fig. 4.30.

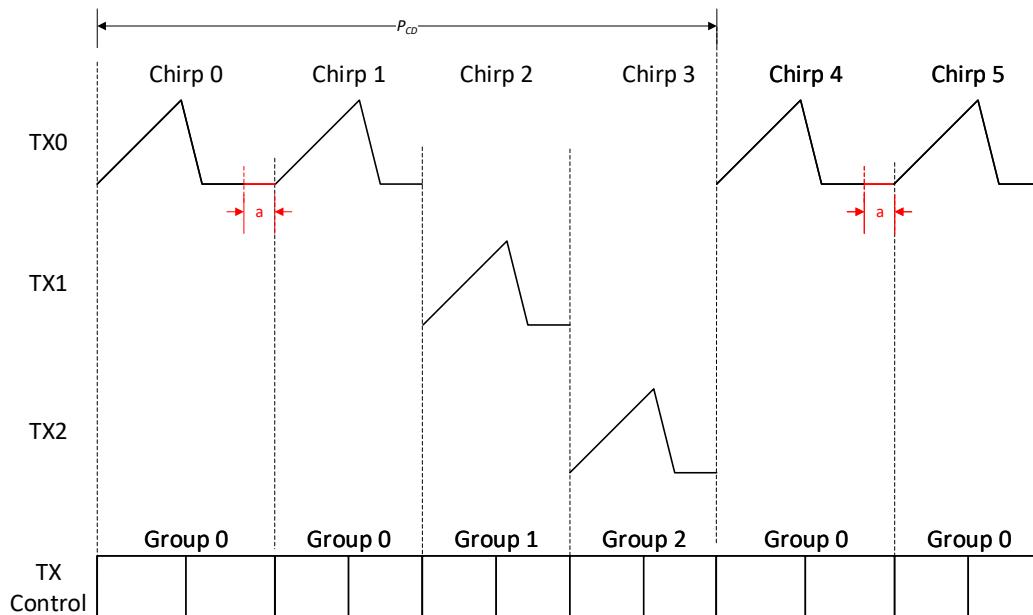


Fig. 4.30: Both Anti-VELAMB CD and VAM Are Enabled, When VAM Period  $P = 3$

- If  $P = 4$ , the controlling sequence for one TX and chirps are shown in Fig. 4.31.

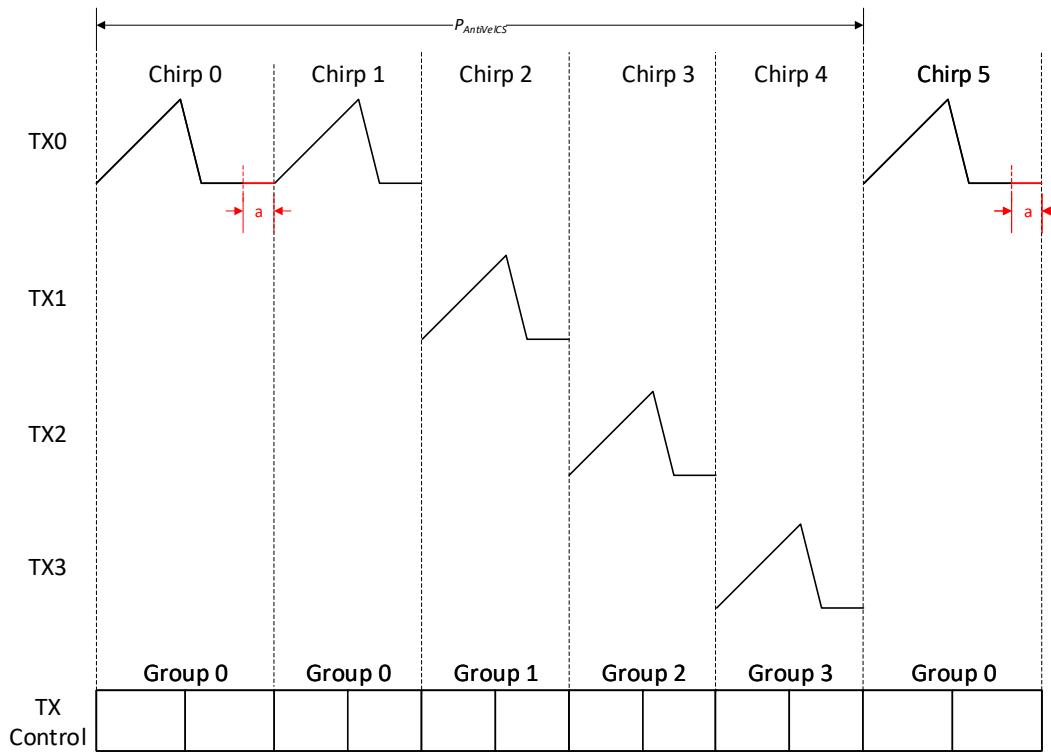


Fig. 4.31: Both Anti-VELAMB CD and VAM Are Enabled, When VAM Period  $P = 4$

Related registers for Anti-VELAMB CD are summarized in Table 4.10.

Table 4.10: Registers for Anti-VELAMB CD

Register	Width	Description
FMCW_VAM_CTL	8	TX control status Divided into 4 groups. For each 2 bits 2'b00 -> off 2'b01 -> in phase 2'b00 -> opposite phase
FMCW_VAM_PRD	2	Period for VAM $FMCW\_VAM\_PRD = \text{Period} - 1$
FMCW_VAM_ENA	1	Enable for VAM 0 disable 1 enable
FMCW_AVA_ENA	1	Enable for Anti-VELAMB CD 0 disable 1 enable
FMCW_ANA_DLY	32	Chirp extended value, based on reference clock cycle

### 4.4.3.6 Auto Gain Control (AGC)

In AGC mode, the internal logic would switch the gain control of RXBB from radio registers to CPU, as illustrated in Fig. 4.32.



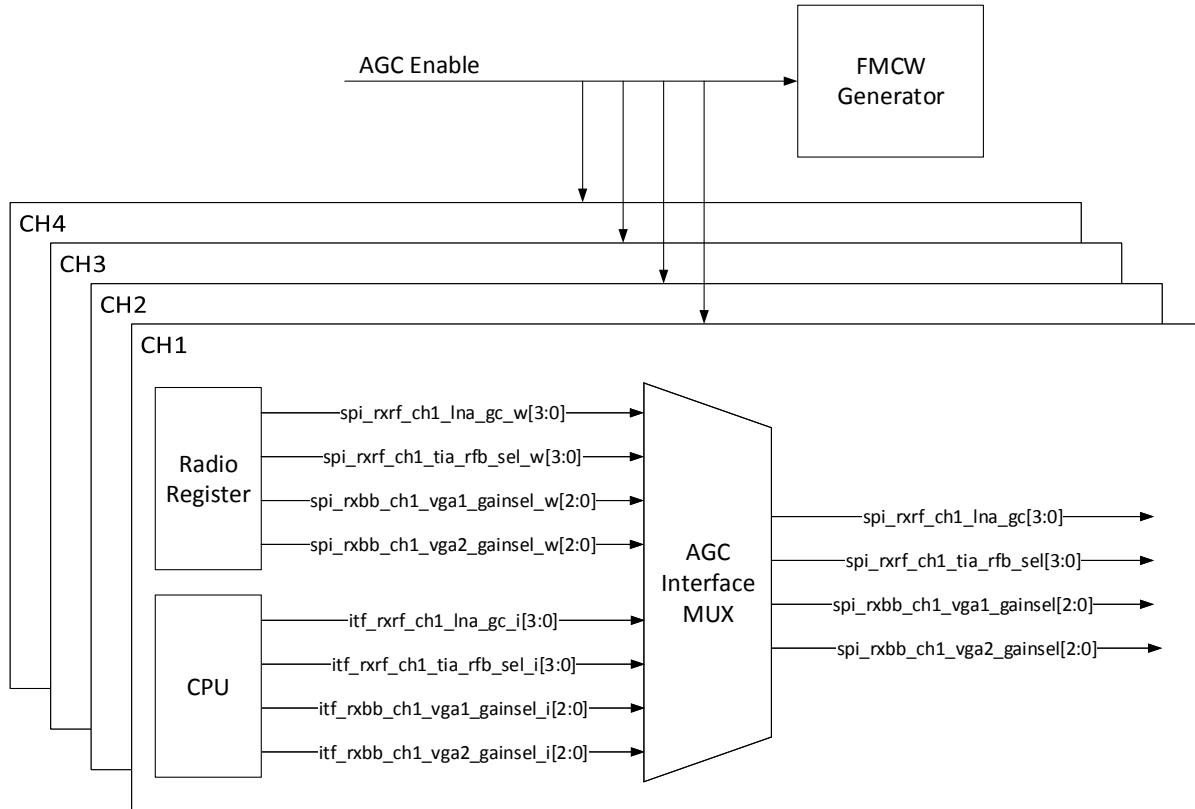


Fig. 4.32: RXBB Gain Control in AGC

Table 4.11: Register for AGC

Register	Width	Description
FMCW_AGC_ENA	1	Enable for AGC 0 disable 1 enable

#### 4.4.3.7 Frame Interleaving

Frame interleaving supports up to 4 different frame types. Each frame type is determined by the following set of FMCW parameters:

- $F_{start}$ ,  $F_{stop}$ ,  $F_{stepup}$ ,  $F_{stepdown}$ ,  $T_{idle}$ ,  $N_{chirp}$
- VAM control
- AGC mode enable/disable
- FH mode on/off
- PS mode on/off
- CS on/off



## 4.4. Programming Interfaces

---

- Anti-VELAMB CD mode on/off

Different frame types can share the same XOR chain for FH mode, PS mode, and CS mode.

FMCW\_FRM\_VEC is a 32-bit frame interleaving controlling register. It is essentially a vector, denoted by *Vec*, and can be divided into 16 sub-vectors, with each having 2 bits, as shown in Fig. 4.33. Every 2 bits defines the type of a specific frame. See descriptions in Table 4.12.

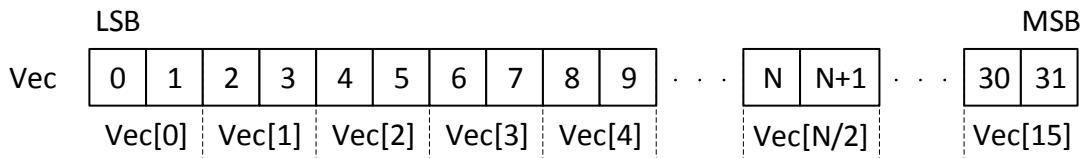


Fig. 4.33: Frame Interleaving Controlling Vector

Table 4.12: Frame Interleaving Controlling Register

Register Value	Frame Type
2'b00	Type I
2'b01	Type II
2'b10	Type III
2'b11	Type IV

FMCW\_FRM\_PRD is a 4-bit register that controls the interleaving period  $P_{interleave}$ . For the  $N$  th frame, its frame type is  $[N \bmod P_{interleave}]$ . The maximum period is 16.

Registers related to this mode are summarized in Table 4.13.

Table 4.13: Registers for Frame Interleaving

Register	Width	Description
FMCW_FRM_ENA	1	Enable for frame interleaving 0 disable 1 enable
FMCW_FRM_PRD	4	Period for frame interleaving $FMCW\_FRM\_PRD = \text{Period} - 1$
FMCW_FRM_VEC	32	Frame type control Divided into 16 groups. For each 2 bits: 2'b00 -> Type I 2'b01 -> Type II 2'b10 -> Type III 2'b11 -> Type IV

Fig. 4.10 shows an example with the register settings listed in Table 4.14.

Table 4.14: Register Settings for Frame Interleaving

Register	Width	Description	Value
FMCW_FRM_ENA	1	Enable for frame interleaving 0 disable 1 enable	0x1
FMCW_FRM_PRD	4	Period for frame interleaving FMCW_FRM_PRD = Period - 1	0x2
FMCW_FRM_VEC	32	Frame type control Divided into 16 groups. For each 2 bits: 2'b00 -> Type I 2'b01 -> Type II 2'b10 -> Type III 2'b11 -> Type IV	00_01_11_00_00_00_00_00_00_00_00_00_00_00_00_00

#### 4.4.4 Coexistence among Features

The embedded features in FMCW generator have interactions with each other.

- Coexistence with anti-interference modes

Anti-interference modes include FH mode, CS mode, and PS mode. All the anti-interference modes can be programmed and enabled independently. So altogether eight combinations of anti-interference modes are supported.

VAM is able to operate with anti-interference modes. When both VAM and PS modes are enabled, the final transmitted phase is determined according to Table 4.15.

When Anti-VELAMB CD mode is on, anti-interference modes are applied to both extended and normal chirps.

Table 4.15: Final Transmitted Phased

		Input from PS	
		in-phase	opposite-phase
Input from VAM	in-phase	in-phase	opposite-phase
	opposite-phase	opposite-phase	in-phase
	off	off	off

- Coexistence with AGC mode

AGC mode updates the behavior on the newly added 3 chirps at the beginning of each frame as follows:

- If FH mode is on and CS mode is off, suppose that FH has two patterns, A and B. For the first 3 chirps, even frames use AAA and odd frames use BBB.



## 4.4. Programming Interfaces

---

- If FH mode is off and CS mode is on, suppose that CS has two patterns, C and D. For the first 3 chirps, even frames use CCC and odd frames use DDD.
- If both FH mode and CS mode are on, suppose that FH has two patterns, A and B and that CS has two patterns, C and D. For the first three chirps, even frames use (A+C)(A+C)(A+C), and odd frames use (B+D)(B+D)(B+D).
- Settings from PS and VAM will be ignored for the first three chirps.
- If Anti-VELAMB CD mode is on, AGC uses normal chirps as the first three chirps.

### 4.4.5 Limitation and Constraints

- For FH mode

Lower side start frequency ( $f_L$ ), higher side stop frequency ( $f_H$ ), up ramp time ( $T_u$ ) and chirp time ( $T_c$ ) are fully programmable, but for better usage, those parameters should meet all of the following requirements:

1.  $f_{H1} - f_{L1} = f_{H2} - f_{L2}$
2.  $T_c$  is constant within the same frame
3.  $T_u$  is constant within the same frame

- For CS mode

The 32-bit delay is programmable, but software should guarantee that the delay is shorter than chirp waiting time, that is,  $\delta < T_{wait}$ , as shown in Fig. 4.34.

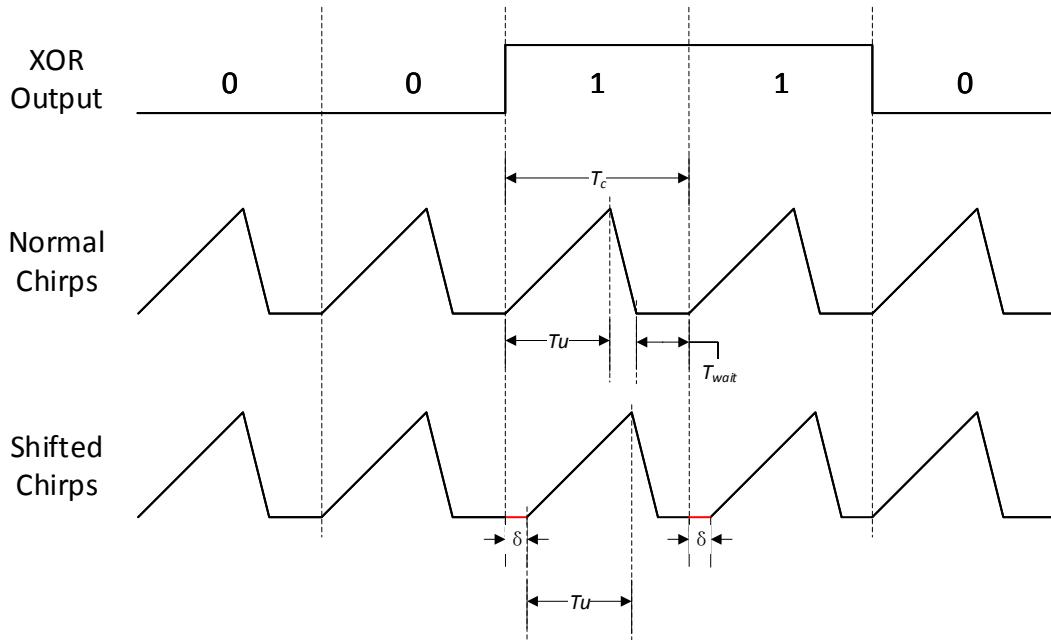


Fig. 4.34: Frame Limitation for CS

## 4.5 Examples

### 4.5.1 Example A: Generating Basic FMCW Waveform

In this example, we want to generate a basic FMCW waveform as described in *Basic Functions*, with parameter settings shown in Table 4.16.

Table 4.16: Example Values of FMCW Parameters

Parameters	Description	Value	Unit
$F_{start}$	Start frequency	76	GHz
$F_{ref}$	Reference clock frequency	400	MHz
$F_{stop}$	Stop frequency	77	GHz
$T_u$	Ramp up time	50	$\mu s$
$T_d$	Ramp down time	10	$\mu s$
$T_{idle}$	Ramp down and idle state time	70	$\mu s$
$N_{chirp}$	Chirp number inside a frame	256	/

---

**Note:**

- $T_u$  and  $T_{idle}$  should meet the following requirement, where  $T_c$  is the time for chirp period.

$$T_u + T_{idle} = T_c$$

- $T_d$  and  $T_{idle}$  should meet the following requirement, where  $T_{wait}$  is the waiting time inside a chirp.

$$T_d + T_{wait} = T_{idle}$$


---

Follow these steps to program and generate the FMCW waveform:

1. Calculate the register values in decimal denotation for the desired parameters, and translate the decimal register values to binary code.

Table 4.17: FMCW Decimal Registers

Parameter	Description
$REG_{start}$	Register value for start frequency
$REG_{stop}$	Register value for stop frequency
$REG_{stepup}$	Register value for ramp up step
$REG_{stepdown}$	Register value for ramp down step
$REG_{idle}$	Register value for idle time
$REG_{chirp}$	Register value for chirp number

1. Obtain  $REG_{start}$  from  $F_{start}$  and  $F_{ref}$ .

$$\begin{aligned}
 REG_{start} &= 125 \times \frac{F_{start}}{F_{ref}} - 16 \\
 &= 125 \times \frac{76}{400} - 16 \\
 &= 7.75_{decimal} \\
 &= 0111\ 1100\ 0000\ 0000\ 0000\ 0000\ 0000_{binary}
 \end{aligned}$$



## 4.5. Examples

---

Assign the binary value of  $REG_{start}$  to the corresponding register bits, as shown in Table 4.18.

Table 4.18: Register Setting for FMCW Start Frequency

Description	Bank Number	Address	Width	Value
Lower 8 bits of fractional part	5	5	8	0000_0000
Middle 8 bits of fractional part	5	6	8	0000_0000
Upper 8 bits of fractional part	5	7	8	0000_0000
4-bit integer part + 4 bits of fractional part	5	8	8	0111_1100

2. Obtain  $REG_{stop}$  from  $F_{start}$  and  $F_{ref}$ .

$$\begin{aligned}
 REG_{stop} &= 125 \times \frac{F_{stop}}{F_{ref}} - 16 \\
 &= 125 \times \frac{77}{400} - 16 \\
 &= 8.0625_{decimal} \\
 &= 1000\ 0001\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000_{binary}
 \end{aligned}$$

Assign the binary value of  $REG_{stop}$  to the corresponding register bits as shown in Table 4.19.

Table 4.19: Register Setting for FMCW Stop Frequency

Description	Bank Number	Address	Width	Value
Lower 8 bits of fractional part	5	9	8	0000_0000
Middle 8 bits of fractional part	5	10	8	0000_0000
Upper 8 bits of fractional part	5	11	8	0000_0000
4-bit integer part + 4 bits of fractional part	5	12	8	1000_0001

3. Obtain  $REG_{stepup}$  from  $F_{start}$ ,  $F_{stop}$ ,  $F_{ref}$ , and  $T_u$ .

$$\begin{aligned}
 REG_{stepup} &= 125 \times \frac{F_{stop} - F_{start}}{(F_{ref})^2 \times T_u} \\
 &= 125 \times \frac{77 - 76}{(400)^2 \times 50} \\
 &= 0.000015625_{decimal} \\
 &= 0000\ 0000\ 0000\ 0001\ 0000\ 0110\ 0010_{binary}
 \end{aligned}$$

Assign the binary value of  $REG_{stepup}$  to the corresponding register bits as shown in Table 4.20

Table 4.20: Register Setting for FMCW Step up Frequency

Description	Bank Number	Address	Width	Value
Lower 8 bits of fractional part	5	13	8	0110_0010
Middle 8 bits of fractional part	5	14	8	0001_0000
Upper 8 bits of fractional part	5	15	8	0000_0000
Highest 4 bits of fractional part	5	16	4	0000

4. Obtain  $REG_{stepdown}$  from  $F_{start}$ ,  $F_{stop}$ ,  $F_{ref}$ , and  $T_u$ .

$$\begin{aligned}
 REG_{stepdown} &= 125 \times \frac{F_{stop} - F_{start}}{(F_{ref})^2 \times T_d} \\
 &= 125 \times \frac{77 - 76}{(400)^2 \times 10} \\
 &= 0.000078125_{decimal} \\
 &= 0000\ 0000\ 0000\ 0101\ 0001\ 1110\ 1011_{binary}
 \end{aligned}$$

Assign the binary value of  $REG_{stepdown}$  to the corresponding register bits as shown in Table 4.21.

Table 4.21: Register Setting for FMCW Step down Frequency

Description	Bank Number	Address	Width	Value
Lower 8 bits of fractional part	5	17	8	1110_1011
Middle 8 bits of fractional part	5	18	8	0101_0001
Upper 8 bits of fractional part	5	19	8	0000_0000
Highest 4 bits of fractional part	5	20	4	0000

5. Obtain  $REG_{idle}$  from  $T_{idle}$  and  $T_{ref}$ .

$$\begin{aligned}
 REG_{idle} &= T_{idle} \times F_{ref} \\
 &= 70 \times 400 \\
 &= 28000_{decimal} \\
 &= 0000\ 0000\ 0000\ 0000\ 0110\ 1101\ 0110\ 0000_{binary}
 \end{aligned}$$

Assign the binary value of  $REG_{idle}$  to the corresponding register bits as shown in Table 4.22.

Table 4.22: Register Setting for FMCW Idle Time

Description	Bank Number	Address	Width	Value
Lower 8 bits of integer part	5	21	8	0110_0000
Middle 8 bits of integer part	5	22	8	0110_1101
Upper 8 bits of integer part	5	23	8	0000_0000
Highest 8 bits of integer part	5	24	8	0000_0000

6. Obtain  $REG_{chirp}$  from  $N_{chirp}$ .

$$\begin{aligned}
 REG_{chirp} &= N_{chirp} \\
 &= 256_{decimal} \\
 &= 0000\ 0001\ 0000\ 0000_{binary}
 \end{aligned}$$

Assign the binary value of  $REG_{chirp}$  to the corresponding register bits as shown in Table 4.23.

Table 4.23: Register Setting for FMCW Chirp Number

Description	Bank Number	Address	Width	Value
Lower 8 bits of integer part	5	25	8	0000_0000
Upper 8 bits of integer part	5	26	8	0000_0001

2. Configure FMCW basic mode.

In this example, we want FMCW generator to generate waveforms by frame. So configure the corresponding register bits as shown in Table 4.24.

## 4.5. Examples

---

Table 4.24: Register Setting for FMCW Basic Mode Selection

Description	Bank Number	Address	Width	Value
Mode selection: By frame	3	44	3	0x2

3. Start FMCW.

Start FMCW with external source (CPU or PAD) with register settings shown in Table 4.25.

Table 4.25: Register Settings for FMCW Start

Description	Bank Number	Address	Width	Value
Start selection from CPU or PAD	3	45	5	0x 16
Start generation	1). From CPU a). Write 0x0 to 0xba_0014 to select CPU b). Write 0x1 to 0xba_0018 to start generating or, 2). From PAD a). Write 0x1 to 0xba_0014 to select PAD b). Write 0x2 to 0xba_023c to MUX FMCW start signal c). Stimulate a pulse with the rising edge to start generation			
Stop generation	Automatically stop generation after transmitting $N_{chirp}$ chirps			

### 4.5.2 Example B: FMCW with VAM Enabled

A basic FMCW waveform can be obtained by steps described in *Example A: Generating Basic FMCW Waveform*. This section furthers the discussion for virtual array mode. In this example, we want to program FMCW with:

- FMCW parameters configured as in Table 4.16, and
- VAM period = 3, and
- The status of 4 TX channels as in Fig. 4.35.



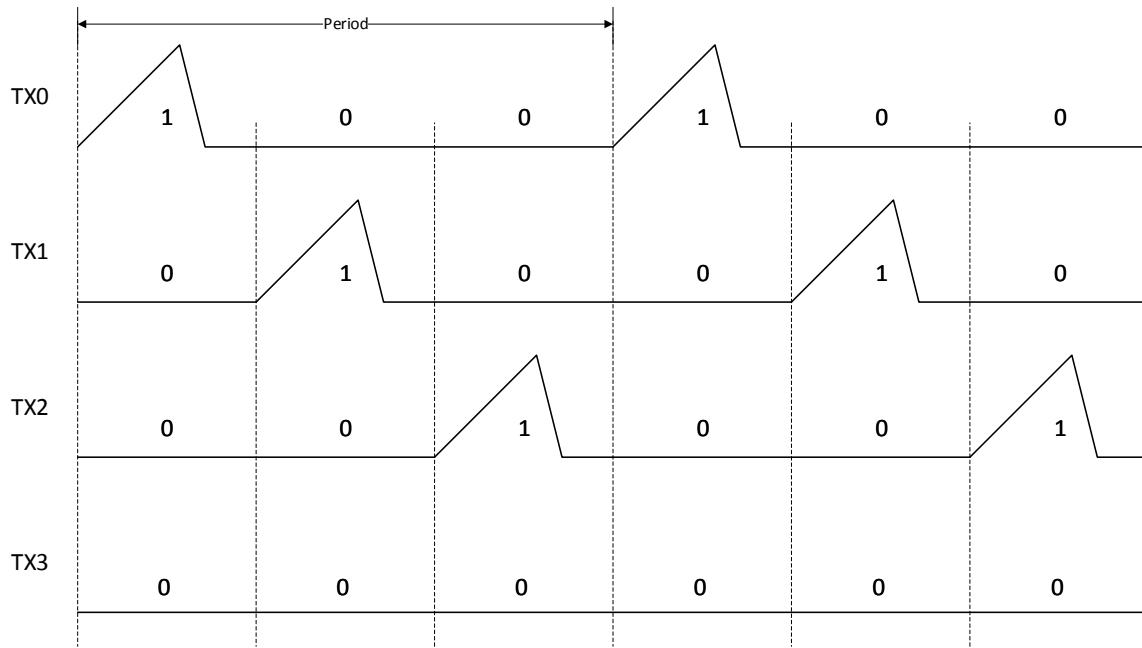


Fig. 4.35: FMCW Virtual Array Mode with Period = 3

Follow these steps for configuration:

1. First configure register values for the parameters  $F_{start}$ ,  $F_{stop}$ ,  $F_{ref}$ ,  $T_u$ ,  $T_d$ , and  $T_{idle}$  by following *Example A: Generating Basic FMCW Waveform*.

Then we will get the register settings as shown in Table 4.26.

## 4.5. Examples

---

Table 4.26: Register Settings for FMCW Basic Parameters

Parameters	Description	Bank Number	Address	Width	Value
$F_{start}$	Lower 8 bits of fractional part	5	5	8	0000_0000
	Middle 8 bits of fractional part	5	6	8	0000_0000
	Upper 8 bits of fractional part	5	7	8	0000_0000
	4-bit integer part + 4 bits of fractional part	5	8	8	0111_1100
$F_{stop}$	Lower 8 bits of fractional part	5	9	8	0000_0000
	Middle 8 bits of fractional part	5	10	8	0000_0000
	Upper 8 bits of fractional part	5	11	8	0000_0000
	4-bit integer part + 4 bits of fractional part	5	12	8	1000_0001
$F_{stepup}$	Lower 8 bits of fractional part	5	13	8	0110_0010
	Middle 8 bits of fractional part	5	14	8	0001_0000
	Upper 8 bits of fractional part	5	15	8	0000_0000
	Highest 4 bits of fractional part	5	16	4	0000_0000
$F_{stepdown}$	Lower 8 bits of fractional part	5	17	8	1110_1011
	Middle 8 bits of fractional part	5	18	8	0101_0001
	Upper 8 bits of fractional part	5	19	8	0000_0000
	Highest 4 bits of fractional part	5	20	4	0000_0000
$T_{idle}$	Lower 8 bits of integer part	5	21	8	0110_0000
	Middle 8 bits of integer part	5	22	8	0110_1101
	Upper 8 bits of integer part	5	23	8	0000_0000
	Highest 8 bits of integer part	5	24	8	0000_0000
$N_{chirp}$	Lower 8 bits of integer part	5	25	8	0000_0000
	Upper 8 bits of integer part	5	26	8	0000_0001

2. Configure VAM related registers.

- Configure TX0 with VAM period = 3 and TX controlling status being in-phase, off and off. Detailed settings are shown in [Table 4.27](#).

Table 4.27: Register Settings of VAM for TX0

Description	Bank Number	Address	Width	Value
Configure TX status controlling	5	63	8	0100_0000
Enable VAM and configure Period to 3	5	64	7	0110_1101

- Configure TX1 with VAM period = 3 and TX controlling status being off, in-phase and off. Detailed settings are shown in [Table 4.28](#).

Table 4.28: Register Settings of VAM for TX1

Description	Bank Number	Address	Width	Value
Configure TX status controlling	5	66	8	0001_0000
Enable VAM and configure Period to 3	5	67	7	0110_1101

- Configure TX2 with VAM period = 3 and TX controlling status being off, off and in-phase. Detailed settings are shown in [Table 4.29](#).

Table 4.29: Register Settings of VAM for TX2

Description	Bank Number	Address	Width	Value
Configure TX status controlling	5	69	8	0000_0100
Enable VAM and configure Period to 3	5	70	7	0110_1101

4. Turn off TX3.

This can be achieved by either of the following alternative ways:

- Shut down TX3 from radio.
- Or, configure TX3 with VAM period = 3 but TX controlling status being off, off and off. Detailed settings are shown in [Table 4.30](#).

Table 4.30: Register Settings of VAM for TX3

Description	Bank Number	Address	Width	Value
Configure TX status controlling	5	72	8	0000_0000
Enable VAM and configure Period to 3	5	73	7	0110_1101

3. Configure FMCW basic mode and start FMCW as described in [Example A: Generating Basic FMCW Waveform](#).

### 4.5.3 Example C: FMCW with FH and VAM Enabled

In this example, we will demonstrate the interactions between frequency hopping mode and VAM mode.

1. The basic FMCW parameters are exactly the same as in [Example A: Generating Basic FMCW Waveform](#) and VAM settings are exactly the same as in [Example B: FMCW with VAM Enabled](#). So we will get the registers settings as shown in [Table 4.31](#).

Table 4.31: Register Settings for FMCW Basic Parameters and VAM

Parameters	Description	Bank Number	Address	Width	Value
$F_{start}$	Lower 8 bits of fractional part	5	5	8	0000_0000
	Middle 8 bits of fractional part	5	6	8	0000_0000
	Upper 8 bits of fractional part	5	7	8	0000_0000
	4-bit integer part + 4 bits of fractional part	5	8	8	0111_1100
$F_{stop}$	Lower 8 bits of fractional part	5	9	8	0000_0000
	Middle 8 bits of fractional part	5	10	8	0000_0000
	Upper 8 bits of fractional part	5	11	8	0000_0000
	4-bit integer part + 4 bits of fractional part	5	12	8	1000_0001
$F_{stepup}$	Lower 8 bits of fractional part	5	13	8	0110_0010
	Middle 8 bits of fractional part	5	14	8	0001_0000
	Upper 8 bits of fractional part	5	15	8	0000_0000
	Highest 4 bits of fractional part	5	16	4	0000_0000
$F_{stepdown}$	Lower 8 bits of fractional part	5	17	8	1110_1011
	Middle 8 bits of fractional part	5	18	8	0101_0001
	Upper 8 bits of fractional part	5	19	8	0000_0000
	Highest 4 bits of fractional part	5	20	4	0000_0000
$T_{idle}$	Lower 8 bits of integer part	5	21	8	0110_0000
	Middle 8 bits of integer part	5	22	8	0110_1101
	Upper 8 bits of integer part	5	23	8	0000_0000
	Highest 8 bits of integer part	5	24	8	0000_0000

continues on next page



## 4.5. Examples

---

Table 4.31 – continued from previous page

Parameters	Description	Bank Number	Address	Width	Value
$N_{chirp}$	Lower 8 bits of integer part	5	25	8	0000_0000
	Upper 8 bits of integer part	5	26	8	0000_0001
Settings for TX0	Configure TX status controlling	5	63	8	0100_0000
	Enable VAM and config Period to 3	5	64	7	0110_1101
Settings for TX1	Configure TX status controlling	5	66	8	0001_0000
	Enable VAM and config Period to 3	5	67	7	0110_1101
Settings for TX2	Configure TX status controlling	5	69	8	0000_0100
	Enable VAM and config Period to 3	5	70	7	0110_1101
Settings for TX3	Configure TX status controlling	5	72	8	0000_0000
	Enable VAM and config Period to 3	5	73	7	0110_1101

2. To achieve frequency hopping, we need the second set of FMCW parameters, including  $FHP_{start}$ ,  $FHP_{stop}$ ,  $THP_u$ ,  $THP_d$ , and  $THP_{idle}$ . Configure the second group of FMCW parameters as listed in Table 4.32.

Table 4.32: The Second Set of FMCW Parameters

Parameters	Description	Value	Unit
$FHP_{start}$	Start frequency	76.5	GHz
$FHP_{stop}$	Stop frequency	77.5	GHz
$THP_u$	Ramp up time	50	us
$THP_d$	Ramp down time	10	us
$THP_{idle}$	Ramp down and idle state time	70	us

3. Translate the decimal register values to binary code following *Example A: Generating Basic FMCW Waveform*, and we will get the corresponding register values as shown in Table 4.33.

Table 4.33: Register Settings for The Second Set of FMCW Parameters

Parameters	Description	Bank Number	Address	Width	Value
$FHP_{start}$	Lower 8 bits of fractional part	5	27	8	0000_0000
	Middle 8 bits of fractional part	5	28	8	0000_0000
	Upper 8 bits of fractional part	5	29	8	1000_0000
	4-bit integer part + 4 bits of fractional part	5	30	8	0111_1110
$FHP_{stop}$	Lower 8 bits of fractional part	5	31	8	0000_0000
	Middle 8 bits of fractional part	5	32	8	0000_0000
	Upper 8 bits of fractional part	5	33	8	1000_0000
	4-bit integer part + 4 bits of fractional part	5	34	8	1000_0011
$FHP_{stepup}$	Lower 8 bits of fractional part	5	35	8	0110_0010
	Middle 8 bits of fractional part	5	36	8	0001_0000
	Upper 8 bits of fractional part	5	37	8	0000_0000
	Highest 4 bits of fractional part	5	38	4	0000_0000
$FHP_{stepdown}$	Lower 8 bits of fractional part	5	39	8	1110_1011
	Middle 8 bits of fractional part	5	40	8	0101_0001
	Upper 8 bits of fractional part	5	41	8	0000_0000
	Highest 4 bits of fractional part	5	42	4	0000_0000
$THP_{idle}$	Lower 8 bits of integer part	5	43	8	0110_0000
	Middle 8 bits of integer part	5	44	8	0110_1101
	Upper 8 bits of integer part	5	45	8	0000_0000
	Highest 8 bits of integer part	5	46	8	0000_0000



4. Configure the hopping tap to 0xFFFF\_FFFF and hopping state to 0xCCCC\_CCCC to realize an output of 00110011... as shown in Fig. 4.36.

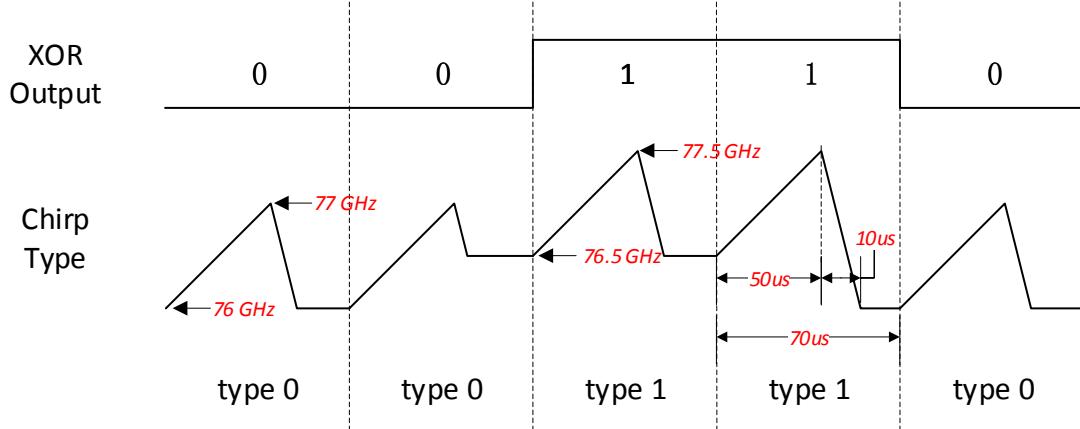


Fig. 4.36: FMCW Frequency Hopping Mode

The related registers settings for frequency hopping mode are shown in Table 4.34.

Table 4.34: Register Settings for Frequency Hopping Mode

Parameters	Description	Bank Number	Address	Width	Value
Tap	Lower 8 bits	3	1	8	1111_1111
	Middle 8 bits	3	2	8	1111_1111
	Upper 8 bits	3	3	8	1111_1111
	Highest 8 bits	3	4	8	1111_1111
State	Lower 8 bits	3	5	8	1100_1100
	Middle 8 bits	3	6	8	1100_1100
	Upper 8 bits	3	7	8	1100_1100
	Highest 8 bits	3	8	8	1100_1100
Running Mode	Configure to one-shot mode <sup>1</sup>	3	9	1	0000_0001
Enable	Frequency hopping enable	5	58	1	0000_0001

1. Configure FMCW basic mode and start FMCW as described in *Example A: Generating Basic FMCW Waveform*.

The transmitted FMCW waveform will look like Fig. 4.37.

<sup>1</sup> One-shot mode means resetting the XOR chain state at the beginning of each frame.

## 4.5. Examples

---

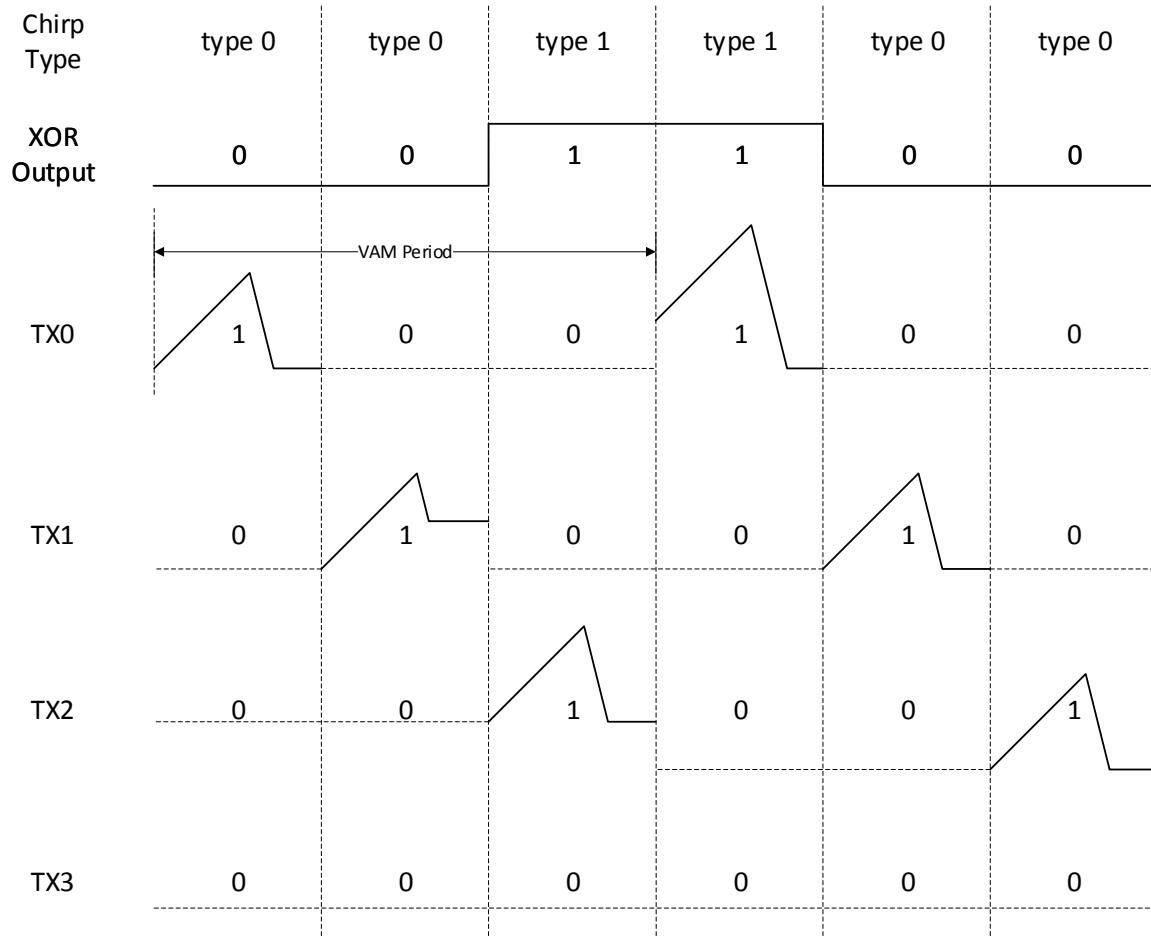


Fig. 4.37: FMCW with Frequency Hopping and VAM Enabled

### 4.5.4 Example D: FMCW with VAM, AGC, and CS Enabled

In this example, We will demonstrate configuring for an FMCW generator with all of the following features are enabled:

- Virtual Array Mode (VAM)
- AGC
- Chirp Shifting (CS)

Follow these steps for configuration:

1. The basic FMCW parameters are exactly the same as in [Example A: Generating Basic FMCW Waveform](#) and VAM settings are exactly the same as in [Example B: FMCW with VAM Enabled](#). So we will get the registers settings as shown in [Table 4.31](#).
2. To realize chirp shifting, the 32-bit delay value denoted by  $\delta$  as shown in Fig. 4.38 is required.

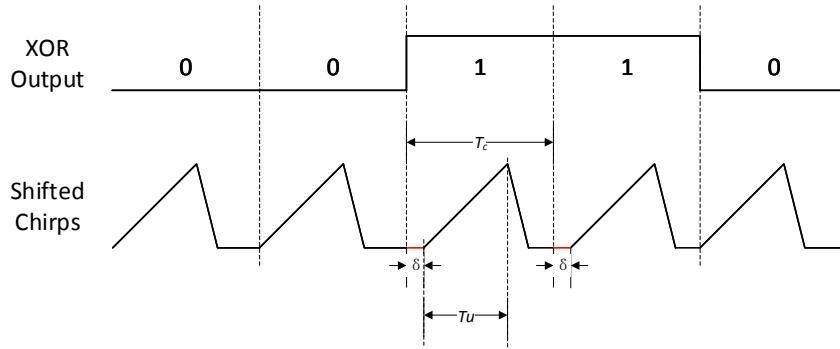


Fig. 4.38: FMCW Chirp Shifting Mode

Configure  $\delta = 5 \text{ } \mu\text{s}$  for chirp shifting mode, as shown in Table 4.35.

Table 4.35: FMCW Chirp Shifting Parameter

Parameters	Description	Value	Unit
$\delta$	32-bit chirp delay value	5	$\mu\text{s}$

- Obtain the corresponding binary code of  $\delta$  by the following formula:

$$REG_{\delta} = \delta \times F_{ref}$$

Where the parameter are defined in Table 4.36.

Table 4.36: FMCW Chirp Shifting Parameter Definition

Parameters	Description	Unit
$\delta$	Chirp delay time	$\mu\text{s}$
$F_{ref}$	Reference clock frequency 400 MHz or 450 MHz	MHz
$REG_{\delta}$	Register value for chirp delay	/

Therefore,

$$\begin{aligned} REG_{\delta} &= \delta \times F_{ref} \\ &= 5 \times 400 \\ &= 2000_{decimal} \\ &= 0000\ 0000\ 0000\ 0000\ 0111\ 1101\ 0000_{binary} \end{aligned}$$

- Assign the binary value of  $REG_{\delta}$  to the corresponding register bits as shown in Table 4.37.

Table 4.37: Register Settings for FMCW Chirp Shifting Delay

Description	Bank Number	Address	Width	Value
Lower 8 bits of integer part	5	53	8	1101_0000
Middle 8 bits of integer part	5	54	8	0000_0111
Upper 8 bits of integer part	5	55	8	0000_0000
Highest 8 bits of integer part	5	56	8	0000_0000



## 4.5. Examples

---

5. Configure the chirp shifting tap to 0xFFFF\_FFFF and chirp shifting state to 0xCCCC\_CCCC to realize an output of 00110011... as shown in Fig. 4.39.

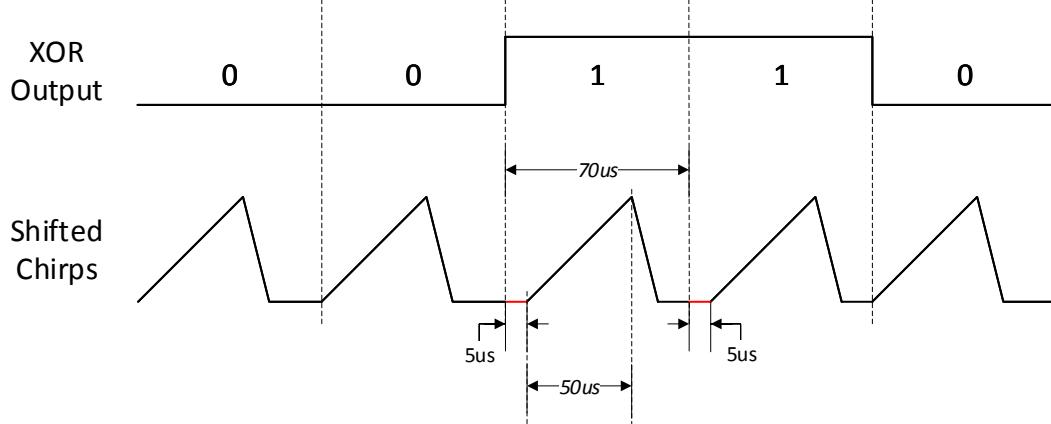


Fig. 4.39: FMCW Chirp Shifting Mode with Delay of 5 us

The related register settings for CS mode are shown in Table 4.38.

Table 4.38: Register Settings for CS Mode

Parameters	Description	Bank Number	Address	Width	Value
Tap	Lower 8 bits	3	19	8	1111_1111
	Middle 8 bits	3	20	8	1111_1111
	Upper 8 bits	3	21	8	1111_1111
	Highest 8 bits	3	22	8	1111_1111
State	Lower 8 bits	3	23	8	1100_1100
	Middle 8 bits	3	24	8	1100_1100
	Upper 8 bits	3	25	8	1100_1100
	Highest 8 bits	3	26	8	1100_1100
Running Mode	Configure to one-shot mode <sup>2</sup>	3	27	1	0000_0001
Enable	Frequency hopping enable	5	57	1	0000_0001

1. Enable AGC according to Table 4.39.

Table 4.39: Register Setting for AGC Enable

Description	Bank Number	Address	Width	Value
Enable AGC	5	47	2	0000_0001

2. Configure FMCW basic mode and start FMCW as described in *Example A: Generating Basic FMCW Waveform*.

The transmitted FMCW waveform will look like Fig. 4.40. The first 3 chirps are generated due to AGC, and the effect of VAM and chirp shifting is demonstrated by the chirps following.

<sup>2</sup> One-shot mode means resetting the XOR chain state at the beginning of each frame.



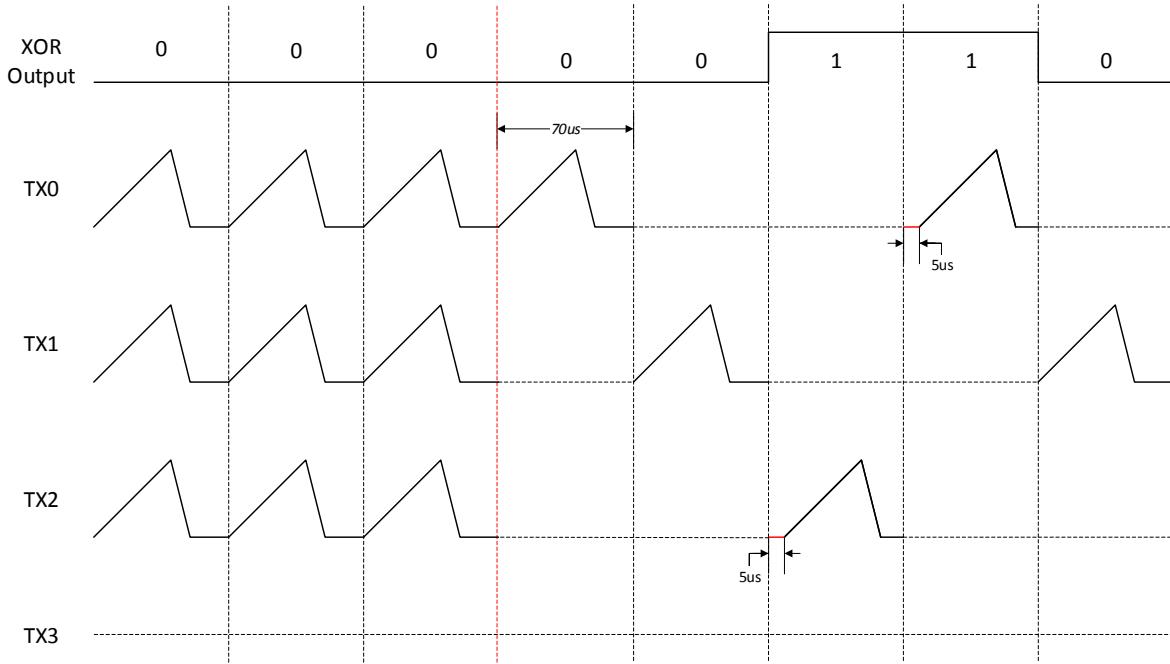


Fig. 4.40: FMCW with VAM, AGC, and CS Enabled

## 4.6 Software and Setting Suggestions

The embedded ADC samples analog signal and then loads those digitalized data into a dual port memory whose clock frequency for writing data is the same as ADC data, shown as Fig. 4.41.

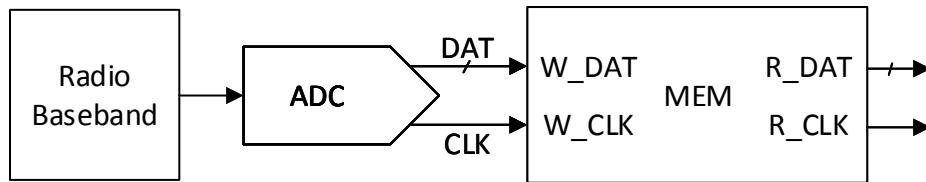


Fig. 4.41: Structure between ADC and Dual Port Memory

ADC samples data continuously after being enabled, but data in memory is taken by a fixed window with the same width, which is the same beginning and ending point relative to the starting of each chirp. A common practice is set a counter denoted as  $N_{length}$  to count the length of each chirp inside a frame, which takes the starting point of the first chirp as the baseline, and the counter will not stop counting until it reaches the length of the first chirp, after that, it will self-cleared to 0, and ready to count the length for the next chirp, whose length is the same with the first chirp, this process will stop at the end of the frame.

Since the counter is in the clock domain of ADC with a frequency denoted as  $F_{ADCOUT}$ , but for a chirp, the up ramp, down ramp and wait state are in the clock domain of FMCW with a frequency denoted as  $F_{ref}$ . To make this radar

## 4.6. Software and Setting Suggestions

---

chip more flexible, we provide 4 different  $F_{ADCOUT}$  values, 20 MHz, 25 MHz, 40 MHz, and 50 MHz, while those of  $F_{ref}$  are 400MHz and 450MHz.

To sample data correctly into memory, the counter length  $T_{counter}$  and the chirp length must be exactly the same, as shown in Fig. 4.42.

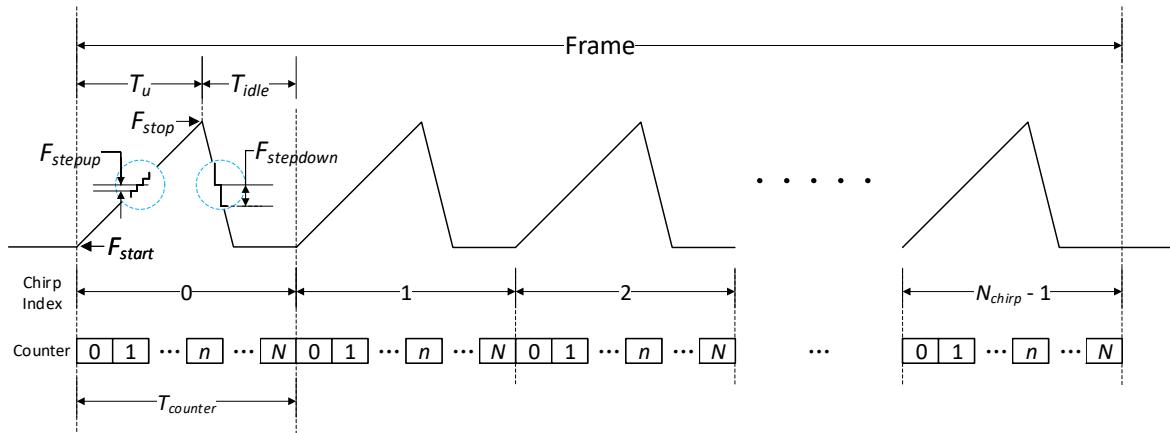


Fig. 4.42: Relationship between Counter Length and Chirp Length

The mathematics presentation for counter length and chirp length is shown in the following formula.

$$T_{counter} = T_u + T_{idle}$$

But in software implementation, the upper formula still holds,  $T_{counter}$  is obtained from counter size and the clock period of ADC output data  $T_{ADCOUT}$ ,  $T_u$  and  $T_{idle}$  are obtained from chirp cycle and the period of the reference clock, which is shown as below.

$$\text{Size}_{counter} \times T_{ADCOUT} = \text{Cycle}_{chirp} \times T_{ref}$$

Where parameters are defined in Table 4.40.

Table 4.40: Parameters Definition

Parameters	Description	Unit
$\text{Size}_{counter}$	Counter size	
$T_{ADCOUT}$	Period of $F_{ADCOUT}$	$\mu s$
$\text{Cycle}_{chirp}$	Cycle of a chirp including up cycle and idle cycle	
$T_{ref}$	Reference clock period	$\mu s$

The cycle of a chirp is defined by  $REG_{start}$ ,  $REG_{stop}$ ,  $REG_{stepup}$ ,  $REG_{idle}$  and  $F_{ref}$ , but due to loss of accuracy when obtaining those register values from decimal value, cycle alignment has to be done to make sure counter length matches chirp length, the general firmware flow for cycle alignment should be designed as the following:

1. Obtaining the greatest common divisor(GCD) from  $F_{ADCOUT}$  and  $F_{ref}$ .
2. Obtaining the total cycle from  $T_c$  based on  $F_{ADCOUT}$  and  $F_{ref}$ .
3. Obtaining  $REG_{start}$ ,  $REG_{stop}$ ,  $REG_{stepup}$ ,  $REG_{stepdown}$  and  $REG_{idle}$  based on  $F_{ref}$ .

Cycle alignment can be done with the following code:

```

//Get the decimal denotation for Fstart and Fstop
#define DIV_RATIO(val, F_SD) ((val) * 1e3 / 8 / (F_SD) - 16) * (1L<<28)

//Get the GCD of a and b
static uint32_t compute_gcd(uint32_t a, uint32_t b)
{
    if(a < b) {
        return compute_gcd(b, a);
    }
    if(a % b == 0) {
        return b;
    }
    return compute_gcd(b, a % b);
}

//Obtaining the register values
void fmcw_radio_compute_reg_value(fmcw_radio_t *radio)
{
    uint32_t gcd = compute_gcd(FADCOUT, Fref); //GCD between ADC output
    ↵data frequency and reference clock
    uint32_t total_cycle = round(Tc * gcd) * Fref / gcd ; //total cycles
    ↵inside a chirp with the length of Tc
    REG_start = DIV_RATIO(Fstart, Fref); //Register value for start
    ↵frequency
    uint32_t REG_stepup = (1L<<28) * (Fstop - Fstart) * 1e3 / (Fref * Tu *
    ↵Fref * 8); //Register value for ramp up step
    uint32_t REG_stepdown = (1L<<28) * (Fstop - Fstart) * 1e3 / (Fref * Td *
    ↵Fref * 8); //Register value for ramp down step
    uint32_t REG_stop_tmp = DIV_RATIO(Fstop, Fref); //Temperary register
    ↵value for Fstop
    uint32_t up_cycle = ceil(1.0 * (REG_stop_tmp - REG_start) / REG_
    ↵stepup); //Cycles in ramp up
    bandwidth = REG_stepup * up_cycle; //Temperary bandwidth
    REG_stop = REG_start + bandwidth; //Register value for stop frequency
    REG_idle = total_cycle - up_cycle ; //Register value for idle time
}

```

## 4.7 Register Descriptions

FMCW related registers could be grouped into two categories, non-banked registers and banked registers. Non-banked registers are located in Bank 3, while the banked registers are located in Banks 5, 6, 7, and 8.

### 4.7.1 Non-Banked Registers

Table 4.41: FMCW Non-Banked Registers

ADDR	Default Value	Bits	Bit Name	Memory Access	Bit Description
0x00	8'b0000_0000	7:4	Reserved	R/W	Not used
		3:0	fmcw_bank_sel[3:0]	R/W	bank selection for FMCW related registers
0x01	8'b0000_0000	7:0	fmcw_fh_tap[7:0]	R/W	32-bit XOR-chain tap for frequency hopping
0x02	8'b0000_0000	7:0	fmcw_fh_tap[15:8]	R/W	
0x03	8'b0000_0000	7:0	fmcw_fh_tap[23:16]	R/W	
0x04	8'b0000_0000	7:0	fmcw_fh_tap[31:24]	R/W	
0x05	8'b0000_0000	7:0	fmcw_fh_sta[7:0]	R/W	32-bit XOR-chain init state for frequency hopping

continues on next page



## 4.7. Register Descriptions

Table 4.41 – continued from previous page

0x06	8'b0000_0000	7:0	fmcw_fh_sta[15:8]	R/W	
0x07	8'b0000_0000	7:0	fmcw_fh_sta[23:16]	R/W	
0x08	8'b0000_0000	7:0	fmcw_fh_sta[31:24]	R/W	
0x09	8'b0000_0001	7:1	Reserved	R/W	Not used
		0	fmcw_fh_rm	R/W	XOR-chain running mode for frequency hopping 1 = reset at the beginning of each frame 0 = continuous running
0x0A	8'b0000_0000	7:0	fmcw_ps_tap[7:0]	R/W	32-bit XOR-chain tap for phase scramble
0x0B	8'b0000_0000	7:0	fmcw_ps_tap[15:8]	R/W	
0x0C	8'b0000_0000	7:0	fmcw_ps_tap[23:16]	R/W	
0x0D	8'b0000_0000	7:0	fmcw_ps_tap[31:24]	R/W	
0x0E	8'b0000_0000	7:0	fmcw_ps_sta[7:0]	R/W	32-bit XOR-chain init state for phase scramble
0x0F	8'b0000_0000	7:0	fmcw_ps_sta[15:8]	R/W	
0x10	8'b0000_0000	7:0	fmcw_ps_sta[23:16]	R/W	
0x11	8'b0000_0000	7:0	fmcw_ps_sta[31:24]	R/W	
0x12	8'b0000_0001	7:1	Reserved	R/W	Not used
		0	fmcw_ps_rm	R/W	XOR-chain running mode for phase scramble 1 = reset at the beginning of each frame 0 = continuous running
0x13	8'b0000_0000	7:0	fmcw_cs_tap[7:0]	R/W	32-bit XOR-chain tap for chirp shifting
0x14	8'b0000_0000	7:0	fmcw_cs_tap[15:8]	R/W	
0x15	8'b0000_0000	7:0	fmcw_cs_tap[23:16]	R/W	
0x16	8'b0000_0000	7:0	fmcw_cs_tap[31:24]	R/W	
0x17	8'b0000_0000	7:0	fmcw_cs_sta[7:0]	R/W	32-bit XOR-chain init state for chirp shifting
0x18	8'b0000_0000	7:0	fmcw_cs_sta[15:8]	R/W	
0x19	8'b0000_0000	7:0	fmcw_cs_sta[23:16]	R/W	
0x1A	8'b0000_0000	7:0	fmcw_cs_sta[31:24]	R/W	
		7:1	Reserved	R/W	Not used
0x1B	8'b0000_0001	0	fmcw_cs_rm	R/W	XOR-chain running mode for chirp shifting 1 = reset at the beginning of each frame 0 = continuous running
0x1C	8'b0000_0000	7:0	fmcw_fil_vecs[7:0]	R/W	32-bit frame interleaving vectors
0x1D	8'b0000_0000	7:0	fmcw_fil_vecs[15:8]	R/W	
0x1E	8'b0000_0000	7:0	fmcw_fil_vecs[23:16]	R/W	
0x1F	8'b0000_0000	7:0	fmcw_fil_vecs[31:24]	R/W	
0x20	8'b0000_0000	7:0	fmcw_fil_timer[7:0]	R/W	32-bit frame interleaving timer used in rotate-mode
0x21	8'b0000_0000	7:0	fmcw_fil_timer[15:8]	R/W	
0x22	8'b0000_0000	7:0	fmcw_fil_timer[23:16]	R/W	
0x23	8'b0000_0000	7:0	fmcw_fil_timer[31:24]	R/W	
0x24	8'b0000_0000	7:1	Reserved	R/W	Not used
		0	fmcw_fil_multi_en	R/W	rotate-mode enable in frame interleaving 1 = enable 0 = disable
0x25	8'b0000_0000	7:4	Reserved	R/W	Not used
		3:0	fmcw_fil_prd[3:0]	R/W	frame interleaving period Period = fmcw_fil_prd + 1
0x26	8'b0000_0000	7:4	Reserved	R/W	Not used
		3:0	fmcw_fil_typ[3:0]	R/W	frame interleaving type after reset released
0x27	8'b0000_0000	7:1	Reserved	R/W	Not used

continues on next page



Table 4.41 – continued from previous page

		0	fmcw_fil_en	R/W	frame interleaving enable 1 = enable 0 = disable
0x2C	8'b0000_0000	7:3	Reserved	R/W	Not used
		2:0	fmcw_mode_sel[2:0]	R/W	FMCW mode selection 3'b000 - single tone 3'b010 - predefined mode
0x2D	8'b0000_0000	7:5	Reserved	R/W	Not used
		4	fmcw_adder_rstn	R/W	reset for FMCW adder 1 = normal working 0 = reset
		3	fmcw_rstn_sdm_3sl	R/W	reset for 3-stage SDM 1 = normal working 0 = reset
		2	fmcw_rstn_sdm_mash	R/W	reset for MASH SDM 1 = normal working 0 = reset
		1	fmcw_start_sel	R/W	FMCW starting source selection 0 = SPI 1 = external source (CPU or PAD)
		0	fmcw_start_spi	R/W	level trigger for FMCW 1 = enable 0 = disable

## 4.7.2 Banked Registers

Table 4.42: FMCW Banked Registers

ADDR	Default Value	Bits	Bit Name	Memory Access	Bit Description
0x00	8'b0000_0000	7:4	Reserved	R/W	not used
		3:0	fmcw_bank_sel[3:0]	R/W	bank selection for FMCW related registers
0x01	8'b1001_0000	7:0	fmcw_idle_start_size[7:0]	R/W	32-bit idle start for each frame
0x02	8'b0101_1111	7:0	fmcw_idle_start_size[15:8]	R/W	
0x03	8'b0000_0001	7:0	fmcw_idle_start_size[23:16]	R/W	
0x04	8'b0000_0000	7:0	fmcw_idle_start_size[31:24]	R/W	
0x05	8'b0000_0000	7:0	fmcw_start_freq[7:0]	R/W	32-bit start frequency
0x06	8'b0000_0000	7:0	fmcw_start_freq[15:8]	R/W	
0x07	8'b0000_0000	7:0	fmcw_start_freq[23:16]	R/W	
0x08	8'b1000_0000	7:0	fmcw_start_freq[31:24]	R/W	
0x09	8'b0000_0000	7:0	fmcw_stop_freq[7:0]	R/W	32-bit stop frequency
0x0A	8'b0000_0000	7:0	fmcw_stop_freq[15:8]	R/W	
0x0B	8'b0000_0000	7:0	fmcw_stop_freq[23:16]	R/W	
0x0C	8'b1000_0000	7:0	fmcw_stop_freq[31:24]	R/W	

continues on next page



## 4.7. Register Descriptions

---

Table 4.42 – continued from previous page

0x0D	8'b0000_0000	7:0	fmcw_step_up_freq[7:0]	R/W	28-bit step up frequency
0x0E	8'b0000_0000	7:0	fmcw_step_up_freq[15:8]	R/W	
0x0F	8'b0000_0000	7:0	fmcw_step_up_freq[23:16]	R/W	
0x10	8'b0000_0000	7:4	Reserved	R/W	not used
		3:0	fmcw_step_up_freq[27:24]	R/W	
0x11	8'b0000_0000	7:0	fmcw_step_dn_freq[7:0]	R/W	28-bit step down frequency
0x12	8'b0000_0000	7:0	fmcw_step_dn_freq[15:8]	R/W	
0x13	8'b0000_0000	7:0	fmcw_step_dn_freq[23:16]	R/W	
0x14	8'b0000_0000	7:4	Reserved	R/W	not used
		3:0	fmcw_step_dn_freq[27:24]	R/W	
0x15	8'b0000_0000	7:0	fmcw_cnt_wait_size_i[7:0]	R/W	32 bits for idle time, which includes down state and wait state
0x16	8'b0000_0000	7:0	fmcw_cnt_wait_size_i[15:8]	R/W	
0x17	8'b0000_0000	7:0	fmcw_cnt_wait_size_i[23:16]	R/W	
0x18	8'b0000_0000	7:0	fmcw_cnt_wait_size_i[31:24]	R/W	
0x19	8'b0000_0000	7:0	fmcw_chirp_size[7:0]	R/W	16-bit chirp number inside a frame
0x1A	8'b0000_0000	7:0	fmcw_chirp_size[15:8]	R/W	
0x1B	8'b0000_0000	7:0	fmcw_start_freq_fh[7:0]	R/W	the secondary 32-bit start frequency for frequency hopping
0x1C	8'b0000_0000	7:0	fmcw_start_freq_fh[15:8]	R/W	
0x1D	8'b0000_0000	7:0	fmcw_start_freq_fh[23:16]	R/W	
0x1E	8'b1000_0000	7:0	fmcw_start_freq_fh[31:24]	R/W	
0x1F	8'b0000_0000	7:0	fmcw_stop_freq_fh[7:0]	R/W	the secondary 32-bit stop frequency for frequency hopping
0x20	8'b0000_0000	7:0	fmcw_stop_freq_fh[15:8]	R/W	
0x21	8'b0000_0000	7:0	fmcw_stop_freq_fh[23:16]	R/W	
0x22	8'b1000_0000	7:0	fmcw_stop_freq_fh[31:24]	R/W	
0x23	8'b0000_0000	7:0	fmcw_step_up_freq_fh[7:0]	R/W	the secondary 28-bit step up frequency for frequency hopping
0x24	8'b0000_0000	7:0	fmcw_step_up_freq_fh[15:8]	R/W	
0x25	8'b0000_0000	7:0	fmcw_step_up_freq_fh[23:16]	R/W	
0x26	8'b0000_0000	7:4	Reserved	R/W	not used
		3:0	fmcw_step_up_freq_fh[27:24]	R/W	
0x27	8'b0000_0000	7:0	fmcw_step_dn_freq_fh[7:0]	R/W	the secondary 28-bit step down frequency for frequency hopping
0x28	8'b0000_0000	7:0	fmcw_step_dn_freq_fh[15:8]	R/W	
0x29	8'b0000_0000	7:0	fmcw_step_dn_freq_fh[23:16]	R/W	
0x2A	8'b0000_0000	7:4	Reserved	R/W	not used
		3:0	fmcw_step_dn_freq_fh[27:24]	R/W	
0x2B	8'b0000_0000	7:0	fmcw_cnt_wait_size_fh_i[7:0]	R/W	the secondary 32-bit idle time for frequency hopping
0x2C	8'b0000_0000	7:0	fmcw_cnt_wait_size_fh_i[15:8]	R/W	

continues on next page



Table 4.42 – continued from previous page

0x2D	8'b0000_0000	7:0	fmcw_cnt_wait_size_fh_i[23:16]	R/W	
0x2E	8'b0000_0000	7:0	fmcw_cnt_wait_size_fh_i[31:24]	R/W	
0x2F	8'b0000_0000	7:1	Reserved	R/W	not used
		0	fmcw_agc_en	R/W	enable for AGC 0 = disable 1 = enable
0x30	8'b0000_0000	7:0	fmcw_cs_ava_dly_size[7:0]	R/W	32-bit chirp extended value in Anti-VELAMB CD, based on reference clock cycle
0x31	8'b0000_0000	7:0	fmcw_cs_ava_dly_size[15:8]	R/W	
0x32	8'b0000_0000	7:0	fmcw_cs_ava_dly_size[23:16]	R/W	
0x33	8'b0000_0000	7:0	fmcw_cs_ava_dly_size[31:24]	R/W	
0x34	8'b0000_0000	7:1	Reserved	R/W	not used
		0	fmcw_cs_ava_en	R/W	enable for Anti-VELAMB CD 0 = disable 1 = enable
0x35	8'b0000_0000	7:0	fmcw_cs_dly_size[7:0]	R/W	32-bit chirp delay value in chirp shifting mode
0x36	8'b0000_0000	7:0	fmcw_cs_dly_size[15:8]	R/W	
0x37	8'b0000_0000	7:0	fmcw_cs_dly_size[23:16]	R/W	
0x38	8'b0000_0000	7:0	fmcw_cs_dly_size[31:24]	R/W	
0x39	8'b0000_0000	7:1	Reserved	R/W	not used
		0	fmcw_cs_en	R/W	enable for chirp shifting mode 0 = disable 1 = enable
0x3A	8'b0000_0000	7:1	Reserved	R/W	not used
		0	fmcw_fh_en	R/W	enable for frequency hopping 0 = disable 1 = enable
0x3D	8'b0000_0010	7:1	Reserved	R/W	not used
		0	fmcw_ps_en	R/W	phase scramble enable 0 = disable 1 = enable
0x3F	8'b0000_0000	7:6	fmcw_tx0_ctrl_vam_1[1:0]	R/W	the first group of VAM control settings for TX0
		5:4	fmcw_tx0_ctrl_vam_2[1:0]	R/W	the second group of VAM control settings for TX0
		3:2	fmcw_tx0_ctrl_vam_3[1:0]	R/W	the third group of VAM control settings for TX0
		1:0	fmcw_tx0_ctrl_vam_4[1:0]	R/W	the fourth group of VAM control settings for TX0
0x40	8'b0110_1000	7	Reserved	R/W	not used
		6	fmcw_tx0_ctrl_vam_rm	R/W	VAM running mode for TX0 0 = continuous 1 = one-shot
		5:3	Reserved	R/W	not used

continues on next page



## 4.7. Register Descriptions

---

Table 4.42 – continued from previous page

		2:1	fmcw_tx0_ctrl_vam_p[1:0]	R/W	VAM period for TX0
		0	fmcw_tx0_ctrl_vam_en	R/W	VAM enable for TX0 0 = disable 1 = enable
0x42	8'b0000_0000	7:6	fmcw_tx1_ctrl_vam_1[1:0]	R/W	the first group of VAM control settings for TX1
		5:4	fmcw_tx1_ctrl_vam_2[1:0]	R/W	the second group of VAM control settings for TX1
		3:2	fmcw_tx1_ctrl_vam_3[1:0]	R/W	the third group of VAM control settings for TX1
		1:0	fmcw_tx1_ctrl_vam_4[1:0]	R/W	the fourth group of VAM control settings for TX1
0x43	8'b0110_1000	7	Reserved	R/W	not used
		6	fmcw_tx1_ctrl_vam_rm	R/W	VAM running mode for TX1 0 = continuous 1 = one-shot
		5:3	Reserved	R/W	not used
		2:1	fmcw_tx1_ctrl_vam_p[1:0]	R/W	VAM period for TX1
		0	fmcw_tx1_ctrl_vam_en	R/W	VAM enable for TX1 0 = disable 1 = enable
0x45	8'b0000_0000	7:6	fmcw_tx2_ctrl_vam_1[1:0]	R/W	the first group of VAM control settings for TX2
		5:4	fmcw_tx2_ctrl_vam_2[1:0]	R/W	the second group of VAM control settings for TX2
		3:2	fmcw_tx2_ctrl_vam_3[1:0]	R/W	the third group of VAM control settings for TX2
		1:0	fmcw_tx2_ctrl_vam_4[1:0]	R/W	the fourth group of VAM control settings for TX2
0x46	8'b0110_1000	7	Reserved	R/W	not used
		6	fmcw_tx2_ctrl_vam_rm	R/W	VAM running mode for TX2 0 = continuous 1 = one-shot
		5:3	Reserved	R/W	not used
		2:1	fmcw_tx2_ctrl_vam_p[1:0]	R/W	VAM period for TX2
		0	fmcw_tx2_ctrl_vam_en	R/W	VAM enable for TX2 0 = disable 1 = enable
0x48	8'b0000_0000	7:6	fmcw_tx3_ctrl_vam_1[1:0]	R/W	the first group of VAM control settings for TX3
		5:4	fmcw_tx3_ctrl_vam_2[1:0]	R/W	the second group of VAM control settings for TX3
		3:2	fmcw_tx3_ctrl_vam_3[1:0]	R/W	the third group of VAM control settings for TX3
		1:0	fmcw_tx3_ctrl_vam_4[1:0]	R/W	the fourth group of VAM control settings for TX3
0x49	8'b0110_1000	7	Reserved	R/W	not used

continues on next page



Table 4.42 – continued from previous page

		6	fmcw_tx3_ctrl_vam_rm	R/W	VAM running mode for TX3 0 = continuous 1 = one-shot
		5:3	Reserved	R/W	not used
		2:1	fmcw_tx3_ctrl_vam_p[1:0]	R/W	VAM period for TX3
		0	fmcw_tx3_ctrl_vam_en	R/W	VAM enable for TX3 0 = disable 1 = enable



## **4.7. Register Descriptions**

---



## SAMPLE PREPROCESS

From this chapter, we discuss each module or feature in Alps baseband processor. It starts from the *sample preprocess* module, which processes data from ADC output before feeding it into the 2D-FFT module.

### 5.1 Overview

In sample preprocess module, time domain processing of ADC data is conducted before 2D FFT, including scaling, filtering, down-sampling, interference detection, and suppression. Sampling related parameters such as sampling frequency, chirp period, and chirp number in a frame, are specified to attain the desired radar performance.

#### 5.1.1 Features

The main features of the sampling module are:

- **ADC compensation:** This feature provides the capability to scale the ADC data by left shifting;
- **Decimation Filter:** An anti-aliasing IIR (Infinite Impulse Response) filter for down-sampling pre-processing;
- **Interference Mitigation:** This feature provides on-line processing to detect and suppress the spike interference from other radar<sup>1</sup>.

### 5.2 Functional Description

The functional diagram and data flow of the sampling module are shown in Fig. 5.2. At the input, 4-channel ADC data sampled from RX 0-4 are combined to one channel, as illustrated in Fig. 5.3, and then fed into ADC compensation shifter, which scales the input simply by right or left shifting by certain bits according to AGC settings. Then the scaled data is low-pass filtered and down-sampled to keep a proper chirp sample length.

For both ADC compensation (Section 10.2.3) and decimation filter, input data and output data are updated at the edge of ADC\_data\_clk, whose frequency is 4 times Fs (ADC sampling frequency). To synchronize the data rate with following baseband modules driven by bb\_clk whose frequency  $f_{bb} = 100\text{MHz}$  or  $200\text{MHz}$ , the RAM MEM\_SAM with independent write and read interfaces is introduced. The length of data written to MEM\_SAM is determined by the register CFG\_SYS\_SIZE\_RNG\_BUF (see Table 5.1). In one chirp duration, the output of the decimation filter is written into MEM\_SAM driven by ADC\_data\_clk. And then driven by bb\_clk, data stored in MEM\_SAM is read out and sent to the interference mitigation module, as shown in Fig. 5.2 . Data stored in MEM\_SAM can be accessed by CPU (see Section 3.2.1.2 for details). The data storage format is shown in Fig. 5.1 .

---

<sup>1</sup> The ADC compensation and interference mitigation functions are implemented in the sampling module, and detailed discussions are covered in Section 10 and Section 11.

## 5.2. Functional Description

---

ADDRESS	High 16bits	Low 16bits	
0x00	Sample 0	Sample 1	RX0
0x04	Sample 0	Sample 1	RX1
0x08	Sample 0	Sample 1	RX2
0x0C	Sample 0	Sample 1	RX3
0x10	Sample 2	Sample 3	RX0
0x14	Sample 2	Sample 3	RX1
0x18	Sample 2	Sample 3	RX2
0x1C	Sample 2	Sample 3	RX3
⋮			

Fig. 5.1: Data Storage Format in MEM\_SAM

The interference mitigation feature is to detect and remove interference samples caused by other radar working within the same frequency bandwidth.

The ADC compensation, decimation filter, and interference mitigation functions can be bypassed by setting corresponding registers. The sampling module also provides some options to dump out ADC data for debugging purpose. Either raw ADC data or filtered ADC can be chosen for debugging by setting `CFG_SAM_DBG_SRC` to 1 or 0 (see Section 15.2.2 for details). Also, as shown in Fig. 5.4, by setting the `CFG_SAM_SINKER` to 1, the output of the sampling module can be sent to the RAM `MEM_BUF` for dumping it out to GPIO (see Section 15.2.1 for details) before fed into the FFT module (see Section 3.6.1.1 for baseband workflow).

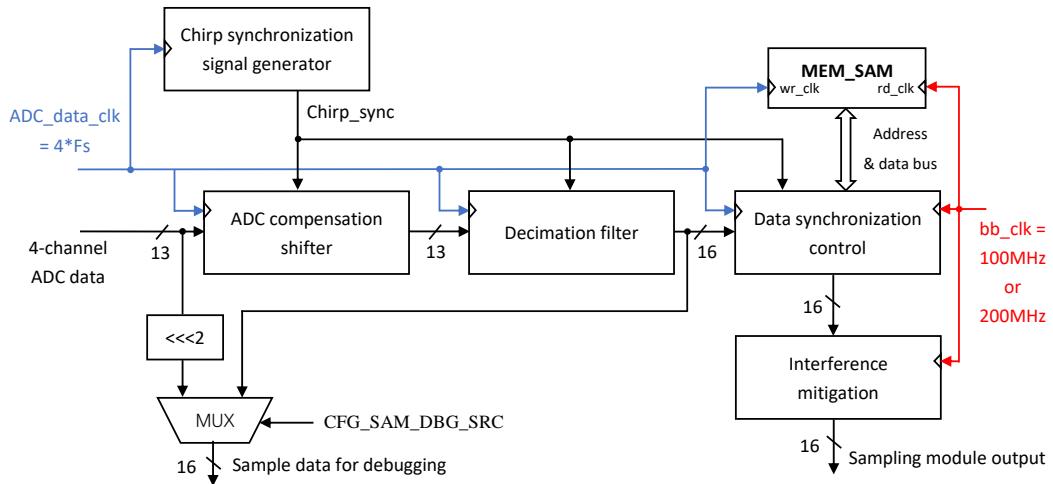


Fig. 5.2: Functional Diagram of the Sampling Module

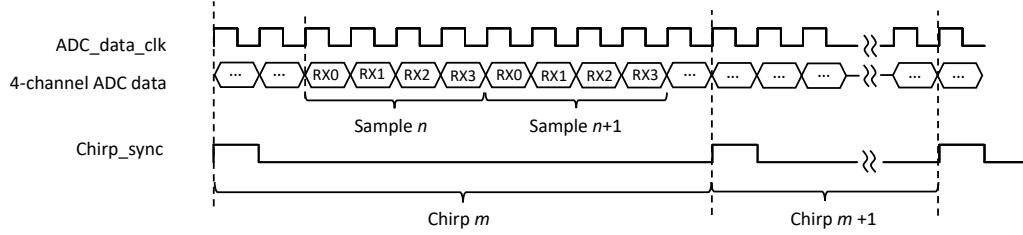


Fig. 5.3: Timing of 4-Channel ADC Data Transmission and Chirp Synchronization

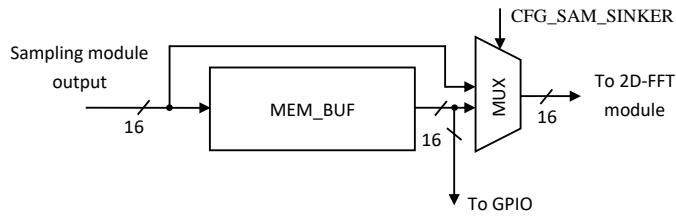


Fig. 5.4: Dumping Sampling Output to GPIO before Fed into 2D-FFT Module

As mentioned in [Section 1.4.1](#), baseband takes ADC output during the ramp-up period of a chirp. Although the RBPU can easily synchronize with ADC sampling at the beginning of each frame, two remaining issues need to be addressed by this module:

- Chirp period could be different depending on applications
- At the points where chirps start to ramp up and ramp down, the frequency synthesizer is in a transition period and radio is *not* in a stable state. It might take some time for the radio to settle down. During these transient periods, ADC data should be discarded.

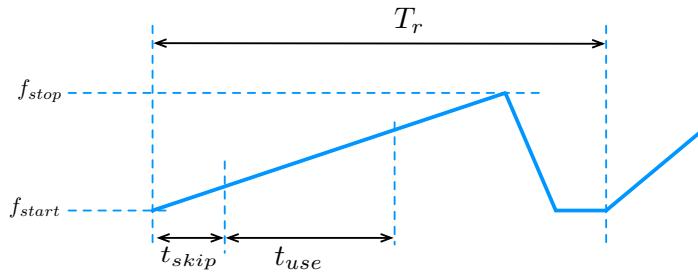


Fig. 5.5: Parameters of Chirp and Its Effective Sampling Positions

Consider a chirp as shown in [Fig. 5.5](#). One can specify chirp sharp and generate FMCW frame through radio configuration [[calterah2020](#)]. To instruct the RBPU to sample at the right time and space, the processing unit needs to be aware of

- Time to skip at the beginning of the chirp  $t_{skip}$ ;
- Actual used sample period  $t_{use}$ ;

## 5.2. Functional Description

---

- Period of chirp  $T_r$ ;
- Total number of chirps in a frame  $N_{ch}$ .

In particular, corresponding registers are set in terms of the number of samples.

The integrated ADC can run at 20/25/40/50 MHz. In some applications, chirp periods could be very long. Although ADC can generate more than 2,048 samples, the supported maximum FFT size is 2,048. In other words, if there is no special process, FFT engine and the remaining system are not able to sufficiently utilize the whole chirp period in this particular situation. Sampling preprocess unit also includes a down-sampling sub-module, whose structure is shown in Fig. 5.6.

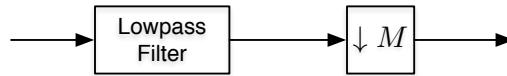


Fig. 5.6: Down-Sampling Structure

Down-sampling factor  $M$  ranges from 1 to 16. The low-pass filter is implemented as an 8-order Chebyshev Type II Filter, with 4 second-order sections. All 4 second-order sections share the same structure, as shown in Fig. 5.7.

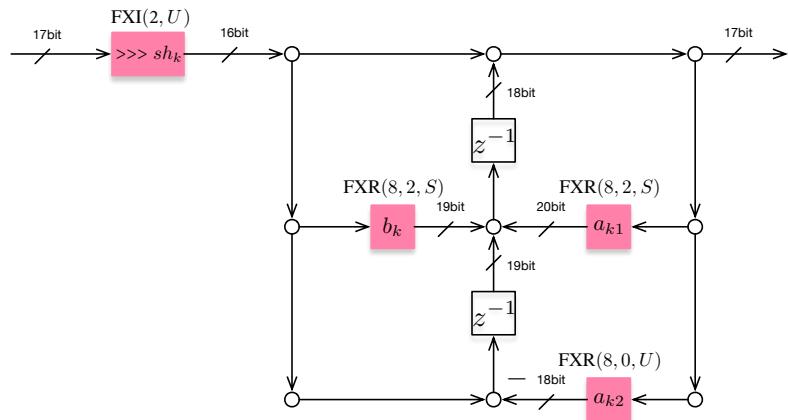


Fig. 5.7: Second-Order Section of Chebyshev Type II Filter

Detailed data types and data flow are also listed in Fig. 5.7. The colored parts (one shifter and three coefficients) in each second-order section are programmable. At the output of the last section, one additional scalar and shifter to balance the total filter gain are designed as shown in Fig. 5.8. The scalar and shifter are also programmable.

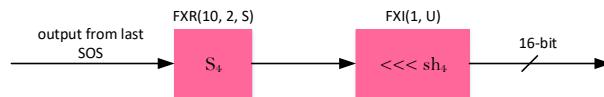


Fig. 5.8: Post-Process at Output of Last (Fourth) Second-Order Section

## 5.3 Programming Interfaces

### 5.3.1 Programming Chirp Information

$t_{skip}$ ,  $t_{use}$ , and  $T_r$  can be specified through registers `CFG_SYS_SIZE_RNG_SKP`, `CFG_SYS_SIZE_RNG_BUF`, and `CFG_SYS_SIZE_RNG_PRD` respectively. The units of the former two are both in ADC sample number **after** decimation filter, and the third one is in ADC sample number **before** decimation filter. In particular, if one needs to skip  $n$  samples at the start of the chirp and feeds total  $m$  samples into the FFT engine, then `CFG_SYS_SIZE_RNG_SKP` and `CFG_SYS_SIZE_RNG_BUF` should be  $n - 1$  and  $m - 1$ , respectively. It is similar for register `CFG_SYS_SIZE_RNG_PRD`. To program the decimation factor  $M$ , the register `CFG_SYS_SIZE_FLT` should be programmed to  $M - 1$ . To program the number of chirps per frame ( $N_{ch}$ ), `CFG_SYS_SIZE_VEL_BUF` should be set to  $N_{ch} - 1$ . An example is shown in Table 5.1.

Table 5.1: Example of Chirp Parameters and Baseband Setup

Parameter	Value	Register Name	Register Value
$T_r$	$50\mu s$	<code>CFG_SYS_SIZE_RNG_PRD</code>	$50 \times 20 - 1 = 999$
$t_{skip}$	$2\mu s$	<code>CFG_SYS_SIZE_RNG_SKP</code>	$2 \times 20 \div 2 - 1 = 19$
$t_{used}$	$40\mu s$	<code>CFG_SYS_SIZE_RNG_BUF</code>	$40 \times 20 \div 2 - 1 = 399$
$F_{ADC}$	20MHz	NA	NA
$M$ (decimate factor)	2	<code>CFG_SYS_SIZE_FLT</code>	1
$N_{ch}$	256	<code>CFG_SYS_SIZE_VEL_BUF</code>	255

### 5.3.2 Programming Decimation Filter

Fig. 5.9 shows the data flow in decimation filter. The decimation filter can be bypassed simply by setting `CFG_SYS_SIZE_FLT` to 0. Then the 13-bit signed input data is signed multiplied by 4, and then sent to the data synchronization module. When the register `CFG_SYS_SIZE_FLT` is set to a value greater than 0, the 13-bit ADC data is fed into 4 cascaded Chebyshev second-order sections(SOS), which are equivalent to an 8-order Chebyshev Type II filter. Then the output of SOS4 passes through the final shifter and scaler. Since the data format of the final scaling factor is FXR(10,2,S), the scaled data is truncated by 8-bit arithmetic right shift to keep a 16-bit effective bit-width.

To set up decimation filter, for the  $k$ th ( $k = 0, 1, 2, 3$ ) second-order section (see Fig. 5.7), the corresponding three coefficients and pre-shifter are specified with `CFG_SAM_F_k_B1`, `CFG_SAM_F_k_A1`, `CFG_SAM_F_k_A2`, and `CFG_SAM_F_k_S1`, respectively. For post-process after the last second-order section, the shifter and scalar are specified with the registers `CFG_SAM_FNL_SHF` and `CFG_SAM_FNL_SCL`, respectively. Data formats and bit-widths of these registers are given in Fig. 5.8 and Fig. 5.7. For details, please refer to Section B.

One thing to note is that `CFG_SAM_F_k_S1 = 0` means the data is not shifted, 1 means an arithmetic right shift by 1 bit, and 2 means an arithmetic right shift by 2 bits.



## 5.4. Limitation and Constraints

---

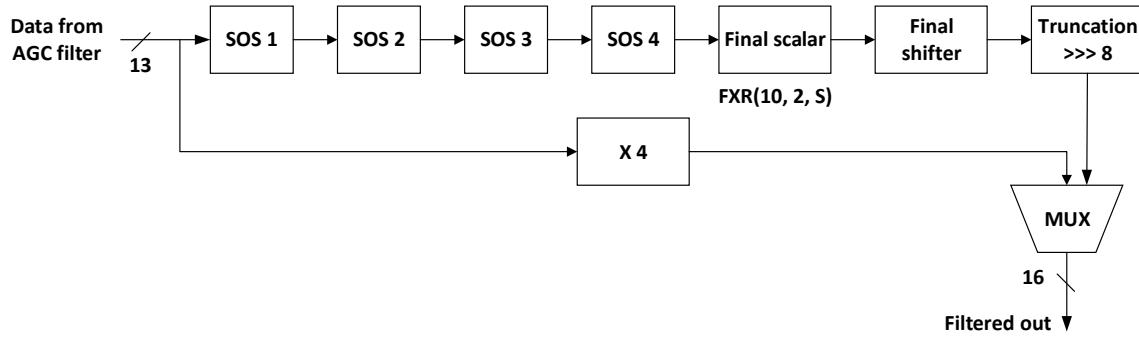


Fig. 5.9: Data Flow in Decimation Filter

## 5.4 Limitation and Constraints

To make sure that the pipeline structure works fine, the following constraint must be satisfied:

$$(t_{chirp} \cdot f_{bb} - \text{ant\_num}) \cdot 2 \geq \text{ant\_num} \cdot \text{range\_fft\_size}$$

Where  $t_{chirp}$  is the chirp period,  $f_{bb}$  is the frequency of baseband core clock, and  $\text{ant\_num}$  denotes the number of ADCs.

The left side of the inequality can be viewed as the system throughput in a chirp, and the right side can be viewed as the number of sample points to process in a chirp.

In Alps,  $\text{ant\_num}$  is 4, so the constraint can be simplified as:

$$t_{chirp} \cdot f_{bb} \geq 4 \cdot (\text{range\_fft\_size}/2 + 1)$$

When the interference mitigation function is enabled, that is, `CFG_SAM_SPK_RM_EN` is set to 1, the throughput is halved and more delays are introduced. The constraint becomes:

$$t_{chirp} \cdot f_{bb} \geq 4 \cdot (\text{range\_fft\_size} + 32)$$

## 5.5 Software and Configuration Suggestions

### 5.5.1 Configuring Decimation Filter for Constant DC Gain

When data passes through the decimation filter, it is suggested that the shifter and scaling registers for each Chebyshev second-order section be configured carefully to maintain the same DC (direct current) gain, just as when the decimation filter is bypassed. To achieve this, example configurations for some common cases are listed in [Table 5.2](#). In Case 0, `CFG_SYS_SIZE_FLT` is set to 0, so the decimation filter is bypassed. In Case 1, `CFG_SYS_SIZE_FLT` is greater than 0, and the coefficients of all 4 Chebyshev second-order sections are set to 0. To achieve the same DC gain as in Case 0, `CFG_SAM_FNL_SHF` and `CFG_SAM_FNL_SCL` are set to 1 and 0x1ff respectively, which equals being multiplied by 4. In Case 2, 3, and 4, IIR filter's coefficients are set to match the decimation factors configured by `CFG_SYS_SIZE_FLT`. The corresponding amplitudes and phase responses of the filters are shown in [Fig. 5.10](#), [Fig. 5.11](#) and [Fig. 5.12](#).

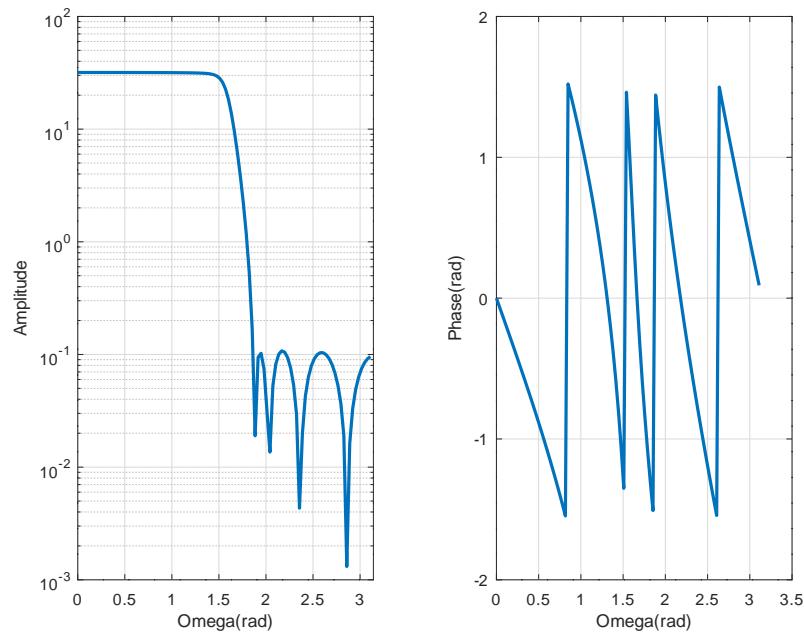


Fig. 5.10: IIR Filter Amplitude and Phase Response for Decimation Factor = 2

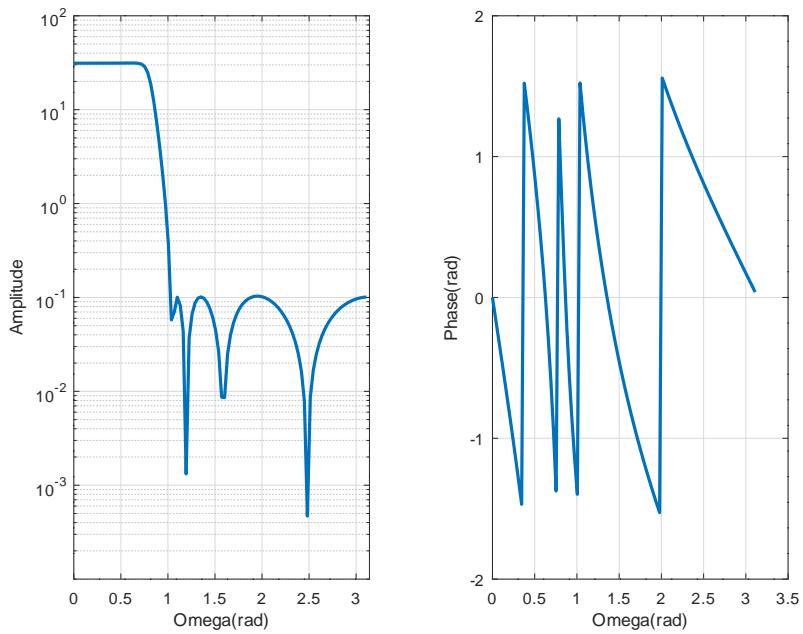


Fig. 5.11: IIR Filter Amplitude and Phase Response for Decimation Factor = 4

## 5.5. Software and Configuration Suggestions

---

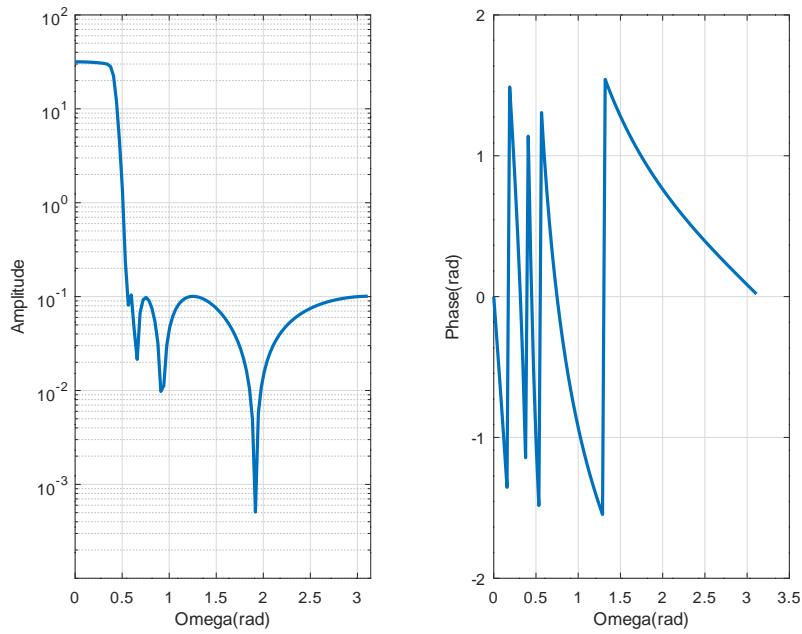


Fig. 5.12: IIR Filter Amplitude and Phase Response for Decimation Factor = 8

Table 5.2: Example Settings of Decimation Filter Coefficients with the Same DC Gain

Register Name	Case 0	Case 1	Case 2	Case 3	Case 4
CFG_SYS_SIZE_FLT	0	> 0	1	3	7
CFG_SAM_F_0_S1	BYPASS	0	0	1	2
CFG_SAM_F_0_B1	BYPASS	0	0x7b	0x65	0x2b
CFG_SAM_F_0_A1	BYPASS	0	0xe7	0x1f	0x46
CFG_SAM_F_0_A2	BYPASS	0	0x0f	0x14	0x4f
CFG_SAM_F_1_S1	BYPASS	0	0	1	1
CFG_SAM_F_1_B1	BYPASS	0	0x5b	0x02	0xb3
CFG_SAM_F_1_A1	BYPASS	0	0xee	0x2c	0x53
CFG_SAM_F_1_A2	BYPASS	0	0x30	0x42	0x7c
CFG_SAM_F_2_S1	BYPASS	0	0	0	0
CFG_SAM_F_2_B1	BYPASS	0	0x39	0xd1	0x9a
CFG_SAM_F_2_A1	BYPASS	0	0xf7	0x3f	0x62
CFG_SAM_F_2_A2	BYPASS	0	0x6c	0x87	0xb4
CFG_SAM_F_3_S1	BYPASS	0	0	0	0
CFG_SAM_F_3_B1	BYPASS	0	0x27	0xc0	0x93
CFG_SAM_F_3_A1	BYPASS	0	0xfe	0x51	0x6f
CFG_SAM_F_3_A2	BYPASS	0	0xc4	0xd4	0xe6
CFG_SAM_FNL_SHF	BYPASS	1	1	1	1
CFG_SAM_FNL_SCL	BYPASS	0x1ff	0x1e1	0x168	0x11f

### 5.5.2 Coexistence with DC Calibration

The device provides an internal DC-level calibration function. To ensure that this function works fine, the path gain from ADC data input to the sampling module output must be constant. There are two other functions that could possibly change the path gain:

- Decimation filter, which contains many shifters and scalers. These registers should be configured to maintain the same path gain according to [Table 5.2](#).
- Phase scrambling. When phase scrambling is enabled, a transmitted wave is phase shifted with 180 degrees in RF TX module and the received wave is multiplied by -1 in the sampling module when `CFG_SAM_DINT_ENA` is set to 1. To avoid this sign inverse, the bit register `CFG_SAM_DINT_ENA` needs to be set to 0 before DC calibration.

## 5.6 Examples

[Table 5.3](#) shows example register settings. Here the ADC sampling frequency is chosen as 20 MHz. Assume FMCW bandwidth is 200MHz, ramp-up time is  $43\mu s$ , ramp-down time is  $5\mu s$  (see [Section 4](#) for how to configure a chirp generation parameters).

The chirp period is  $50\mu s$ , i.e.,  $2\mu s$  is idle in a chirp. The time length to skip in the header of ramp-up section is  $2\mu s$ . The length of used samples in ramp-up section is chosen as  $40\mu s$ , i.e. 800 sample points to use given 20MHz sampling frequency. The down\_sample rate is 2. So the true number of sample points skipped is  $2\mu s \times 20\text{MHz} \div 2 = 20$  and number of sample points to use for range FFT is  $40\mu s \times 20\text{MHz} \div 2 = 400$ . The range FFT size is 512. The chirp number in a frame and velocity FFT size are both 256. The related registers are configured as in [Table 5.3](#). Decimation filter coefficients are chosen as the same values as in case 2 in [Table 5.2](#).



## 5.6. Examples

---

Table 5.3: Example Settings of Sampling Related Registers

Register Name	Register Value	Corresponding Parameters
CFG_SYS_SIZE_RNG_PRD	999	Chirp period is $50\mu s$ ( $F_{ADC} = 20MHz$ )
CFG_SYS_SIZE_FLT	1	Down sampling rate is 2
CFG_SYS_SIZE_RNG_SKP	19	Skipped sample length from chirp start is $2\mu s$ ( $F_{ADC} = 20MHz$ )
CFG_SYS_SIZE_RNG_BUF	399	Used sample length after skipping for range FFT is $40\mu s$ ( $F_{ADC} = 20MHz$ )
CFG_SYS_SIZE_RNG_FFT	511	Range FFT size is 512
CFG_SYS_SIZE_BPM	0	Virtual array mode is off
CFG_SYS_SIZE_VEL_BUF	255	Chirp number in a frame is 256
CFG_SYS_SIZE_VEL_FFT	255	Velocity FFT size is 256
CFG_SAM_F_0_S1	0	input arithmetic right shift by 0 bit in SOS1
CFG_SAM_F_0_B1	0x7b	coefficient b1 in SOS1
CFG_SAM_F_0_A1	0xe7	coefficient a1 in SOS1
CFG_SAM_F_0_A2	0x0f	coefficient a2 in SOS1
CFG_SAM_F_1_S1	0	input arithmetic right shift by 0 bit in SOS2
CFG_SAM_F_1_B1	0x5b	coefficient b1 in SOS2
CFG_SAM_F_1_A1	0xee	coefficient a1 in SOS2
CFG_SAM_F_1_A2	0x30	coefficient a2 in SOS2
CFG_SAM_F_2_S1	0	input arithmetic right shift by 0 bit in SOS3
CFG_SAM_F_2_B1	0x39	coefficient b1 in SOS3
CFG_SAM_F_2_A1	0x62	coefficient a1 in SOS3
CFG_SAM_F_2_A2	0x6c	coefficient a2 in SOS3
CFG_SAM_F_3_S1	0	input arithmetic right shift by 0 bit in SOS4
CFG_SAM_F_3_B1	0x27	coefficient b1 in SOS4
CFG_SAM_F_3_A1	0xfe	coefficient a1 in SOS4
CFG_SAM_F_3_A2	0xc4	coefficient a2 in SOS4
CFG_SAM_FNL_SHF	1	final arithmetic left shift by 1 bit
CFG_SAM_FNL_SCL	0x1e1	final scaling $481/256 = 1.8789$

## FAST FOURIER TRANSFORM (FFT)

### 6.1 Overview

The FFT module takes input from sample preprocess and performs FFT, including 1D-FFT and 2D-FFT. 1D-FFT is performed in the range dimension, and 2D-FFT handles data in the Doppler dimension based on the output of 1D-FFT. This chapter is devoted to the FFT module.

#### 6.1.1 Features

- **Programmable FFT Windows:** Include windows for both range and Doppler gates;
- **FFT Engine:** Performs FFT computations;
- **Zero Doppler Removal:** The capability to remove all objects with no radical velocity related to the radar system.
- **Phase Compensation:** It performs phase compensation after 2D-FFT when FMCW triggers virtual array.

## 6.2 Functional Description

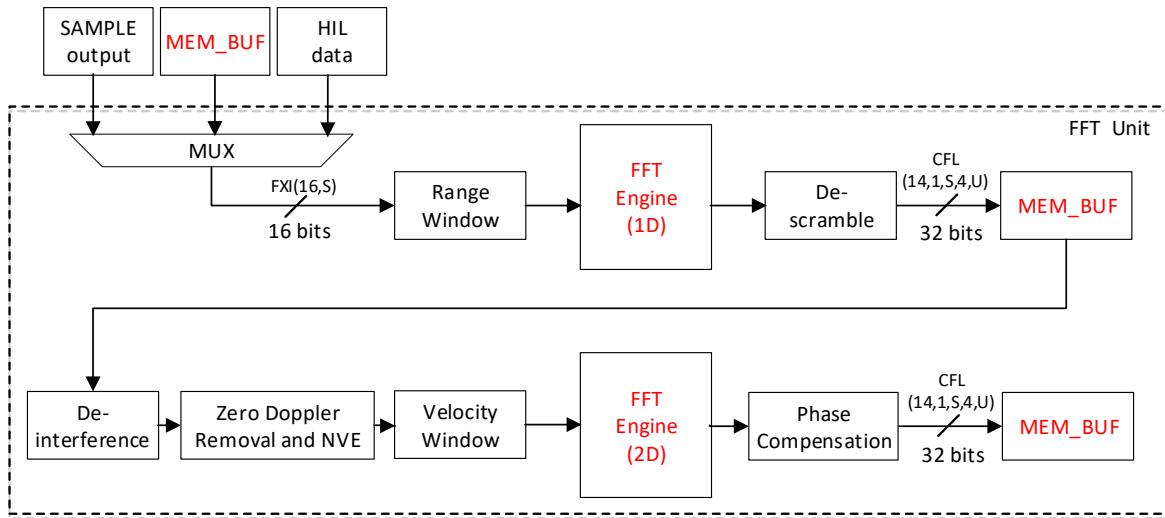


Fig. 6.1: Diagram of FFT Computing Unit

As shown in Fig. 6.1, FFT computation goes through the following steps.

1. Input mux.

16-bit data is selected as input depending on the configuration from one of the following:

- SAMPLE output
- MEM\_BUF data
- HIL data

2. Range windowing.

16-bit input (per channel) data stream is applied with range windowing coefficients from MEM\_WIN.

3. 1D-FFT.

The output data of range windowing goes into the 1D-FFT engine for the fixed-point FFT computation.

4. De-scrambling.

De-scrambling handles phase scrambling. For more information, see *Phase Scrambling*.

5. 1D-FFT output.

1D-FFT result is 32-bit with the format of CFL(14, 1, S, 4, U). The result data is stored into the shared 2-MB RAM (MEM\_BUF).

---

**Note:** Since the input is real data, only half amount of the data is saved due to the symmetric property of FFT.

---

6. Anti-interference.

Anti-interference handles frequency hopping and chirp shifting. For more information, see [Frequency Hopping](#) and [Chirp Shifting](#).

7. Zero Doppler removal and noise variance estimation (NVE).

This step is to handle zero Doppler removal and noise estimation. For more information on NVE, see [Noise Estimator](#).

8. Velocity windowing.

16-bit input (per channel) data stream is applied with velocity windowing coefficients from MEM\_WIN.

9. 2D-FFT.

The output data of velocity windowing goes into the 2D-FFT engine for the fixed-point FFT computation.

10. Phase compensation.

Phase compensation is for compensating phase in virtual array.

11. 2D-FFT output.

2D-FFT result is 32-bit with the format of CFL(14, 1,  $S$ , 4,  $U$ ). The result data is stored into the shared 2-MB RAM (MEM\_BUF). Refer to the *Data Order* in [SAMPLE Output Data](#) for detailed information.

### 6.2.1 FFT Source Selection

As shown in [Fig. 6.1](#), there are 3 kinds of input data, *SAMPLE output*, *MEM\_BUF data* and *HIL data*. But only one can be selected at a time. No matter which input is selected, the input data is considered as ADC data. 2 parallel channels of 16-bit input data will directly go into range windowing.

Generally, it is suggested that *SAMPLE output* be selected, which will directly go into FFT and thus save time. The other 2 types of input, *MEM\_BUF data* and *HIL data*, are used for debugging. For detailed debugging information, refer to [Data Collection and Hardware-in-the-Loop \(HIL\)](#).

### 6.2.2 FFT Windows

Windowing can improve FFT spectral leakage that is caused by discontinuities in the original, non-integer number of periods in a signal. Windowing reduces the amplitude of the discontinuities at the boundaries of each finite sequence acquired by the digitizer. No window is often called the uniform or rectangular window because there is still a windowing effect.

There are 2 windowing processes, corresponding to 1D-FFT and 2D-FFT respectively. 1D-FFT window is called range window, and 2D-FFT window is called velocity window. These 2 windows are programmable and stored in different RAM. Each entry is in the format of FXR(14, 0,  $U$ )<sup>1</sup>.

Range window supports maximum 2048 point FFT, so the maximum window size is 2048. Range window will store the whole window data to MEM\_WIN, which provides the capability to deal with complicated applications, such as spur suppression.

Velocity window supports maximum 1024 point FFT, so the maximum window size is 1024. Considering the symmetric property of window, velocity window will only store half of the window data to MEM\_WIN, thus saving the memory.

---

<sup>1</sup> Please refer to [Data Representation](#) for its detail meaning.

## 6.2. Functional Description

---

### 6.2.3 FFT Engine

The FFT engine has some flexibility. The FFT engine is based on radix-2 decimation in time and the supporting FFT sizes are shown in [Table 6.1](#). Essentially, FFT size configuration is limited by 2-MB RAM. Maximum FFT sizes along range dimension and Doppler dimension are 2048 and 1024 respectively. 1D-FFT and 2D-FFT use the same FFT engine, and both outputs are saved in the same RAM named MEM\_BUF. So 2D-FFT output will rewrite 1D-FFT output.

Table 6.1: Supported FFT Sizes

	64	128	256	512	1024	2048
64	Y	Y	Y	Y	Y	Y
128	Y	Y	Y	Y	Y	Y
256	Y	Y	Y	Y	Y	
512	Y	Y	Y	Y		
1024	Y	Y	Y			

[Fig. 6.2](#) shows some detailed implementation of the butterfly structure. There are two right shifters at its output, (i.e., divided by 2). For each FFT stage, users can choose to apply the shifter or not, so as to achieve better numeric performance.

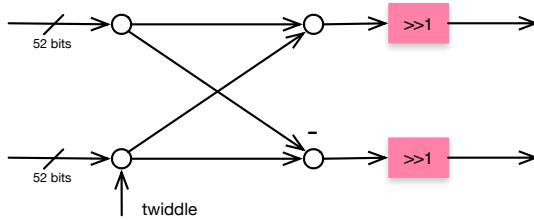


Fig. 6.2: A Diagram of Radix-2 Butterfly Used in FFT Engine

### 6.2.4 Zero Doppler Removal

The Zero Doppler removal feature is aimed to eliminate the static objects before 2D-FFT in the scenarios where static objects do not need to be considered. This feature can be enabled or disabled through the register CFG\_FFT\_ZER\_DPL\_ENB.

If zero Doppler removal is disabled, 1D-FFT results will directly go into the Doppler dimension. If zero Doppler is enabled, 1D-FFT result minus the average value  $\hat{x}$ , which is the arithmetic mean along the Doppler dimension, will be sent for velocity windowing. The average value  $\hat{x}$  is obtained by  $\hat{x} = \frac{1}{n} \sum_{i=0}^{n-1} x_i$ , where  $n$  is the number of chirps in a frame and equals the value of CFG\_SYS\_SIZE\_VEL\_BUF.

## 6.2.5 Phase Compensation

There will be phase differences of reflect signals of different Rx antennas in virtual array, see [Phase Compensation in Virtual Array](#).

# 6.3 Programming Interfaces

## 6.3.1 FFT Source Selection

The registers `CFG_SAM_SINKER` and `CFG_SYS_ENABLE` are used for selecting the input data.

### 6.3.1.1 `CFG_SAM_SINKER`

The register `CFG_SAM_SINKER` determines whether ADC data will go into the 1D-FFT engine directly or not, which will take effect when the *SAMPLE state* is enabled. The value can be set to 0 or 1.

- If `CFG_SAM_SINKER` is set to 0, ADC data will go into `MEM_BUF` while ADC data is streaming in, which is used for data debugging. After all ADC data is collected, `MEM_BUF` data will then be output for 1D-FFT calculation. The baseband workflow is shown in [Section 3.6.1.1](#).
- If `CFG_SAM_SINKER` is set to 1, ADC data will go into the 1D-FFT engine while ADC data is streaming in, which is time-saving because 1D-FFT calculation time is mostly shared with chirp sequence time. The baseband workflow is shown in [Section 3.6.1.2](#).

### 6.3.1.2 `CFG_SYS_ENABLE`

The register `CFG_SYS_ENABLE` determines which states will be enabled. This register has 9 bits, each bit corresponding to a baseband processing state.

- Bit value of 1 means enabling the corresponding state.
- Bit value of 0 means disabling the corresponding state.

*SAMPLE* (Bit 2 in `CFG_SYS_ENABLE`) and *HIL* (Bit 1 in `CFG_SYS_ENABLE`) are the 2 states used for FFT data input, and they are mutually exclusive. So either Bit 2 or Bit 1 of `CFG_SYS_ENABLE` must be set to enabled before running FFT calculation.

## 6.3.2 FFT Windows

### 6.3.2.1 Window Size

The window size is determined by registers `CFG_SYS_SIZE_RNG_BUF` and `CFG_SYS_SIZE_VEL_BUF`, which correspond to the range window and the velocity window.

`CFG_SYS_SIZE_RNG_BUF` determines the effective data size in the range dimension, which is the FFT window size in the range dimension. Suppose the effective data size in the range dimension is  $2^{n_1}$ . The value of this register should be programmed to  $2^{n_1} - 1$ .

`CFG_SYS_SIZE_VEL_BUF` determines the effective chirp number in the Doppler dimension, which is the FFT window size of the Doppler dimension. Suppose the effective chirp number in the Doppler dimension is  $2^{n_2}$ . Then the value of this register should be programmed to  $2^{n_2} - 1$ .



### 6.3.2.2 Window RAM

Physically, range window RAM and velocity window RAM are 2 independent RAM. For programming simplicity, they are both activated by MEM\_WIN. To program the windowing RAM, first assign the memory space starting from 0xE0\_0000 to MEM\_WIN by programming CFG\_SYS\_MEM\_ACT (0xC0\_0014) to 2. Then program the window coefficients in MEM\_WIN. Each coefficient should be stored at an address of a multiply of 4. Coefficients of the range window and velocity window should be programmed in an alternative way as shown in Fig. 6.3, The address mapping in this figure is designed from CPU vision, which is a virtual mapping. In physical layer, window RAM is cut 4 bank areas, bank start addresses(byte) of range window is 0, 0x2000, 0x4000 and 0x6000 mapped to bank 0, 1, 2, 3. bank start addresses(byte) of velocity window is 0, 0x800, 0x1000, and 0x1800 mapped to bank 0, 1, 2, 3.

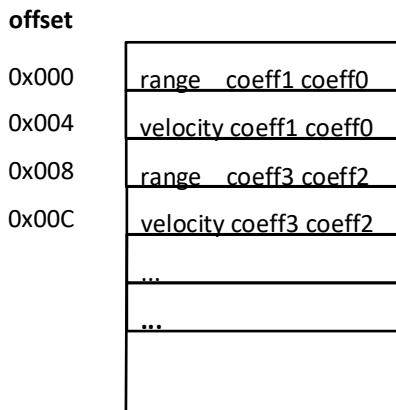


Fig. 6.3: Memory Table Layout of FFT Windows

### 6.3.2.3 No Windowing

Windowing can be disabled by setting the register CFG\_FFT\_NO\_WIN to 1.

## 6.3.3 FFT Engine

Two basic knobs are designed for FFT computing units, which are FFT size and FFT stage shifters.

### 6.3.3.1 FFT Size

2D-FFT size is configured through the register CFG\_SYS\_SIZE\_RNG\_FFT (0xC0\_0030) and CFG\_SYS\_SIZE\_VEL\_FFT (0xC0\_003C), corresponding to the range dimension and the Doppler dimension respectively. Suppose the size of 2D-FFT is  $2^{n_1} \times 2^{n_2}$ . Then program the two registers with the value of  $2^{n_1} - 1$  and  $2^{n_2} - 1$ , respectively.



### 6.3.3.2 FFT Stage Shifter

FFT stage shifter is determined by registers `CFG_FFT_SHFT_RNG` and `CFG_FFT_SHFT_VEL`. Implemented with the radix-2 butterfly structure, FFT with the size of  $2^n$  has  $n$  stages. For each stage, one can choose to do the right-shift (divided by 2) or not, by setting the registers `CFG_FFT_SHFT_RNG` (0xC0\_0600) and `CFG_FFT_SHFT_VEL` (0xC0\_0604), respectively.

For each register, its  $i$ th bit controls  $i$ th stage (1 means shifting, 0 means not shifting)<sup>2</sup>. These knobs are used to achieve better numeric performance. For example, with strong reflectors, one can apply shifters at the beginning stages of 1D-FFT so that the butterfly engine will not be saturated or clipped. Another example is that with very weak signals, one can disable shifters at the ending stages of 2D-FFT to reduce quantization errors introduced by the butterfly engine.

### 6.3.4 Zero Doppler Removal

To turn on the zero Doppler removal feature, set the register `CFG_FFT_ZER_DPL_ENB` to 1.

### 6.3.5 Phase Compensation

`CFG_FFT_DBPM_ENA` and `CFG_FFT_DBPM_DIR` are used to control the phase compensation in virtual array.

- `CFG_FFT_DBPM_ENA` is used to enable or disable this feature. Value with 1 means enabling and 0 means disabling.
- `CFG_FFT_DBPM_DIR` is used to choose the compensation direction. Value with 0 means positive phase compensation and 1 means negative phase compensation.

## 6.4 Limitation and Constraints

### 6.4.1 FFT Source Selection

The registers `CFG_SAM_SINKER` and `CFG_SYS_ENABLE` are used to decide the input data selection. Only one of the 3 kinds of input data can be selected. Otherwise, baseband will go wrong. Specifically, Bit 1 and Bit 2 in `CFG_SYS_ENABLE` should not be enabled at the same time.

- To select *SAMPLE output* as the FFT source:

In `CFG_SYS_ENABLE`, enable Bit 2 (*SAMPLE state*) and disable Bit 4 (*1D-FFT state*), and set `CFG_SAM_SINKER` to 1.

In this case, 1D-FFT state will share the state of SAMPLE and the input will directly go into the 1D-FFT engine.

- To select *MEM\_BUF data* as the FFT source:

In `CFG_SYS_ENABLE`, enable Bit 2 (*SAMPLE state*) and enable Bit 4 (*1D-FFT state*), and set `CFG_SAM_SINKER` to 0.

1D-FFT state should be enabled because the input is from *MEM\_BUF* and 1D-FFT state cannot be shared with other baseband states.

- To select *HIL data* as the FFT source:

In `CFG_SYS_ENABLE`, enable Bit 1 (*HIL state*) and disable Bit 4 (*1D-FFT state*), and `CFG_SAM_SINKER` is irrelevant here.

In this case, 1D-FFT state will share the state of HIL and the input will directly go into the 1D-FFT engine.

<sup>2</sup> In this user guide, bits are always enumerated from its LSB within a word or a byte.



### 6.4.2 FFT Windows

It is suggested that `CFG_SYS_SIZE_RNG_BUF` be configured to a value equivalent or approximately equivalent to that of `CFG_SYS_SIZE_RNG_FFT`. Due to the symmetry of coefficients in `MEM_WIN`, make sure that the size of range window is even. Suppose the size is  $2n$ . Then `CFG_SYS_SIZE_RNG_BUF` should be set to  $2n - 1$ .

`CFG_SYS_SIZE_VEL_BUF` is recommended to be configured to a value equivalent to that of `CFG_SYS_SIZE_VEL_FFT`. Likewise, make sure that the size of velocity window is even. Suppose the size is  $2m$ . Then `CFG_SYS_SIZE_RNG_BUF` should be set to  $2m - 1$ .

`CFG_FFT_NO_WIN` is recommended to be set to 0 for most applications and users.

### 6.4.3 FFT Size

FFT size is limited by `MEM_BUF` size. Please refer to [2MB Limited](#) for more detailed information.

### 6.4.4 Phase Compensation

When FMCW triggers virtual array but does not trigger anti velocity ambiguity, this feature should be enabled to compensate for the phase difference caused by velocity. Please refer to [Phase Compensation in Virtual Array](#) for detail information.

When FMCW triggers virtual array and also triggers anti velocity ambiguity, this feature should be disabled. Another compensation will be enabled in DoA, see [Phase Compensation for Virtual Array](#).

## 6.5 Software and Config Suggestions

### 6.5.1 Sizes of FFTs

There are several factors impact the choice of FFT sizes. For the range gate, FFT size is determined by accuracy or resolution. For the range gate domain, the total number of samples fed into FFT engine should be no greater than the range-gate FFT size; for the Doppler gate domain, the total number of chirps of each channel should be no greater than the doppler gate FFT size. For example, suppose the total number of samples are 400, one should choose FFT size to be at least 512.

On the other hand, it is recommended to that the total number of samples should be as close to FFT size as possible, because it benefits resolution. For example, suppose Doppler-gate FFT is 128, it is preferred to use 128 chirps than 120 chirps.

### 6.5.2 FFT Stage Shifters

In standard FFT, there should be a right shifter for each FFT stage, which ensures there is no output overflow (saturation) at each stage output. However, since the FFT is implemented in fixed-point format, the right shifter drops the LSB at each stage, which could increases the quantization noise.

Regarding to the saturation situations, if the input is not saturated, output has less probability of saturation. Therefore, it is recommended to enable the shifters at early stages and disable them at the last stages. For example, one can enable all shifters for the range gate and disable the last three stages for the Doppler gate.



## 6.6 Examples

### 6.6.1 Register Settings

Table 6.2: FFT Configuration Suggestions

Registers Name	Example Value	Description
CFG_SYS_SIZE_RNG_PRD	999 (0x3E7)	Chirp period is $50\mu s$ ( $F_{ADC} = 20MHz$ )
CFG_SYS_SIZE_FLT	1	Down sampling rate is 2
CFG_SYS_SIZE_RNG_SKP	19 (0x13)	Skipped sample length from chirp start is $2\mu s$ ( $F_{ADC} = 20MHz$ )
CFG_SYS_SIZE_RNG_BUF	399 (0x18F)	Used sample length after skipping for range FFT is $40\mu s$ ( $F_{ADC} = 20MHz$ )
CFG_SYS_SIZE_RNG_FFT	511 (0x1FF)	Range FFT size is 512
CFG_SYS_SIZE_BPM	0	Virtual array mode is off
CFG_SYS_SIZE_VEL_BUF	255 (0xFF)	Chirp number in a frame is 256
CFG_SYS_SIZE_VEL_FFT	255 (0xFF)	Velocity FFT size is 256
CFG_FFT_SHFT_RNG	511 (0x1ff)	Enable 9 stage shifter, stage 1~9
CFG_FFT_SHFT_VEL	255 (0xff)	Enable 8 stage shifter, stage 1~8

### 6.6.2 FFT Windows

The following pseudo-code shows how to program FFT windows to RAM.

```

1 #define SYS_SIZE_RNG_BUF 399
2 #define SYS_SIZE_VEL_BUF 255
3 #define MEM_WIN 1
4 #define CFG_SYS_MEM_ACT 0xC0_0014
5 #define BANK_ACT 0
6
7 int main()
8 {
9     int rng_win[SYS_SIZE_RNG_BUF] = {0, 1, ... };           // whole range window
10    int vel_win[SYS_SIZE_VEL_BUF] = {0, 1, ... };           // half velocity window
11
12    // switch to FFT windows RAM
13    reg_write(CFG_SYS_MEM_ACT, MEM_WIN);
14
15    // programming range window in even address
16    int rng_bnk_offset = BANK_ACT * 0x2000;
17    int rng_win_cnt = 0;
18    for(int i = 0; i < (SYS_SIZE_RNG_BUF + 1)/2; i++) {
19        int coe0 = rng_win(2*i);
20        int coe1 = rng_win(2*i + 1);
21        mem_win_write((rng_bnk_offset + (rng_win_cnt*2) << 2), (coe1<<16)|coe0); // ←two coes in a word
22        rng_win_cnt = rng_win_cnt + 1;
23    }
24
25    // programming velocity window in odd address
26    int vel_bnk_offset = BANK_ACT * 0x800;
27    int vel_win_cnt = 0;
28    for(int i = 0; i < ((SYS_SIZE_VEL_BUF + 1)/2 + 1)/2; i++) {

```

(continues on next page)



## 6.6. Examples

---

(continued from previous page)

```
29     int coe0 = vel_win(2*i);
30     int coe1 = vel_win(2*i + 1);
31     mem_win_write((vel_bnk_offset + (vel_win_cnt*2 + 1) << 2), (coe1<<16) | coe0);
32     vel_win_cnt = vel_win_cnt + 1;
33 }
34
35     return 0;
36 }
```



## CONSTANT FALSE ALARM RATE (CFAR) DETECTORS

### 7.1 Overview

Constant false alarm rate (CFAR) detector is a standard object-detection sub-module in modern radar systems. It takes output from 2D-FFT and outputs FFT indices, which correspond to reflectors in range-Doppler domain. Since each channel outputs 2D-FFT results independently, CFAR module combines the 2D-FFT results from different channels into one 2D array (or matrix) before its object searching. CFAR modules not only supports multiple choices of CFAR algorithms, but also can be configured in different ways for different applications.

#### 7.1.1 Features

- CFAR combiners
  - SISO Combination (non-coherent combination)
  - MIMO Combination (coherent combination)
- Region dependent CFAR framework
  - Up to 8 programmable regions
  - Different CFAR schemes for different regions
- 2D sliding windows
  - Rectangular window
  - Cross window
- CFAR algorithms
  - Traditional CFAR algorithms with flexible configuration and extension
  - Calterah proprietary CFAR algorithms

### 7.2 Functional Description

#### 7.2.1 CFAR Combiner

Before searching for objects, 2D-FFT results need to be combined into one 2D-array called Range-Doppler matrix (RDM). Object searching is restricted within a 2D-array instead of a 3D-array, which reduces searching effort. Alps implements two ways of combination. One is called SISO (single-input-single-output) and the other is called MIMO (multiple-input-multiple-output). The former one is also called non-coherent combination and the latter is also known as coherent combination.

## 7.2. Functional Description

---

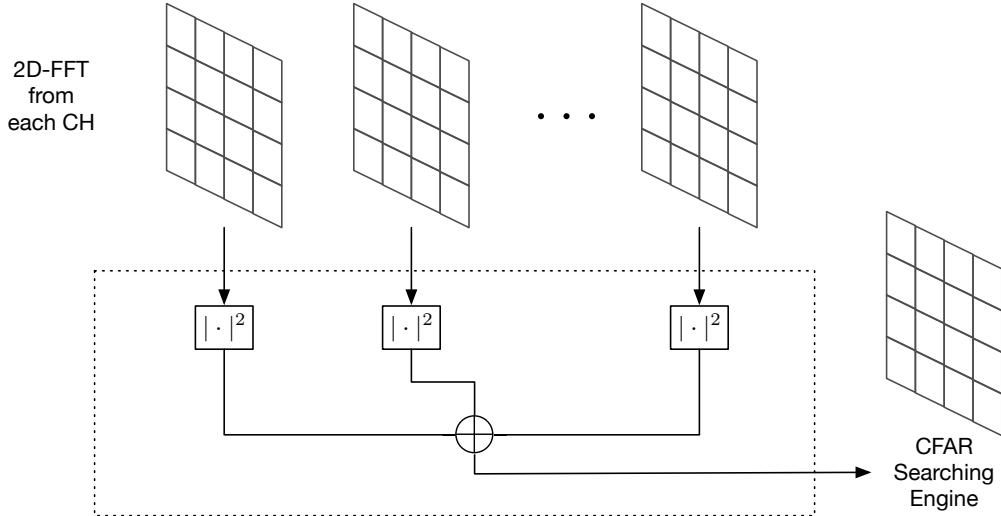


Fig. 7.1: Diagram of SISO Combination

Fig. 7.1 shows the basic data processing diagram of SISO combination. For each channel's 2D-FFT, SISO combination element-wisely takes magnitude-square (power) and then combines them in power domain. Intuitively, it assumes that all objects can be located regardless of the angle related to the radar module. So the combination does not take any prior information of the objects' directions.

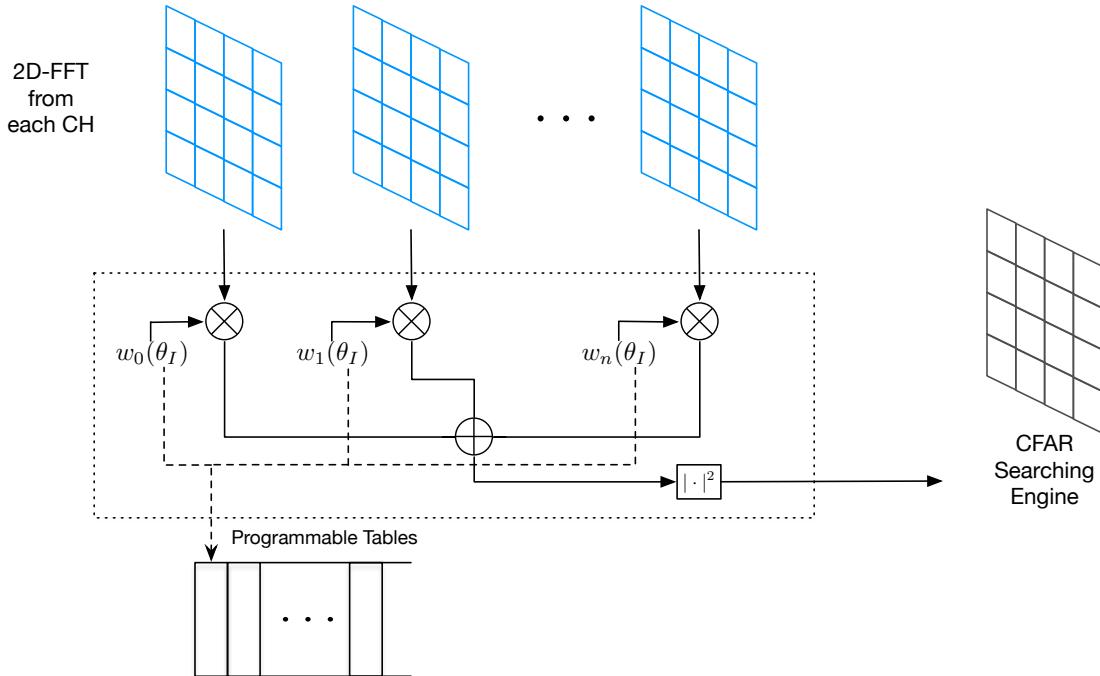


Fig. 7.2: MIMO Combination

MIMO combination does something slightly different, as shown in Fig. 7.2. For each channel's 2D-FFT, it takes (complex) weighted sum and then takes magnitude-square. Intuitively, the weights before the adder are steering RX

channels towards a certain direction. If there is an object near around this direction, the post-combination signal-to-noise (SNR) can be higher compared against SISO combination, thus a higher detection probability in CFAR searching engine. However, it is possible that the object is not near around this direction so that the post-combination SNR could be very poor. For example, when the object is sitting at the nulling angle of the virtual RX antenna pattern, the SNR could be  $-\infty$  in theory. To overcome this, multiple groups of weights are needed to cover multiple pre-combination directions. Alps allows users to specify up to 16 groups of (up to 32) complex weights. With MIMO combination, each combined RDM is fed into an object searching engine. For a particular detection bin, as long as it pass the CFAR threshold in one of the RDMs, the corresponding FFT indices will be output.

The major drawback of MIMO combination is the time consumption compared to SISO combination. With SISO combination, only one RDM is fed into searching engines, whereas MIMO generates multiple RDMs and searching engines do the searching sequentially.

## 7.2.2 RDM Region Partition

Alps provides a way of doing so-called region dependent CFAR. Namely, one can partition the RDM into regions and use different object searching schemes for different regions.

Alps supports up to 8 regions in RDM and the partition typically looks as shown in Fig. 7.3. Different numbers stand for different regions. In particular, RDM is cut along the range gate into four strips. For each strip, one can define two regions. Based on the cyclic-shifting property along the Doppler gate, even-numbered regions have wrapping-around property along the Doppler gate.

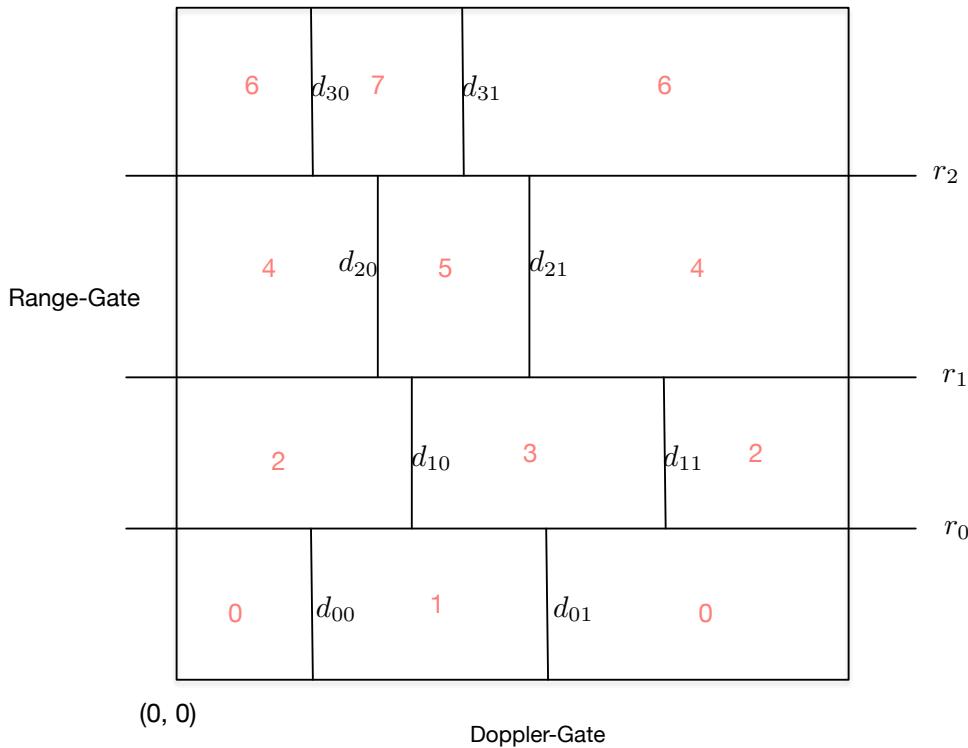


Fig. 7.3: Region Partition in RDM

More precisely, let  $r_i$ ,  $i = 0, 1, 2$  denote the cut boundaries along the range gate and assume that  $r_0 \leq r_1 \leq r_2$ . Also let  $d_{lj}$ ,  $j = 0, 1$  denote the boundaries along the Doppler gate within a strip  $l$  and assume that  $d_{l0} \leq d_{l1}$ . Then Table

## 7.2. Functional Description

---

7.1 is the decision table of all regions, where  $(k, p)$  is location of the cell (or bin) under test (CUT).

Table 7.1: Region Decision Table of Bin with Index  $(k, p)$

Region	Condition
0	$k \leq r_0$ and $(p \leq d_{00} \text{ or } p > d_{01})$
1	$k \leq r_0$ and $d_{00} < p \leq d_{01}$
2	$r_0 < k \leq r_1$ and $(p \leq d_{10} \text{ or } p > d_{11})$
3	$r_0 < k \leq r_1$ and $d_{10} < p \leq d_{11}$
4	$r_1 < k \leq r_2$ and $(p \leq d_{20} \text{ or } p > d_{21})$
5	$r_1 < k \leq r_2$ and $d_{20} < p \leq d_{21}$
6	$r_2 < k$ and $(p \leq d_{30} \text{ or } p > d_{31})$
7	$r_2 < k$ and $d_{30} < p \leq d_{31}$

Note that values of  $r_i$  and  $d_{lj}$  are all programmable so that one can achieve any partition with the same pattern as shown in Fig. 7.3.

Most CFAR related parameters are region dependent. But some of them are not. Readers should be aware of that throughout remaining discussions of this chapter and refer to summary in Section 7.4.

### 7.2.3 Sliding Windows

Most of CFAR algorithms are designed in a sliding window fashion. In other words, for each cell under test (CUT), one collects certain cells as the reference cells within a local (2D) window, whose center is usually CUT itself. By comparing reference cells and CUT, it determines whether the CUT contains pure noise or signals from reflectors. Based on the shape of sliding windows, Alps provides rectangular window and cross window.

---

**Note:** Sliding window type can not be region dependent. In other words, either rectangular window or cross window can be applied to all subregions.

---

#### 7.2.3.1 Rectangular Window

A rectangular window is defined by a bit mask of size  $11 \times 11$ . Value 1 means that the corresponding cell is selected as reference, value 0 means otherwise. Therefore, the maximum possible number of reference points are 121. In some applications, a larger window is preferred to avoid masking effects or edging. One can do decimation as shown in Fig. 7.4 along the range gate or the Doppler gate or both. Note that the maximum decimation factor is 1, so the maximum effective window size is  $21 \times 21$ .

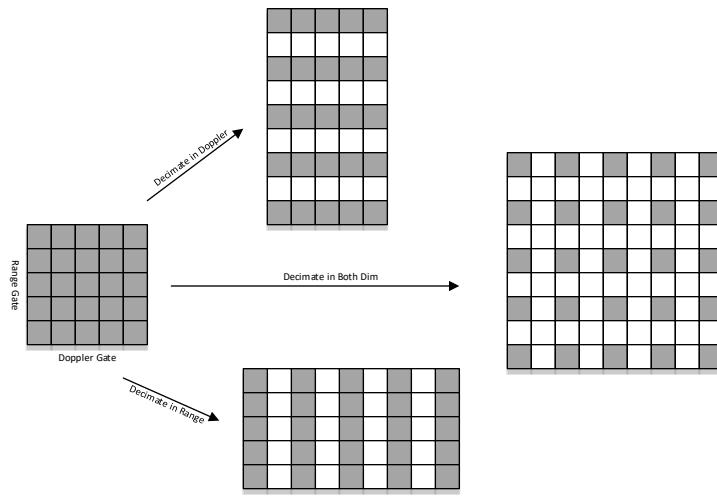


Fig. 7.4: Decimation of Rectangular Window Mask

---

**Note:** Both the bit mask and the decimation scheme can be region-dependent.

---

### 7.2.3.2 Cross Window

A typical cross window is shown in Fig. 7.5. The maximum length of the four wings is 9. Lengths of the two wings along the range gate are the same. Lengths of the two wings along the Doppler gate are also the same. The number of guard cells along each wing can be  $0 \sim 15$ , but guard cell numbers along the two wings along the range gate are the same and so are the guard cell numbers along the two wings along the Doppler gate.

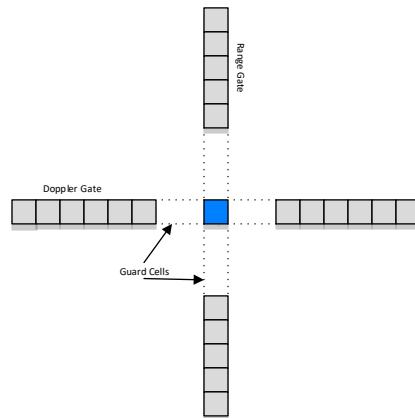


Fig. 7.5: Example Cross Window



## 7.2. Functional Description

---

---

**Note:** Cross window parameters are **not** region dependent.

---

### 7.2.3.3 Boundary Behavior

If a CUT lies near the boundary of RDM, the sliding window will be out of boundary. Alps has the following behaviors.

- Out of range gate boundary: Padding zeros for reference cells.
- Out of Doppler-gate boundary: Using cells wrapping around the Doppler gate.

### 7.2.4 Object Searching

For object searching, Alps provides the following schemes:

- Peak Detector
- CA-CFAR
- OS-CFAR
- SOGO-CFAR
- NR-CFAR

Within a region dependent framework, only one XXX-CFAR object searching scheme can be chosen for each region.

For each region, one can also choose to enable the peak detector or not. Once the peak detector is enabled, the final object output must satisfy both of the following conditions:

1. It is a peak;
2. It passes the corresponding CFAR threshold.

Different peak detectors can be configured for different regions.

#### 7.2.4.1 Peak Detector

Peak detector is also implemented in a sliding window fashion. But the size of its bit mask is  $5 \times 5$  and independent of the sliding window used for CFAR searching. In other words, one can choose a different sliding window for peak detector from that for CFAR.

#### 7.2.4.2 CA-CFAR

Logically, Alps CA-CFAR is implemented as shown in Fig. 7.6:

1. Collect all powers of RCs based on sliding window and CUT position;
2. Remove all 0 powers;
3. Remove  $n$  maximum powers ( $n = 0, 1, 2$ );
4. Compute average power of RCs;
5. Compare the scaled power of CUT with the average power to make the final decision.

---

**Note:**

---

- It is possible that powers of certain RCs are zeros, which could happen when CUT is near the boundary of RDM. The design of Step 2 above is to prevent these points from biasing the decision.
- Design of Step 3 is to prevent large reflectors around CUT from biasing the decision.
- The parameters marked in red in Fig. 7.6 are configurable and can be region dependent.

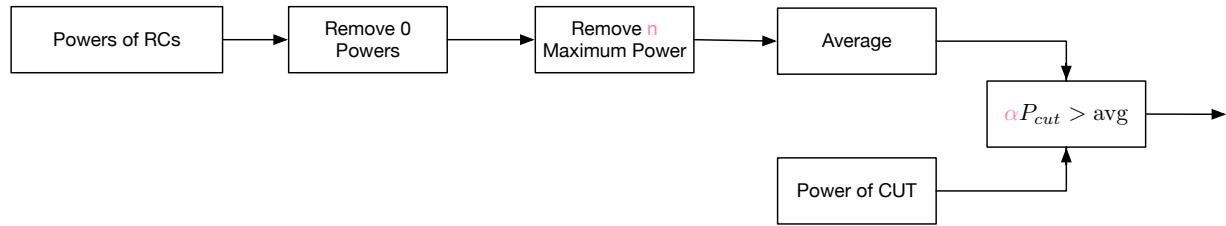


Fig. 7.6: Diagram of CA-CFAR

#### 7.2.4.3 OS-CFAR

Logically, Alps OS-CFAR is implemented as shown in Fig. 7.7:

1. Collect all powers of RCs based on sliding window and CUT position;
2. Remove all 0 powers;
3. Find the rank among RCs ( $r$ );
4. Compare  $r$  with  $T_{rank}$  to make the final decision, where  $T_{rank}$  comes from one of the following sources:
  - A pre-programmed fixed value  $T_{dec}$ ;
  - A related rank, which is product of a programmable factor  $\eta$  and size of RCs.

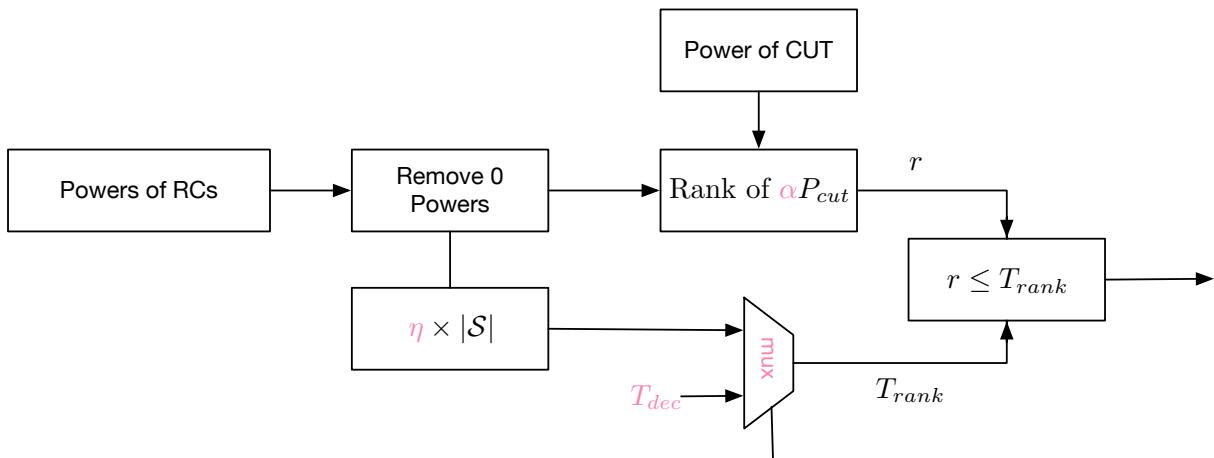


Fig. 7.7: Diagram of OS-CFAR

**Note:**

## 7.2. Functional Description

---

- It is possible that powers of certain RCs are zeros, which could happen when CUT is near the boundary of RDM. The design of Step 2 above is to prevent these points from biasing the decision.
  - Design of related rank can adapt to situations where CUT is near RDM boundary.
  - The parameters marked in red in Fig. 7.7 are configurable and can be region dependent.
- 

### 7.2.4.4 SOGO-CFAR

SOGO-CFAR only works with cross windows. Logically, Alps SOGO-CFAR is implemented as shown in Fig. 7.8:

1. For each side within a cross window, collect all powers of RCs;
2. For each side, compute the average power;
3. Select average powers based on a programmable mask;
4. Select  $i$  th smallest average power;
5. Compare the scaled CUT power with the output average power to make the final decision.

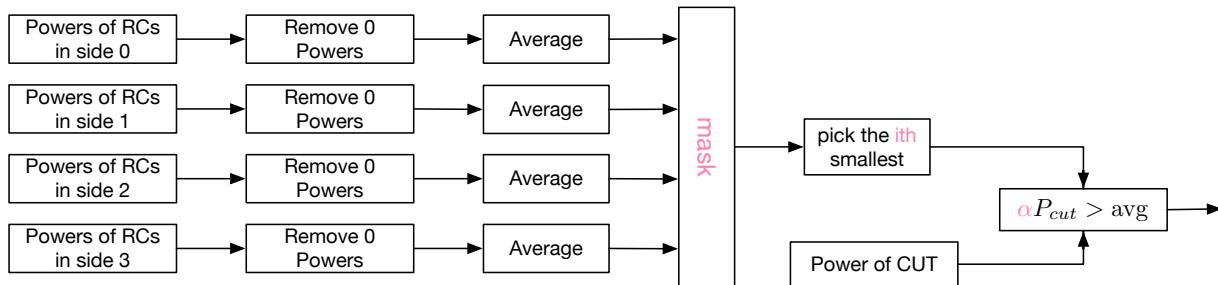


Fig. 7.8: Diagram of SOGO-CFAR

---

#### Note:

- It is possible that powers of certain RCs are zeros, which could happen when CUT is near the boundary of RDM. The design of Step 2 above is to prevent these points from biasing the decision.
  - The parameters marked in red in Fig. 7.8 are configurable and can be region dependent.
- 

### 7.2.4.5 NR-CFAR

NR-CFAR is a Calterah proprietary CFAR algorithm, where NR stands for **noise reference**. It utilizes global noise estimation result to do the CFAR (See Section 14).

Logically, NR-CFAR is implemented as shown in Fig. 7.9. It looks much more complicated than other CFAR algorithms, but the idea behind it is very simple. Generally, it depends on three inputs:

- $\hat{\sigma}^2$ : Noise variance estimate used in the algorithm;
- $P_{cut}$ : Power of CUT;
- $\mathcal{S}$ : Set of reference cells' powers.

NR-CFAR can be divided into three sub-schemes:

- Scheme 0: It simply compares noise estimate and a scaled CUT's power. It does not depend on  $\mathcal{S}$ .
- Scheme 1: It is quite similar to CA-CFAR except the construction of set  $\mathcal{S}$ .
- Scheme 2: It is quite similar to OS-CFAR except the construction of set  $\mathcal{S}$ .

One can select one of the three schemes.

Some comments on  $\tilde{\sigma}^2$  and  $\mathcal{S}$ :

- $\tilde{\sigma}^2$  is obtained from the noise estimator by comparing the value of the prior range gate and current range gate. The design of this is due to overshooting of the noise estimator (see [Section 14](#))
- $\mathcal{S}$  is constructed based on the noise estimate. It basically excludes cells with powers quite different from  $\tilde{\sigma}^2$ .

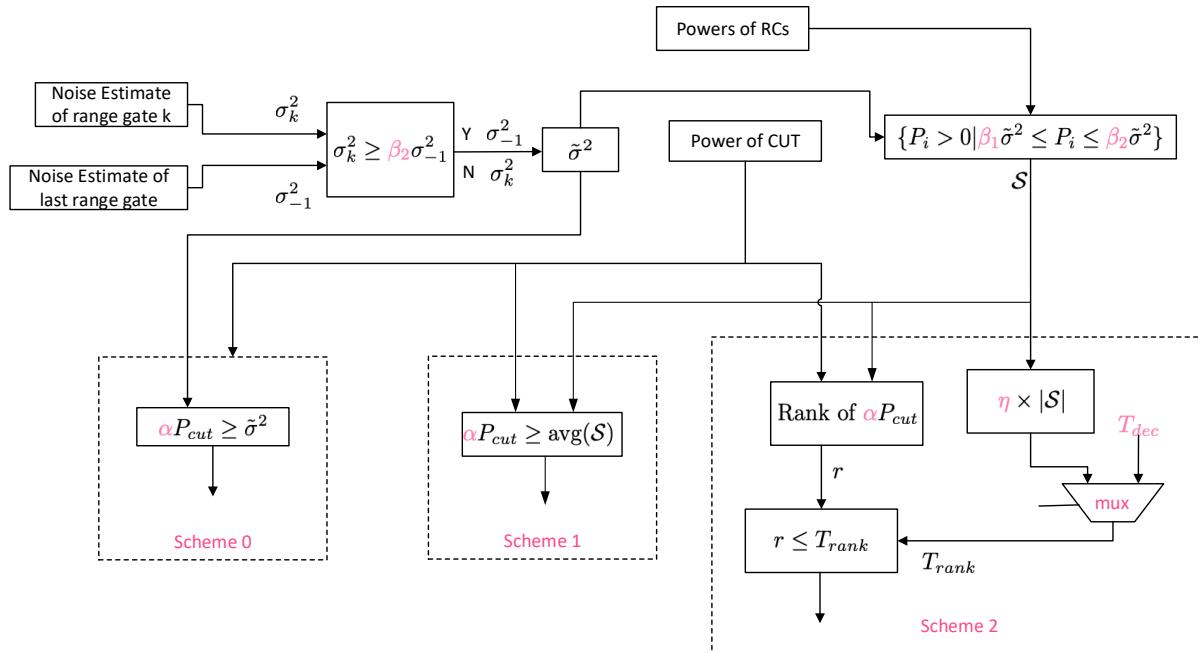


Fig. 7.9: Diagram of NR-CFAR

#### Note:

- It is possible that powers of certain RCs are zeros, which could happen when CUT is near the boundary of RDM. The design of Step 2 above is to prevent these points from biasing the decision.
- The parameters marked in red in [Fig. 7.9](#) are configurable and can be region dependent.

## 7.3 Programming Interfaces

### 7.3.1 Combination Engines

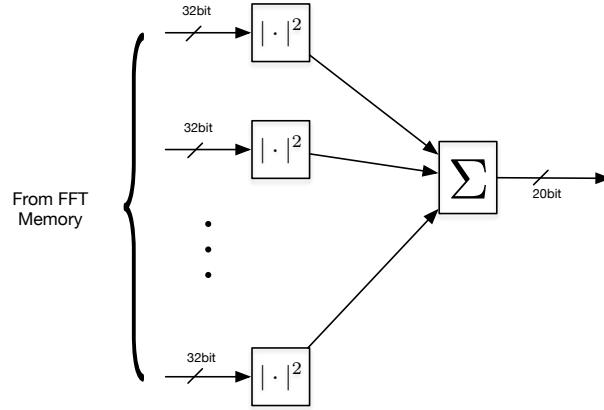


Fig. 7.10: SISO Combination Engine Diagram

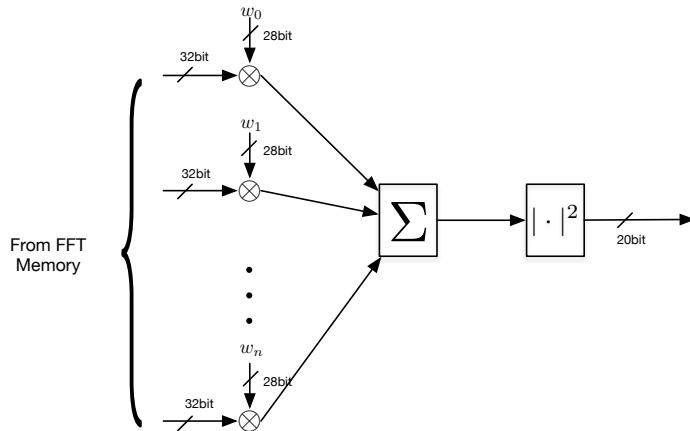


Fig. 7.11: MIMO Combination Engine Diagram

[Fig. 7.10](#) and [Fig. 7.11](#) show detailed implementation of SISO combination engine and MIMO combination engine, respectively. Input data is from FFT memory, which is 32-bit data in the format of CFL(14, 1,  $S$ , 4,  $U$ ). For both engines, output data is 20-bit in the format of FLR(15, 1,  $U$ , 5,  $U$ ). Since the output of two engines share the same format, they can share the same hardware in the post-process.

Weights (or steering vectors) used in MIMO combination engine are programmable. Alps supports up to 16 groups of up to 16-size weights. Each weight is 28-bit in the format of CFX(14, 1,  $S$ ).

One can choose either SISO combination or MIMO combination by setting register `CFG_CFR_TYPE_CMB` (0xC0\_0808) to 0 or 1, respectively. Little configuration for SISO combination is needed. So the remaining of this subsection will focus on MIMO combination.

To program the steering vectors, one needs to program `CFG_SYS_MEM_ACT` (0xC0\_0014) to 5, which switches the memory space starting from 0xE0\_0000 to digital beamforming steering table. Configurable offset addresses are supported, as this memory space is shared by:

- steering vectors used by CFAR;
- DBPM coefficients used in BPM, which is discussed in [Section 9](#);
- steering vectors used by digital beamforming engine, which is discussed in [Section 8.3.2](#).

One can program `CFG_CFR_MIMO_ADR` (0xC0\_0810) to an address of a multiple of 16 to specify where to start the steering vectors. Each vector element is aligned at an address of a multiple of 4 (See [Section 9](#) for detailed memory layout).

To specify the number of combination directions  $d$ , just let `CFG_CFR_MIMO_NUM` (0xC0\_080C) be  $d - 1$ . `CFG_CFR_MIMO_NUM` is 4-bit with the format of FXI(4, 0).

### 7.3.2 RDM Region Partition

To program the RDM region partition, one only needs to let hardware know the value of  $r_i$  and  $d_{lj}$  shown in Fig. 7.3, where  $i = 0, 1, 2$ ,  $l = 0, 1, 2, 3$ , and  $j = 0, 1$ . The corresponding control registers are `CFG_CFR_PRT_RNG_0i` and `CFG_CFR_PRT_VEL_lj`, respectively.

### 7.3.3 Sliding Window

To choose either rectangular sliding window or cross sliding window, set `CFG_CFR_CS_ENA` (0xC0\_0960) to 0 or 1, respectively.

#### 7.3.3.1 Rectangular Window

In hardware, the bit mask size  $11 \times 11$  is stored in the register `CFG_CFR_MSK_DS_ij`, where  $i = 0, \dots, 7$  indicating the region numbering and where  $j = 0, 1, 2, 3$  because each register only has 32 bits.

The bit mask corresponding to the range and Doppler gates are shown in [Fig. 7.12](#).



### 7.3. Programming Interfaces

---

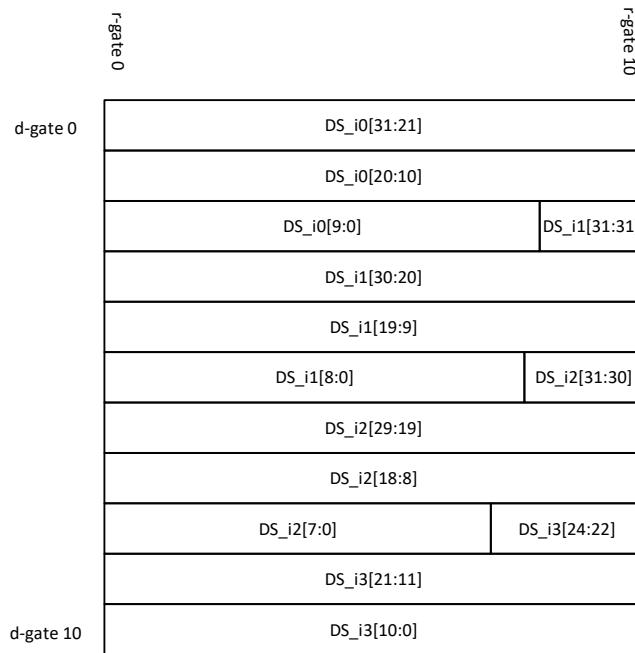


Fig. 7.12: Bit Mask Packing in Register CFG\_CFR\_MSK\_DS\_i [0-3]

The decimation scheme is controlled by the register CFG\_CFR\_TYP\_DS (0xC0\_0844), with Bit 0 and Bit 1 for Region 0, Bit 2 and Bit 3 for Region 1 and so on. The bit format is:

- 00b: No decimation on both dimensions
- 10b: Decimate along the Doppler gate
- 01b: Decimate along the range gate
- 11b: Decimate along both dimensions

#### 7.3.3.2 Cross Window

Cross windows are not region dependent. A cross window is defined by four parameters listed below.

Table 7.2: Cross Window Parameters and Its Corresponding Registers

Parameter Name	Register Name	Register Addr	Numeric Range
length along range gate	CFG_CFR_CS_SIZ_RNG	0xC0_0970	1~9
length along Doppler gate	CFG_CFR_CS_SIZ_VEL	0xC0_096C	1~9
size of guard cells along range gate	CFG_CFR_CS_SKP_RNG	0xC0_0968	0~15
size of guard cells along Doppler gate	CFG_CFR_CS_SKP_VEL	0xC0_0964	0~15

### 7.3.4 Object Searching

#### 7.3.4.1 Peak Detector

The sliding window for peak detector is also maintained in a bit mask fashion. For rectangular window, the supported size is up to  $5 \times 5$ , while for cross window, size of up to  $3 \times 3$  is supported. The masks are stored in registers `CFG_CFR_MSK_PK_0i`, where  $i = 0, \dots, 7$ , for 8 regions. The bit map is shown in Fig. 7.13.

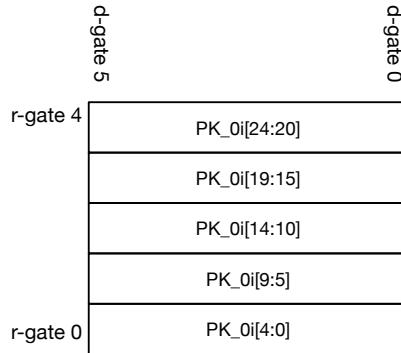


Fig. 7.13: Bit Map of `CFG_CFR_MSK_PK_0i`

Peak detector for each region can be disabled and enabled independently and it is controlled by the register `CFG_CFR_PK_ENB` with Bit 7 for Region 7 and Bit 0 for Region 0.

#### 7.3.4.2 CFAR Parameter Space Management

For different regions, different CFAR schemes can be chosen independently. CFAR scheme choice is controlled by the register `CFG_CFR_TYP_AL` (0xC0\_0840). Algorithm definition is coded in 2-bit format:

- 00b: CA-CFAR
- 10b: SOGO-CFAR
- 01b: OS-CFAR
- 11b: NR-CFAR

The first 2 bits (LSB) are for Region 0, and the second 2 bits are for Region 1, and so on.

To support up to 8 regions and flexible configuration for different CFAR algorithms, the parameter space is organized as follows:

- Logically, Alps has 8 parameters to accommodate the largest number of CFAR parameters;
- Each parameter has 8 groups to support 8 regions;
- When a specific CFAR scheme is selected for a particular region, hardware interprets underlying hardware value accordingly.

For example, suppose that one parameter has 28 bits stored in hardware. If the corresponding region selects to use CA-CFAR, this parameter will be interpreted as CA-CFAR's parameter. Similarly, if OS-CFAR is selected, it will be interpreted as OS-CFAR's parameter.

### 7.3. Programming Interfaces

---

The 8 parameters are stored in 8 groups of registers  $\text{CFG\_CFR\_PRM}_{-i\text{-}x}$ , where  $i = 0 \dots 7$ . Each group has a different number of registers due to different bit widths. The bit maps of Parameter [0-6] are shown in Fig. 7.14 to Fig. 7.20. Parameter 7 simply has 1 bit and  $\text{CFG\_CFR\_PRM}_{-7\text{-}0}$  has 8 bits with Bit  $i$  for Region  $i$ .

PRM_0_0[27:21]	PRM_0_0[20:14]	PRM_0_0[13:7]	PRM_0_0[6:0]
region 7	region 6	region 5	region 4
PRM_0_1[27:21]	PRM_0_1[20:14]	PRM_0_1[13:7]	PRM_0_1[6:0]
region 3	region 2	region 1	region 0

Fig. 7.14: Parameter 0: Bit Map of  $\text{CFG\_CFR\_PRM}_{-0\text{-}0}$  and  $\text{CFG\_CFR\_PRM}_{-0\text{-}1}$

PRM_1_0[29:20]	PRM_1_0[19:10]	PRM_1_0[9:0]
region 7	region 6	region 5
PRM_1_1[29:20]	PRM_1_1[19:10]	PRM_1_1[9:0]
region 4	region 3	region 2
PRM_1_2[19:10]	PRM_1_2[9:0]	
region 1	region 0	

Fig. 7.15: Parameter 1: Bit Map of  $\text{CFG\_CFR\_PRM}_{-1\text{-}0}$ ,  $\text{CFG\_CFR\_PRM}_{-1\text{-}1}$ , and  $\text{CFG\_CFR\_PRM}_{-1\text{-}2}$

PRM_2_0[29:20]	PRM_2_0[19:10]	PRM_2_0[9:0]
region 7	region 6	region 5
PRM_2_1[29:20]	PRM_2_1[19:10]	PRM_2_1[9:0]
region 4	region 3	region 2
PRM_2_2[19:10]	PRM_2_2[9:0]	
region 1	region 0	

Fig. 7.16: Parameter 2: Bit Map of  $\text{CFG\_CFR\_PRM}_{-2\text{-}0}$ ,  $\text{CFG\_CFR\_PRM}_{-2\text{-}1}$ , and  $\text{CFG\_CFR\_PRM}_{-2\text{-}2}$

PRM_3_0[23:12]	PRM_3_0[11:0]	PRM_3_1[23:12]	PRM_3_1[11:0]
region 7	region 6	region 5	region 4
PRM_3_2[23:12]	PRM_3_2[11:0]	PRM_3_3[23:12]	PRM_3_3[11:0]
region 3	region 2	region 1	region 0

Fig. 7.17: Parameter 3: Bit Map of CFG\_CFR\_PRM\_3\_0, CFG\_CFR\_PRM\_3\_1, CFG\_CFR\_PRM\_3\_2, and CFG\_CFR\_PRM\_3\_3

PRM_4_0[27:21]	PRM_4_0[20:14]	PRM_4_0[13:7]	PRM_4_0[6:0]
region 7	region 6	region 5	region 4
PRM_4_1[27:21]	PRM_4_1[20:14]	PRM_4_1[13:7]	PRM_4_1[6:0]
region 3	region 2	region 1	region 0

Fig. 7.18: Parameter 4: Bit Map of CFG\_CFR\_PRM\_4\_0 and CFG\_CFR\_PRM\_4\_1

PRM_5_0[27:21]	PRM_5_0[20:14]	PRM_5_0[13:7]	PRM_5_0[6:0]
region 7	region 6	region 5	region 4
PRM_5_1[27:21]	PRM_5_1[20:14]	PRM_5_1[13:7]	PRM_5_1[6:0]
region 3	region 2	region 1	region 0

Fig. 7.19: Parameter 5: Bit Map of CFG\_CFR\_PRM\_5\_0 and CFG\_CFR\_PRM\_5\_1

PRM_6_0[23:12]	PRM_6_0[11:0]	PRM_6_1[23:12]	PRM_6_1[11:0]
region 7	region 6	region 5	region 4
PRM_6_2[23:12]	PRM_6_2[11:0]	PRM_6_3[23:12]	PRM_6_3[11:0]
region 3	region 2	region 1	region 0

Fig. 7.20: Parameter 6: Bit Map of CFG\_CFR\_PRM\_6\_0, CFG\_CFR\_PRM\_6\_1, CFG\_CFR\_PRM\_6\_2, and CFG\_CFR\_PRM\_6\_3

## 7.3. Programming Interfaces

---

### 7.3.4.3 CA-CFAR

Tuning knobs for CA-CFAR are marked in red in Fig. 7.6.

- $n$ : Number of maximum powers to remove before averaging;
- $\alpha$ : Scalar of CUT's power before comparison.

The following table summarizes the mapping from these knobs to registers.

Table 7.3: Parameters for CA-CFAR

Parameter Name	Register Group	Data Format	Supported Numeric Range
$n$	CFG_CFR_PRM_0_x	FXI(2, $U$ )	0 ~ 2
$\alpha$	CFG_CFR_PRM_1_x	FLR(6, 6, $U$ , 4, $U$ )	0 ~ 63

### 7.3.4.4 OS-CFAR

Tuning knobs for OS-CFAR are marked in red in Fig. 7.7.

- $T_{dec}$ : The preselected rank threshold;
- $\alpha$ : The scalar of CUT's power before ranking;
- Rank Selector: The mux control to select the final source of rank threshold;
- $\eta$ : The scalar of related rank.

The following table summarizes the mapping from these knobs to registers.

Table 7.4: Parameters for OS-CFAR

Parameter Name	Register Group	Data Format	Supported Numeric Range
$T_{dec}$	CFG_CFR_PRM_0_x	FXI(7, $U$ )	0 ~ 121
$\alpha$	CFG_CFR_PRM_1_x	FLR(6, 6, $U$ , 4, $U$ )	0 ~ 63
Rank Selector	CFG_CFR_PRM_2_x	FXI(1, $U$ )	0 or 1
$\eta$	CFG_CFR_PRM_3_x	FXR(12, 0, $U$ )	0 ~ 0.9997

### 7.3.4.5 SOGO-CFAR

Tuning knobs for SOGO-CFAR are marked in red in Fig. 7.8.

- $\alpha$ : The scalar of CUT's power before comparison;
- mask: The mask control to select the sides of the cross window;
- $i$ : The  $i$  th smallest over the averages from the selected sides.

The following table summarizes the mapping from these knobs to registers.

Table 7.5: Parameters for SOGO-CFAR

Parameter Name	Register Group	Data Format	Supported Numeric Range
$i$	CFG_CFR_PRM_0_x	FXI(2, $U$ )	0 ~ 3
$\alpha$	CFG_CFR_PRM_1_x	FLR(6, 6, $U$ , 4, $U$ )	0 ~ 63
mask	CFG_CFR_PRM_2_x	FXI(4, $U$ )	0x1 ~ 0xF



### 7.3.4.6 NR-CFAR

Tuning knobs for NR-CFAR are marked in red in Fig. 7.9.

- Scheme Selector: Select output from one of three sub-schemes;
- $\alpha$ : Scalar of power of CUT before comparison or ranking;
- $\beta_1$ : Lower bound scalar in construction of set of reference powers;
- $\beta_2$ : Upper bound scalar in construction of set of reference powers;
- $T_{dec}$ : Pre-programmed rank threshold for Scheme 2;
- $\eta$ : The scalar of related rank for Scheme 2;
- Rank Selector: The mux control to select the final source of rank threshold in Scheme 2.

The following table summarizes the mapping from these knobs to registers.

Table 7.6: Parameters for NR-CFAR

Parameter Name	Register Group	Data Format	Supported Numeric Range
Scheme Selector	CFG_CFR_PRM_0_x	FXI(2, U)	0 ~ 2
$\alpha$	CFG_CFR_PRM_1_x	FLR(6, 6, U, 4, U)	0 ~ 63
$\beta_1$	CFG_CFR_PRM_2_x	FLR(6, 6, U, 4, U)	0 ~ 63
$\beta_2$	CFG_CFR_PRM_3_x	FLR(6, 6, U, 4, U)	0 ~ 63
$T_{dec}$	CFG_CFR_PRM_5_x	FXI(7, 0)	0 ~ 121
$\eta$	CFG_CFR_PRM_6_x	FXR(12, 0, U)	0 ~ 0.9997
Rank Selector	CFG_CFR_PRM_7_x	FXI(1, U)	0 or 1

## 7.4 Limitation and Constraints

- SISO Combiner and MIMO Combiner cannot be chosen dynamically for different regions.
- The size of a rectangular sliding window can be  $11 \times 11$  without decimation. With decimation, it can be  $21 \times 21$  but the total number of reference cells must be no greater than 121.
- For each side of a cross sliding window, the number of reference cells must be no greater than 9 and the size of guard cells must be no greater than 15.
- For peak detector, the rectangular window can be up to  $5 \times 5$  and the cross window can be up to  $3 \times 3$ .
- During one frame, one cannot choose a rectangular window for one region and a cross window for another. In other words, sliding window type needs to be uniform for all regions.
- Configuration related to cross Windows is **not** region dependent.
- SOGO-CFAR is supported only for cross Windows.



# 7.5 Software and Configuration Suggestions

General guidelines on choosing and tuning CFAR related parameters are discussed here.

## 7.5.1 SISO vs MIMO Combiners

Compared to MIMO Combiner, SISO Combiner is more straightforward and requires little configuration. MIMO combination can provide additional performance gain but requires additional tuning.

Performance gain of MIMO combination comes from knowing the directions of reflectors beforehand. However, it is generally not true. Alps provides 16 programmable steering vectors, which essentially let user choose 16 different directions to cover the field of view of the radar system.

Another pitfall of using MIMO combination is the case of using virtual array. When virtual array is in use, moving objects can cause additional phase offsets, which are not fully known at CFAR stage<sup>1</sup>. In this case, it is recommended that one should **only** use non-virtualized RX channels to do the MIMO combination by setting weights of other channels to zero.

## 7.5.2 Cross Window vs Rectangular Window

Large sliding windows improve the CFAR performance in the situation where the reflector extends in the range gate or Doppler gate. For example, a large shuttle or ship can cause a spread spectrum along the range gate. For indoor applications, moving people could cause a spread spectrum along the Doppler gate. Therefore, a rectangular window is suitable with the situation where spectrum spread happens at either the range gate or Doppler gate but not both; while a cross window works better in the situation where both dimensions have serious spread.

## 7.5.3 Peak Detector

Although one can program any value with the  $5 \times 5$  bit mask, it is recommended to use a regular one such as all-one in  $5 \times 5$  or  $3 \times 3$ . Another two particular options are to use  $3 \times 1$  and  $1 \times 3$ , which are equivalently to disable peak detector in one dimension but not in the other one.

## 7.5.4 CFAR Tuning

Most CFAR algorithms compare local noise level with the power of CUT. Therefore, one can consider the parameter of  $\alpha$  in the CFAR algorithms provided in Alps as an inverse of signal-to-noise ratio (SNR). All other quantities on the other side of inequalities are some form of “noise estimate”.

Supported CFAR algorithms generally either use averaging or ranking. Thus, roughly speaking, these algorithms generally fall into either CA-CFAR or OS-CFAR category (with certain variations). When tuning the CFAR knobs, readers are recommended to refer to classical papers for more insight, such as [Rohling1983].

---

<sup>1</sup> If there is no velocity ambiguity, such offsets can be fully known. However, when using virtual array, the system often suffers from velocity ambiguity. For velocity ambiguity, please see [Section 13](#).



### 7.5.5 Region Dependent CFAR Schemes

Region dependent searching gives the flexibility of deploying different CFAR algorithms for different regions in RDM. A flexible tuning way usually comes with larger tuning effort. Therefore, it is recommended:

- Turn on the region dependent feature only when it helps directly<sup>2</sup>.
- Keep the number of regions as small as possible.

In Alps, there are two high-pass filters in analog RX chain. Consequently, the noise floor has a high-pass shape in the frequency domain and suppresses the power of the reflector near the radar system. Therefore, if a universal CFAR algorithm cannot display a good performance, one can tune the CFAR criterion to overcome this effect. On the other hand, one can also choose to lower the CFAR threshold for reflectors that are relatively far away from the radar receiver.

## 7.6 Examples

Suppose that the per-channel FFT size is  $256 \times 128$ , and CA-CFAR is chosen with a cross sliding window without the region dependent feature. Then the corresponding register values are:

Table 7.7: Example Settings for CA-CFAR

Register Name	Address	Value	Comments
CFG_CFR_PRT_RNG_01	0xC00834	255	Index starting from 0
CFG_CFR_PRT_RNG_02	0xC00838	255	Index starting from 0
CFG_CFR_PRT_RNG_03	0xC0083C	255	Index starting from 0
CFG_CFR_PRT_VEL_00	0xC00814	127	Index starting from 0
CFG_CFR_PRT_VEL_01	0xC00818	127	Index starting from 0
CFG_CFR_PRT_VEL_10	0xC0081C	127	Index starting from 0
CFG_CFR_PRT_VEL_11	0xC00820	127	Index starting from 0
CFG_CFR_PRT_VEL_20	0xC00824	127	Index starting from 0
CFG_CFR_PRT_VEL_21	0xC00828	127	Index starting from 0
CFG_CFR_PRT_VEL_30	0xC0082C	127	Index starting from 0
CFG_CFR_PRT_VEL_31	0xC00830	127	Index starting from 0
CFG_CFR_CS_ENA	0xC00960	1	Enable cross window
CFG_CFR_CS_SIZ_RNG	0xC00970	9	The largest number of range gates in sliding window
CFG_CFR_CS_SIZ_VEL	0xC0096C	9	The largest number of Doppler gates in sliding window
CFG_CFR_CS_SKP_RNG	0xC00968	15	The largest number of guard cells in the range-gate domain
CFG_CFR_CS_SKP_VEL	0xC00964	15	The largest number of guard cells in the Doppler-gate domain
CFG_CFR_TYP_AL	0xC00840	0x0	Select CA-CFAR for all regions (only Region 0 in use)
CFG_CFR_PRM_0_1	0xC0084C	2	Remove 2 maximum powers before computing the average
CFG_CFR_PRM_1_2	0xC00858	0x339	0.1 in format of $FLR(6, 6, U, 4, U)$

In another setup, one may consider using SOGO-CFAR (with SO) instead. However, due to the boundary effect and high pass effect of RF, using SO may cause some issues when detecting objects near DC. Thus, one can turn to region dependent setup. For example, take a two-region approach with a range-gate boundary at range gate 30. Then the corresponding register values are:

Table 7.8: Example Settings for SOGO-CFAR

Register Name	Address	Value	Comments
CFG_CFR_PRT_RNG_00	0xC00834	30	Index starting from 0
CFG_CFR_PRT_RNG_01	0xC00838	255	Index starting from 0
CFG_CFR_PRT_RNG_02	0xC0083C	255	Index starting from 0
CFG_CFR_PRT_VEL_00	0xC00814	127	Index starting from 0
CFG_CFR_PRT_VEL_01	0xC00818	127	Index starting from 0
CFG_CFR_PRT_VEL_10	0xC0081C	127	Index starting from 0
CFG_CFR_PRT_VEL_11	0xC00820	127	Index starting from 0

continues on next page

<sup>2</sup> One can disable the region dependent feature by setting  $r_i$  and  $d_{l_j}$  to the largest values. In this way, only Region 0 is in use.



## 7.6. Examples

---

Table 7.8 – continued from previous page

Register Name	Address	Value	Comments
CFG_CFR_PRT_VEL_20	0xC00824	127	Index starting from 0
CFG_CFR_PRT_VEL_21	0xC00828	127	Index starting from 0
CFG_CFR_PRT_VEL_30	0xC0082C	127	Index starting from 0
CFG_CFR_PRT_VEL_31	0xC00830	127	Index starting from 0
CFG_CFR_CS_ENA	0xC00960	1	Enable cross window
CFG_CFR_CS_SIZ_RNG	0xC00970	9	The largest number of range gates in sliding window
CFG_CFR_CS_SIZ_VEL	0xC0096C	9	The largest number of Doppler gates in sliding window
CFG_CFR_CS_SKP_RNG	0xC00968	15	The largest number of guard cells in the range-gate domain
CFG_CFR_CS_SKP_VEL	0xC00964	15	The largest number of guard cells in the Doppler-gate domain
CFG_CFR_TYP_AL	0xC00840	0x22	Select SOGO-CFAR for Region 0 and Region 2
CFG_CFR_PRM_0_1	0xC0084C	0x1	Region 0 takes second least average and region 2 takes least average
CFG_CFR_PRM_1_2	0xC00858	0x339	Region 0 0.1 in format of FLR(6, 6, U, 4, U)
CFG_CFR_PRM_1_1	0xC00854	0x339	Region 2 0.1 in format of FLR(6, 6, U, 4, U)
CFG_CFR_PRM_2_2	0xC00864	0xF	Region 0 takes all four sides
CFG_CFR_PRM_2_1	0xC00860	0xF	Region 2 takes all four sides



## DIRECTION OF ARRIVAL (DOA) ESTIMATORS

### 8.1 Overview

Calterah Alps integrates two engines for DoA estimation: One is classical digital beamforming (DBF) and the other one is deterministic maximum likelihood (DML). They are both implemented with programmable steering vector tables. In other words, when evaluating power along a certain direction, Alps uses the *inner product* to compute the value directly<sup>1</sup>. By making all steering vectors programmable, the engines not only accommodate all kinds of antenna array design and but also work well with most antenna calibration methods.

Both engines provide the capability of separating multiple objects inside the same range gate and Doppler gate. Besides some fine-tuning knobs, both engines support 2D DoA estimation, which is covered in [Section 9](#).

#### 8.1.1 Features

- Digital Beamforming Engine
  - Supports up to 360 steering vectors;
  - Maximum steering vector size is  $32^2$ ;
  - Support up-to-4-object separation within the same range gate and Doppler gate.
- Deterministic Maximum Likelihood Engine
  - Supports up to 760 steering vectors;
  - Maximum steering vector size is  $32^2$ ;
  - Support up-to-2-object separation within the same range gate and Doppler gate.

<sup>1</sup> An alternative approach is to utilize FFT. It is faster but it assumes that RX antennas are located at equal distances or on a regular grid. This assumption might not be true due to radiation impact among antennas and manufacture inaccuracy.

<sup>2</sup> Alps alone has maximum 4 TX and 4 RX channels. Therefore, it can generate 16 channels in total. The support for vector size of 32 is to accommodate possible future extension.

## 8.2 Functional Description

### 8.2.1 Digital Beamforming

#### 8.2.1.1 Overview

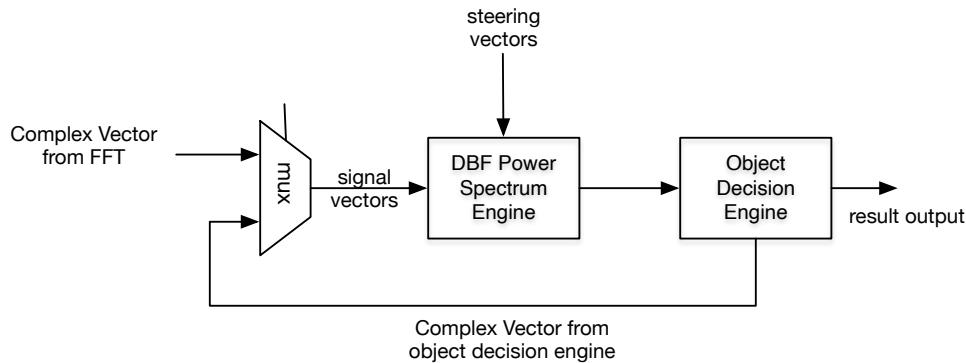


Fig. 8.1: Diagram of DBF Engine for Estimation of DoA

Fig. 8.1 shows the structure of the DBF engine. If DBF is enabled as the DoA estimator, for each CFAR output, the DBF engine is called and estimates the DoA of reflectors . In particular, each CFAR output provides the range-gate and Doppler-gate indices. Based on these two indices, hardware queries the corresponding FFT result from each channel and forms a vector to feed into the DBF engine. Then DBF runs automatically without any further CPU intervention.

The DBF engine consists of two parts: *DBF spectrum power engine* and *object decision engines*. The former is to compute the power spectrum over different directions based on the input signal vector and steering vectors. The latter involves DoA estimator related decisions based on the input power spectrum, including:

- How does the object searching is done? (Iterative or non-iterative)
- Which direction has potential objects?
- Should the searching be stopped?

The remaining of Section 8.2.1 discusses functions of these two parts in detail.

### 8.2.1.2 DBF Power Spectrum Engine

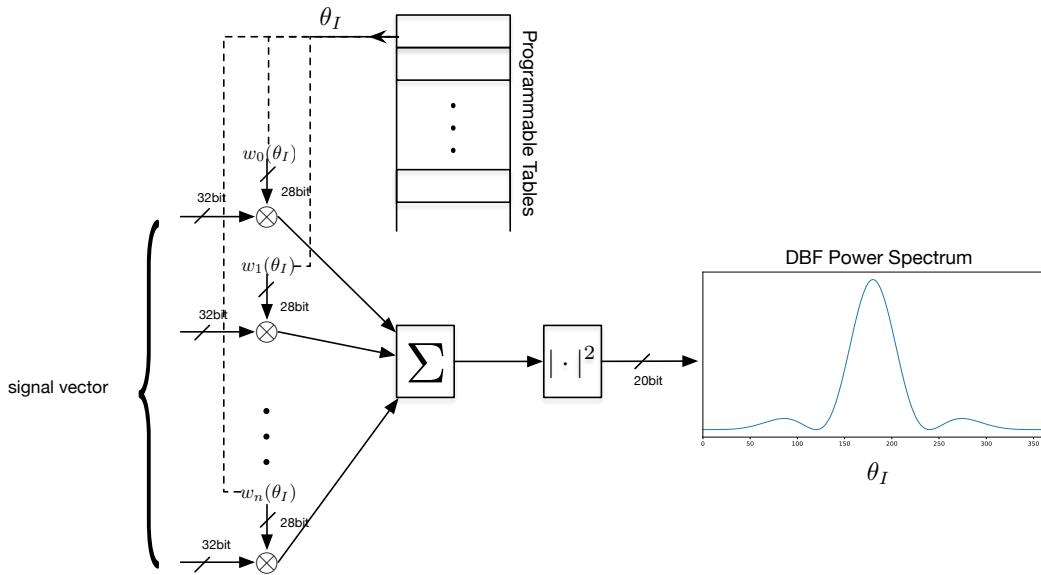


Fig. 8.2: Diagram of DBF Power Spectrum Engine for Estimation of DoA

Elements of the input signal vector are all 32-bit in the format of CFL(14, 1,  $S$ , 4,  $U$ ). For each index  $\theta_I$ , the engine queries a steering vector (with the same size as the input signal vector) from a programmable table and computes the inner product between the signal vector and the steering vector. Taking the magnitude square of the inner product, it outputs a power (20 bits with the format of FLR(15, 1,  $U$ , 5,  $U$ )) along a particular angle indexed by  $\theta_I$ . Collecting all power values corresponding to all  $\theta_I$ , one can get a power spectrum that visually looks like the graph shown on the right side of Fig. 8.2.

Steering vectors are stored in one programmable look-up table (LUT). Each element of the steering vectors is 28-bit and encoded in the format of CFX(14, 1,  $S$ ). The maximum supported steering vector size is 32 and the maximum supported number of  $\theta_I$  is 360.

### 8.2.1.3 Object Decision Engine

The object decision engine supports separation of multiple objects (up to 4) within the same range and Doppler gates. It has the following modes:

- Orthogonal match pursuit mode (OMP)
- Iterative peak mode (IPM)
- Non-iterative peak mode (Non-IPM)
- Full peak mode (FPM)

---

**Note:** These modes are mutually exclusive. The DBF engine can only work with one mode.

---

## 8.2. Functional Description

---

### 8.2.1.3.1 Orthogonal Match Pursuit Mode (OMP)

As indicated in the name, this mode borrows the idea from standard orthogonal match pursuit [Cai2011]. The decision process can be described as follows:

1. Let  $\mathbf{r}$  denote the signal vector, let  $\mathcal{V}$ ,  $\Theta$  and  $\mathcal{P}$  denote three empty sets, and let  $O_{\max}$  denote the maximum possible number of objects;
2. If the size of  $\Theta$  ( $|\Theta|$ ) is greater or equal to  $O_{\max}$ , stop searching and go to the last step; otherwise, go to the next step;
3. Find the largest peak in DBF power spectrum generated by  $\mathbf{r}$ , and put its  $\theta_I$ , steering vector  $\mathbf{w}(\theta_I)$ , and the power corresponding to the index of  $\theta_I$  into the sets  $\Theta$ ,  $\mathcal{V}$ , and  $\mathcal{P}$ , respectively;
4. Find the orthogonal projection of  $\mathbf{r}$  onto the subspace generated by vectors in  $\mathcal{V}$ , and use  $\mathbf{r}_{\mathcal{V}}$  to denote it;
5. Set  $\mathbf{r}$  to be  $\mathbf{r} - \mathbf{r}_{\mathcal{V}}$ ;
6. If the power  $\|\mathbf{r}\|^2$  is less than the noise level threshold  $T_{noi}$ , then stop and go to the last step; otherwise, repeat from Step 2;
7. Output  $\Theta$  and  $\mathcal{P}$ .

The algorithm flow is quite fixed. However, one can tweak the searching scheme by tuning  $O_{\max}$  and  $T_{noi}$ .  $O_{\max}$  does not depend on the outputs of CFAR while  $T_{noi}$  is an SNR dependent parameter. In particular, the value of SNR is computed in hardware by using the power and noise level  $noi$  reported from CFAR. Based on the regime (low, high, mid) which this SNR value (in dB) falls into, baseband core chooses a coefficient  $\alpha$ . Then  $T_{noi} = \alpha \times noi$ .

### 8.2.1.3.2 Iterative Peak Mode (IPM)

IPM is similar to the OMP mode. The decision process can be described as follows:

1. Let  $\mathbf{r}$  denote the signal vector, let  $\mathcal{V}$ ,  $\Theta$  and  $\mathcal{P}$  denote three empty sets, and let  $O_{\max}$  denote the maximum possible number of objects;
2. Find the largest  $O_{\max}$  peaks in DBF spectrum generated by the signal vector  $\mathbf{r}$ ;
3. Count the number of peaks whose power is greater than  $P_{\max} \times \alpha$ , where  $P_{\max}$  is power of the largest peak and  $\alpha$  is a programmable coefficient less than 1; Let  $N_{pk}$  denote the number;
4. Set  $\tilde{O}_{\max}$  to be  $\min(O_{\max}, N_{pk})$ ;
5. If the size of  $\Theta$  ( $|\Theta|$ ) is greater or equal to  $\tilde{O}_{\max}$ , stop searching and go to the last step; otherwise, go to the next step;
6. Find the largest peak in DBF power spectrum generated by  $\mathbf{r}$ , and put its  $\theta_I$ , steering vector  $\mathbf{w}(\theta_I)$ , and the power corresponding to the index of  $\theta_I$  into the sets  $\Theta$ ,  $\mathcal{V}$ , and  $\mathcal{P}$ , respectively;
7. Find the orthogonal projection of  $\mathbf{r}$  onto the subspace generated by vectors in  $\mathcal{V}$ , and use  $\mathbf{r}_{\mathcal{V}}$  to denote it;
8. Set  $\mathbf{r}$  to be  $\mathbf{r} - \mathbf{r}_{\mathcal{V}}$ ;
9. If the power  $\|\mathbf{r}\|^2$  is less than the noise level threshold  $T_{noi}$ , then stop and go to the last step; otherwise, repeat from Step 5;
10. Output  $\Theta$  and  $\mathcal{P}$ .

---

#### Note:

- This mode is similar to OMP except that the value  $O_{\max}$  can be changed based on the number of relatively strong peaks in the original DBF power spectrum;

- 
- Besides  $O_{\max}$  and  $T_{noi}$ ,  $\alpha$  is also tunable. See Section 8.3.3.
- 

### 8.2.1.3.3 Non-Iterative Peak Mode (Non-IPM)

Non-IPM is similar to IPM except that it is not in an iterative fashion.

1. Let  $\mathbf{r}$  denote the signal vector, let  $\mathcal{V}$ ,  $\Theta$  and  $\mathcal{P}$  denote three empty sets, and let  $O_{\max}$  denote the maximum possible number of objects;
2. Find the largest  $O_{\max}$  peaks in DBF spectrum generated by the signal vector  $\mathbf{r}$ ;
3. Count the number of peaks whose power is greater than  $P_{\max} \times \alpha$ , where  $P_{\max}$  is power of the largest peak and  $\alpha$  is a programmable coefficient less than 1; Let  $N_{pk}$  denote the number;
4. Set  $O_{\max}$  to be  $\min(O_{\max}, N_{pk})$ ; and set  $i = 1$ ;
5. If the size of  $\Theta$  ( $|\Theta|$ ) is greater or equal to  $O_{\max}$ , stop searching and go to the last step; otherwise, go to the next step;
6. Find the  $i$ -th largest peak in DBF power spectrum, and put its  $\theta_I$ , steering vector  $\mathbf{w}(\theta_I)$ , and the power corresponding to the index of  $\theta_I$  into the sets  $\Theta$ ,  $\mathcal{V}$ , and  $\mathcal{P}$ , respectively;
7. Set  $\mathbf{r}$  to be  $\mathbf{r} - \mathbf{r}_{\mathcal{V}}$  and set  $i = i + 1$ ;
8. If the power  $\|\mathbf{r}\|^2$  is less than the noise level threshold  $T_{noi}$ , then stop and go to the last step; otherwise, repeat from Step 5;
9. Output  $\Theta$  and  $\mathcal{P}$ .

---

#### Note:

- This mode is similar to IPM except that Non-IPM does not require repeating computing DBF power spectrum based on the updated  $\mathbf{r}$ ;
- 

### 8.2.1.3.4 Full Peak Mode (FPM)

This mode is the simplest one. It finds all peaks (up to 12) in DBF power spectrum and outputs the corresponding positions and powers of the peaks.

### 8.2.1.3.5 Raw-Fine Searching

The first two modes need to re-compute DBF power spectrum during each iteration, which is quite time-consuming. On the other hand, in each iteration, only the largest peak matters. Therefore, we can take this advantage to do raw-fine searching.

The idea is quite straight-forward. To find the largest peak, we do it in two phases. During the first phase, computation and searching are done every  $S_{raw}$  index. During the second phase, searching only around the position of the peak ( $\pm R_{fine}$ ) found during the first phase. This can largely reduce the searching time.

It is possible that one ends up at a wrong global peak using raw-fine searching. But because both  $S_{raw}$  and  $R_{fine}$  are programmable and Alps only supports a relatively small number of channels (up to 32), it is not difficult to find a good pair of values for all kinds of applications.



### 8.2.2 Deterministic Maximum Likelihood (DML)

#### 8.2.2.1 DML Algorithm

This section gives a brief introduction to the basic principle of the DML algorithm. Let  $X$  denote the receiving signal vector from RX antenna array, and let  $A$  denote the steering matrix comprised of  $D$  steering vectors, where  $D$  is the number of reflectors. Then  $A$  can be expressed as  $A(\theta) = [sv(\theta_0), sv(\theta_1), \dots, sv(\theta_{D-1})]$ , where  $sv(\theta_i)$  is the steering vector for  $\theta_i$  and  $\theta = [\theta_0, \theta_1, \dots, \theta_{D-1}]$  is the vector with each element representing the angle of each reflector.

The DML algorithm estimates  $D$  reflectors' angles  $\theta$  by:

$$\hat{\theta} = \arg \max_{\theta} \text{tr} \left( P_{A(\theta)} \hat{R} \right) \quad (8.1)$$

Where  $\text{tr}()$  is the trace function of the matrix,  $P_{A(\theta)} = A(\theta) (A^H(\theta) A(\theta))^{-1} A^H(\theta)$ , and  $\hat{R} = \frac{1}{M} \sum_{i=1}^M X(i) X^H(i)$  where  $M$  is the number of snapshots.

According to the result of simulation, DML performs well in some respects. It can separate incoming signals with small angle differences even in single snapshot case, so in our DML engine, we use only one snapshot to do the angle estimation, i.e.  $M = 1$ .

DML's good performance comes along with prices. First of all, it has to know beforehand the number of reflectors  $D$ . Second the time complexity is comparatively big. As  $D$  increases, the algorithm's time complexity becomes unacceptable. If  $N$  denotes the number of candidate angles, the time complexity of the algorithm is  $O(N^D)$ .

The DML engine in Calterah Alps chip chooses to support maximum two candidate angles ( $D = 2$ ). There are basically two reasons behind the choice. First, the simulation shows that even in the case of more than two candidates, DML is able to output the two correct candidate angles with maximum power. The second reason is that by increasing the resolutions of range and velocity, most reflectors can be separated in the range or velocity domain, leaving the situation of more than 2 reflectors within the same range and Doppler gate is not likely to happen.

#### 8.2.2.2 Two-Step Searching

Since  $D = 2$ , the DML performs a 2D search over the angle space formed by Cartesian product of two identical 1D vector  $\theta_{1D} = [\theta_{c0}, \theta_{c1}, \dots, \theta_{c(N-1)}]^T$ , where  $\theta_{ci}$  is the  $i$  th candidate angle. The time complexity  $O(N^2)$  for 2D search is still too big for the maximum number of candidate angles ( $N = 760$ ). To solve the problem, the DML engine in the Alps device breaks the 2D angle space searching procedure into two steps:

##### 1. Coarse Search.

Perform a 2D search over the angle space formed by Cartesian product of two identical 1D vectors  $\theta_{1D} = [\theta_{c(start)}, \theta_{c(start+step)}, \theta_{c(start+2*step)} \dots, \theta_{c(start+N_1*step)}]^T$ , where  $start$  and  $step$  are integers defining the starting point and the pace of Coarse Search, and  $N_1$  is the biggest integer which satisfies  $(start + N_1 \cdot step) \leq N - 1$ .

##### 2. Refined Search.

If the result of Coarse Search is  $(\theta_{1i}, \theta_{1j})$ , perform a 2D search over the angle space formed by Cartesian product of  $[\theta_{1i-\frac{step}{2}}, \theta_{1i-\frac{step}{2}+1}, \dots, \theta_{1i+\frac{step}{2}-1}]$  and  $[\theta_{1j-\frac{step}{2}}, \theta_{1j-\frac{step}{2}+1}, \dots, \theta_{1j+\frac{step}{2}-1}]$ . The result of this step  $(\theta_{2i}, \theta_{2j})$  is output as the result of 2D searching.

Two-step searching lowers the time complexity of the algorithm from  $O(N^2)$  to  $O\left(\left(\frac{N}{step}\right)^2 + step^2\right)$ . When  $step$  is so selected as  $step = \sqrt{N}$ , the complexity becomes linear to the input size.

As mentioned before, the goal of this algorithm is to find the global extreme point in angle space formed by Cartesian product of two identical 1D vector  $\theta_{1D} = [\theta_{c0}, \theta_{c1}, \dots, \theta_{c(N-1)}]^T$ . Let's assume it is  $(\theta_i, \theta_j)$ . If it is too far away



from the result of Step 1 ( $\theta_{1i}, \theta_{1j}$ ) and not covered by the searching space of Step 2, the result of Step 1 falls into a local extreme point. In this case, the overall searching fails. To prevent this failure, a random variation within the range  $[0, \frac{\text{step}}{2}]$  may be added into the *start* parameter during the initialization of the algorithm at the beginning of every frame. Then any failing frames would be ignored by the tracking algorithm in firmware. Simulations also suggest that a smaller *step* would also give a lower chance of such failure. A choice to balance between time consumption of DML engine and the chance of falling into local extreme point should be made.

### 8.2.2.3 Distribution of Candidate Angles

Because for DML engine, the maximum number of reflectors supported within a CFAR tone is 2, Equation (8.1) can be rewritten to:

$$\hat{\theta} = \arg \max_{\theta} \text{tr} \left( P_{A(\theta)} \hat{R} \right) = \arg \max_{\theta} \left( X^H A(\theta) (A(\theta)^H A(\theta))^{-1} A(\theta)^H X \right) \quad (8.2)$$

Where  $A(\theta) = [sv(\theta_{s0}), sv(\theta_{s1})]$  is the steering matrix containing two steering vectors associated with two angles.

Define the metric function of DML as:

$$C(\theta_{s0}, \theta_{s1}) \triangleq X^H A(\theta) (A(\theta)^H A(\theta))^{-1} A(\theta)^H X \quad (8.3)$$

Then the metric function can be further derived as:

$$C(\theta_{s0}, \theta_{s1}) = [a, b] \begin{bmatrix} c & d \\ d^H & c \end{bmatrix}^{-1} \begin{bmatrix} a^H \\ b^H \end{bmatrix} = \frac{c(a^H \cdot a + b^H \cdot b) - d \cdot a \cdot b^H - d^H \cdot a^H \cdot b}{c^2 - d^H \cdot d} \quad (8.4)$$

Where:

Table 8.1: Metric Function Description

Variable in Equation (8.4)	Description
$a$	$X^H \cdot sv(\theta_{s0})$
$b$	$X^H \cdot sv(\theta_{s1})$
$c$	$sv(\theta_{s0})^H \cdot sv(\theta_{s0})$ , element number in $X$
$d$	$sv(\theta_{s0})^H \cdot sv(\theta_{s1})$

As can be seen in Equation (8.4), the calculation of metric function involves the calculations of intermediate variables  $a$ ,  $b$ , and  $d$ . The procedure of calculating  $d$  can be further resolved as:

$$d = sv(\theta_{s0})^H \cdot sv(\theta_{s1}) = \sum_{i=0}^{N-1} e^{j \cdot 2\pi \cdot d_i \cdot (\sin(\theta_{s1}) - \sin(\theta_{s0}))} \quad (8.5)$$

Where  $d_i$  is  $i$  th receiving antenna's position normalized by wave length. For more information about steering vectors programming, please refer to [Section 8.5.2](#).

The Equation (8.5) shows that  $d$  is a function of two candidate angles  $\theta_{s0}$  and  $\theta_{s1}$ . If  $N$  candidate angles are so distributed as an arithmetic progression, any pairs of candidate angles would result in different values of  $d$ , and the number of all possible  $d$  would be  $O(N^2)$ , which requires that the calculation of  $d$  be performed on-the-fly. However, if  $N$  candidate angles are so distributed that the value of their sine functions form an arithmetic progression, the number of all possible  $d$  will shrink to  $O(N - 1)$ . It is now possible to store all possible  $d$  beforehand and save run time of the algorithm drastically.

Base on this reason, the candidate angles used by the DML engine are so discrete that their sine functions form an arithmetic progression, and all  $N - 1$  possible  $d$  values are generated during system initialization and stored in memory for further use by DML engine.

Similarly, let the factor  $k$  denote  $\frac{1}{c^2 - d^H \cdot d}$  in Equation (8.4). Then  $k$  can also be calculated during system initialization and stored in memory, based on what is done for  $d$ .



### 8.3. Programming Interfaces

---

The overall run time is shortened by those two tricks of calculating  $d$  and  $k$  adopted by Alps baseband.

$$k = \frac{1}{c^2 - d^H \cdot d} \quad (8.6)$$

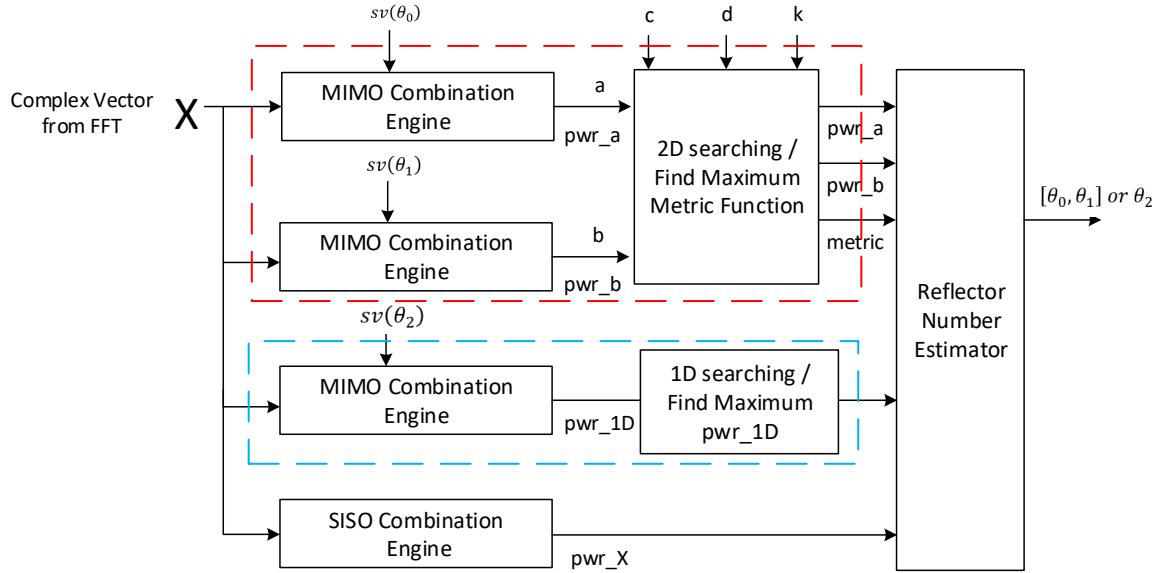


Fig. 8.3: DML Engine Diagram

#### 8.2.2.4 Reflector Number Estimator (RNE)

One of the drawbacks of the DML algorithm is that it has to know the number of reflectors beforehand. Alps baseband integrates two DML engines. One is called 2O-DML engine and is for 2-object detection, which assumes the number of reflectors is 2. The other one is called 1O-DML engine which assumes that 1 reflector is out there. Alps baseband integrates a reflector number estimator, which makes the final decision on which DML engine's output should be treated as the final output of the DML algorithm.

Both DML engines can be running simultaneously, and together they output 3 candidate angles in total, associated with 3 projecting powers. Reflector number estimator uses these 3 powers together with the power of the input signal  $X$  and the metric function of 2O-DML engine to estimate the number of reflectors.

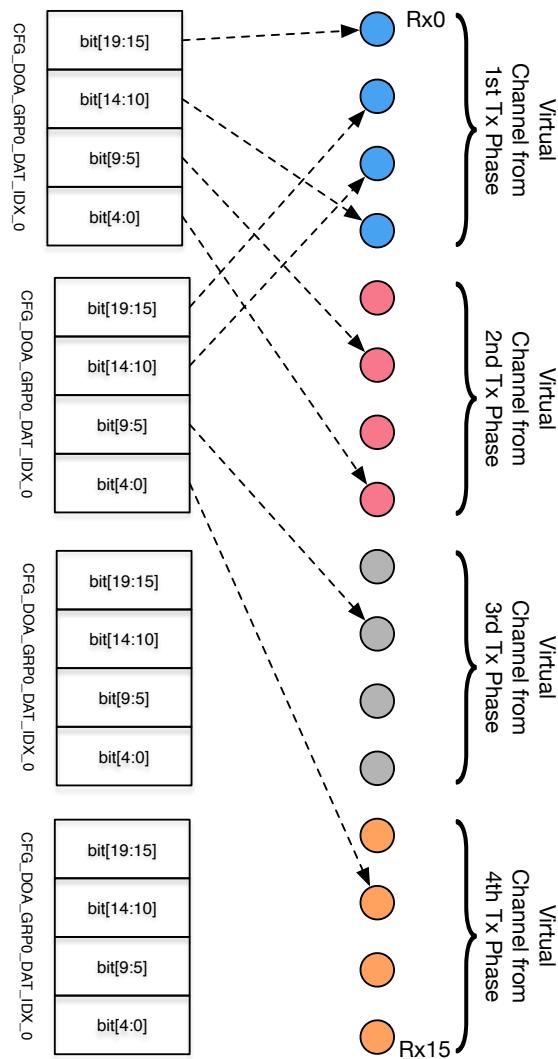
The system is depicted in Fig. 8.3.

## 8.3 Programming Interfaces

For DoA estimation, either the DBF engine or the DML engine can be selected. Both engines should be aware of which channels are used to form the signal vectors and from what location in memory to get the steering vectors.

### 8.3.1 Control of Signal Vectors

Alps has up to 4 TX and 4 RX. Therefore, by using virtual antenna (discussed in [Section 9](#)), one can virtualize up to 16 RX channels. [Fig. 8.4](#) briefly illustrates the idea. In some applications, one may only use part of these channels, or only part of these channels exist. Therefore, both the DBF power spectrum engine and DML engine need to know which channels to use. In hardware, this information is maintained with a group of mapping registers `CFG_DoA_GRP0_DAT_IDX_t` ( $t = 0 \dots 7$ ), each of that has four 5-bit values. As shown in [Fig. 8.4](#), `CFG_DoA_GRP0_DAT_IDX_0 [19:15]` stores the channel ID corresponding to the first element of steering vector, `CFG_DoA_GRP0_DAT_IDX_0 [14:10]` stores the channel ID corresponding to the second element of steering vector, and so on.



[Fig. 8.4:](#) Signal Vector Mapping for Rx channels with Virtual Array.

A single Alps uses the first 4 registers, and the remaining 4 registers are for future extension.

### 8.3.2 Programming Steering Vectors

Steering vectors should be programmed into table starting from the address 0xE0\_0000 with the register CFG\_SYS\_MEM\_ACT set to 5. Each element of the vectors is 28-bit with the format of CFX(14, 1, S). The brief layout is shown in Fig. 8.5.

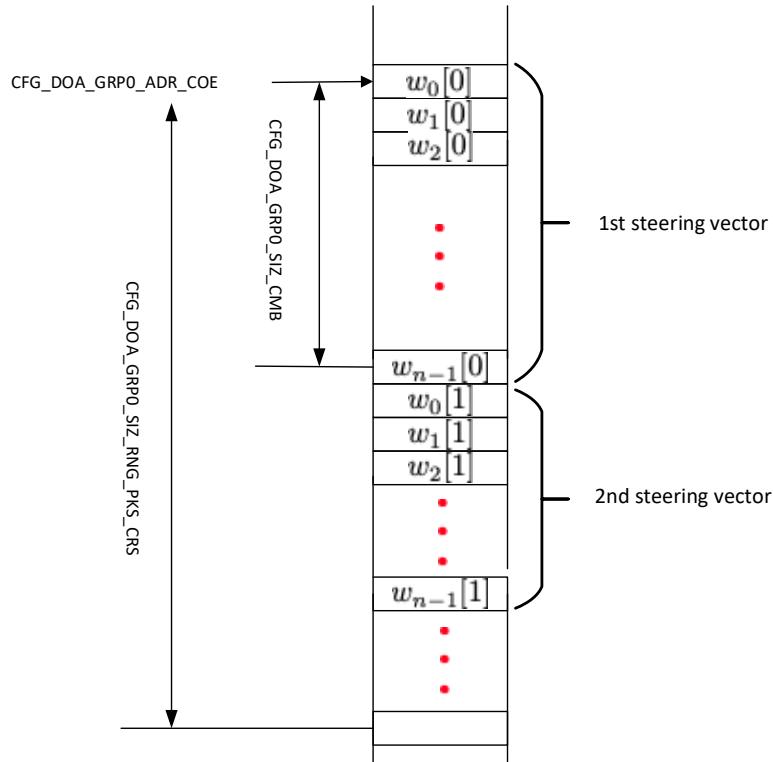


Fig. 8.5: Memory Layout for Steering Vector Programming

Since elements of steering vectors are written continuously, the starting address (CFG\_DoA\_GRP0\_ADR\_COE), vector size (CFG\_DoA\_GRP0\_SIZ\_CMB), and total number of steering vectors  $N_{dbf}$  should be indicated correctly so that DBF or DML engine knows where to start and stop.

It should be noted that CFG\_DoA\_GRP0\_SIZ\_RNG\_PKS\_CRS has different meanings with  $N_{dbf}$ . In order to support raw-fine searching of DBF, CFG\_DoA\_GRP0\_SIZ\_RNG\_PKS\_CRS is designed to denote the number of steering vectors ( $N'_{dbf}$ ) to compute DBF power spectrum iteratively, where  $N'_{dbf} = \frac{N_{dbf}}{S_{raw}}$ .

- CFG\_DoA\_GRP0\_ADR\_COE should be an address of a multiple of 4;
- CFG\_DoA\_GRP0\_SIZ\_CMB should be vector size minus 1;
- CFG\_DoA\_GRP0\_SIZ\_RNG\_PKS\_CRS should be the number of steering vectors minus 1, that's  $N'_{dbf} - 1$ .
- CFG\_DoA\_GRP0\_SIZ\_STP\_PKS\_CRS should be programmed with the value of  $S_{raw} - 1$ , and for DML or FPM mode of DBF, it should be programmed to be 0.

### 8.3.3 Digital Beamforming

#### 8.3.3.1 DBF Power Spectrum Engine

To make DBF power spectrum engine work properly, one only needs to do the following:

1. Let the engine know which channels are used to form the signal vectors ([Section 8.3.1](#));
2. Program steering vectors;
3. Let the engine know where to get the steering vectors ([Section 8.3.2](#)).

#### 8.3.3.2 Object Decision Engine

As mentioned in [Section 8.2](#), there are several different modes. The selection of modes is controlled by the register `CFG_DoA_GRP0_MOD_SCH`:

Table 8.2: DBF Mode Control

Value	Mode
0	FPM
1	NIPM
3	IPM
4	OMP

##### 8.3.3.2.1 Orthogonal Match Pursuit Mode (OMP)

As mentioned in [Section 8.2.1.3.1](#), Alps provides controls on the existing criteria of iterative searching: one is the maximum number of detected objects  $O_{\max}$  and the other is noise level threshold  $T_{noi}$ .

$O_{\max}$  is determined by two values: the pre-set value denoted by  $O_{1,\max}$  and the signal vector size (or the steering vector size) denoted by  $O_{2,\max}$ . In particular, we have

$$O_{\max} = \min(O_{1,\max}, O_{2,\max} - 1)$$

In turn,  $O_{1,\max}$  and  $O_{2,\max}$  are related to the registers `CFG_DoA_GRP0_NUM_SCH` and `CFG_DoA_GRP0_SIZ_CMB` as follows:

Register	Value
<code>CFG_DoA_GRP0_NUM_SCH</code>	$O_{1,\max} - 1$
<code>CFG_DoA_GRP0_SIZ_CMB</code>	$O_{2,\max} - 1$

---

#### Note:

- `CFG_DoA_GRP0_NUM_SCH` is in the format of  $FXI(2, U)$  and can take a value from 0, 1, 2, and 3. One can program it directly to control the value of  $O_{1,\max}$ .
  - `CFG_DoA_GRP0_SIZ_CMB` is in the format of  $FXI(5, U)$ . This register cannot be programmed arbitrarily because it depends on the array design as mentioned in [Section 8.3.3.1](#).
- 

As mentioned in [Section 8.2.1.3.1](#),  $T_{noi}$  is determined by SNR and the noise level of CFAR output. SNR is computed in hardware to determine the SNR regime:



### 8.3. Programming Interfaces

---

Condition	SNR regime
$\text{SNR} < \text{CFG\_DoA\_GRP0\_THR\_SNR\_0}$	low
$\text{CFG\_DoA\_GRP0\_THR\_SNR\_0} \leq \text{SNR} < \text{CFG\_DoA\_GRP0\_THR\_SNR\_1}$	mid
$\text{SNR} \geq \text{CFG\_DoA\_GRP0\_THR\_SNR\_1}$	high

For each regime, we have a scalar programmed in `CFG_DoA_GRP0_SCL NOI_k`, where  $k = 0, 1, \text{ or } 2$ .  $T_{noi}$  is determined by the product between the noise level reported from CFAR and this scalar.

---

**Note:**

- `CFG_DoA_GRP0_SCL NOI_k` is in the format of  $\text{FLR}(6, 6, U, 4, U)$ . Usually a scalar larger than 1 should be used. The larger the value is, the earlier the iteration stops.
  - `CFG_DoA_GRP0_THR_SNR_0` and `CFG_DoA_GRP0_THR_SNR_1` are both in the format of  $\text{FXI}(7, U)$  in dB scale. Alps provides these for fine tuning of `CFG_DoA_GRP0_SCL NOI_k`.
- 

#### 8.3.3.2.2 (Non-)Iterative Peak Mode

IPM is quite similar to OMP mode except that the maximum number of objects used as the stopping criterion is  $\tilde{O}_{\max}$  instead of  $O_{\max}$ , where  $\tilde{O}_{\max}$  is the minimum between  $O_{\max}$  and the number of peaks whose powers are strong enough. Here “strong enough” is defined as that the power of the peak is above the power of the strongest power times a scalar. The scalar is stored in the programmable register `CFG_DoA_GRP0_SCL POW` with the format of  $\text{FLR}(6, 6, U, 4, U)$ . Usually this value should be less than 1.

The tuning knobs mentioned in OMP mode are all available to IPM mode with the same meaning.

For Non-IPM, the tuning knobs are the same as in IPM mode.

#### 8.3.3.2.3 Full Peak Mode (FPM)

There is no tuning knob for FPM.

#### 8.3.3.2.4 Raw-Fine Searching

There are only two parameters to program to enable the raw-fine searching feature: raw search step  $S_{raw}$  and fine search range  $R_{fine}$ .

- `CFG_DoA_GRP0_SIZ_STP_PKS_CRS` is in the format of  $\text{FXI}(4, U)$ . It should be programmed with the value of  $S_{raw} - 1$ ;
- `CFG_DoA_GRP0_SIZ RNG_PKS_RFD` is in the format of  $\text{FXI}(6, U)$ . It should be programmed with the value of  $R_{fine}$ .

There is a corner case: Suppose during raw searching phase, the maximum peak is found at angle index  $\theta_I$ . Then  $\theta_I - R_{fine}$  could be less than 0, or  $\theta_I + R_{fine}$  could be greater than the total number of steering vectors. In such a case, the overflow is handled internally by hardware as follows:

- If  $\theta_I - R_{fine} < 0$ , then fine searching will be done from 0 to  $2R_{fine}$ ;
- If  $\theta_I + R_{fine} \geq N_{dbf}$ , where  $N_{dbf}$  is the total number of steering vectors, then fine searching will be done from  $N_{dbf} - 2R_{fine} - 1$  to  $N_{dbf} - 1$ .

**Note:**

- To use DBF, the value of CFG\_DoA\_GRP0\_SIZ RNG\_PKS\_RFD should be less than 512;
- $N_{dbf}$  should be a multiple of  $S_{raw}$ .

### 8.3.4 Deterministic Maximum Likelihood

As discussed in Section 9.2.4, DML supports Normal Mode and Combined Mode.

For Normal Mode, only one group named Group A is supported. The following discussion uses Group A as the example to show how to program the DML engine.

Similar to the DBF engine, the DML engine also needs to know how to form the input signal vector and how/where to get steering vectors. Refer to Section 8.3.1 and Section 8.3.2.

#### 8.3.4.1 Two-Step Searching

As mentioned in Section 8.3.2, the number of steering vectors  $N$  is configured by setting the register CFG\_DoA\_GRP0\_SIZ RNG\_PKS\_CRS to  $N - 1$ . Let's define  $N$  candidate angles as  $\theta_0, \theta_1, \dots, \theta_{N-1}$ , and the corresponding steering vectors are  $sv(\theta_0), sv(\theta_1), \dots, sv(\theta_{N-1})$ . To avoid the problem of falling into local extreme point for a series of consecutive frames, it is not recommended that 2O-DML searching always start from Index 0. The starting angle index denoted by *start* should be configured in the register CFG\_DML\_GRP0\_SV\_START (0xC0\_3588). Symmetrically, 2O-DML engine also provides a register to configure the ending angle index denoted by *end*. *end* should be configured in the register CFG\_DML\_GRP0\_SV\_END (0xC0\_3592) and must be less than  $N$ .

The angle *step* used in Coarse Search as defined in Section 8.2.2.2 is configured in the register CFG\_DML\_GRP0\_SV\_STP (0xC0\_3584). The candidate angles being searched in Coarse Search are listed as  $\theta_{start}, \theta_{start+step}, \theta_{start+2\cdot step}, \dots, \theta_{start+N_1\cdot step}$ , where  $N_1$  is the biggest integer which satisfies  $start + N_1 \cdot step \leq end$ .

The result of Coarse Search is denoted by  $[\theta_i, \theta_j]$ , where  $start \leq \theta_i < \theta_j \leq end$ . The candidates being searched for the 1st angle in the refined phase are listed as  $\theta_{st}, \theta_{st+1}, \dots, \theta_{i+\frac{step}{2}-1}$ , where  $st = \max(i - \frac{step}{2}, start)$ . For the 2nd angle, the candidate list is  $\theta_{j-\frac{step}{2}}, \theta_{j-\frac{step}{2}+1}, \dots, \theta_{lt}$ , where  $lt = \min(j + \frac{step}{2} - 1, end)$ . The Refined Search has a boundary and the angles are restricted by the range from *start* to *end*.

#### 8.3.4.2 Distribution of Candidate Angles

While Section 8.3.2 gives an introduction to where/how to store steering vectors, this current section focus on how to generate steering vectors. As mentioned in Section 8.2.2.3, the candidate angles to generate steering vectors should be discrete so that their sine functions form an arithmetic progression. If  $\theta_L$  denotes the leftmost angle and  $\theta_R$  denotes the rightmost angle within the angle range to be detected, the candidate angle should be  $\theta_i = \text{asin} \left( \sin(\theta_L) + \frac{i}{N-1} \cdot (\sin(\theta_R) - \sin(\theta_L)) \right)$ , where  $i = 0, 1, \dots, N - 1$ .

The length of steering vectors  $c$ , should be configured in the register CFG\_DoA\_GRP0\_SIZ\_CMB.

The intermediate values  $d$  and  $k$  as mentioned in Section 8.2.2.3 should also be set in the memory defined in Section 3.2.1.2 with CFG\_SYS\_MEM\_ACT (0xC0\_0014) set to 9. The memory address starts from 0xE0\_0000 with a length of  $4096 \times 64$  bits and is capable of storing 4 banks of  $d$  and  $k$ , each bank occupying a memory of  $1024 \times 64$  bits. For more information about banks, please refer to Section 12.

The format of  $d$  is CFL(18,1,S,5,S) and the format of  $k$  FLR(18,1,U,5,S). So, a pair of  $d$  and  $k$  occupy a piece of memory of 64 bits. The total number of  $d$  is  $N - 1$  and so is the total number of  $k$ . The layout of the memory storing  $d$  and  $k$  is illustrated by Fig. 8.6.



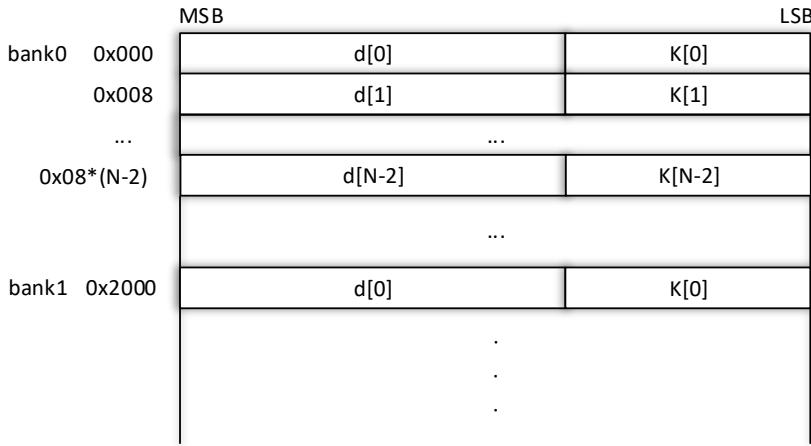


Fig. 8.6: Memory Layout of  $d$  and  $k$

Register `CFG_DML_MEM_BASE_ADDR` (0xC0\_3664) should be configured with the value  $N - 1$ , where  $N$  is the total angle numbers configured for Group A. For more information about groups, please refer to Section 9.2.4.

#### 8.3.4.3 Reflector Number Estimator (RNE)

The number of reflectors output from the DML engine can be fixed to 1 or 2, or decided by the reflector number estimator within the DML engine. Specifically, it is controlled by the register `CFG_DoA_GRP0_NUM_SCH` (0xC0\_2620), which is in the format of  $FXI(2, U)$  and can take a value from 0, 1, 2, and 3.

- When `CFG_DoA_GRP0_NUM_SCH` is set to 0, the output number is fixed to 1.

In this case, the baseband behavior is further controlled by the register `CFG_DML_GRP0_EXTRA_EN` (0xC0\_3616), which is a 1-bit enabler to switch on 1O-DML.

- When 1O-DML is activated by setting `CFG_DML_GRP0_EXTRA_EN` to 1, output from 1O-DML is selected as the final output.
- While 1O-DML is switched off by setting `CFG_DML_GRP0_EXTRA_EN` to 0, one of the two output angles from 2O-DML with the bigger projecting power is selected as the final output.

- When `CFG_DoA_GRP0_NUM_SCH` is set to 1, the output number is fixed to 2.

In this case, `CFG_DML_GRP0_EXTRA_EN` should be set to 0. And output from 2O-DML is taken directly as the final output.

- When `CFG_DoA_GRP0_NUM_SCH` is set to 2 or 3, the number of reflectors is decided by the reflector number estimator within the DML engine, which could be either 1 or 2.

As mentioned in Section 8.2.2.4, the DML engine uses 4 powers and the metric function to estimate the number of reflectors within the same CFAR bin. 5 coefficients  $c_i$ ,  $i = 0, 1, 2, 3, 4$  are configured in the register list `CFG_DML_GRP0_DC_COE_i` where  $i = 0, 1, 2, 3, 4$  to form an inequality (8.7). And only one reflector's DoA will be output from DML engine when Inequality (8.7) holds. Otherwise, 2 reflectors' DoA results will be output for a CFAR bin.

Since the powers from 1O-DML/2O-DML are generated by the internal `MIMO_COMBINE_ENGINE` or `SIMO_COMBINE_ENGINE` module (Section 7.3.1), the 4 powers and metric function are scaled by differ-

ent factors. To avoid ambiguity, the inequality with precise definitions of related variables is shown as:

$$c_0 \cdot P_{00} + c_1 \cdot P_{12} + c_2 \cdot P_1 + c_3 \cdot P_2 \geq c_4 \cdot 2^c \cdot P_X \quad (8.7)$$

Table 8.3: Variables in Inequality (8.7)

Vari- ables	Explanation
$c$	$\text{ceil}(\log_2(\text{sh}))$ , where sh is the element number in signal vector $X$
$P_{00}$	$\ X^H \cdot \text{sv}(\theta_2)\ ^2$ , the projecting power from 1O-DML, where $\theta_2$ is the output angle from 1O-DML
$P_{12}$	$X^H A(\theta) (A(\theta)^H A(\theta))^{-1} A(\theta)^H X$ , metric function from 2O-DML
$P_1$	$P_1 = \max \left\{ \ X^H \cdot \text{sv}(\theta_0)\ ^2, \ X^H \cdot \text{sv}(\theta_1)\ ^2 \right\}$ where $\theta_0$ and $\theta_1$ are two angles output from 2O-DML
$P_2$	$P_2 = \min \left\{ \ X^H \cdot \text{sv}(\theta_0)\ ^2, \ X^H \cdot \text{sv}(\theta_1)\ ^2 \right\}$ where $\theta_0$ and $\theta_1$ are two angles output from 2O-DML
$P_X$	$X^H \cdot X$ , the power of signal vector

The register list `CFG_DML_GRP0_DC_COE_i` is in the format of FXR(14, 1, S). The 5 coefficients  $c_i$  should be so normalized by the same factor that the biggest absolute value among them is 1.

$P_{00}$  and  $P_{12}$  in Equation (8.7) could be ignored if:

- Setting the register `CFG_DML_GRP0_EXTRA_EN` to 0; or
- Setting the register `CFG_DML_GRP0_DC_COE_2_EN` (0xC0\_3620) to 0; or
- Setting both `CFG_DML_GRP0_DC_COE_2` and `CFG_DML_GRP0_DC_COE_3` to 0.

### 8.3.5 Number of CFAR Outputs for DoA

DoA estimates the angles of reflectors for each CFAR output. The number of CFAR outputs is determined by `FDB_CFR_NUMB_OBJ`, `CFG_DoA_NUMB_OBJ`, and `CFG_DoA_MODE_RUN`.

- `FDB_CFR_NUMB_OBJ` is read-only, and indicates the number of CFAR outputs from hardware.
- `CFG_DoA_NUMB_OBJ` is configurable and used for the CPU to specify a CFAR output number for DoA.
- `CFG_DoA_MODE_RUN` determines whose value DoA takes as the number of CFAR outputs, `FDB_CFR_NUMB_OBJ` or `CFG_DoA_NUMB_OBJ`.
  - When the chip is used in cascade mode, `CFG_DoA_MODE_RUN` must be set to 1. And DoA takes the value of `CFG_DoA_NUMB_OBJ` as the number of CFAR outputs.
  - When the chip is used in non-cascade mode, `CFG_DoA_MODE_RUN` can be set to 0 or 2.
    - \* When `CFG_DoA_MODE_RUN` is set to 0, DoA takes `FDB_CFR_NUMB_OBJ` as the number of CFAR outputs. This is the default setting.
    - \* When `CFG_DoA_MODE_RUN` is set to 2, DoA takes `CFG_DoA_NUMB_OBJ` as the number of CFAR outputs.



# 8.4 Limitation and Constraints

Although DoA Estimators are designed to be flexible so that it can accommodate all kinds of applications, there is always some limitation. This part summarizes the limitation and constraints.

## 8.4.1 Digital Beamforming

- CFG\_DoA\_GRP0\_SIZ RNG\_PKS\_CRS should be no greater than 360, which means that the total number of steering vectors is no greater than 360.
- CFG\_DoA\_GRP0\_SIZ\_CMB should be no greater than 32, which means that in DBF power spectrum engine, the size of the signal vector should be no greater than 32.
- CFG\_DoA\_GRP0\_THR\_SNR\_0 should be no greater than CFG\_DoA\_GRP0\_THR\_SNR\_1.
- CFG\_DoA\_GRP0\_SIZ RNG\_PKS\_RFD should be less than 512.
- CFG\_DoA\_GRP0\_SIZ RNG\_PKS\_CRS+1 should be a multiple of CFG\_DoA\_GRP0\_SIZ\_STP\_PKS\_CRS+1, as mentioned in [Section 8.3.3.2.4](#).
- CFG\_DoA\_GRP0\_SIZ\_STP\_PKS\_CRS should be 0 when DBF is selected and CFG\_DoA\_GRP0\_SIZ\_CMB is no less than 28, for example, 28, 29, 30, or 31.

## 8.4.2 Deterministic Maximum Likelihood

- CFG\_DML\_GRP0\_SV\_END should be less than  $N$ , the number of candidate angles.
- CFG\_DML\_GRP0\_SV\_STP should be greater than 2 and less than 64
- $(CFG\_DML\_GRP0\_SV\_END - CFG\_DML\_GRP0\_SV\_START) / CFG\_DML\_GRP0\_SV\_STP$  should be less than 128.
- CFG\_DML\_GRP0\_SV\_START should be less than CFG\_DML\_GRP0\_SV\_END.
- If CFG\_DoA\_GRP0\_NUM\_SCH is set to 1, CFG\_DML\_GRP0\_EXTRA\_EN should be set to 0.
- If CFG\_DoA\_GRP1\_NUM\_SCH is set to 1, CFG\_DML\_GRP1\_EXTRA\_EN should be set to 0.
- $(CFG\_DoA\_GRP0\_SIZ\_RNG\_PKS\_CRS +1) \times ((CFG\_DoA\_GRP0\_SIZ\_CMB +4) \gg 2) \times 4$  should be less than 11,520.
- CFG\_DoA\_GRP0\_SIZ\_STP\_PKS\_CRS should be set to 0.
- CFG\_DoA\_GRP0\_MOD\_SCH should be set to 1.
- If DoA\_GRP0\_SIZ\_CMB is greater than 7 and CFG\_DoA\_GRP0\_NUM\_SCH is not set to 2, baseband core reset (Address 0xb2030c) must be called after running DML. See the Reset chapter in *Calterah Alps Reference Manual* for baseband core reset.
- If DoA\_GRP1\_SIZ\_CMB is greater than 7 and CFG\_DoA\_GRP1\_NUM\_SCH is not set to 2, baseband core reset (Address 0xb2030c) must be called after running DML. See the Reset chapter in *Calterah Alps Reference Manual* for baseband core reset.

As discussed in [Section 9.2.4](#), multiple groups can be configured for DoA estimation. For the limitation of this kind of configuration, please refer to [Section 9.4.3.2](#).



## 8.5 Software and Configuration Suggestions

### 8.5.1 DBF vs DML

Only one of these two engines can be chosen for DoA estimation. A rough comparison between them is listed below:

- Advantages of DBF engine over DML engine:
  - Supporting more than two objects for the same range and Doppler gate
  - Faster
- Advantages of DML engine over DBF engine:
  - Higher resolution with the same antenna array design

### 8.5.2 Steering Vectors

Both DBF and DML require the programming of steering vectors. Given one RX channel's position, say  $p$ , its output signal can be modeled as  $A \exp(j2\pi \frac{p}{\lambda} \sin(\theta))$  when a reflected wave comes from the angle  $\theta$ , as shown in Fig. 8.7.

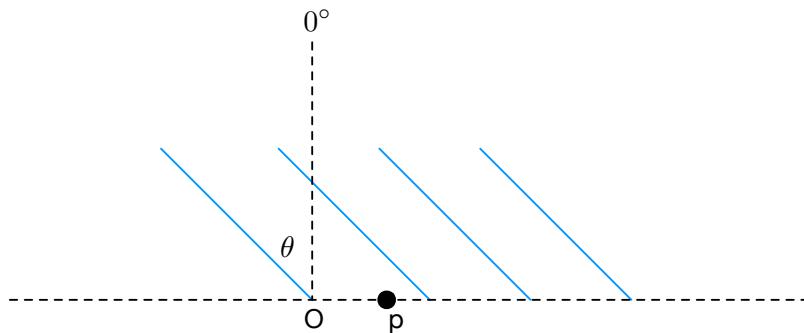


Fig. 8.7: Illustration of Reflected Wave and Its Position Related to Rx Channels

Steering vectors are of form

$$\left[ \exp(j2\pi \frac{p_0}{\lambda} \sin(\theta)), \exp(j2\pi \frac{p_1}{\lambda} \sin(\theta)), \dots, \exp(j2\pi \frac{p_{n-1}}{\lambda} \sin(\theta)) \right]$$

Where  $p_i$  is the position of the  $i$ -th Rx channel. By varying  $\theta$ , one can get a group of steering vectors. For DBF, the maximum supported number of steering vectors is 360. One can choose various ways to distribute the 360 points within the FoV of a radar system.

One simple solution is distributing  $\theta$  within the FoV with equal space.

Another possible way is distributing  $\sin(\theta)$  within  $[-1, 1]$  with equal space. For example, suppose that FoV is  $[-90, 90]$ . Then  $\sin(\theta) \in [-1, 1]$  when  $\theta \in [-90, 90]$ . Generate steering vectors based on

$$\left[ \exp(j2\pi \frac{p_0}{\lambda} u), \exp(j2\pi \frac{p_1}{\lambda} u), \dots, \exp(j2\pi \frac{p_{n-1}}{\lambda} u) \right] \quad (8.8)$$

Where  $u$  is equal-space distributed in  $[-1, 1]$ . This method trades accuracy around  $90^\circ$  for accuracy around  $0^\circ$ .

Both ways are applicable for DBF. However, for DML, only the second way is applicable as described in Section 8.2.2.3 and Section 8.3.4.2. If FoV  $[-90, 90]$  in the previous example is discrete into  $N$  angles, the second way would have  $u_i = -1 + \frac{2i}{N-1}$  in Equation (8.8), where  $i = 0, 1, \dots, N-1$ , and the corresponding angles are  $\theta_i = \arcsin(u_i)$ .

### 8.5.3 Digital Beamforming

#### 8.5.3.1 Modes of Object Decision Engine

Modes for object decision engines are related in one way or another. Here are some general suggestions:

- **OMP mode vs IPM**

The only difference between these two modes is the maximum number of objects, which could cause some detection difference in some corner cases.

For example, there are two objects that are so close to each other that the two peaks in DBF power spectrum merged into one as shown in Fig. 8.8. In this case, if DBF is configured to allow outputting more than 2 objects, IPM might output only one object at  $0^\circ$ , while OMP might output three objects with one at  $0^\circ$  and two on both sides of the first object.

Note that it does not always happen that two objects are so very close. However, if it does happen, in both modes it is difficult to separate the two objects. Therefore, the choice between them should depend on the applications and how software algorithm to handle this type of corner case.

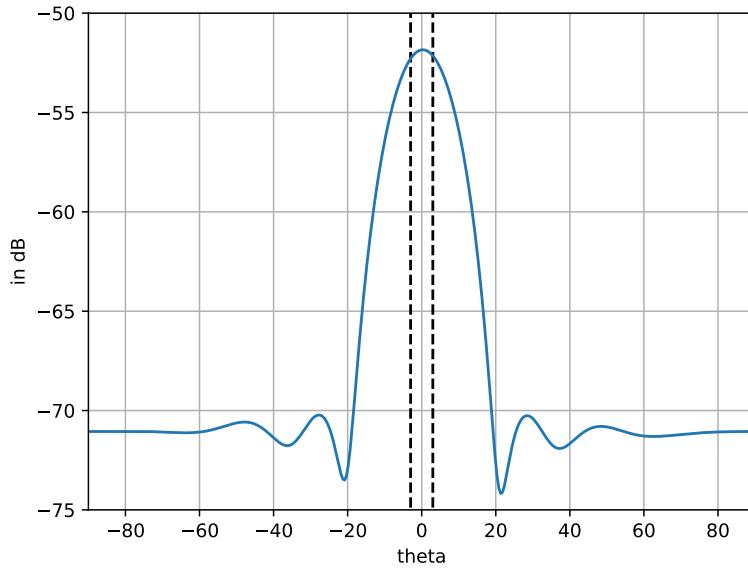


Fig. 8.8: DBF Power Spectrum with Two Objects Closed to Each Other

- **IPM vs Non-IPM**

The only difference between these two modes is whether to search object recursively. Generally, recursive searching have higher resolution in certain cases.

- **FPM vs Others**

It outputs up to 12 peaks and highly depends on the post process to determine the number of objects in the same range and Doppler gate.

### 8.5.3.2 Tuning Knobs

- CFG\_DoA\_GRP0\_NUM\_SCH

As discussed in [Section 8.3.3.2.1](#), this register takes effect only when its value is less than the value of CFG\_DoA\_GRP0\_SIZ\_CMB. Therefore, it is suggested that one should set this value less than CFG\_DoA\_GRP0\_SIZ\_CMB.

- CFG\_DoA\_GRP0\_SCL NOI\_k

It is recommended that one should set this register to a value greater than 1, which means that the final noise level used for exiting iteration is higher than the noise level reported from CFAR.

It is also recommended that one should not do SNR regime-dependent setting first (by setting the same value for all  $k$ ). Turn on SNR regime-dependent setting only when it is really helpful in the application.

- CFG\_DoA\_GRP0\_SCL POW

This register is used to exclude peaks caused by side lobes. One can determine it by antenna pattern and antenna weights added to steering vectors.

- CFG\_DoA\_GRP0\_SIZ\_STP\_PKS\_CRS and CFG\_DoA\_GRP0\_SIZ\_RNG\_PKS\_RFD

Generally, a larger value of CFG\_DoA\_GRP0\_SIZ\_STP\_PKS\_CRS should be paired with a larger value of CFG\_DoA\_GRP0\_SIZ\_RNG\_PKS\_RFD to avoid stuck-at local peaks. Also, the sharper the main lobe of the designed antenna array is, the smaller number of CFG\_DoA\_GRP0\_SIZ\_STP\_PKS\_CRS is required.

## 8.5.4 Deterministic Maximum Likelihood

### 8.5.4.1 Steering Vectors

How to generate steering vectors is discussed in [Section 8.5.2](#). For DML, the second way described in [Section 8.5.2](#) is used. Sine functions of all candidate angles have to be evenly spaced within  $[\sin(\theta_L), \sin(\theta_R)]$ , where  $[\theta_L, \theta_R]$  is the FoV of a radar system.

### 8.5.4.2 Configuration of $d$

As shown in [Equation \(8.5\)](#),  $d$  is actually a function of the difference between the sine functions of two candidate angles. If we have  $N$  candidate angles, the number of such difference is  $N - 1$ . So the number of all possible  $d$  is  $N - 1$ . The list of  $d$  should be generated through [Equation \(8.9\)](#) and put into the memory described in [Section 8.3.4.2](#).

$$d(k) = sv(\theta_{k+1})^H \cdot sv(\theta_0) = \sum_{i=0}^{n-1} e^{\frac{j \cdot 2\pi \cdot p_i \cdot (k+1)(\sin(\theta_R) - \sin(\theta_L))}{\lambda \cdot (N-1)}} \quad (8.9)$$

### 8.5.4.3 Configuration of $k$

Theoretically,  $k$  is programmed according to [Equation \(8.6\)](#). However, in engineering reality, an adjustment has to be applied to [Equation \(8.6\)](#). Because the calculation of metric function in [Equation \(8.4\)](#) involves an inverse operation of the matrix as  $\begin{bmatrix} c & d \\ d^H & c \end{bmatrix}^{-1}$ , when  $\theta_{S0}$  and  $\theta_{S1}$  in [Equation \(8.4\)](#) is close to each other, the matrix  $\begin{bmatrix} c & d \\ d^H & c \end{bmatrix}$  will also become nearly invertible. Due to the fact that only finite bit length can be used for internal registers in the DML engine, an adjusting factor is applied to [\(8.6\)](#) to suppress the effect of quantization error. The generation of  $k$  list is shown in [Equation \(8.10\)](#).

$$k(i) = \frac{g(i)}{c^2 - d(i)^H \cdot d(i)}, \quad i = 0, 1, \dots, N - 2 \quad (8.10)$$



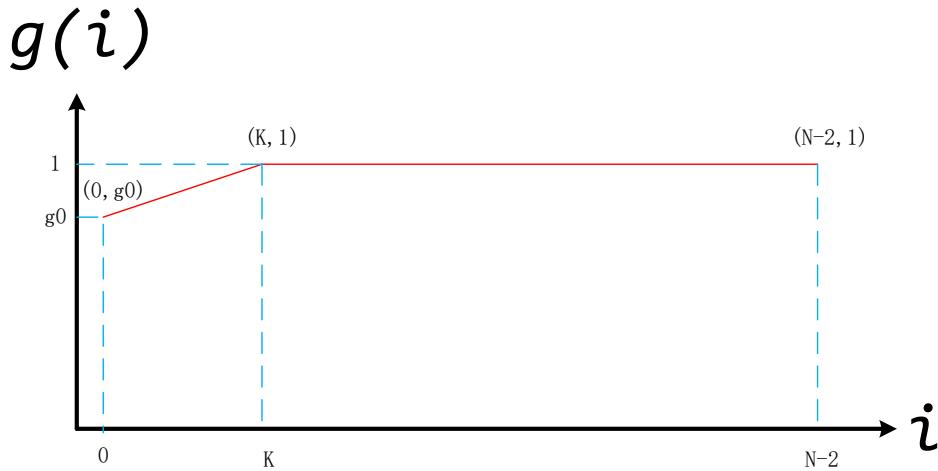


Fig. 8.9: K Factors  $g(i)$

As shown in Fig. 8.9, the definition of  $g(i)$  is given by:

$$g(i) = \begin{cases} g_0 + \frac{i \cdot (1-g_0)}{K} & , \text{ when } i < K \\ 1 & , \text{ otherwise} \end{cases} \quad (8.11)$$

Where  $g_0$  is the initial factor. it is suggested that  $g_0$  be set with the value 0.8. The  $K$  should be set to the biggest integer that satisfies:

$$\frac{K \cdot (\sin(\theta_R) - \sin(\theta_L))}{N - 1} < \frac{\lambda}{D} \quad (8.12)$$

### 8.5.4.4 Configuration of *start*, *step* and *end*

The parameters *start*, *step*, and *end* control the range and pace of angle searching in 2O-DML.

As have been discussed in Section 8.2.2.2, a smaller *step* can result in a smaller chance of falling into local extreme point when iterating all pairs of candidate angles. However, a small *step* may result in unacceptable long run time as the best *step* in terms of time-consumption is  $\text{step} = \sqrt{N}$ . To balance these conflicts, the *step* value should be the integer that satisfies both of the following two conditions.

- $\frac{\text{step} \cdot (\sin(\theta_R) - \sin(\theta_L))}{N} < \frac{\lambda}{2 \cdot D}$ , where  $\lambda$  is the wave length and  $D$  is the antenna aperture.
- As close as possible to  $\sqrt{N}$ . This condition is necessary when run time is critical to the system. Otherwise, a smaller *step* is preferred.

During the initialization of the DML engine at the beginning of every frame, a random offset with uniform distribution in range  $[0, \frac{\text{step}}{2}]$  should be added in *start*.

The parameter *end* should be less than the number of candidate angles  $N$ .

#### 8.5.4.5 Reflector Number Estimator (RNE)

As can be seen in [Section 8.3.4.3](#), baseband provides a very flexible way to configure the behavior of RNE. The best combination of  $c_i$  ( $i = 0, 1, 2, 3, 4$ ) depends on many factors, such as the configuration of TX and RX antennas of the radar system. Data statistics and training may be required. However, simulation shows that some simple rules are also good enough for some simple scenarios. For example, the difference of two projecting powers of two angles resulting from 2O-DML can be compared to a threshold (e.g. 10 dB), exceeding it means one angle's power is the dominating one and one reflector is likely out there. In this case, the  $c_i$  ( $i = 0, 1, 2, 3, 4$ ) may be configured as:

Table 8.4: Coefficient Configuration of RNE

Coefficient	Value	Coefficient	Value
$c_0$	0	$c_1$	0
$c_2$	0.1	$c_3$	-1
$c_4$	0		

## 8.6 Examples

### 8.6.1 Digital Beamforming

As an example given here, it is supposed that the radar system has 4 Rx channels and DBF is in used to detect up-to 2 objects. It is also assumed that 360 steering vectors will be used for DoA estimator. The corresponding register values are

Table 8.5: Table of Example with DBF

Register Name	Address	Value	Comments
CFG_DoA_GRP0_SIZ RNG_PKS_CRS	0xC00A48	359	$360 - 1 = 359$
CFG_DoA_GRP0_SIZ_CMB	0xC00A54	3	$4 - 1 = 3$
CFG_DoA_GRP0_MOD_SCH	0xC00A38	4	Using OMP mode
CFG_DoA_GRP0_NUM_SCH	0xC00A3C	1	$2 - 1 = 1$
CFG_DoA_GRP0_THR_SNR_0	0xC00A7C	10	Stopping criterion is <i>not</i> SNR dependent
CFG_DoA_GRP0_THR_SNR_1	0xC00A80	10	
CFG_DoA_GRP0_SCL NOI_0	0xC00A88	0x283	Total power is no less than 5 times the noise level
CFG_DoA_GRP0_SCL NOI_1	0xC00A8C	0x283	Total power is no less than 5 times the noise level
CFG_DoA_GRP0_SCL NOI_2	0xC00A90	0x283	Total power is no less than 5 times the noise level
CFG_DoA_GRP0_DAT_IDX_0	0xC00A58	0x00443	Antenna indices for using 0, 1, 2, 3 Rx Channels

For examples of programming of steering vectors, one can refer to [Section 9](#).

### 8.6.2 Deterministic Maximum Likelihood

This section gives an example of how to set up the DML engine. The configuration of the DML engine has much to do with the system's antenna layout. For example, the antenna's position will impact the generation of steering vectors; The aperture of virtualized antenna will impact the step of Coarse Search as described in [Section 8.2.2.2](#) and [Section 8.5.4.4](#); Finally, the layout of virtualized antenna will impact the generation of the coefficients used in RNE as described in [Section 8.3.4.3](#).

Take an AIP system as an example. Suppose the antenna board has 4 RX antenna and 4 TX antenna. Using RX0 as reference, RX antenna's position is listed as [0, 0.6, 2.1, 3.2]. Note that the position is normalized by wave length of the signal. Using TX1 as reference, TX antenna's position is listed as [6.2, 0, 1, 1.5]. If we use TX0 and TX1 to form a virtual array with  $N_{va} = 2$ , the virtualized antenna position will be listed as  $[d_i | i = 0, 1, \dots, 7] = [0, 0.6, 2.1, 3.2, 6.2, 7.8, 8.3, 9.4]$ . The maximum antenna aperture  $D$  is  $9.4 \cdot \lambda$ .

The parameters of DML can be determined following these steps in the phase of system initialization:

## 8.6. Examples

---

1. The corresponding coefficients for RNE is trained as  $[c_i | i = 0, 1, \dots, 4] = [0.16, -1, 0.18, 0.03, 0.20]$ .
2. The FoV of the board is around  $[-30, 30]$ , so we can set  $\theta_L = -30^\circ$  and  $\theta_R = 30^\circ$ .
3. Set the candidate angle number to 300.
4. Generate corresponding steering vectors as described by (8.8).
5. Set *start* to 0 and *end* to 299.
6. According to Section 8.5.4.4, set the *step* to 13.
7. To generate the *k* list, set the *K* as described in section (8.12) to 26.
8. Generate the *d* list as described by (8.9).

During the initialization of each frame, a random value with uniform distribution among the range of  $[0, \frac{\text{step}}{2}]$  can be added to *start*.

Table 8.6: Baseband Settings to Enable MIMO

Register Name	Offset	Bit Width	Default Value	Value and description
CFG_SYS_SIZE_BPM	0x34	2	0	1 ( $N_{va} = 2$ )
CFG_DoA_DBPM_ENA	0xA1C	1	0	0 (TDM)
CFG_DoA_DBPM_ADR	0xA20	12	0	0 (TDM)

Table 8.7: Baseband Settings to Activate Normal Mode

Register Name	Offset	BiW	Default value	value
CFG_DoA_MODE_GRP	0xA2C	2	0	0 (Normal Mode)
CFG_DoA_NUMB_GRP	0xA30	2	0	0 (Normal Mode)

Table 8.8: Signal Vector Settings

Register Name	Offset	BiW	Default value	value
CFG_DoA_GRP0_DAT_IDX_0	0xA58	20	0	0b00000,00001,00010,00011
CFG_DoA_GRP0_DAT_IDX_1	0xA5C	20	0	0b00100,00101,00110,00111
CFG_DoA_GRP0_DAT_IDX_2	0xA60	20	0	0
CFG_DoA_GRP0_DAT_IDX_3	0xA64	20	0	0
CFG_DoA_GRP0_DAT_IDX_4	0xA68	20	0	0
CFG_DoA_GRP0_DAT_IDX_5	0xA6C	20	0	0
CFG_DoA_GRP0_DAT_IDX_6	0xA70	20	0	0
CFG_DoA_GRP0_DAT_IDX_7	0xA74	20	0	0

Registers  $CFG\_DoA\_GRP1\_DAT\_IDX_t$   $t = 0, 1, \dots, 7$  are all cleared as 0. Registers  $CFG\_DoA\_GRP2\_DAT\_IDX_t$   $t = 0, 1, \dots, 7$  are all cleared as 0.

Table 8.9: Steering Vectors of Settings for Combined Mode

Register Name	Offset	BiW	Default value	value
CFG_DoA_GRP0_ADR_COE	0xA40	12	0	0 (TDM + SISO-CFAR)
CFG_DoA_GRP0_SIZ RNG_PKS_CRS	0xA48	10	0	299
CFG_DoA_GRP0_SIZ_CMB	0xA54	5	0	7

DML related registers are set as shown in Table 8.10.

Table 8.10: DML Register Settings for Combined Mode

Register Name	Offset	BiW	Default value	value
CFG_DML_GRP0_SV_START	0xE04	9	0	0
CFG_DML_GRP0_SV_STP	0xE00	6	0	13
CFG_DML_GRP0_SV_END	0xE08	10	0	299
CFG_DML_GRP1_DC_COE_0	0xE34	14	0	Numerical Value for $c_0$ of group0
CFG_DML_GRP1_DC_COE_1	0xE38	14	0	Numerical Value for $c_1$ of group0
CFG_DML_GRP1_DC_COE_2	0xE3C	14	0	Numerical Value for $c_2$ of group0
CFG_DML_GRP1_DC_COE_3	0xE40	14	0	Numerical Value for $c_3$ of group0
CFG_DML_GRP1_DC_COE_4	0xE44	14	0	Numerical Value for $c_4$ of group0
CFG_DML_GRP0_EXTRA_EN	0xE20	1	0	1
CFG_DML_GRP0_DC_COE_2_EN	0xE24	1	0	1
CFG_DoA_GRP0_NUM_SCH	0xA3C	2	0	2 (RNE detect Obj number dynamically)
CFG_DML_MEM_BASE_ADDR	0xE50	10	0	299
CFG_DoA_GRP0_MOD_SCH	0xA38	3	0	1
CFG_DoA_GRP0_SIZ_STP_PKS_CRS	0xA4C	10	0	0
CFG_DoA_GRP1_MOD_SCH	0xAC0	3	0	1
CFG_DoA_GRP0_SIZ_STP_PKS_CRS	0xAD4	10	0	0

Please refer to Section 9.6.7 for the example code of generating of steering vector and  $d$  and  $k$ .

## **8.6. Examples**

---



## VIRTUAL ARRAY (MIMO)

### 9.1 Overview

In general, the performance of DoA estimation is limited by the aperture of RX antenna. It is possible to increase the aperture by virtualizing more RX antenna channels through signal modulation at TX antennas. Alps supports time-division modulation (TDM) and binary-phase modulation (BPM).

#### 9.1.1 Background

With more and more requirements for angle measurement of current automotive radar systems, such as high angular resolution and measurements of both azimuth angle and elevation angle, the traditional way of increasing the number of RX channels has been difficult to meet the requirements, so virtual array is proposed.

Virtual array (MIMO) is a way to virtualize more RX channels by increasing the number of TX channels. Take a radar system with 1 TX channel and 4 RX channels for example. It is straight forward that the reflected signal corresponding to the signal transmitted by the TX will get 4 copies at the receiver, as illustrated by Fig. 9.1. The phase difference between every two adjacent copies is  $\omega = (2\pi/\lambda)d \sin(\theta)$ .

But if we get an additional TX channel and let the two TX channels transmit the same signal at the same time, there will be a time interval between the reflected signals from TX0 and from TX1. So there will be altogether 8 copies of the reflected signal at the receiver end, as illustrated by Fig. 9.2. It is equivalent to extending the size of the antenna array to 8 at the receiver end, which is illustrated by Fig. 9.3. That's the essence of virtual array.

Different from the case where we have 8 RX channels, with virtual array, each RX channel involves two copies of data and we have to figure out a way to separate them. Only if we can separate them, can we realize the virtualization of 8 RX channels. A common idea is to make the multiple copies of data received at the same RX channel orthogonal.

TDM and BPM realize data orthogonalization from the time domain and the code domain respectively, which are the main topics of this chapter.

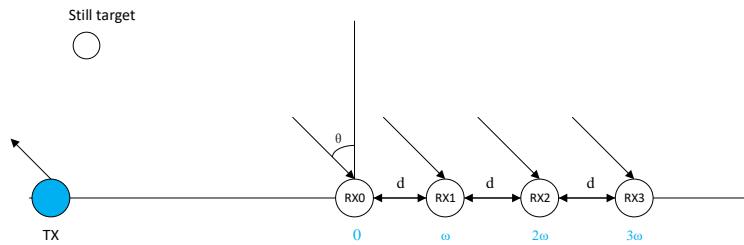


Fig. 9.1: 1T4R Radar

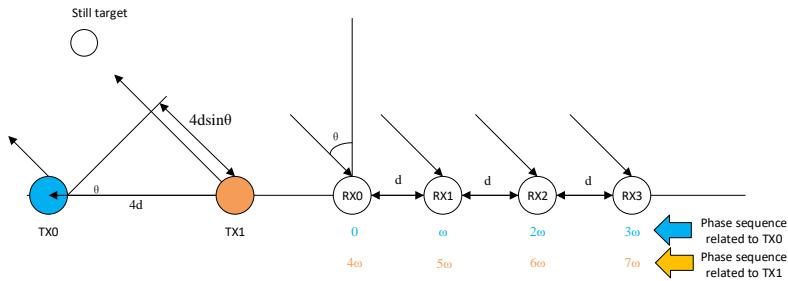


Fig. 9.2: 2T4R MIMO Radar

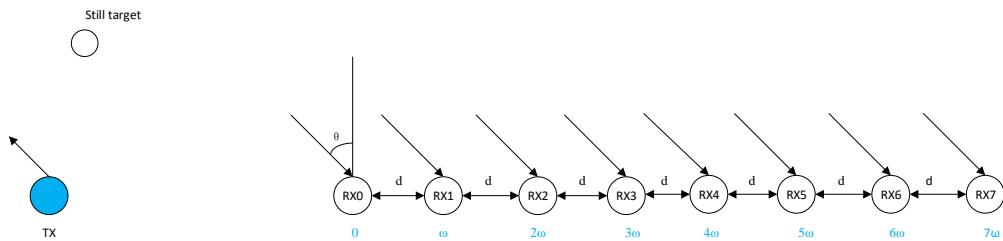


Fig. 9.3: 1T8R Radar

As mentioned, virtual array can virtualize more RX channels or a new antenna array, but it does not determine the angle measurement. It is the design of the antenna array that determines the angle measurement. For example, the aperture of the antenna array determines angular resolution; and to measure both azimuth angle and elevation angle, a planar antenna array is required. Angle measurement is another important topic of this chapter.

### 9.1.2 Features

- TDM
- BPM
- 2D DoA
  - Normal mode
  - Single shot mode
  - Combined mode

## 9.2 Functional Description

### 9.2.1 TDM (Time-Division Modulation) MIMO

TDM scheme virtualizes RX channels through time domain modulation. Since Alps has 4 TX channels and 4 RX channels, it is possible to virtually create 8 or 12 or 16 RX channels. In particular, Fig. 9.4, Fig. 9.5, and Fig. 9.6 show the three schemes, respectively. Generally, TDM scheme only turns on one TX antenna for particular chirps within a frame. To achieve such virtualization, one needs to design the antennas so as to make the separation of TX antennas equal the total aperture of RX antenna array. For example, if RX antenna array is a union linear array (ULA) with separation of  $\lambda/2$ , then TX antenna separation should be  $2\lambda$  to achieve corresponding virtual array with TDM of maximal aperture.

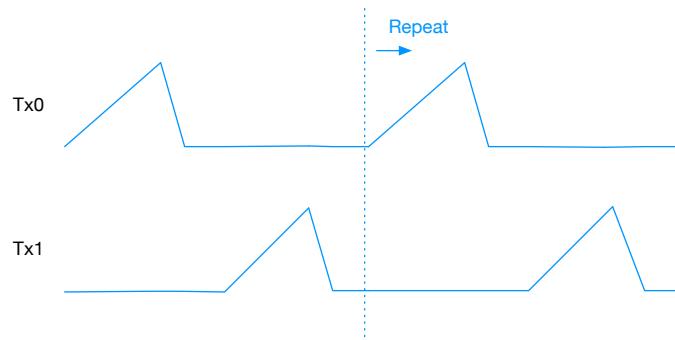


Fig. 9.4: TDM Virtual Array Scheme with 2 TX Channels

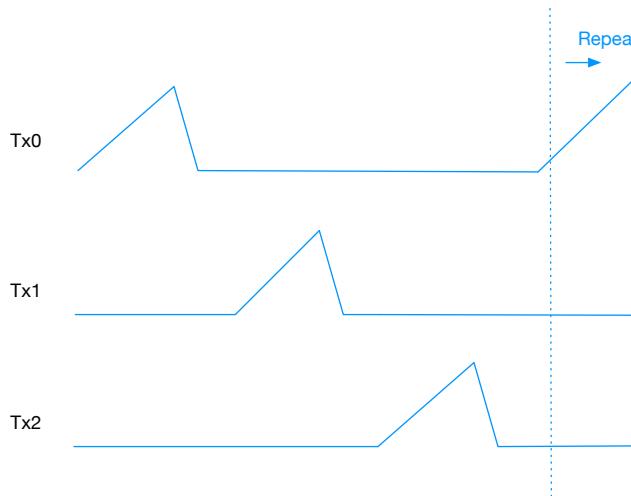


Fig. 9.5: TDM Virtual Array Scheme with 3 TX Channels

## 9.2. Functional Description

---

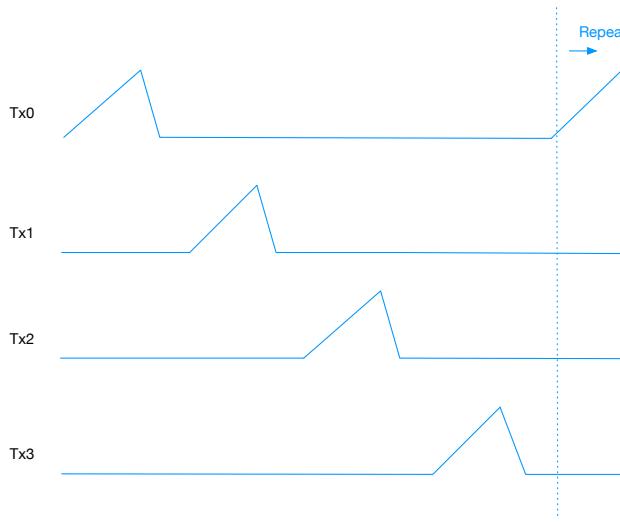


Fig. 9.6: TDM Virtual Array Scheme with 4 TX Channels

It is not hard to see from the figures above that the data of different RX channels are distinguished by chirp.

If  $T_r$  denotes the length of a chirp,  $T'_r$  denotes the period of the chirp in TDM scheme, and  $N_{va}$  denotes the number of TX channels (the number of virtual array groups), then  $T'_r = N_{va}T_r$ .

The lengthened chirp period is a drawback of MIMO and we will discuss it in [Section 13.2.1](#) and [Section 13.2.2](#).

### 9.2.2 BPM (Binary-Phase Modulation) MIMO

BPM scheme virtualizes RX channels through the coding domain, which utilizes Hadamard matrix. Compared with TDM, BPM has larger total TX power. However, since  $3 \times 3$  Hadamard does not exist, BPM scheme can only virtually create 8-RX array or 16-RX array. In particular, [Fig. 9.7](#) and [Fig. 9.8](#) show the two schemes, respectively. Labels inside chirps indicate the phases of corresponding chirps at particular TX channels. 1 indicates phase  $0^\circ$  and  $-1$  indicates phase  $180^\circ$ . The antenna design for BPM scheme follows the same principle as for TDM scheme.

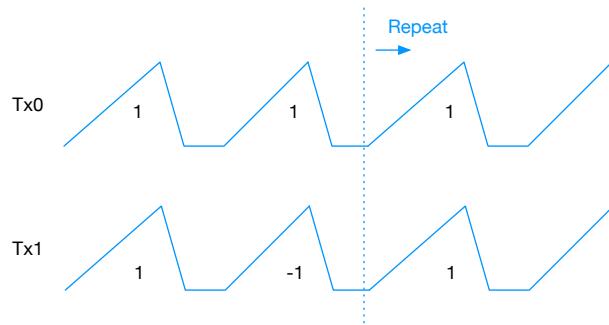


Fig. 9.7: BPM Virtual Array Scheme with 2 TX Channels

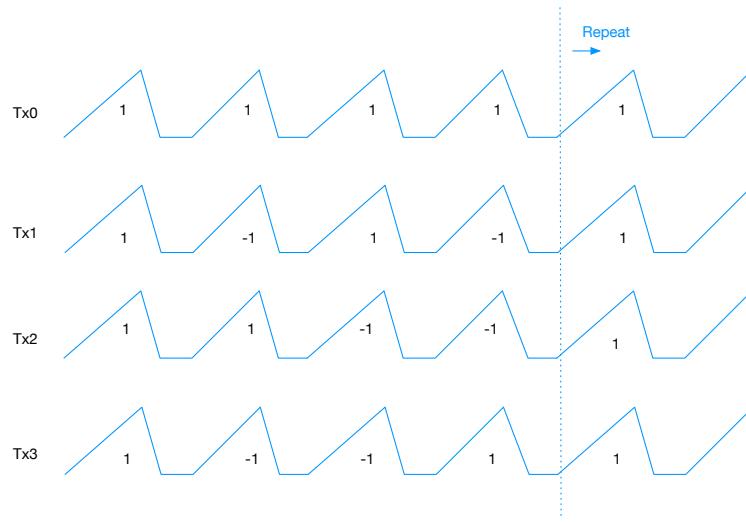


Fig. 9.8: BPM Virtual Array Scheme with 4 TX Channels

It is not difficult to figure out the difference between BPM and TDM from the figures above. Multiple copies of data corresponding to the number of TX channels are mixed in a chirp, we must have the corresponding number of chirps to solve the equations and separate the data of different RX channels. Because the data is linear superposed, we can do the separation at any stage before DoA.

To reduce the computations, Alps supports doing the separation (demodulation) after CFAR and just before DoA.

Suppose that the 2D-FFT vector corresponding to the TDM scheme with 2 TX channels shown in Fig. 9.4 is  $[x_0, \dots, x_7]^T$ , then for BPM as shown in Fig. 9.7, it should be demodulated as the following way:

$$\begin{bmatrix} \tilde{x}_i \\ \tilde{x}_{i+4} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} x_i \\ x_{i+4} \end{bmatrix}, i = 0, 1, 2, 3$$

As for BPM shown in Fig. 9.8, it should be demodulated as the following way:

$$\begin{bmatrix} \tilde{x}_i \\ \tilde{x}_{i+4} \\ \tilde{x}_{i+8} \\ \tilde{x}_{i+12} \end{bmatrix} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ x_{i+4} \\ x_{i+8} \\ x_{i+12} \end{bmatrix}, i = 0, 1, 2, 3$$

Because all TX channels transmit at the same time, the total power of BPM is stronger than TDM, which means an improvement in SNR. But BPM still has the same drawback as TDM, which is the lengthened chirp period.

### 9.2.3 Phase Compensation in Virtual Array

For a SIMO system as shown in Fig. 9.1, the phase differences of reflect signals between different RX channels are caused by DoA and antenna spacing. For a MIMO system, however, there is another factor that gives an extra phase difference between reflected signals from different TX channels and this factor is the movement of targets. See Fig. 9.9.

## 9.2. Functional Description

---

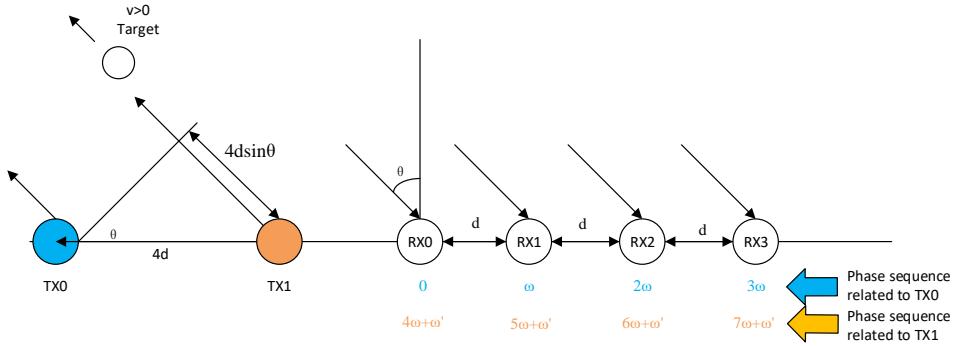


Fig. 9.9: Moving Target Causing Extra Phase Differences in MIMO

In a MIMO system, if the Doppler frequency of a target is  $f_{\text{Doppler}}$ , then the extra phase difference is  $\omega' = 2\pi f_{\text{Doppler}} T_{itv}$ . Here  $T_{itv}$  denotes the time interval for TX channels to transmit signal.

If the phase differences are not eliminated and the data of these RX channels is used to compute DoA, the results will go wrong.

Alps supports the compensation for phase difference caused by the movement of targets in a MIMO system at two alternative stages. One stage is after 2D-FFT and the other stage is after CFAR but before DoA.

Generally, let  $k$  and  $p$  denote the frequency indices of 2D-FFT output.  $k$  is the frequency index of range FFT (RNG-FFT) and  $p$  is the frequency index of velocity FFT (VEL-FFT). Let  $L_z$  denote the size of VEL-FFT.  $T_r$  is the length of a chirp and  $T'_r$  is the period of the chirp in virtual array. Then for the virtual array group with the index  $va$  ( $va = 0 \dots N_{va} - 1$ ), the phase compensation value  $\text{Ph}(va)$  will be computed in the following two alternative stages.

- If the system does not have the problem of velocity ambiguity, phase compensation is done after 2D-FFT according to the following formula.

By default, phase compensation is done at this stage.

$$\text{-- For } p \leq \frac{L_z}{2}, f_{\text{Doppler}} = \frac{p}{T'_r L_z}.$$

$$\text{Ph}(va) = \exp \left( j2\pi \left( \frac{p}{T'_r L_z} \right) T_r va \right) \quad (9.1)$$

where  $T'_r = N_{va} T_r$ .

$$\text{-- For } p > \frac{L_z}{2}, f_{\text{Doppler}} = \frac{p-L_z}{T'_r L_z}.$$

$$\text{Ph}(va) = \exp \left( j2\pi \left( \frac{p-L_z}{T'_r L_z} \right) T_r va \right) \quad (9.2)$$

where  $T'_r = N_{va} T_r$ .

- If there is the problem of velocity ambiguity, phase compensation has to be done after CFAR but before DoA, with the ambiguity factor  $q$  known, as shown in Section 13.2.1.3. Users can refer to Section 13.2.1 for more information.

The compensation of phase is implemented as multiplying the result of 2DFFT by  $\text{Ph}(va)$ .

## 9.2.4 2D DoA in Virtual Array

2D DoA is to get both azimuth and elevation DoA estimates of the targets.

### 9.2.4.1 Background

Take the coordinate system shown in Fig. 9.10 as an example. The position of the target is at  $\mathbf{T}$ . The planar antenna array (viewer) is located in the XOZ plane, and  $\mathbf{OU}$  is the projection of  $\mathbf{OT}$  in the XOY plane. Then the azimuth angle  $\theta$  is the angle between  $\mathbf{OU}$  and the Y axis and the elevation angle  $\varphi$  is the angle between  $\mathbf{OT}$  and  $\mathbf{OU}$ .

It is obvious that if either the azimuth angle or elevation angle becomes 0 degree, the problem of the direction of arrival (DoA) of the target can be solved by using a linear antenna array and we call it 1D DoA. As for the target shown in Fig. 9.10, it is a target in three-dimensional space. Neither the azimuth nor the elevation angle is zero, so the problem of the direction of arrival (DoA) of the target has to be solved by using a planar antenna array and we call it as 2D DoA.

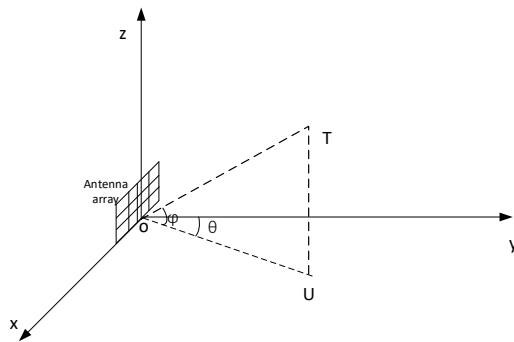


Fig. 9.10: Example of Coordinate System

The reflector signal generated by 2D DoA can be modeled as follows:

$$S = A * \exp \{j2\pi d_x / \lambda \sin \theta \cos \varphi\} \exp \{j2\pi d_z / \lambda \sin \varphi\} \quad (9.3)$$

Where  $(d_x, d_z)$  is the position of the RX in the planar antenna array and regardless of any noise.

In a radar system with the feature of 2D scan of spatial domain, the azimuth angle  $\theta$  is estimated by receiving the reflected signal from the object using multiple receivers on the x-axis. These receivers are spaced apart with the same distance  $d_x$  or different intervals. The phase shift of these receivers can be expressed as  $2\pi d_{x_i} / \lambda \sin \theta \cos \varphi$ , where  $d_{x_i}$  is the position of an RX channel on the x-axis. Here we introduce an intermediate angle  $\alpha$ , which is a combination of angle  $\theta$  and angle  $\varphi$ . The relationship of them is shown as follows:

$$\sin \alpha = \sin \theta \cos \varphi \quad (9.4)$$

It is easy to know we can get the intermediate angle  $\alpha$  first, and when we get the angle  $\varphi$ , we can use Equation (9.4) to figure out the angle  $\theta$ , which is the main idea of the 2D DoA algorithm.

The planar antenna array can be virtualized by virtual array (MIMO). See the example in Fig. 9.11. Users can choose FFT data from different groups of RX channels and different 2D scan methods to estimate azimuth DoA and elevation DoA. For details about DoA methods used in 2D DoA, see Section 8.2.1 and Section 8.2.2.

Alps supports three modes for 2D scan:

- *Normal Mode*

## 9.2. Functional Description

- Single Shot Mode
- Combined Mode

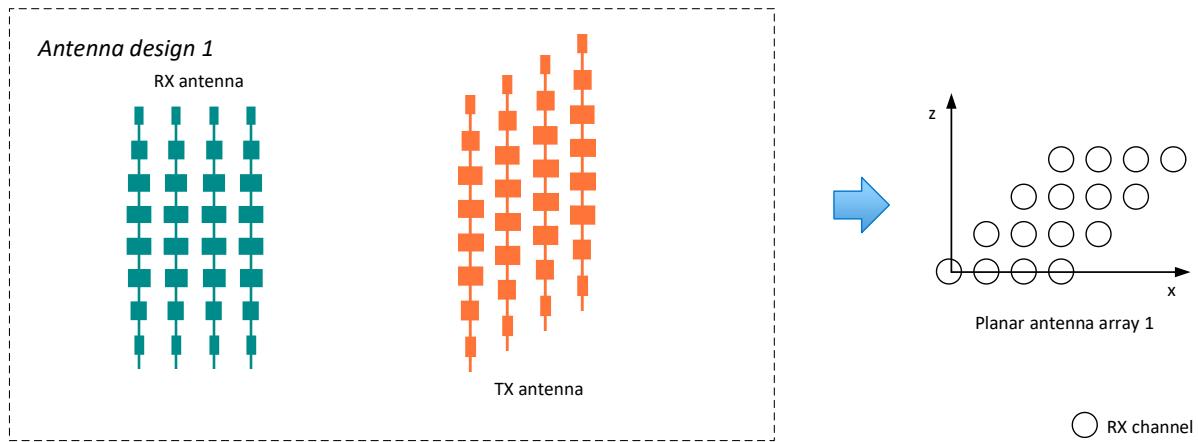


Fig. 9.11: Planar Array with Virtual Array

### 9.2.4.2 Normal Mode

Normal mode is an extension of 1D DoA and uses 3 groups of channels to do DBF with different sets of steering vectors. Group A is used for estimation of the intermediate angle  $\alpha$ , and Group B is used for elevation angle estimation. Results from Group C are viewed as aided information to solve the pairing issue in firmware.

It is recommended that the RX channels chosen for a group meet the following requirements:

- The RX channels are in a straight line.
- The phase differences in the RX group for the estimation of the intermediate angle  $\alpha$  is caused only by the azimuth angle and elevation angle.
- The phase differences in the RX group for elevation angle estimation is caused only by the elevation angle.

Take a 4T4R radar system as an example. The maximum number of RX channels realized through virtual array is 16. Suppose an equivalent array is generated as shown on the left side of Fig. 9.12.

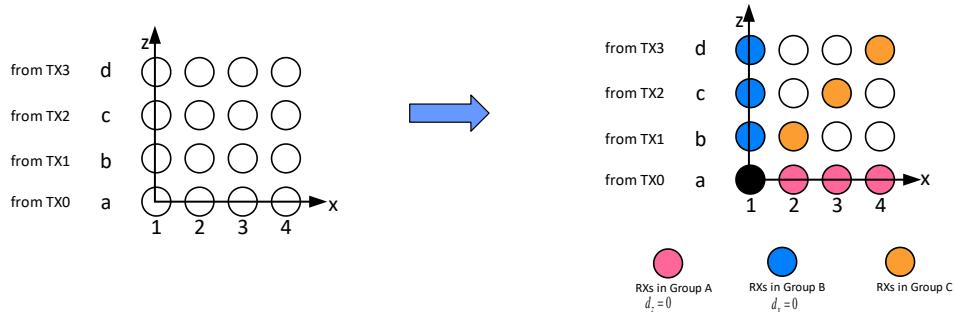


Fig. 9.12: Example of Selecting 3 Different Groups for 2D-DoA with Normal Mode

For 2D-DoA with normal mode, one can choose three groups of channels as shown on the right side of Fig. 9.12.



Table 9.1 lists the details the selected groups.

Table 9.1: Example of Selecting 3 Different Groups for 2D-DoA with Normal Mode

Group	Channels in Group
Group A	(1, a), (2, a), (3, a), (4, a)
Group B	(1, a), (1, b), (1, c), (1, d)
Group C	(1, a), (2, b), (3, c), (4, d)

The flow chart of normal mode is shown in Fig. 9.13.

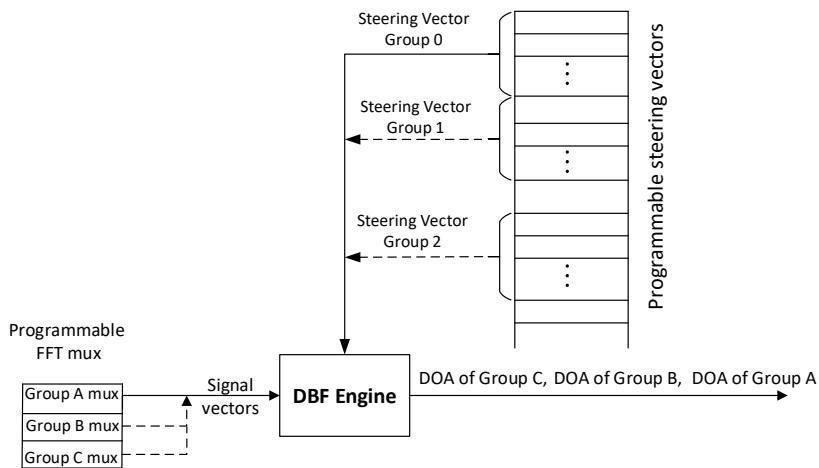


Fig. 9.13: Flow Chart of 2D DoA with Normal Mode

As shown in Fig. 9.13, hardware calls the DBF engine in sequence and outputs the DoA result of each group in sequence. For each group, based on the range-gate and Doppler-gate indices provided by CFAR, hardware queries the corresponding FFT result from given channels (programmable channels in a group) and form a vector to feed into DBF engine. As for the steering vectors of each group are computed in advance as Section 8.5.2 and fed into DBF engine too.

#### 9.2.4.3 Single Shot Mode

Single shot mode is similar to normal mode in choosing groups of channels to estimate DoA, but in one frame it supports only one of the three groups mentioned in **Normal Mode** for DBF and the group is programmable.

It uses full peak mode mentioned in Section 8.2.1.3.4 to measure DoA. The structure is shown in Fig. 9.14.

## 9.2. Functional Description

---

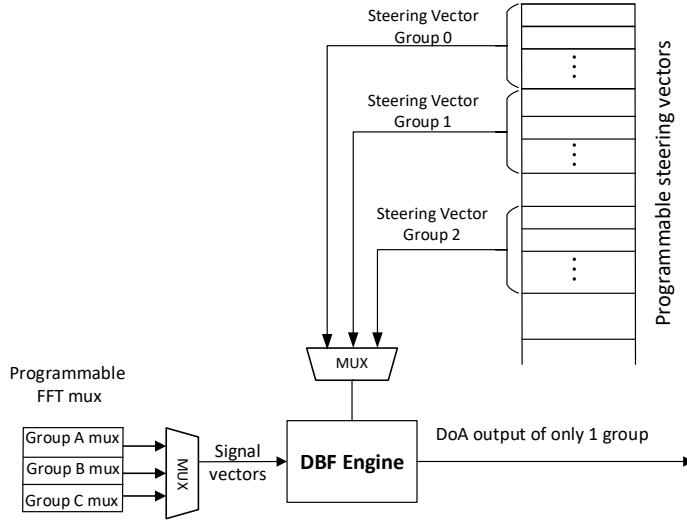


Fig. 9.14: Flow Chart of 2D DoA with Single Shot Mode

### 9.2.4.4 Combined Mode

Combined mode is different from the two modes mentioned above in estimation of the elevation DoA.

The elevation angle  $\varphi$  in combined mode is estimated by receiving the reflected signal from the object using multiple groups of receivers that are parallel to the x-axis. We call these groups as **C2D Groups**, and the multiple receivers on the x-axis used to estimate the intermediate angle  $\alpha$  are grouped as the **Principal Group**.

Suppose a receiver of a **C2D Group** is located at the position  $\{d_x, d_z\}$ . Then the signal arriving at the receiver is delayed by  $(d_x \sin \alpha + d_z \sin \varphi)$ , which causes a phase shift of  $\frac{2\pi}{\lambda} (d_x \sin \alpha + d_z \sin \varphi)$ . Only if the intermediate angle  $\alpha$  is got, we can subtract it from the total phase shift and get the phase shift caused only by the elevation angle  $\varphi$ . After subtraction, all receivers in the same **C2D Group** are of the same phase shift  $\frac{2\pi}{\lambda} d_z \sin \varphi$  and can be further coherently combined. This phase shift between each of the **C2D Groups** and the **Principal Group** is used to estimate the elevation angle  $\varphi$ . That is the main idea of combined mode.

Let  $r_{az}$  denote the vector formed by FFT results from the channels in a Principal Group.

Let  $c_g$  denote the vectors formed by FFT results from the channels in C2D Groups  $g$  ( $g = 1 \dots n_{c2d}$ ,  $1 \leq n_{c2d} \leq 3$ ).

Then the vector of phase difference of these group (denoted by  $r_{ev}$ ) arising from the elevation angle can be obtained by calculating the projection of all groups of channels to a same intermediate angle  $\alpha$ .

After  $r_{ev}$  is obtained, the elevation angle corresponding to the intermediate angle can be computed by calling DoA engine once again.

The whole process can be described as follows:

1. Use  $r_{az}$  and Steering Vector Group 0 to estimate the intermediate angle  $\alpha$ , and let  $\Theta$  denote the set of intermediate angles.  $|\Theta|$  is the size of  $\Theta$ .
2. Compute the inner products  $r_{az} \bullet w_p(\theta_i)$  and  $c_g \bullet w_{c_g}(\theta_i)$  respectively, and collect all inner products to form  $r_{ev}$ , where;
  - $\theta_i$  denotes one element in the set  $\Theta$ .
  - $w_p(\theta_i)$  denotes the steering vector formed by antenna positions  $(d_x, d_z)$  of Principal Group from  $\theta_i$ .
  - $w_p(\theta_i)$  is programmable in the same way as Steering Vector Groups 0 and 1.

- $\mathbf{w}_{c_g}(\theta_i)$  denotes the steering vector formed by antenna positions  $((d_x, d_z))$  of C2D Group  $g$  from  $\theta_i$ .  $\mathbf{w}_{c_g}(\theta_i)$  is programmable in the same way as Steering Vector Groups 0 and 1, too.
3. Use  $\mathbf{r}_{ev}$  and Steering Vector Group 1 to estimate the elevation angle, and let  $\Phi_{\theta_i}$  denote the set of elevation angles with the same intermediate angle  $\theta_i$ .  $|\Phi_{\theta_i}|$  is the size of  $\Phi_{\theta_i}$ .
  4. Repeat Step 2 and Step 3 with different elements in the set  $\Theta$  until all the elements in the set  $\Theta$  get matched set of elevation angles.
  5. Output  $\Theta$  and  $\Phi_{\theta_i}$  ( $i = 0 \dots (|\Theta| - 1)$ ).

The maximal numbers of intermediate angles  $|\Theta|$  and elevation angles  $|\Phi_{\theta_i}|$  are programmable.

The main processing flow of combined mode is shown in Fig. 9.15. It should be noted that the MIMO combination in Fig. 9.15 is implemented by averaging the inner products of two vectors.

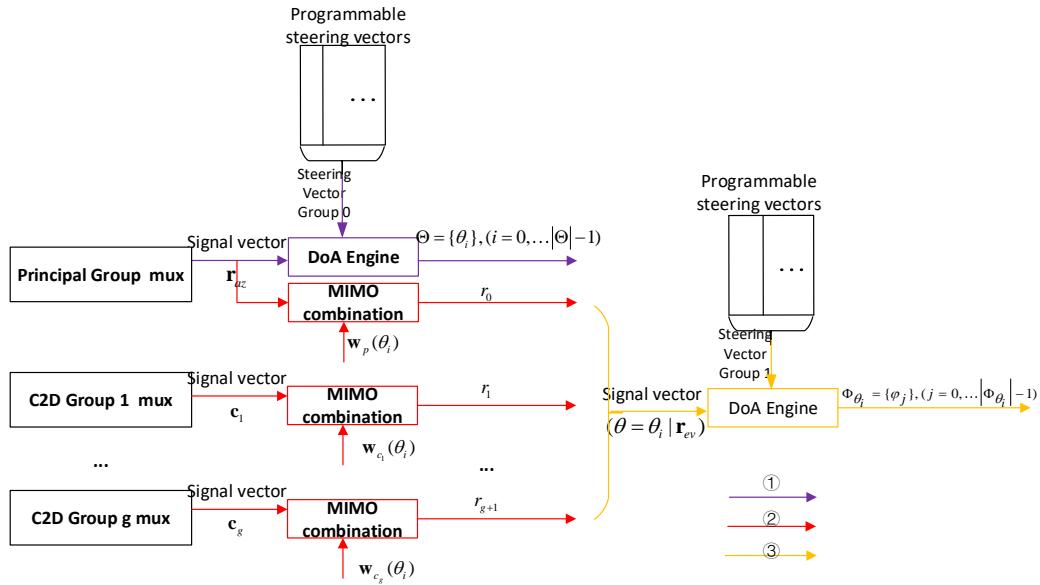


Fig. 9.15: Main Processing Flow of 2D DoA with Combined Mode

Suppose an equivalent array is generated as shown on the left side of Fig. 9.16. Then to do 2D-DoA with combined mode, one can choose four groups of channels as shown on the right side of Fig. 9.16.

Table 9.2 shows the details of the selected groups.

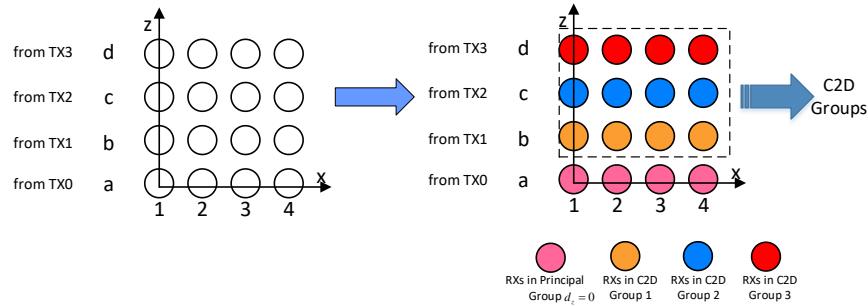


Fig. 9.16: Example of Selecting Groups of Channels for 2D-DoA with Combined Mode

Table 9.2: Settings for 2D-DoA with Combined Mode

Group	Channels in Group
Principal Group	(1, a), (2, a), (3, a), (4, a)
C2D Group 1	(1, b), (2, b), (3, b), (4, b)
C2D Group 2	(1, c), (2, c), (3, c), (4, c)
C2D Group 3	(1, d), (2, d), (3, d), (4, d)

Obviously, using combined mode will skip the pairing issue that exists in normal mode and single shot mode.

## 9.3 Programming Interfaces

### 9.3.1 Virtual Array

To active virtual array, one needs to configure both radio and baseband. For radio configuration, refer to Fig. 4.21. Here, the focus is on baseband setup. As mentioned before, different numbers of TX channels ( $N_{va}$ ) are supported to realize MIMO and there are three alternative modes of 2D DoA. The following table summarizes the registers related.

Table 9.3: Programming Interfaces for virtual array

Register Name	Offset	Bit Width	Description
CFG_SYS_SIZE_BPM	0x34	2	Size of virtual array group. It should be set to $N_{va} - 1$ .
CFG_FFT_DBPM_ENA	0x608	1	Enable bit for phase compensation for virtualized RX channels after 2DFFT. 0-> disable; 1-> enable.
CFG_DoA_DBPM_ENA	0xA1C	1	Enable bit for demodulation. 0-> disable; 1-> enable.
CFG_DoA_DBPM_ADR	0xA20	12	Offset address for the BPM demodulation (DBPM) coefficients. Format: FXI(12, U) Let x denote the offset address, BPM demodulation (DBPM) coefficients are stored from base_address_of_mem_coe + x * 16 (in bytes and in terms of APB access).

As described in [Section 9.2.2](#), if BPM scheme is chosen for virtual array, an operation of demodulation is needed. The BPM demodulation (DBPM) coefficients are stored in RAM. To program the DBPM coefficients, one needs to set the memory programming knob CFG\_SYS\_MEM\_ACT to 5. After that, the shared memory space starts from base\_address\_of\_mem\_coe, which is 0xE0\_0000. Because this memory is shared for storing DoA steering vectors, CFAR MIMO combination steering vectors, and the DBPM coefficients, CFG\_DoA\_DBPM\_ADR is used to specify the offset address for DBPM coefficients.

### 9.3.2 2D DoA

To make 2D DoA work properly, follow these steps:

1. Let the hardware know which mode is chosen for doing 2D DoA.

Table 9.4: Programming Interfaces for 2D DoA

Register Name	Offset	Bit Width	Description
CFG_DoA_MODE_GRP	0xA2C	2	Selection bits for 2D DoA mod. 0 => normal mode; 1 => combined mode; 2 => single shot mode.



### 9.3. Programming Interfaces

---

2. Let the hardware know which DoA engine is used to do 2D DoA;

Table 9.5: Programming Interfaces for 2D DoA

Register Name	Offset	Bit Width	Description
<b>CFG_DoA_GRP<math>k</math>_TYP_SCH</b> ( $k = 0 \dots 2$ )	0xA34+ $k$ *0x88	2	Indicates which DoA engine is chosen in Group $k$ . 0 => DBF engine; 2 => DML engine; 1 => reserved.

---

**Note:**

- If normal mode is chosen and DML engine is used, DoA estimation can only be done for either the horizontal or elevation angle in a single frame. In other words, it is 1D DoA.
  - DML engine does not support single shot mode. In other words, if single shot mode is chosen for doing DoA, DML engine cannot be chosen.
  - 2D DoA can only work with one kind of DoA engine in a frame.
- 

3. Let the DoA engine know which channels in each group are used to form the signal vectors;

Table 9.6: Programming Interfaces for 2D DoA Signal Vectors

Register Name	Offset	Bit Width	Description
CFG_DoA_GRP $k$ _DAT_IDX $t$	0xA58 + $k$ * 0x88 + $t$ * 0x4	20	The $t$ th 4 channel index to form the signal vectors in Group $k$ , $k=0,1,2$ and $t=0 \sim 7$ . For bit mapping rule, see table <a href="#">Table 9.7</a> .

CFG\_DoA\_GRP $x$ \_DAT\_IDX $t$  ( $x = 1, 2$ ,  $t = 0, \dots, 7$ ) is an extension of CFG\_DoA\_GRP0\_DAT\_IDX $t$  ( $t = 0, \dots, 7$ ) to support 2D DoA and it has the same bit mapping rule with CFG\_DoA\_GRP0\_DAT\_IDX $t$  ( $t = 0, \dots, 7$ ).

As for CFG\_DoA\_GRP0\_DAT\_IDX $t$  ( $t = 0, \dots, 7$ ), the eight registers are used for saving 32 RX channels' indices. Each register can save 4 indices, with every 5 bits representing the index information of one RX channel. The channel indices are packed in the 20-bit registers CFG\_DoA\_GRP0\_DAT\_IDX $t$  ( $t = 0, \dots, 7$ ) as shown in [Table 9.7](#).

Table 9.7: Bit Mapping of CFG\_DoA\_GRP0\_DAT\_IDX $t$

Bits[19:15]	Bits[14:10]	Bits[9:5]	Bits[4:0]
RX index for the $(4t)$ th signal	RX index for the $(4t+1)$ th signal	RX index for the $(4t+2)$ th signal	RX index for the $(4t+3)$ th signal

As for how to number the antenna of a specific antenna array, you can refer to [Fig. 8.4](#).

**Note:**

- When using normal mode or single shot mode, the registers `CFG_DoA_GRP0_DAT_IDX_t` ( $t = 0, \dots, 7$ ) are used to store RX indices of *Group A* shown in [Table 9.1](#) and the registers `CFG_DoA_GRPx_DAT_IDX_t` ( $x = 1, 2, t = 0, \dots, 7$ ) are used to store RX indices of *Group B*, *Group C* respectively.
- When using combined mode, the signal vector of Group 1 to get the elevation angle comes from the inner products and is not related to some specific RX channels, so the registers `CFG_DoA_GRPx_DAT_IDX_t` ( $x = 1, 2; t = 0, \dots, 7$ ) are not used in combined mode.
- When using combined mode, the registers `CFG_DoA_GRP0_DAT_IDX_t` ( $t = 0, \dots, 7$ ) are used to store RX indices of Principal Group and C2D Groups. The RX indices of C2D Groups will be stored successively after Principal Group.

4. Program steering vectors of each group to let the engine know where to get the steering vectors (where to start and stop) of each group.

Table 9.8: Programming Interfaces for Steering Vectors of 2D DoA

Register Name	Offset	Bit Width	Description
<code>CFG_DoA_GRPk_ADR_COE</code>	<code>0xA40 + k * 0x88</code>	12	Offset address for the DBF/DML coefficients in Group k, $k=0,1,2$ .
<code>CFG_DoA_GRPk_SIZ_RNG_PKS_CRS</code>	<code>0xA48 + k * 0x88</code>	10	Total number of steering vectors in Group k when raw search step is 1, $k=0,1,2$ .
<code>CFG_DoA_GRPk_SIZ_CMB</code>	<code>0xA54 + k * 0x88</code>	5	Total channel number in Group k, $k=0,1,2$ .
<code>CFG_DoA_C2Dg_ADR_COE</code>	<code>0xBCC + (g-1) * 0x8</code>	12	Offset address for the DBF/DML coefficients in Combined Steering Vector Group g when using combined mode, $g=1,2,3$ .
<code>CFG_DoA_C2Dg_SIZ_CMB</code>	<code>0xBD0 + (g-1) * 0x8</code>	5	Total channel number in Combined Steering Vector Group g when using combined mode, $g=1,2,3$ .

As described in [Fig. 9.13](#) and [Fig. 9.14](#), there are 3 groups of steering vectors to be written into RAM. For each group, the 3 parameters `CFG_DoA_GRPk_ADR_COE`, `CFG_DoA_GRPk_SIZ_RNG_PKS_CRS`, and `CFG_DoA_GRPk_SIZ_CMB` determine the steering vectors. There is no difference between 1D DoA and 2D DoA in the programming of steering vectors. Refer to [Fig. 8.5](#) to know the programming process.

As for combined mode, there are at most 5 groups of steering vectors to be written into RAM. You can refer to [Fig. 9.15](#) to know the corresponding relationship between the interfaces and the function. It should be noted that in combined mode, `CFG_DoA_GRP2_ADR_COE`, `CFG_DoA_GRP2_SIZ_RNG_PKS_CRS` and `CFG_DoA_GRP2_SIZ_CMB` are not used and ignored. And the total number of steering vectors in different combined steering vector groups should be identical and equal the total number of steering vectors in Steering Vector Group 0.

`CFG_DoA_GRPk_SIZ_RNG_PKS_CRS` is in format of FXI(10, U). When using DBF, it should be less than 360.

5. Let the hardware know how many groups in total, in other words, how many times are needed to call the DoA engine to get the all DoA information of one target.



Table 9.9: Programming Interfaces for 2D DoA

Register Name	Offset	Bit Width	Description
CFG_DoA_NUMB_GRP	0xA30	2	<p>Depends on the 2D DoA mode.</p> <ul style="list-style-type: none"> <li>- To run 2D DoA in normal mode, CFG_DoA_NUMB_GRP should be set to be <math>N_{group} - 1</math>. Value 0 means 1D DoA.</li> <li>- To run 2D DoA in combined mode, CFG_DoA_NUMB_GRP should be set to 1.</li> <li>- To run 2D DoA in single shot mode, CFG_DoA_NUMB_GRP should be set to be <i>group index</i> (beginning from 0), because single shot mode only supports one group of channel to do DBF in a frame as mentioned in Fig. 9.14.</li> </ul>

#### 9.3.2.1 Digital Beamforming

As discussed in *Direction of Arrival (DoA) Estimators*, there are two kinds of DoA methods. In this section, we will introduce the programming interfaces of using DBF to do 2D DoA.

##### 9.3.2.1.1 Normal Mode

Alps supports at most 3 groups of channels to do DBF in a single frame for 2D DoA. So all DBF related registers mentioned in Section 8.3.3 are extended to 3 groups. The following table summarizes the registers related to using the DBF engine to solve 2D DoA in normal mode.



Table 9.10: Programming Interfaces for 2D DoA Using DBF in Normal Mode

Register Name	Offset	Bit Width	Description
CFG_DoA_GRPk_MOD_SCH	0xA38 + k * 0x88	3	DBF mode in Group k, k=0~2. Shown in <a href="#">Table 8.2</a>
CFG_DoA_GRPk_NUM_SCH	0xA3C + k * 0x88	2	Maximum object number shared in a bin in Group k, k=0~2.
CFG_DoA_GRPk_SIZ_STP_PKS_CRS	0xA4C + k * 0x88	4	Raw search step in Group k, k=0~2. Mentioned in <a href="#">Section 8.3.3.2.4</a> .
CFG_DoA_GRPk_SIZ_RNG_PKS_RFD	0xA50 + k * 0x88	6	Fine search range in Group k, k=0~2. Mentioned in <a href="#">Section 8.3.3.2.4</a> .
CFG_DoA_GRPk_THR_SNR_y	0xA7C + k * 0x88 + y * 0x4	7	SNR regime thresholds in Group k, k=0~2; y=0,1.
CFG_DoA_GRPk_SCL_POW	0xA84 + k * 0x88	10	Power scaler to determine the peak number in Group k when DBF runs in (Non-)IPM mode, k=0~2.
CFG_DoA_GRPk_SCL NOI_y	0xA88 + k * 0x88 + y * 0x4	10	Noise scaler to determine whether to stop the 1D DoA engine when SNR falls into a different SNR regime in Group k, k=0~2, y=0~2.

**Note:**

- CFG\_DoA\_GRPk\_NUM\_SCH is in the format of  $FXI(2, U)$  and can take a value from 0, 1, 2, and 3. Value 0 means that the maximum number of detected objects in a bin is 1.
- CFG\_DoA\_GRPk\_SIZ\_STP\_PKS\_CRS is in the format of  $FXI(4, U)$ . It should be programmed with the value of  $S_{raw} - 1$ .
- CFG\_DoA\_GRPk\_SIZ\_RNG\_PKS\_RFD is in the format of  $FXI(6, U)$ . It should be programmed with the value of  $R_{fine}$ .  $R_{fine}$  should be a multiple of  $S_{raw}$ . This register is effective only when DBF runs in OMP mode or IPM mode.
- CFG\_DoA\_GRPk\_SIZ\_RNG\_PKS\_CRS is in the format of  $FXI(10, U)$ . It should be programmed with the value of  $N'_{dbf} - 1$ , where  $N'_{dbf}$  is the number of steering vectors that are used to compute DBF power spectrum and  $N'_{dbf} = \frac{N_{dbf}}{S_{raw}}$ .  $N_{dbf}$  is the total number of steering vectors and it should be a multiple of  $S_{raw}$ .
- CFG\_DoA\_GRPk\_THR\_SNR\_0 and CFG\_DoA\_GRPk\_THR\_SNR\_1 are both in the format of  $FXI(7, U)$ . It is in dB scale. Alps provides these for fine tuning of CFG\_DoA\_GRPk\_SCL NOI\_y.
- CFG\_DoA\_GRPk\_SCL\_POW is in the format of  $FLR(6, 6, U, 4, U)$ . Usually this value should be less than 1. This register is effective only when DBF runs in IPM mode or Non-IPM mode.
- CFG\_DoA\_GRPk\_SCL NOI\_y is in the format of  $FLR(6, 6, U, 4, U)$ . Usually a scalar greater than 1 should be used. The greater the value is, the earlier the iteration stops.



## 9.3. Programming Interfaces

---

### 9.3.2.1.2 Single Shot Mode

Because single shot mode uses the full peak mode of DBF, only the register `CFG_DoA_GRP $k$ _MOD_SCH` ( $k = 0, 1, 2$ ) needs to be configured to 0.

### 9.3.2.1.3 Combined Mode

As discussed in [Section 9.2.4.4](#), there is no pairing issue in combined mode, so combined mode only needs 2 groups of DBF related registers mentioned in [Section 8.3.3](#). Users can refer to [Table 9.10](#) and let  $k = 0, 1$ . Note that `CFG_DoA_GRP1_NUM_SCH` can only take a value from 0 and 1, which means that the maximal objects shared in a bin is 2 in combined mode.

### 9.3.2.2 Deterministic Maximal Likelihood

This section discusses how to program DoA engine when DML is selected to perform 2D DoA.

#### 9.3.2.2.1 Normal Mode

To choose normal mode, set the register `CFG_DoA_MODE_GRP` to 0.

To choose DML for 2D DoA, set the register `CFG_DoA_GRP0_TYP_SCH` to 2.

Since in normal mode, only 1D DoA is allowed for DML, the register `CFG_DoA_NUMB_GRP` should be set to 0, and the registers `CFG_DoA_GRP1_TYP_SCH` and `CFG_DoA_GRP2_TYP_SCH` are ignored. Based on the same reason, the registers with either `CFG_DoA_GRP1_` or `CFG_DoA_GRP2_` as the prefix are ignored.

The following table summarizes the registers to enable the normal mode and DML feature. For detailed usage of DML internal registers, users can refer to [Section 8.3.4](#).

Table 9.11: Programming Interfaces for Normal Mode Using DML

Register Name	Offset	Bit Width	Description
<code>CFG_DoA_MODE_GRP</code>	0xA2C	2	Cleared to 0 to indicate normal mode
<code>CFG_DoA_NUMB_GRP</code>	0xA30	2	Cleared to 0 to indicate 1D DoA
<code>CFG_DoA_GRP0_TYP_SCH</code>	0xA34	2	Set to 2 to indicate DML
<code>CFG_DoA_GRP0_ADR_COE</code>	0xA40	12	Offset address for DML coefficients
<code>CFG_DoA_GRP0_SIZ_RNG_PKS_CRS</code>	0xA48	10	Total number of steering vectors
<code>CFG_DoA_GRP0_SIZ_CMB</code>	0xA54	5	Total channel number

#### 9.3.2.2.2 Single Shot Mode

DML does not support single shot mode.



### 9.3.2.2.3 Combined Mode

To choose combined mode, set CFG\_DoA\_MODE\_GRP to 1.

To choose DML for 2D DoA, set the register CFG\_DoA\_GRP0\_TYP\_SCH to 2.

In combined mode, Alps baseband requires two groups. So, to choose DML for 2D DoA, set both CFG\_DoA\_GRP0\_TYP\_SCH and CFG\_DoA\_GRP1\_TYP\_SCH to 2. Basically, registers with name having GRP0 as part of it is used by the first group and GRP1 by the second group.

The following table summarizes the register settings for combined mode using DML.

Table 9.12: Programming Interfaces for Combined Mode Using DML

Register Name	Offset	Bit Width	Description
CFG_DoA_MODE_GRP	0xA2C	2	Cleared to 1 to indicate combined mode
CFG_DoA_NUMB_GRP	0xA30	2	Set to 1 for combined mode
CFG_DoA_GRPk_TYP_SCH	0xA34 + k * 0x88	2	Set to 2 to indicate DML, k=0,1
CFG_DML_GRPk_SV_START	0xE04 + k * 0x28	9	FXI(9,U). The starting position for DML searching for Group k, k=0,1
CFG_DML_GRPk_SV_STP	0xE00 + k * 0x28	6	FXI(6,U). The searching step for DML engine in Group k, k=0,1
CFG_DML_GRPk_SV_END	0xE08 + k * 0x28	10	FXI(10,U). The end position for DML searching for Group k, k=0,1
CFG_DML_GRPk_DC_COE_j	0xE0C + k * 0x28 + j * 4	14	FXR(14,1,S). The coefficients used by RNE for Group k, k=0,1. The number of coefficients is 5, so j=0,1,...4.
CFG_DML_GRPk_EXTRA_EN	0xE20 + k * 0x28	1	Set to 1 to enable extra 10-DML searching as described in <a href="#">Section 8.2.2.4</a> for Group k, k=0,1
CFG_DML_GRPk_DC_COE_2_EN	0xE24 + k * 0x28	1	Set to 1 to enable using $c_2$ and $c_3$ in Inequality (8.7) for Group k, k=0,1
CFG_DML_MEM_BASE_ADR	0xE50	10	FXI(10,U). Offset address to store d and k for Group 1.

## 9.4 Limitation and Constraints

This section summarizes the limitation and constraints of Alps baseband for virtual array.

### 9.4.1 Functional Constraints

Alps has 4 TX channels and 4 RX channels and it can virtually create an 8-RX array or 16-RX array or 12-RX array (TDM). The maximal size of array is 16.



## 9.4. Limitation and Constraints

---

### 9.4.2 Memory Constraints

- Maximum MEM\_COE size to store steering vectors is  $32 * 360 = 11520$  words, where 32 is the maximum number of channels, and 360 is the maximum number of vectors in all groups.

For 1D DoA, only one group of steering vectors need to be stored in the RAM and it is hard to exceed the limit of RAM size. However, for 2D DoA in combined mode, it is possible to get at most 5 groups of steering vectors (one azimuth group, one elevation group, and three combined steering vector groups), where the number of steering vectors in the azimuth group equals that of each combined steering vector group. Especially for DML, which supports maximum 760 vectors, it is possible to exceed the limit of RAM size if there is no tradeoff against the total number of channels, reducing the number of channels).

- Maximum MEM\_RLT size to store DoA information (angle and power) of objects within the same range gate and Doppler gate is limited to storing maximal 12 copies of data.

### 9.4.3 Constraints for 2D DoA

When multiple groups are enabled in normal mode or combined mode, the groups have to be configured with the same DoA engine, either DBF or DML. The scenario where DBF and DML are configured for different groups within the same frame is not supported.

#### 9.4.3.1 Using Digital Beamforming

There is no extra limitation for each group running in 2D DoA mode with DBF compared to 1D DoA with DBF. Refer to Section 8.4.1 to learn about the limitation of 1D DoA with DBF.

Note that to run in combined mode, the following equation should be met:

$$\left( \lceil \frac{N_p}{4} \rceil \times 4 + \sum_{g=1}^{n_{c2d}} \lceil \frac{N_{c_g}}{4} \rceil \times 4 \right) \times N_{dbf}^{az} + \lceil \frac{n_{c2d} + 1}{4} \rceil \times 4 \times N_{dbf}^{ev} \leq 11520 \quad (9.5)$$

Table 9.13: Descriptions of Parameters

Parameter	Description
$N_{dbf}^{az}$	total number of steering vectors of Group 0
$N_{dbf}^{ev}$	total number of steering vectors of Group 1
$N_p$	the size of vector $\mathbf{r}_{az}$
$n_{c2d}$	the number of C2D Groups
$N_{c_1}$	the size of vector $\mathbf{c}_1$
$N_{c_2}$	the size of vector $\mathbf{c}_2$
$N_{c_3}$	the size of vector $\mathbf{c}_3$

---

#### Note:

- CFG\_DoA\_GRP0\_SIZ RNG\_PKS\_CRS should be programmed as  $\frac{N_{dbf}^{az}}{S_{raw}^{az}} - 1$ .
  - CFG\_DoA\_GRP1\_SIZ RNG\_PKS\_CRS should be programmed as  $\frac{N_{dbf}^{ev}}{S_{raw}^{ev}} - 1$ .
  - CFG\_DoA\_GRP0\_SIZ\_CMB should be programmed as  $N_p - 1$ .
  - CFG\_DoA\_GRP1\_SIZ\_CMB should be programmed as  $n_{c2d}$ .
  - CFG\_DoA\_C2Dg\_SIZ\_CMB should be programmed as  $N_{c_g} - 1$ .  $g = 1 \dots n_{c2d}$ .
- 



### 9.4.3.2 Using Deterministic Maximum Likelihood

#### 9.4.3.2.1 Memory Limitation

As mentioned in [Section 9.4.2](#), memory limitation should be taken into consideration when configuring DML.

- When configured in Normal Mode, only one group is supported for DML. The limitation would be:
  - $(\text{CFG\_DoA\_GRP0\_SIZ\_RNG\_PKS\_CRS} + 1) \times ((\text{CFG\_DoA\_GRP0\_SIZ\_CMB} + 4) \gg 2) \times 4 < 11520$
- When configured in Combined Mode:
  - The maximum value that can be configured in `CFG_DoA_GRP1_SIZ_CMB` is 3. In other words, the signal channel number in the second group (or elevation group) has to be less than or equal to 4.
  - For easy description, we rename our numerical values configured in registers from their lengthy names to shortened names as described in [Table 9.14](#). The memory constraint is defined as Inequality [\(9.6\)](#).

Table 9.14: Nick Name for Values in Registers

Numerical Value in Register	Nick Name
<code>CFG_DoA_GRP0_SIZ_CMB</code>	$L_{sv0}$
<code>CFG_DoA_GRP1_SIZ_CMB</code>	$L_{sv1}$
<code>CFG_DoA_C2D1_SIZ_CMB</code>	$L_{svC2D1}$
<code>CFG_DoA_C2D2_SIZ_CMB</code>	$L_{svC2D2}$
<code>CFG_DoA_C2D3_SIZ_CMB</code>	$L_{svC2D3}$
<code>CFG_DoA_GRP0_SIZ_RNG_PKS_CRN</code>	$N_{sv0}$
<code>CFG_DoA_GRP1_SIZ_RNG_PKS_CRN</code>	$N_{sv1}$

$$A_0 + A_1 + A_2 + A_3 + A_4 \leq 11520 \quad (9.6)$$

Where:

$$A_0 = (N_{sv0} + 1) \times ((L_{sv0} + 4) \gg 2) \times 4 \quad (9.7)$$

$$A_1 = (N_{sv0} + 1) \times ((L_{svC2D1} + 4) \gg 2) \times 4 \quad (9.8)$$

$$A_2 = L_{sv1} > 1? ((N_{sv0} + 1) \times ((L_{svC2D2} + 4) \gg 2) \times 4) : 0 \quad (9.9)$$

$$A_3 = L_{sv1} > 2? ((N_{sv0} + 1) \times ((L_{svC2D3} + 4) \gg 2) \times 4) : 0 \quad (9.10)$$

$$A_4 = (N_{sv1} + 1) \times ((L_{sv1} + 4) \gg 2) \times 4 \quad (9.11)$$

Besides steering vector memory restriction, DML uses an internal memory to store  $d$  and  $k$  as described in [Section 8.2.2.3](#). The memory can store altogether 1024  $d$  and  $k$ , and the number of  $d$  or  $k$  for a specific group is decided by the corresponding number of steering vectors. When configured in Normal Mode, the restriction is naturally satisfied due to the fact that the bit length of `CFG_DoA_GRP0_SIZ_RNG_PKS_CRN` is 10 and the maximum number of steering vector for group0 is restricted by the length of the register to 1024. However, when configured in combined mode, the criteria `CFG_DoA_GRP0_SIZ_RNG_PKS_CRN + CFG_DoA_GRP1_SIZ_RNG_PKS_CRN` should be less than or equal to  $1024-2=1022$ .

- $(\text{CFG\_DoA\_GRP0\_SIZ\_RNG\_PKS\_CRN} + \text{CFG\_DoA\_GRP1\_SIZ\_RNG\_PKS\_CRN}) \leq 1022$



### 9.4.3.2.2 Two-Step Searching

Symmetrically, 2O-DML engine uses 2 groups of parameters. There are some hidden mutual influences, and the end angle of one group must not equal the start angle of the other group in the Refined Search, which are shown in the following two formulas.

$$(start_0 + (n - \frac{1}{2}) \cdot step_0) \neq end_1 \quad (9.12)$$

$$(start_1 + (n - \frac{1}{2}) \cdot step_1) \neq end_0 \quad (9.13)$$

Where:

- $start_0$  represents the start angle of Group 0;
- $end_0$  represents the end angle of Group 0;
- $step_0$  represents the refined search range of Group 0;
- $(start_0 + (n - \frac{1}{2}) \cdot step_0)$  represents all the potential start angles of the Refined Search of Group 0;
- $start_1$  represents the start angle of Group 1;
- $end_1$  represents the end angle of Group 1;
- $step_1$  represents the refined search range of Group 1;
- $(start_1 + (n - \frac{1}{2}) \cdot step_1)$  represents all the potential start angles of the Refined Search of Group 1;
- $n$  is an integer, which can be 0, 1, 2, 3, . . . .

## 9.5 Software and Configuration Suggestions

### 9.5.1 Software and Configuration Flow

To activate the virtual array feature, one needs to configure both radio and baseband.

If the planar array is obtained, to activate 2D DoA, usually one only needs to configure baseband.

It is suggested that one should follow these steps to design the firmware:

1. Configure the radio part according to [Section 4](#) to generate the transmit pattern of TX.

There are 3 kinds of register bits in the radio part for virtual array, which indicate:

- transmit period
- transmit pattern
- control bit for virtual array mode

2. Configure the baseband to get the virtualized planar array.

3. Configure the baseband to enable 2D DoA.

The choice of the 2D DoA mode should depend on the applications.

- **Normal mode vs single shot mode:** Normal mode gives maximal 4 angles along each direction and there are maximum 3 directions (azimuth, elevation, and an aided direction) in a frame, while single shot mode gives maximal 12 angles along only one of the 3 directions in a frame. Because single shot mode uses the full peak mode of DBF, which does not need iteration, single shot mode works faster than normal mode.

- **Normal mode vs combined mode:** In normal mode, when multiple objects exist, extra DoA estimation along an aided direction is needed to solve pairing issue. In combined mode, however, the extra DoA estimation is not needed. If the pairing issue is not considered, take the planar array shown in Fig. 9.12 for example, combined mode will occupy more memory than normal mode.
4. Program signal vectors and steering vectors.
  5. Configure DoA engine related registers.

## 9.5.2 Software Suggestions for DoA Related Registers

### 9.5.3 Digital Beamforming

For each group running for 2D DoA mode with DBF, users can refer to Section 8.5.3 to design the firmware. Note that since elements of steering vectors of each group are written continuously, the starting address (CFG\_DoA\_GRP<sub>k</sub>\_ADR\_COE, ( $k = 0, 1, 2$ )) should be programmed correctly so that different groups of steering vectors will not be overwritten. It is suggested that one should configure the CFG\_DoA\_GRP<sub>k</sub>\_ADR\_COE as follows:

```
CFG_DoA_GRPk_ADR_COE = CFG_DoA_GRP(k-1)_ADR_COE + int((CFG_DoA_GRPk_SIZ_CMB +4)/4) * 4
* (CFG_DoA_GRPk_SIZ_RNG_PKS_CRS +1)
```

#### Note:

- When the vector size (CFG\_DoA\_GRP<sub>k</sub>\_SIZ\_CMB +1 ( $k = 0, 1, 2$ )) is less than multiply of 4, the memory should be extended to multiply of 4 by filling 0 as the added steering vector.
- CFG\_DoA\_GRP<sub>k</sub>\_ADR\_COE ( $k = 0, 1, 2$ ) should be transformed by division of 4 to satisfy the meets that APB access in terms of 16 Bytes(4 words) a time.

## 9.5.4 Deterministic Maximal Likelihood

The procedure of configuring steering vectors of DML follows exactly the same rules as that of DBF except that the angles should be so distributed within FoV that their sine function is evenly spaced as described in Section 8.5.4.1. For how/where to store the steering vectors, refer to Section 9.5.3.

When configured in combined mode, the register CFG\_DML\_MEM\_BASE\_ADDR should be configured to  $8 \cdot (N - 1)$ , where  $N$  is the number of steering vectors configured for Group 0.

## 9.6 Examples

The following example is about how to configure Alps to work under BPM virtual array scheme with 4 TX channels ( $N_{va} = 4$ ) as shown in Fig. 9.8.

Antenna pattern is shown in Fig. 9.17.



## 9.6. Examples

---

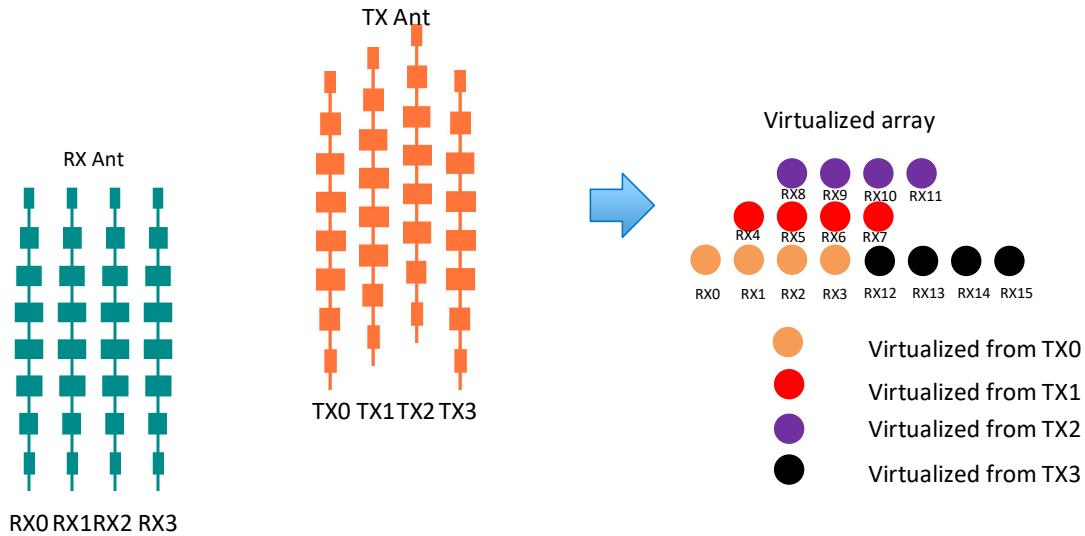


Fig. 9.17: Example of Planar Array with Virtual Array

And combined mode is used to do 2D DoA and the groups of channels are chosen as [Table 9.15](#).

**Note:** The principle to choose combined steering vector groups is that all channels in a combined steering vector group should be of the same height and there is height difference between C2D groups.

---

Table 9.15: Example of Channel Selection for 2D-DoA Using Combined mode

Parameter	Value	Description
DoA_GRP0_FFT_MUX	0xF00F	RX0~RX3 and RX12~RX15 are chosen to do azimuth angle estimation
DoA_C2D_GRP1_FFT_MUX	0x00F0	RX4~RX7 are chosen to generate a projection value on azimuth angle
DoA_C2D_GRP2_FFT_MUX	0x0F00	RX8~RX11 are chosen to generate a projection value on azimuth angle

The coefficient memory shared for storing the DoA steering vectors, CFAR MIMO combination steering vectors, and the BPM demodulation (DBPM) coefficients is arranged as shown in [Fig. 9.18](#).  $N_1$  denotes the maximal number of CFAR MIMO combination steering vectors and it is  $16 * 16 = 256$ .  $M$  denotes the maximal number of DBPM coefficients and it is 16. If DBF engine is used to do 2D DoA, the total number of steering vectors in Group 0 (to get azimuth angle) is 360 and the total number of steering vectors in Group 1 (to get elevation angle) is 180.

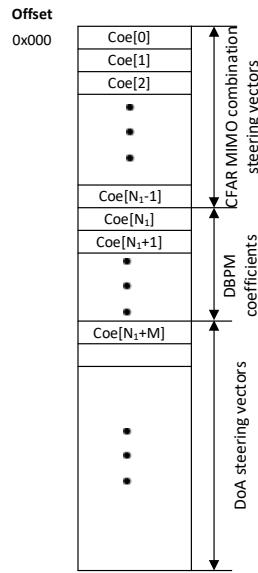


Fig. 9.18: Example of Memory Layout of Coefficient Memory

The following sub-sections describe the programming procedures in detail.

### 9.6.1 Configuring Radio Part

Program the radio part to enable the BPM MIMO scheme for virtual array. Table 9.16 summarized the radio settings.

Table 9.16: Radio Settings to Enable BPM MIMO

Register Name	Offset	Bit Width	Item	Default Value	Value
RADIO_BK5_FMCW_TX0_CTRL_1_1	0x3F	8	VAM patten	0	b'01010101
RADIO_BK5_FMCW_TX1_CTRL_1_1	0x42	8	VAM patten	0	b'01100110
RADIO_BK5_FMCW_TX2_CTRL_1_1	0x45	8	VAM patten	0	b'01011010
RADIO_BK5_FMCW_TX3_CTRL_1_1	0x48	8	VAM patten	0	b'01101001
RADIO_BK5_FMCW_TXY_CTRL_1_2	0x40 + y * 0x3	8	VAM period	Bits[2:1]: 0	3
RADIO_BK5_FMCW_TXY_CTRL_1_2	0x40 + y * 0x3	8	VAM EN	Bit[0]: 0	1
RADIO_BK5_FMCW_TXY_CTRL_1_2	0x40 + y * 0x3	8	VAM TX on/off time	Bit[3]: 1	0

## 9.6. Examples

---

### 9.6.2 Configuring Baseband for Virtual Array

Program the Baseband part to get the virtualized array. Table 9.17 summarizes the baseband settings.

Table 9.17: Baseband Settings to Enable BPM MIMO

Register Name	Offset	Bit Width	Default Value	Value
CFG_SYS_SIZE_BPM	0x34	2	0	3
CFG_FFT_DBPM_ENA	0x608	1	0	1
CFG_DoA_DBPM_ENA	0xA1C	1	0	1
CFG_DoA_DBPM_ADR	0xA20	12	0	256/4

Set the memory programming knob CFG\_SYS\_MEM\_ACT to 5. After that, program the DBPM coefficients starting from the offset address CFG\_DoA\_DBPM\_ADR as the figure shown in Fig. 9.19.

0x100	0x07ffc000
0x101	0x07ffc000
0x102	0x07ffc000
0x103	0x07ffc000
0x104	0x07ffc000
0x105	0x08000000
0x106	0x07ffc000
0x107	0x08000000
0x108	0x07ffc000
0x109	0x07ffc000
0x10A	0x08000000
0x10B	0x08000000
0x10C	0x07ffc000
0x10D	0x08000000
0x10E	0x08000000
0x10F	0x07ffc000

Fig. 9.19: Example of Memory Layout of DBPM Coefficients

### 9.6.3 Configuring Baseband for 2D DoA

Program the Baseband part to choose which mode for 2D DoA.

For settings for normal mode, see summary in Table 9.18.

Table 9.18: Example for 2D DoA in Normal Mode

Register Name	Offset	BiW	Default Value	Value
CFG_DoA_MODE_GRP	0xA2C	2	0	0
CFG_DoA_NUMB_GRP	0xA30	2	0	2

For settings for combined mode, see summary in Table 9.19.

Table 9.19: Example for 2D DoA in Combined Mode

Register Name	Offset	BiW	Default Value	Value
CFG_DoA_MODE_GRP	0xA2C	2	0	1
CFG_DoA_NUMB_GRP	0xA30	2	0	1



### 9.6.4 Configuring Signal Vectors

Regarding Table 9.7, use the following C code to translate the parameter DoA\_\*\_FFT\_MUX to signal vector related registers.

```

1  uint16_t sigvect_gn(uint32_t mux, uint32_t reg_value)
2  {
3      cnt = 0;
4      uint32_t antidx[32];
5      for (int i = 0; i < 32; i++) {
6          antidx[i] = 0;
7          if ((mux >> i) & 0x1) {
8              antidx[cnt] = i;
9              cnt++;
10         }
11     }
12     for (int i = 0; i < 8; i++)
13         reg_value[i] = (antidx[4*i + 0] << (3 * DoA_ANT_IDX_BW)) | (antidx[4*i + 1] <<_
14         ↪ (2 * DoA_ANT_IDX_BW)) | (antidx[4*i + 2] << DoA_ANT_IDX_BW) | antidx[4*i + 3];
15     return cnt; //channel number
}

```

Table 9.20 summarizes the settings for signal vector related registers.

Table 9.20: Example Settings for Signal Vectors of 2D DoA in Combined Mode

Register Name	Offset	Bit Width	Default value	Value
CFG_DoA_GRP0_DAT_IDX_0	0xA58	20	0	0x000000443
CFG_DoA_GRP0_DAT_IDX_1	0xA5C	20	0	0x000635cf
CFG_DoA_GRP0_DAT_IDX_2	0xA60	20	0	0x000214c7
CFG_DoA_GRP0_DAT_IDX_3	0xA64	20	0	0x0004254b
CFG_DoA_GRP0_DAT_IDX_4	0xA68	20	0	0
CFG_DoA_GRP0_DAT_IDX_5	0xA6C	20	0	0
CFG_DoA_GRP0_DAT_IDX_6	0xA70	20	0	0
CFG_DoA_GRP0_DAT_IDX_7	0xA74	20	0	0

---

**Note:** It can be learned from Table 9.15 that 16 channels are selected, so registers CFG\_DoA\_GRP0\_DAT\_IDX\_t ( $t = 4, \dots, 7$ ) are configured to 0.

---

### 9.6.5 Configuring Steering Vectors

The total number of steering vectors of Group 0 (to get azimuth angle) is 360 and the total number of steering vectors of Group 1 (to get elevation angle) is 180. We use OMP mode of DBF for estimation of azimuth angle and Non-IPM mode for estimation of elevation angle. The raw search step of OMP mode  $S_{raw}$  is 4. Table 9.21 summarizes the settings for steering vectors related registers.



## 9.6. Examples

---

Table 9.21: Example Settings for Steering Vectors of 2D DoA in combined mode

Register Name	Offset	Bit Width	Value	Comment
CFG_DoA_GRP0_ADR_COE	0xA40	12	$(256 + 16)/4$	BPM + MIMO-CFAR
CFG_DoA_GRP0_SIZ RNG_PKS_CRS	0xA48	10	$89(360/4 - 1)$	$\frac{N_{dbf}}{S_{raw}} - 1$
CFG_DoA_GRP0_SIZ_CMB	0xA54	5	7	$8 - 1 = 7$
CFG_DoA_GRP1_ADR_COE	0xAC8	12	$(256 + 16 + 8 * 360)/4$	(grp0_coe + grp0_len)
CFG_DoA_GRP1_SIZ RNG_PKS_CRS	0xAD0	10	179	$180 - 1 = 179$
CFG_DoA_GRP1_SIZ_CMB	0xADC	5	2	$3 - 1 = 2$
CFG_DoA_C2D1_ADR_COE	0xBCC	12	$(256 + 16 + 8 * 360 + 4 * 180)/4$	(grp1_coe + grp1_len)
CFG_DoA_C2D1_SIZ_CMB	0xBD0	5	3	$4 - 1 = 3$
CFG_DoA_C2D2_ADR_COE	0xBD4	12	$(256 + 16 + 8 * 360 + 4 * 180 + 4 * 360)/4$	(c2d1_coe + c2d1_len)
CFG_DoA_C2D2_SIZ_CMB	0xBD8	5	3	$4 - 1 = 3$

Set the memory programming knob `CFG_SYS_MEM_ACT` to 5. After that, program the steering vectors of each group as described in [Section 8.5.2](#) and store them starting from the corresponding offset address.

### 9.6.6 Configuring for Digital Beamforming

If we choose DBF as the DoA engine to do 2D DoA, we need set `CFG_DoA_GRPk_TYP_SCH` ( $k = 0, 1$ ) to 0, and then DBF related registers are required to be configured.

In this scenario, we use OMP mode of DBF for estimation of azimuth angle and Non-IPM mode for estimation of elevation angle. The raw search step of OMP mode  $S_{raw}$  is 4 and  $R_{fine}$  is 20.  $O_{max}$  for azimuth angle is 4.  $O_{max}$  for elevation angle is 2.

[Table 9.22](#) summarizes the settings for DBF related registers.



Table 9.22: Example Settings for Combined Mode Using DBF

Register Name	Offset	Bit Width	Value	Comment
CFG_DoA_GRP0_MOD_SCH	0xA38	3	4	OMP
CFG_DoA_GRP0_NUM_SCH	0xA3C	2	3	$O_{\max} - 1 = 3$
CFG_DoA_GRP0_SIZ RNG PKS_RFD	0xA50	6	20	$R_{fine} = 20$
CFG_DoA_GRP0_THR_SNR_0	0xA7C	7	15	medium SNR threshold
CFG_DoA_GRP0_THR_SNR_1	0xA80	7	30	high SNR threshold
CFG_DoA_GRP0_SCL_POW	0xA84	10	0	not used in OMP mode
CFG_DoA_GRP0_SCL NOI_0	0xA88	10	0x283	When in low SNR regime, the iteration stops if the total power is less than 5 times the noise level.
CFG_DoA_GRP0_SCL NOI_1	0xA8C	10	0x282	When in medium SNR regime, the iteration stops if the total power is less than 10 times the noise level.
CFG_DoA_GRP0_SCL NOI_2	0xA90	10	0x281	When in high SNR regime the iteration stops if total power is less than 20 times the noise level.
CFG_DoA_GRP1_MOD_SCH	0xAC0	3	1	Non-IPM
CFG_DoA_GRP1_NUM_SCH	0xAC4	2	0	$O_{\max} - 1 = 1$
CFG_DoA_GRP1_SIZ_STP_PKS_CRS	0xAD4	4	0	not used in Non-IPM mode
CFG_DoA_GRP1_SIZ RNG PKS_RFD	0xAD8	6	0	not used in Non-IPM mode
CFG_DoA_GRP1_THR_SNR_0	0xB04	7	15	medium SNR threshold
CFG_DoA_GRP1_THR_SNR_1	0xB08	7	30	high SNR threshold
CFG_DoA_GRP1_SCL_POW	0xB0C	10	0x349	A peak is counted if its power is bigger than 0.1 times the maximal peak
CFG_DoA_GRP1_SCL NOI_0	0xB10	10	0x283	see CFG_DoA_GRP0_SCL NOI_0
CFG_DoA_GRP1_SCL NOI_1	0xB14	10	0x282	see CFG_DoA_GRP0_SCL NOI_1
CFG_DoA_GRP1_SCL NOI_2	0xB18	10	0x281	see CFG_DoA_GRP0_SCL NOI_2

### 9.6.7 Deterministic Maximal Likelihood

While Section 8.6.2 gives an example of how to configure DML in normal mode, this section gives an example of programming DML in combined mode. The basic system configuration is listed as:

- The planar array as shown in Fig. 9.17 is assumed. TX0 and TX3 are virtualized to form a 1T8R array with the maximum aperture  $D$  hypothetically equals  $10\lambda$ .
- Channel selection for azimuth and elevation angles is defined by Table 9.15.
- CFAR: SISO Combination
- Virtual Array Mode: TDM



## 9.6. Examples

---

- For azimuth, the angle range to be detected is defined as:  $[\theta_L, \theta_R] = [-\frac{\pi}{6}, \frac{\pi}{6}]$ . The number of steering vector is defined as  $N_0 = 300$ . Use RNE to detect the number of reflectors in this dimension. Based on the principle described in [Section 8.5.4.4](#), the *step* parameter is set as 15. The parameter  $K$  in Equation (8.11) is set as 30.
- For elevation angle, the range to be detected is define as  $[\theta_D, \theta_U] = [-\frac{\pi}{12}, \frac{\pi}{12}]$ . The number of steering vector is defined as  $N_1 = 64$ . The number of reflectors in this dimension is fixed as 1.

Then relevant registers are set as tables from [Table 9.23](#) to [Table 9.27](#).

**Table 9.23: Baseband Settings to Enable MIMO**

Register Name	Offset	Bit Width	Default Value	Value and description
CFG_SYS_SIZE_BPM	0x34	2	0	3 ( $N_{va} = 4$ )
CFG_DoA_DBPM_ENA	0xA1C	1	0	0 (TDM)
CFG_DoA_DBPM_ADR	0xA20	12	0	0 (TDM)

**Table 9.24: Baseband Settings for 2D DoA in Combined Mode**

Register Name	Offset	Bit Width	Default Value	Value
CFG_DoA_MODE_GRP	0xA2C	2	0	1 (combined mode)
CFG_DoA_NUMB_GRP	0xA30	2	0	1 (combined mode)

**Table 9.25: Signal Vector Settings**

Register Name	Offset	Bit Width	Default Value	Value
CFG_DoA_GRP0_DAT_IDX_0	0xA58	20	0	b'00000,00001,00010,00011
CFG_DoA_GRP0_DAT_IDX_1	0xA5C	20	0	b'01100,01101,01110,01111
CFG_DoA_GRP0_DAT_IDX_2	0xA60	20	0	b'00100,00101,00110,00111 (D2D1)
CFG_DoA_GRP0_DAT_IDX_3	0xA64	20	0	b'01000,01001,01010,01011 (D2D2)
CFG_DoA_GRP0_DAT_IDX_4	0xA68	20	0	0
CFG_DoA_GRP0_DAT_IDX_5	0xA6C	20	0	0
CFG_DoA_GRP0_DAT_IDX_6	0xA70	20	0	0
CFG_DoA_GRP0_DAT_IDX_7	0xA74	20	0	0

---

**Note:** Registers  $CFG\_DoA\_GRP1\_DAT\_IDX\_t$   $t = 0, 1, \dots, 7$  are all cleared to 0. Registers  $CFG\_DoA\_GRP2\_DAT\_IDX\_t$   $t = 0, 1, \dots, 7$  are all cleared to 0.

---



Table 9.26: Steering Vector Settings for Combined Mode

Register Name	Offset	Bit Width	Default Value	Value
CFG_DoA_GRP0_ADR_COE	0xA40	12	0	0 (TDM + SISO - CFAR)
CFG_DoA_GRP0_SIZ RNG_PKS_CRS	0xA48	10	0	299
CFG_DoA_GRP0_SIZ_CMB	0xA54	5	0	7
CFG_DoA_GRP1_ADR_COE	0xAC8	12	0	8*4*300
CFG_DoA_GRP1_SIZ RNG_PKS_CRS	0xAD0	10	0	63
CFG_DoA_GRP1_SIZ_CMB	0xADC	5	0	2
CFG_DoA_C2D1_ADR_COE	0xBCC	12	0	8*4*300 + 4*4*64
CFG_DoA_C2D1_SIZ_CMB	0xBD0	5	0	3
CFG_DoA_C2D2_ADR_COE	0xBD4	12	0	8*4*300 + 4*4*64 + 4*4*300
CFG_DoA_C2D2_SIZ_CMB	0xBD8	5	0	3

Table 9.27: DML Register Settings for Combined Mode

Register Name	Offset	Bit Width	Default Value	Value
CFG_DML_GRP0_SV_START	0xE04	9	0	0
CFG_DML_GRP0_SV_STP	0xE00	6	0	15
CFG_DML_GRP0_SV_END	0xE08	10	0	299
CFG_DML_GRP1_SV_START	0xE2C	9	0	0
CFG_DML_GRP1_SV_STP	0xE28	6	0	4
CFG_DML_GRP1_SV_END	0xE30	10	0	8
CFG_DML_GRP1_DC_COE_0	0xE34	14	0	numerical value for $c_0$ of Group 0
CFG_DML_GRP1_DC_COE_1	0xE38	14	0	numerical value for $c_1$ of Group 0
CFG_DML_GRP1_DC_COE_2	0xE3C	14	0	numerical value for $c_2$ of Group 0
CFG_DML_GRP1_DC_COE_3	0xE40	14	0	numerical value for $c_3$ of Group 0
CFG_DML_GRP1_DC_COE_4	0xE44	14	0	numerical value for $c_4$ of Group 0
CFG_DML_GRP0_EXTRA_EN	0xE20	1	0	1
CFG_DML_GRP0_DC_COE_2_EN	0xE24	1	0	1
CFG_DML_GRP1_EXTRA_EN	0xE48	1	0	1
CFG_DML_GRP1_DC_COE_2_EN	0xE4C	1	0	0
CFG_DoA_GRP0_NUM_SCH	0xA3C	2	0	2 (RNE detected object number dynamically)
CFG_DoA_GRP1_NUM_SCH	0xAC4	2	0	0 (Object number is fixed to 1)
CFG_DML_MEM_BASE_ADDR	0xE50	10	0	299 * 8

**Note:**

- Since for the elevation direction (Group 1), the RNE output is fixed to 1, which means the output from 1O-DML is taken, The *start*, *step* and *end* parameter for 2O-DML for this direction is not relevant. Setting [0,4,8] to [*start*, *step*, *end*] is to eliminate the running time of 2O-DML.
- $c_0, c_1, \dots, c_4$  for the elevation direction (Group 1) is not relevant.



## 9.6. Examples

The following sample code gives an example of generating steering vectors in Group 0. The same rules applies to generating steering vectors in Group 1, Combined Steering Vector Group 1, and Combined Steering Vector Group 2.

```
/* ----- Programming Steering Vectors begins ----- */

/* Steering Vector Number for group 0 */
write_reg(CFG_DoA_GRP0_SIZ_RNG_PKS_CRS, 299);
/* Steering Vector Size for group 0 */
write_reg(CFG_DoA_GRP0_SIZ_CMB, 7);

/* TDM and SISO-CFAR, the memory is occupied by Steering Vector, so
   the Starting offset for steering vector is zero */
write_reg(CFG_DoA_GRP0_ADR_COE, 0);

U32 * SV_Offset = 0xE00000;
U32 SV_Tracker = 0;

write_reg(CFG_SYS_MEM_ACT, 5); /* selecting DoA SV memory */

float sin_theta = SIN_L; /* SIN_L is defined as -sin(pi/6) and SIN_R is sin(pi/
←6) */
float sin_step = (SIN_R - SIN_L) / 299;

/* generating steering vector for group 0 */
for (int sv_idx = 0; sv_idx < 300; sv_idx++) {
    int idx_Tx[8] = {0, 0, 0, 0, 3, 3, 3, 3};
    for (int ele_idx = 0; ele_idx < 8; ele_idx++) {
        float real = cos(2 * pi * (Rx_ant_position_X[ele_idx % 4] + Tx_ant_
→position_X[idx_Tx[ele_idx]]) * sin_theta);
        float imag = sin(2 * pi * (Rx_ant_position_X[ele_idx % 4] + Tx_ant_
→position_X[idx_Tx[ele_idx]]) * sin_theta);

        complex_t sv_elem = complex_t(real, imag);
        write_reg(SV_Offset + SV_Tracker, complex_to_cfx(&sv_elem, 14, 1, true));
        SV_Tracker++;
    }
    /* The address of a steering vector should be 4 bytes alignment */
    SV_Tracker = ((SV_Tracker + 3) >> 2) * 4;
    sin_theta += sin_step;
}

/* The first group occupies 4 * SV_Tracke memory, so the next group starts from 4_
→* SV_Tracke */
write_reg(CFG_DoA_GRP1_ADR_COE, 4 * SV_Tracker);
```

The following sample code gives an example of generating parameter lists  $d$  and  $k$  in Group 0 as described in Section 8.3.4.2.

```
/* Programming d and k */
U64 * dk_Offset = 0xE00000;
write_reg(CFG_SYS_MEM_ACT, 9); /* selecting DML dk memory */

U64 d;
U64 k;

U64 dk_comb;

/* d,k for group0 */
```

(continues on next page)



(continued from previous page)

```

/* SIN_L is defined as -sin(pi/6) and SIN_R is sin(pi/6) */
sin_step = (SIN_R - SIN_L) / (300 - 1);
for (int dk_idx = 0; dk_idx < 300 - 1; dk_idx++) {
    float real_sum = 0;
    float imag_sum = 0;

    int idx_Tx[8] = {0, 0, 0, 0, 3, 3, 3, 3};

    for (int ch_idx = 0; ch_idx < 8; ch_idx++) {
        real_sum += cos(2 * pi * (Rx_ant_position_X[ch_idx % 4] + Tx_ant_position_
→X[idx_Tx[ch_idx]]) * (dk_idx + 1) * sin_step);
        imag_sum += sin(2 * pi * (Rx_ant_position_X[ch_idx % 4] + Tx_ant_position_
→X[idx_Tx[ch_idx]]) * (dk_idx + 1) * sin_step);
    }

    complex_t d_cplx = complex_t(real_sum, imag_sum);
    d = u64_complex_to_cfl(&d_cplx, 18, 1, true, 5, true);

    /* d_amp = d * conj(d) */
    float d_amp = get_complex_power(d_cplx);

    float g = 1;
    if (dk_idx < K_CALC_KLIST) {
        g = 0.8 + 0.2 * (dk_idx) / K_CALC_KLIST;
    }

    float k_flp = g / (8 * 8 - d_amp);
    k = u64_float_to_flr(k_flp, 18, 1, false, 5, true);

    dk_comb = k | d << 18;

    dk_Offset++ = dk_comb;
}

/* d,k for group1 */
write_reg(CFG_DML_MEM_BASE_ADDR, 8 * (NUM_SV_0 - 1));

```



## **9.6. Examples**

---



## AUTO GAIN CONTROL (AGC)

### 10.1 Overview

On the field, there are many wacky situations, such as large reflectors close to radar modules, interference from other devices or other radar systems. One common impact is saturation of radar RX channels. If RX channels are saturated, radar system is basically “blind”. Thus, baseband cannot work well. In particular, if the saturation is caused by interference, any interference avoidance mechanism will not work, either. Auto gain control (AGC) is designed for countering these types of situations.

#### 10.1.1 Features

- Basic AGC
  - Programmable gain table is supported that covers LNA/TIA/VGA1/VGA2
  - Independent AGC between different RX RF chains
- AGC align
- ADC compensation
- AGC IRQs: AGC IRQs are used to inform Error Management Unit (EMU) of the status of ADC data.

### 10.2 Functional Description

#### 10.2.1 Basic AGC

Generally, when AGC mode is enabled, three additional chirps will be added in front of normal chirps for AGC purpose.

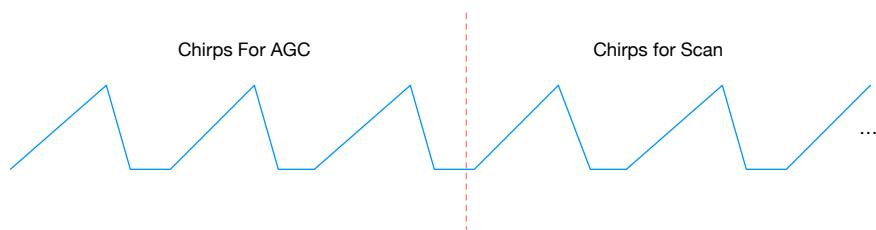


Fig. 10.1: FMCW Frequency with AGC Mode On

## 10.2. Functional Description

---

For each RX RF chain, there are three saturation detectors, which monitor the outputs of TIA, VGA1, and VGA2. All detectors have level-outputs. High-level output indicates that the output signal exceeds the upper threshold. Low-level output indicates that the output signal does not reach the lower threshold. See Fig. 10.2. During the first two chirps, baseband adjusts the RX gain based on the outputs of these saturation detectors. During the third chirp, baseband computes the “best” RX gain based on ADC outputs. See Fig. 10.1.

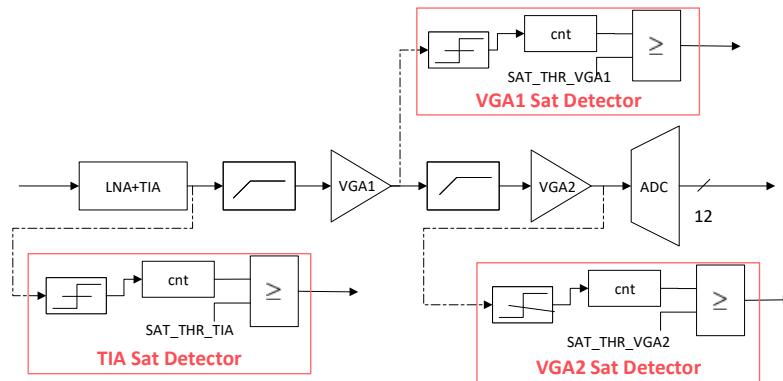


Fig. 10.2: RX Chain Block and Its Saturation Detectors

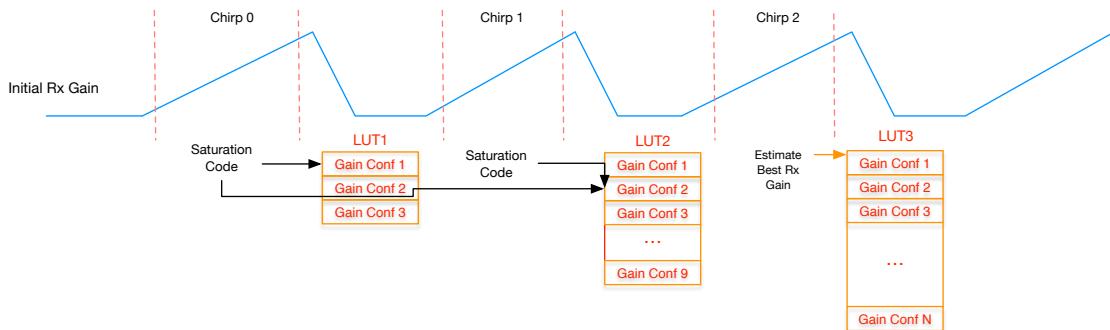


Fig. 10.3: Detailed Mechanisms of AGC

As shown in Fig. 10.3, at the beginning of the frame, each stage of RX chain is set to a default value, which is called *init gain* (programmable). During the first chirp, baseband samples all 3 detectors for each RX chain. To prevent glitches, baseband claims saturation only when the total number of samples of high-level exceeds certain thresholds  $\text{AGC\_SAT\_THR\_TIA}$ ,  $\text{AGC\_SAT\_THR\_VGA1}$ , and  $\text{AGC\_SAT\_THR\_VGA2}$  (programmable). At the end of the first chirp, baseband has 3-bit saturation status for each RX chain. Based on this, baseband chooses a gain setting for the next chirp. The gain is chosen based on the look-up-table (LUT) shown in Table 10.1.

Table 10.1: Gain Settings for First Chirp

TIA	VGA1	VGA2	Gain Setting
not saturated	not saturated	not saturated	init gain
saturated	X	X	entry 0
not saturated	saturated	X	entry 1
not saturated	not saturated	saturated	entry 2

During the second chirp, baseband observes the three detectors for each RX chain again to determine the RX gain for

the second chirp. This time, the LUT is larger since it is based on the saturation status of both the first chirp and the second chirp. See summary in [Table 10.2](#).

Table 10.2: Gain Settings for Second Chirp

First Chirp			Second Chirp			Gain Setting
TIA	VGA1	VGA2	TIA	VGA1	VGA2	
saturated	X	X	saturated	X	X	entry 0
saturated	X	X	not saturated	saturated	X	entry 1
saturated	X	X	not saturated	not saturated	saturated	entry 2
not saturated	saturated	X	saturated	X	X	entry 3
not saturated	saturated	X	not saturated	saturated	X	entry 4
not saturated	saturated	X	not saturated	saturated	saturated	entry 5
not saturated	not saturated	saturated	saturated	X	X	entry 6
not saturated	not saturated	saturated	not saturated	saturated	X	entry 7
not saturated	not saturated	saturated	not saturated	not saturated	saturated	entry 8
otherwise						gain setting from the first chirp

The purpose of the first two chirps are used to position RX RF chain so that RX does not get saturated. Therefore, programming of LUTs for the first and second chirps should reflect such intention. For example, if one detector reports saturation, the corresponding RF block should lower its gain. During the third chirp, AGC algorithm not only observes saturation detectors but also computes the ADC output power. If during the third chirp, RX chain is still saturated, AGC algorithm goes to minimal gain directly. If no saturation happens during the third chirp, ADC power is estimated to find a “best” gain by using a third LUT.

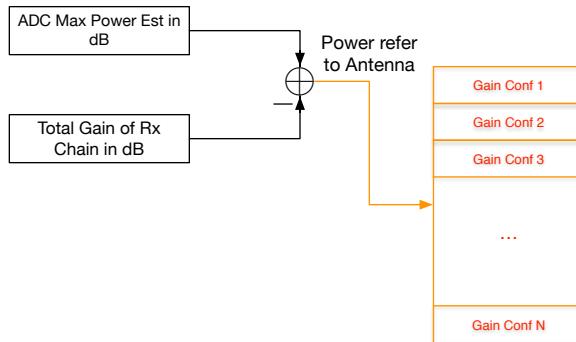


Fig. 10.4: HW Structure of Final RX Gain

As shown in Fig. 10.4, AGC algorithm estimates ADC's maximal power based on AGC\_DAT\_MAX\_SEL (programmable) and converts it to dB. We use the antenna referred input power to look up the programming table. Therefore, the final gain table should be programmed as following:

1. The first entry corresponds to the minimal input power, which can be obtained by targeted ADC level minus maximal RX gain.
  2. The last entry corresponds to the maximal input power, which can be obtained by targeted ADC level minus minimal RX Gain.
  3. Each entry is a gain setting instead of power. Sweep all possible input powers, each time with 1 dB increase. Given the input power, the total RX gain to reach the target ADC output level is determined. To meet the total gain budget, when allocating gain to each RF block, it is suggested one begin with LNA and then TIA, VGA1,

## 10.2. Functional Description

---

and VGA2, and allocate as large gain as possible. Make sure that the output power of each RF block does not trigger the saturation detector.

### 10.2.2 AGC Alignment

When the basic AGC function works, each RX RF chain controls the final gain setting independently. Generally, different RX channels may be set with different LNA/TIA/VGA1/VGA2 gains, which is OK to solve the range and velocity information of targets, but not good for solving angle information. The reason is that a gain difference of RX channels can cause an extra phase difference of RX channels, which is harmful to the estimation of DoA.

The AGC alignment feature guarantees that all RX RF chains are set with the same final gain setting, which is related to the biggest entry index of all channels before alignment and it means the lowest gain of all.

### 10.2.3 ADC Compensation

As has been introduced before, AGC algorithm refers current ADC's maximal power to antenna input power and determines the final gain settings by computing the total RX gain needed to reach the target ADC output level. Usually, the smaller the input power, the greater the gain needed. But when the input power is smaller than the minimal input power, even if the maximal gain setting (the first entry of the final gain table) is allocated, there is still a misgain between the target ADC output level and the actual ADC output level. This situation may be rare, but existent. To cope with this situation, ADC compensation feature is provided.

ADC compensation algorithm estimates the misgain between the target ADC output level and the actual ADC output level and converts it to shifters according to the *ADC Compensation Level* (programmable). When the ADC compensation feature is enabled, the ADC data will be compensated by left shifting the shifters, as shown in Fig. 10.5. It should be noted that if the misgain is negative, the shifters will be zero.

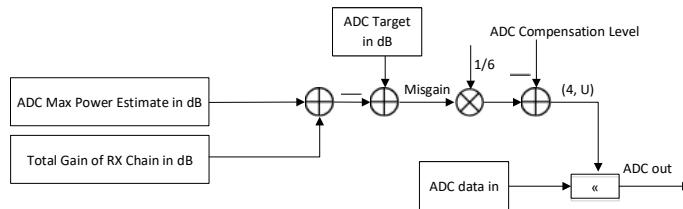


Fig. 10.5: Hardware Structure for ADC Compensation Mode

Furthermore, if the ADC alignment feature is enabled at the same time, ADC data of different RX channels will be compensated by left shifting the same shifter that is the minimum one.

### 10.2.4 AGC IRQs

AGC IRQs indicate whether ADC data is in a reasonable range, neither too large (close to saturation) nor too small (close to noise floor). AGC state has 12 IRQs, corresponding to 4 channels, with each channel having 3 IRQs. See [IRQ-Related AGC Registers](#). These 12 IRQs will perform *OR* operation and then change to one signal connected to EMU. In other words, as long as one of these 12 IRQs is raised, the signal after *OR* operation will be raised and sent to EMU.

---

**Note:** For more EMU details, refer to the EMU section in *Calterah Alps Reference Manual*.

---

AGC IRQs are not on the list of CPU IRQs, but CPU can also check the status of AGC IRQs by reading the register FDB\_AGC\_IRQ\_STATUS, which has 12 bits corresponding to the 12 IRQs.

## 10.3 Programming Interfaces

### 10.3.1 Basic AGC

The design of the AGC algorithm is simple, but compared to other features, more configuration knobs are required to be programmed. Discussion is divided into several parts.

#### 10.3.1.1 Gain Code and Gain Value

Baseband supports 2 levels of LNA setting, 4 levels of TIA setting, 6 levels of VGA1 and VGA2 settings. Therefore, gain code is packed as shown in Table 10.3.

Table 10.3: Gain Code Mapping

8	7	6	5	4	3	2	1	0
LNA	TIA	VGA1			VGA2			

The corresponding gain value is programmed in 8 bits with the format of FXR(8, 7, U) and the unit of a gain value is dB.

The code of LNA and TIA is stored in registers. See Table 10.4.

Table 10.4: Programming Interfaces for Gain Code of LNA and TIA

Register Name	Offset	Bit Width	Description
CFG_AGC_CODE_LNA_x	0x214 ~ 0x218	4	RF bits for level x of LNA, x=0, 1.
CFG_AGC_CODE_TIA_x	0x21C ~ 0x228	4	RF bits for level x of TIA, x=0, 1, 2, 3.



## 10.3. Programming Interfaces

---

### 10.3.1.2 Glitch Protection

To prevent hardware glitch, some protection is designed and shown in [Table 10.5](#). You can refer to [Fig. 10.2](#) to learn about the mechanism of saturation detectors and refer to [Fig. 10.4](#) for the usage of `CFG_AGC_DAT_MAX_SEL`.

[Table 10.5: Programming Interfaces for Saturation Detection](#)

Register Name	Offset	Bit Width	Bits	Description
<code>CFG_AGC_SAT_THR_TIA</code>	0x200	24	7:0	The sample number that high-level TIA saturation detector exceeds. Indicates a true saturation of TIA in a chirp.
<code>CFG_AGC_SAT_THR_VGA1</code>	0x204	24	7:0	The sample number that high-level VGA1 saturation detector exceeds. Indicates a true saturation of VGA1 in a chirp.
<code>CFG_AGC_SAT_THR_VGA2</code>	0x208	24	7:0	The sample number that high-level VGA2 saturation detector exceeds. Indicates a true saturation of VGA2 in a chirp.
<code>CFG_AGC_DAT_MAX_SEL</code>	0x210	2	1:0	Option to select the $n$ th largest (absolute value) ADC sample for the power estimation. $n$ can be 1, 2, 3, corresponding to the value of 0, 1, 2, respectively

For `CFG_AGC_SAT_THR_TIA`, `CFG_AGC_SAT_THR_VGA1` and `CFG_AGC_SAT_THR_VGA2` in [Table 10.5](#), the 24 bits are organized as shown in [Table 10.6](#).

[Table 10.6: Bit Mapping of 24 Bits of Saturation Registers](#)

Bits[23:16]	Bits[15:8]	Bits[7:0]
successive frame number of saturation to indicate an IRQ	successive chirp number of saturation to indicate a true saturation of a frame	sample number to indicate a true saturation of a chirp

The upper 16 bits of the saturation registers are related to IRQ triggers, which will be introduced in [Table 10.11](#).

### 10.3.1.3 Clip Tables

The first two clip tables *LUT1* and *LUT2* and initial RX gain as described in [Fig. 10.3](#) are stored in registers instead of RAM. See [Table 10.7](#).

[Table 10.7: Programming Interfaces for Init gain and First Two Clip Tables](#)

Register Name	Offset	Bit Width	Description
<code>CFG_AGC_CDGN_INIT</code>	0x0230	17	Initial gain code and its gain value
<code>CFG_AGC_CDGN_C0_x</code>	0x0234 ~ 0x023C	17	Table for Clip1 x=0, 1, 2.
<code>CFG_AGC_CDGN_C1_x</code>	0x0240 ~ 0x0260	17	Table for Clip1 x=0~8.

For each register in [Table 10.7](#), the 17 bits are organized as shown in [Table 10.8](#).

[Table 10.8: Bit Mapping of 17 Bits for LUTs and Initial Gain Code](#)

Bits[16:8]	Bits[7:0]
Gain Code	Gain Value



The final LUT *LUT3* as described in Fig. 10.3 is stored in RAM. To program the final LUT, one needs to set the memory programming knob `CFG_SYS_MEM_ACT` (0x90\_0014) to 0. After that, the LUT begins from the address 0x80\_0000. For the final LUT, each entry is organized as shown in Table 10.8.

To help determine proper final gain settings, refer to Fig. 10.4. The minimum antenna input power `CFG_AGC_GAIN_MIN` and target ADC output level `CFG_AGC_DB_TARGET` should be programmed. See Table 10.9. `CFG_AGC_DB_TARGET` can be obtained by  $20 \log_{10} 2^{\text{AGC\_ADC\_target level}}$ . `CFG_AGC_GAIN_MIN` can be obtained by targeted ADC output level (in dB) minus maximal RX gain (in dB).

Table 10.9: Programming Interfaces for Choosing Final Gain

Register Name	Offset	Bit Width	Description
<code>CFG_AGC_GAIN_MIN</code>	0x22C	10	Minimum antenna input power (in dB), with the format of: FXR(10, 9, S)
<code>CFG_AGC_DB_TARGET</code>	0x278	8	Targeted ADC output level (in dB), with the format of: FXR(8, 7, S)

#### 10.3.1.4 Monitoring-Related AGC Registers

There are also some registers that monitor AGC and store certain status of the AGC algorithm. They are not only for the debugging purpose but also useful for applications. See Table 10.10.

Table 10.10: Interfaces for Storing Final AGC Status

Register Name	Offset	Bit Width	Description
<code>FDB_AGC_COD_Cx</code>	0x284 ~ 0x290	9	Final gain code for current frame of Channel x, x=0,1,2, or 3.
<code>FDB_AGC_SAT_CNT_sss_Cx</code>	0x294 ~ 0x2C0	24	Bits[7:0]: Saturation counters for sss block of Channel x in a chirp; Bits[15:8]: Saturation counters for sss block of Channel x in a frame; Bits[23:16]: Number of saturated frames for sss block of Channel x; where sss = TIA, VGA1, or VGA2 and x = 0 or 1. The counter excludes AGC's first three chirps.
<code>FDB_AGC_DAT_MAX_yST_Cx</code>	0x2C4 ~ 0x2F0	12	The y th largest ADC absolute value of Channel x, where y=1,2, or 3 and x=0,1,2, or 3.

## 10.3. Programming Interfaces

---

### 10.3.1.5 IRQ-Related AGC Registers

There are also some registers that control when to trigger IRQ. The mechanism of AGC IRQ is like this. When AGC IRQ is enabled, the IRQ signal is given by checking the ADC buffer saturation state. If the number of frames continuously saturated in an RF block exceeds the corresponding threshold, an IRQ is triggered. Generally, when the IRQ is received, the frame counter for the saturation indicator and the IRQ status need to be cleared to recover the workflow. The programming interfaces are shown in [Table 10.11](#).

Table 10.11: Programming Interfaces for Setting AGC IRQ

Register Name	Offset	Bit Width	Description
CFG_AGC_IRQ_ENA	0x27C	12	Enable bits for IRQ
CTL_AGC_IRQ_CLR	0x280	12	Clear IRQ status
CTL_AGC_SAT_CNT_CLR_FRA	0x20C	1	Clear frame counter for saturation indicator
CFG_AGC_CHCK_ENA	0x264	1	Enable bit for checking ADC buffer saturation
CFG_AGC_SAT_THR_TIA	0x200	24	Bits[15:8]: Successive TIA saturation chirp number to indicate a true frame saturation of TIA. Bit[23:16]: Successive TIA saturation frame number to indicate an IRQ of TIA.
CFG_AGC_SAT_THR_VGA1	0x204	24	Bits[15:8]: Successive VGA1 saturation chirp number to indicate a true frame saturation of VGA1. Bit[23:16]: Successive VGA1 saturation frame number to indicate an IRQ of VGA1.
CFG_AGC_SAT_THR_VGA2	0x208	24	Bits[15:8]: Successive VGA2 saturation chirp number to indicate a true frame saturation of VGA2. Bit[23:16]: Successive VGA2 saturation frame number to indicate an IRQ of VGA2.
FDB_AGC_IRQ_STATUS	0x2F4	12	Feedback current IRQ status

For CFG\_AGC\_IRQ\_ENA in [Table 10.11](#), the 12 bits are organized as shown in [Table 10.12](#).

Table 10.12: Bit mapping rule of register CFG\_AGC\_IRQ\_ENA

Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Rx0 TIA	Rx1 TIA	Rx2 TIA	Rx3 TIA	Rx0 VGA1	Rx1 VGA1	Rx2 VGA1	Rx3 VGA1	Rx0 VGA2	Rx1 VGA2	Rx2 VGA2	Rx3 VGA2

For CTL\_AGC\_IRQ\_CLR and FDB\_AGC\_IRQ\_STATUS in [Table 10.11](#), the 12 bits are organized as shown in [Table 10.13](#).



Table 10.13: Bit mapping rule of register CTL<sub>\_</sub>AGC<sub>\_</sub>IRQ<sub>\_</sub>CLR and FDB<sub>\_</sub>AGC<sub>\_</sub>IRQ<sub>\_</sub>STATUS

Bit 11	Bit 10	Bit 9	Bit 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Rx0 TIA	Rx0 VGA1	Rx0 VGA2	Rx1 TIA	Rx1 VGA1	Rx1 VGA2	Rx2 TIA	Rx2 VGA1	Rx2 VGA2	Rx3 TIA	Rx3 VGA1	Rx3 VGA2

### 10.3.2 AGC Alignment

For AGC alignment, only one knob CFG<sub>\_</sub>AGC<sub>\_</sub>ALIGN<sub>\_</sub>EN needs to be configured. See Table 10.14.

Table 10.14: Programming Interface for AGC Alignment

Register Name	Offset	Bit Width	Description
CFG <sub>_</sub> AGC <sub>_</sub> ALIGN <sub>_</sub> EN	0x268	1	Enable bit for aligning AGC of 4 channels

### 10.3.3 ADC Compensation

As discussed in [Section 10.2.3](#), the ADC compensation algorithm converts the misgains after basic AGC and AGC alignment or just basic AGC to shifters. The final shifter is adjusted only by the parameter *ADC Compensation Level* (see [Fig. 10.5](#)). So there is only one knob related to the algorithm. Summary of registers about this feature is shown in [Table 10.15](#).

Table 10.15: Programming Interfaces for ADC Compensation

Register Name	Offset	Bit Width	Description
CFG <sub>_</sub> AGC <sub>_</sub> CMPN <sub>_</sub> EN	0x26c	1	Enable bit for ADC compensation
CFG <sub>_</sub> AGC <sub>_</sub> CMPN <sub>_</sub> ALIGN <sub>_</sub> EN	0x270	1	Enable bit for aligning shifters of 4 channels to the minimum one
CFG <sub>_</sub> AGC <sub>_</sub> CMPN <sub>_</sub> LVL	0x274	1	ADC compensation referred level

### 10.3.4 AGC IRQs

The following describes the registers about AGC IRQs.

- To enable AGC IRQs, in the register CFG<sub>\_</sub>AGC<sub>\_</sub>IRQ<sub>\_</sub>ENA, set the related bit to 1.
- To check AGC IRQs, in the register FDB<sub>\_</sub>AGC<sub>\_</sub>IRQ<sub>\_</sub>STATUS, read the related bit .
- To clear IRQs, firstly set CTL<sub>\_</sub>AGC<sub>\_</sub>SAT<sub>\_</sub>CNT<sub>\_</sub>CLR<sub>\_</sub>FRA to 1 and then in the register CTL<sub>\_</sub>AGC<sub>\_</sub>IRQ<sub>\_</sub>CLR, set the related bit to 1.

See [Setting IRQ-Related AGC](#) for more details.



### 10.3.5 AGC Coexistence with Virtual Array

As discussed in [Section 9](#), there is a specific pattern for chirps transmitted by different TX when radar works in virtual array mode. So it should be considered how to transmit the additional 3 chirps needed for AGC mode when the radar works in virtual array mode. For example, if the virtual array number is 2 and TDM is chosen, there are two options to transmit the additional 3 chirps. One is always letting one of the two TX transmitters transmit the needed chirps, and the other is letting the two TX transmitters transmit the needed chirps in turn. Refer to [Section 4](#) to get the information of programming corresponding interfaces.

## 10.4 Limitation and Constraints

- AGC supports at most 2 levels of LNA setting, 4 levels of TIA setting, 6 levels of VGA1, and 6 levels of VGA2.
- The 2 levels of LNA are 18 dB and 24 dB and it is suggested one should use the second level of LNA in the final gain table, because the higher gain, the lower noise figure.
- The 4 levels of TIA are 0 dB, 6 dB, 12 dB and 18 dB, and 6 dB is the minimum gain step.
- The 6 levels of VGA1 are 7 dB, 10 dB, 13 dB, 16 dB, 19 dB, and 22 dB, and 3 dB is the minimum gain step.
- The 6 levels of VGA2 are 5 dB, 8 dB, 11 dB, 14 dB, 17 dB, and 20 dB, and 3 dB is the minimum gain step.

So the range of the gain that AGC can provide to antenna input power is from 30 dB to 84 dB with 3-dB step.

---

**Note:** When two Alps CAL77S244-AE chips are cascaded, AGC is not supported.

---

When AGC IRQs are raised, the CPU should clear the status of IRQs. Otherwise, the IRQ status will not change for the next frame and the status will be invalid.

## 10.5 Software and Configuration Suggestions

Because of the large number of registers and RAM required to be programmed and the relationship with FMCW, it is suggested that one should follow these steps to design the firmware:

1. Configure the radio part according to [Section 4](#).
  - There are 3 kinds of registers in the radio part about AGC mode, which are reference voltage thresholds, saturation signal, and control bit for transmitting additional 3 chirps.
2. Configure the baseband part.
  - Programming gain code of LNA and TIA.
    - a. The gain values of the two levels of LNAs are 18 dB and 24 dB. And the gain codes for 18 dB and 24 dB is  $h'4$  and  $h'F$  respectively.
    - b. The gain values of the 4 levels of TIAs are 0 dB, 6 dB, 12 dB and 18 dB, and the corresponding gain codes are  $h'1$ ,  $h'2$ ,  $h'4$ , and  $h'8$ .
  - Programming for glitch protection
  - Programming initial gain and the first two clip tables
    - a. The total gain of the initial gain table should not be too weak and it's better to be medium-level.

- b. Programming of LUTs for the first and second chirps should reflect the intention that no saturation happens during the third chirp. A general principle is, if one detector reports saturation, the corresponding RF block should lower its gain. And the gain of the RF blocks before it should be lowered, too, because the current saturation can be triggered by previous stages, too.
  - Setting the final gain table.
    - a. The first entry corresponds to the minimal input power, which can be obtained by targeted ADC level minus maximal RX gain.
    - b. The last entry corresponds to the maximal input power, which can be obtained by targeted ADC level minus minimal RX Gain.
    - c. Sweep all possible input powers, each time with 1 dB increase.
    - d. Given the input power, traverse all combinations of LNA/TIA/VGA1/VGA2 meeting the total gain budget and triggering no saturation to any RF block.
    - e. Choose the combination of LNA/TIA/VGA1/VGA2 that is of as large total gain as possible.
  - Setting IRQ-related AGC
  - Configuring for AGC alignment
  - Configuring for ADC compensation
3. Configure the control bit to enable AGC.

## 10.6 Examples

The following example is about how to configure Alps to work under AGC mode. We use 2 levels of LNA setting, 4 levels of TIA setting, 6 levels of VGA1 and VGA2 mentioned in [Section 10.4](#). Summary of the gain table and gain code is shown in [Table 10.16](#). Here we set `AGC_ADC_target_level` to 10 bits.

Table 10.16: Radio Gain Setting

RF Block	Gain Table	Gain Code
LNA	[18 24]	$h'4 \rightarrow 18\text{dB}; h'F \rightarrow 24\text{dB}$
TIA	[0 6 12 18]	$h'1 \rightarrow 0\text{dB}; h'2 \rightarrow 6\text{dB}; h'4 \rightarrow 12\text{dB}; h'8 \rightarrow 18\text{dB};$
VGA1	[7 10 13 16 19 22]	$h'1 \rightarrow 7\text{dB}; h'2 \rightarrow 10\text{dB}; h'3 \rightarrow 13\text{dB}; h'4 \rightarrow 16\text{dB}; h'5 \rightarrow 19\text{dB}; h'6 \rightarrow 22\text{dB};$
VGA2	[5 8 11 14 17 20]	$h'1 \rightarrow 5\text{dB}; h'2 \rightarrow 8\text{dB}; h'3 \rightarrow 11\text{dB}; h'4 \rightarrow 14\text{dB}; h'5 \rightarrow 17\text{dB}; h'6 \rightarrow 20\text{dB};$

### 10.6.1 Configuring Radio Part

Programming radio part to enable AGC, see summary in [Table 10.17](#).



## 10.6. Examples

---

Table 10.17: Radio & RF BB Settings to Enable AGC

Register Name	Off-set	Bit Width	Item	Default Value	Value
RADIO_BK3_FMCW_CTRL_1_0	0x65	8	AGC_EN_1	Bit[4]: 0	1
RADIO_BK0_CH1_RX_PDT	0x49	8	BB_PKD_VGA	Bit[2]: 0	1
RADIO_BK0_CH1_RX_PDT	0x49	8	BB_PKD_EN	Bit[3]: 0	1
RADIO_BK0_CH1_RX_PDT	0x49	8	RF_SAT_EN	Bit[7]: 0	1
RADIO_BK0_CH1_RX_PDT	0x49	8	RF_TIA_SAT_VREF_SEL	Bits[5:4]: 3	3
RADIO_BK0_CH1_RX_PDT	0x49	8	BB_PKD_VTHSEL	Bits[1:0]: 0	0
RADIO_BK0_CH2_RX_PDT	0x57	8	BB_PKD_VGA	Bit[2]: 0	1
RADIO_BK0_CH2_RX_PDT	0x57	8	BB_PKD_EN	Bit[3]: 0	1
RADIO_BK0_CH2_RX_PDT	0x57	8	RF_SAT_EN	Bit[7]: 0	1
RADIO_BK0_CH2_RX_PDT	0x57	8	RF_TIA_SAT_VREF_SEL	Bits[5:4]: 3	3
RADIO_BK0_CH2_RX_PDT	0x57	8	BB_PKD_VTHSEL	Bits[1:0] 0	0
RADIO_BK0_CH3_RX_PDT	0x65	8	BB_PKD_VGA	[Bit2] 0	1
RADIO_BK0_CH3_RX_PDT	0x65	8	BB_PKD_EN	[Bit3] 0	1
RADIO_BK0_CH3_RX_PDT	0x65	8	RF_SAT_EN	[Bit7] 0	1
RADIO_BK0_CH3_RX_PDT	0x65	8	RF_TIA_SAT_VREF_SEL	Bits[5:4]: 3	3
RADIO_BK0_CH3_RX_PDT	0x65	8	BB_PKD_VTHSEL	Bits[1:0]: 0	0
RADIO_BK0_CH4_RX_PDT	0x73	8	BB_PKD_VGA	Bit[2]: 0	1
RADIO_BK0_CH4_RX_PDT	0x73	8	BB_PKD_EN	Bit[3]: 0	1
RADIO_BK0_CH4_RX_PDT	0x73	8	RF_SAT_EN	Bit[7]: 0	1
RADIO_BK0_CH4_RX_PDT	0x73	8	RF_TIA_SAT_VREF_SEL	Bits[5:4]: 3	3
RADIO_BK0_CH4_RX_PDT	0x73	8	BB_PKD_VTHSEL	Bits[1:0]: 0	0

## 10.6.2 Configuring Baseband Part

We divide this part into several sub-parts.

### 10.6.2.1 Configuring Gain Code

Program the gain code of LNA and TIA, as shown in Table 10.18, according to Table 10.4.

Table 10.18: AGC LNA Code and TIA Code

Register Name	Offset	Bit Width	Default Value	Value
CFG_AGC_CODE_LNA_0	0x214	4	0	4
CFG_AGC_CODE_LNA_1	0x218	4	0	15
CFG_AGC_CODE_TIA_0	0x21C	4	0	1
CFG_AGC_CODE_TIA_1	0x220	4	0	2
CFG_AGC_CODE_TIA_2	0x224	4	0	4
CFG_AGC_CODE_TIA_3	0x228	4	0	8



### 10.6.2.2 Configuring Glitch Protection

Program the thresholds to indicate saturation and pick the maximal ADC data of the third chirp to compute the final gain code, as shown in [Table 10.19](#), according to [Table 10.5](#).

Table 10.19: Thresholds to Indicate Saturation and Maximal ADC Data Selection

Register Name	Offset	Bit Width	Default Value	Value
CFG_AGC_SAT_THR_TIA	0x200	24	0	Bits[7:0]: 1
CFG_AGC_SAT_THR_VGA1	0x204	24	0	Bits[7:0]: 1
CFG_AGC_SAT_THR_VGA2	0x208	24	0	Bits[7:0]: 1
CFG_AGC_DAT_MAX_SEL	0x210	2	0	0

### 10.6.2.3 Configure Initial Gain and First Two Clip Tables

Following we will give a detail example to show how to configure the registers related to initial gain and the first two clip tables.

1. Initial gain table is chosen to be [1,2,4,4], that is LNA is set to be 24dB, TIA is set to be 12dB, VGA1 is 19dB and VGA2 is 17dB. The total gain of this setting is 72dB. The initial gain should be a relatively large gain since we have two chances back off the radio gains.
2. The first clip table mentioned in [Table 10.1](#) is set as following:
  - Entry 0: [1, 0, 2, 2]. The reasoning behind is that AGC choosing this entry only when saturation happens at TIA. Therefore it is reasonable to back off TIA's gain first.
  - Entry 1: [1, 1, 0, 2]. Similarly, selecting this gain implicates that saturation happens at VGA1 but not TIA. Therefore, VGA1's gain need to be backed off.
  - Entry 2: [1, 1, 0, 0]. This follows similar reasoning for VGA2.
3. The second clip table mentioned in [Table 10.2](#) can be programmed as following. It is based on the similar reasoning as programming of first clip table.
  - Entry 0: [0, 0, 1, 2]
  - Entry 1: [1, 0, 0, 2]
  - Entry 2: [1, 0, 2, 0]
  - Entry 3: [1, 0, 0, 2]
  - Entry 4: [1, 0, 0, 2]
  - Entry 5: [1, 1, 0, 0]
  - Entry 6: [1, 0, 0, 0]
  - Entry 7: [1, 0, 0, 0]
  - Entry 8: [1, 1, 0, 0]

Regarding to [Table 10.8](#), we can use the following C code to translate initial gain and the first two clip tables to gain code related registers.

```

1 uint32_t agc_cdgn(lna_gain_tbl[], tia_gain_tbl[], v1_gain_tbl[], v2_gain_tbl[], table)
2 {
3     //table = {lna_gain_lvl, tia_gain_lvl, v1_gain_lvl, v2_gain_lvl}
4     uint32_t gain = 0;

```

(continues on next page)



## 10.6. Examples

(continued from previous page)

```

5   uint32_t cod = 0;
6   gain = (uint32_t)lna_gain_tbl[table[0]] + (uint32_t)tia_gain_tbl[table[1]] ;
7   gain = gain + v1_gain_tbl[table[2]] + v2_gain_tbl[table[3]];
8   gain = gain * 2; //GAIN_FXR(8, 7, U)
9   cod = (table[3]+1) | ((table[2]+1)<<VGA_BiW) | (table[1]<<(2 * VGA_BiW)) | _  

10  ↳(table[0]<<(2 * VGA_BiW + TIA_BiW));
11  cod = (cod << GAIN_BiW) + gain;
12  return cod;
}

```

Summary to configure the initial gain/clip table to gain code related registers is shown in Table 10.20.

Table 10.20: AGC Initial Gain Code and Clip1/Clip2 Code

Register Name	Offset	Bit Width	Default Value	Value(hex)
CFG_AGC_CDGN_INIT	0x230	17	0	h'1ad90
CFG_AGC_CDGN_C0_0	0x234	17	0	h'11b60
CFG_AGC_CDGN_C0_1	0x238	17	0	h'14b60
CFG_AGC_CDGN_C0_2	0x23C	17	0	h'14954
CFG_AGC_CDGN_C1_0	0x240	17	0	h'0134e
CFG_AGC_CDGN_C1_1	0x244	17	0	h'10b54
CFG_AGC_CDGN_C1_2	0x248	17	0	h'11954
CFG_AGC_CDGN_C1_3	0x24C	17	0	h'10b54
CFG_AGC_CDGN_C1_4	0x250	17	0	h'10b54
CFG_AGC_CDGN_C1_5	0x254	17	0	h'14954
CFG_AGC_CDGN_C1_6	0x258	17	0	h'10948
CFG_AGC_CDGN_C1_7	0x25C	17	0	h'10948
CFG_AGC_CDGN_C1_8	0x260	17	0	h'14954

### 10.6.2.4 Setting Final Gain Table

Program the final LUT to RAM. How to design the final gain level table?

The following pseudo C code is an example to generate a final gain level table:

```

1 min_power_db = adc_target_db - max_rx_gain;
2 max_power_db = adc_target_db - min_rx_gain;
3 for (int p = min_power_db; p <= max_power_db; p = p + 1) {
4     mis_gain = adc_target_db - p;
5     //call_func
6     power2gain(mis_gain, p);
7 }
8 gain_level_array power2gain(mis_gain, ant_power) {
9     //first, choose LNA and TIA
10    for(l=0;l<lna_level;l++) {
11        for(t=0;t<tia_level;t++) {
12            gain = lna_gain_tbl[l]+tia_gain_tbl[t];
13            if((ant_power+gain)<=thres_db_TIA) and (gain<mis_gain) //not saturate and
14            ↳not exceed adc_target_db
15                store l,t,gain;
16        }
17    }
18    sort(gain) // from low to high
choose the (l,t) that of highest gain

```

(continues on next page)



(continued from previous page)

```

19   if (size(l,t)>=2)
20     choose the one that has higher LNA gain.
21
22 //then choose VGA1 and VGA2 as the same principle
23 ...
24   return l,t,v1,v2;
25 }
```

When the final gain level table is ready, you can use the same mapping rule in [Table 10.8](#) to translate it to gain code with gain value and write them to RAM.

Do not forget to configure the following two registers. You can refer to [Table 10.9](#) to get the formula.

Table 10.21: Set Minimum Input Power and Target dB

Register Name	Offset	Bit Width	Description
CFG_AGC_GAIN_MIN	0x22C	10	-48
CFG_AGC_DB_TARGET	0x278	8	120

### 10.6.2.5 Setting IRQ-Related AGC

Since IRQ can indicate the AGC status of the RBU, it is suggested that IRQ be enabled. As discussed in [Table 10.11](#), this feature involves the interaction between the RBU and other units. A summary of IRQ-related AGC settings is shown in [Table 10.22](#).

---

**Note:** Whenever IRQ is triggered, CTL\_AGC\_IRQ\_CLR and CTL\_AGC\_SAT\_CNT\_CLR\_FRA should be re-configured.

---

Table 10.22: Setting IRQ-Related AGC

Register Name	Offset	Bit Width	Default Value	Value
CFG_AGC_IRQ_ENA	0x27C	12	0	h'FFF
CTL_AGC_IRQ_CLR	0x280	12	0	h'FFF
CTL_AGC_SAT_CNT_CLR_FRA	0x20C	1	0	h'1
CFG_AGC_CHCK_ENA	0x264	1	0	h'1
CFG_AGC_SAT_THR_TIA	0x200	24	0	h'010101
CFG_AGC_SAT_THR_VGA1	0x204	24	0	h'010101
CFG_AGC_SAT_THR_VGA2	0x208	24	0	h'010101

### 10.6.2.6 Configuring AGC Alignment

To enable the AGC alignment feature, set CFG\_AGC\_ALIGN\_EN as shown in [Table 10.23](#).

Table 10.23: Enabling AGC Alignment

Register Name	Offset	Bit Width	Default Value	Value
CFG_AGC_ALIGN_EN	0x268	1	0	1



## 10.6. Examples

---

### 10.6.2.7 Configuring ADC Compensation

Program for the ADC compensation feature as shown in [Table 10.24](#). Usually the greater the value of ADC compensation level is, the harder it is to trigger ADC compensation. So it is not recommended that CFG\_AGC\_CMPN\_LVL is set with a small value.

Table 10.24: Enabling ADC Compensation

Register Name	Offset	Bit Width	Default Value	Value
CFG_AGC_CMPN_EN	0x26c	1	0	1
CFG_AGC_CMPN_ALIGN_EN	0x270	1	0	1
CFG_AGC_CMPN_LVL	0x274	4	0	2

### 10.6.3 Configuring Control Bit to Enable AGC

As a sub-function of the RBPU, AGC corresponds to a control bit of the system register CFG\_SYS\_ENABLE. The configuration is shown in [Table 10.25](#).

Table 10.25: Enabling AGC Mode

Register Name	Offset	Bit Width	Item	Default Value	Value
CFG_SYS_ENABLE	0x18	9	AGC	[Bit0]: 0	1



## INTERFERENCE AVOIDANCE AND MITIGATION

### 11.1 Overview

Multiple radar systems using overlapped frequency bandwidth may cause interference among each other. Alps provides 3 modes to avoid interference among systems and 1 mechanism to mitigate interference.

### 11.2 Functional Description

#### 11.2.1 Interference Avoidance Modes

The interference avoidance modes include:

- Phase Scrambling (PS)
- Chirp Shifting (CS)
- Frequency Hopping (FH)

All three modes select random chirps within a frame and change one of the parameters of the chirps.

- For PS, the changed parameter is the phase of the waveform.
- For CS, the changed parameter is the starting point of ramp-up time.
- For FH, the changed parameters are the start and stop frequencies of the sweep, while the sweep bandwidth remains unchanged.

All three modes can be activated independently.

##### 11.2.1.1 XOR Chain

Alps uses 3 individual XOR chains to generate random patterns for 3 modes respectively.

Each XOR chain is implemented by a 32-bit self-shift register, as shown in Fig. 11.1. During each working cycle, one bit is right-shifted. There are maximum 32 binary addition operations, with input taken from bits of the shift register. The addition operation for each bit is controlled by a corresponding bit from a 32-bit *tap*. Both the initial state of the self-shift register and the tap are programmable. The resulting bit of addition operations is fed into the MSB of the self-shift register after right shifting. The LSB of the self-shift register is output as one bit of a pseudo-random sequence that controls the behavior of the chirps in interference avoidance mode.

## 11.2. Functional Description

---

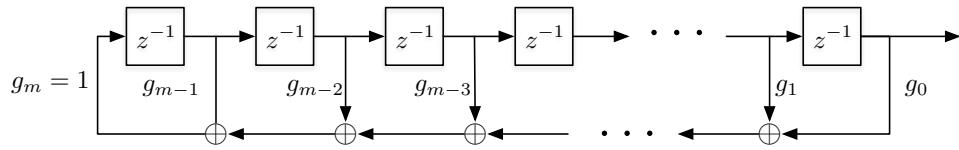


Fig. 11.1: XOR Chain Used to Generate Random Sequence

### 11.2.1.2 Phase Scrambling

Phase scrambling is to randomize the phase of each chirp, as shown in Fig. 11.2. +1 indicates phase  $0^\circ$  and -1 indicates phase  $180^\circ$ . Note that phase scrambling is applied across all TX channels, which is different from BPM discussed in Section 9. The pattern of +1 and -1 sequence as shown in Fig. 11.2 is controlled by the corresponding XOR chain. The output 1 indicates a  $180^\circ$  phase shift, and 0 indicates no phase shift.

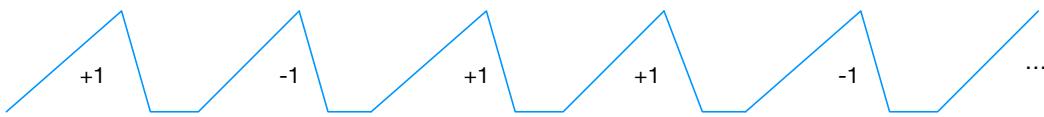


Fig. 11.2: Phase Scrambling Frame

The inverted phase has to be compensated in baseband. From interference's perspective, the compensation operation is a process of whitening itself. The power of the interfering spectrum is so evenly spread along the Doppler dimension that it cannot form a distinct tone after 2D-FFT, as explained in Section 6.

A straightforward way of compensation is simply inverting the sign of the signal of the phase-shifted chirps. However, due to the imperfection of analog circuits, the actual phase shift applied to different TX virtual groups may be different, close to but not exactly  $180^\circ$ .

A more sophisticated and refined compensation is described in Section 9 that enables users to compensate for the phases of signals corresponding to different TX virtual groups.

### 11.2.1.3 Frequency Hopping

Frequency hopping mode uses two different types of chirps inside a frame, as illustrated by the second sub-figure in Fig. 11.3. The two different chirps have different start and stop frequencies but the same bandwidth and chirp period. Suppose the two different start frequencies are  $f_{L1}$  (the higher one) and  $f_{L2}$  (the lower one), and the difference between  $f_{L1}$  and  $f_{L2}$  is  $f_{shift}$ . The chirp type inside the frame is decided by the output of FH XOR chain. Outputting 0 indicates that the chirp start frequency is  $f_{L2}$  and outputting 1 indicates that the chirp start frequency is  $f_{L1}$ .

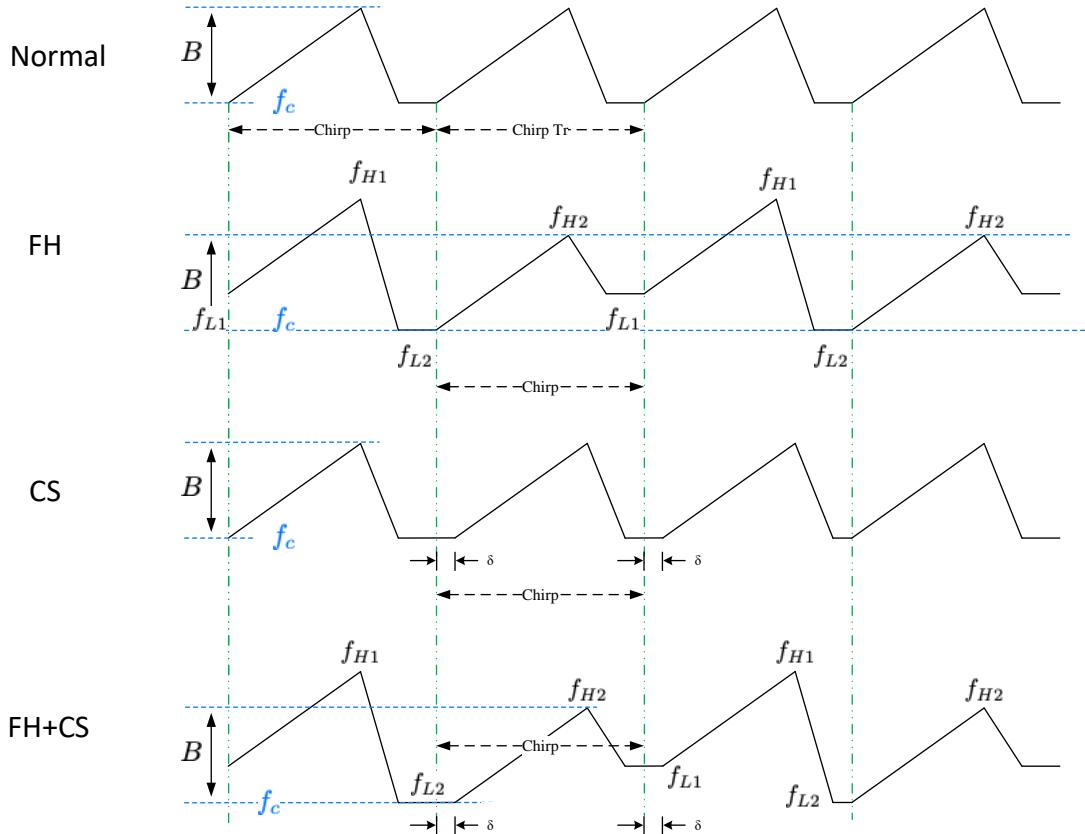


Fig. 11.3: Frequency Hopping & Chirp Shifting

According to the FMCW radar principle, the returned signal is mixed with the transmitted signal.

- In FH mode, if the returned signal is desirable, the frequency of its mixed signal is unvaried for all chirps. Only a phase difference exists between hopped chirps and non-hopped chirps. Suppose a reflector's range is  $R$ . Then the phase difference between hopped chirps and normal chirps is given by:

$$\theta_{FH} = -2\pi \frac{2f_{shift}R}{c} \quad (11.1)$$

Since it is relevant to the reflector's range, the phase compensation is chosen to be done between RNG-FFT and VEL-FFT. Therefore, RNG-FFT index here can be regarded as proportional to distance. The compensation is further given by:

$$\theta_{FH} = -2\pi \frac{T_u}{B} \frac{F_S \cdot f_{shift} \cdot k}{N_{rng}} \quad (11.2)$$



## 11.2. Functional Description

---

Where:

Table 11.1: Parameter Descriptions

Item	Explanation
$B$	system bandwidth
$T_u$	chirp ramp-up time
$k$	RNG-FFT index
$F_s$	sampling frequency of input for RNG-FFT
$N_{rng}$	RNG-FFT size

- In FH mode, if the returned signal is undesirable, which means an interference signal, the frequency of its mixed signal in misaligned chirps will become bigger than the cutoff frequency of the low-pass filter (LPF) before ADC, and thus not visible to baseband.

### 11.2.1.4 Chirp Shifting

Similar to frequency hopping, chirp shifting mode also uses two different types of chirps inside a frame, as illustrated by the third sub-figure in Fig. 11.3. The ramp-up time for one type of chirp has zero offset compared to the starting point of the chirp, and  $\delta$  offset for the other type of chirp. The type of chirp is indicated by the output of CS XOR chain. Outputting 1 from CS XOR chain indicates a  $\delta$  delay for the ramp-up of the chirp, and outputting 0 indicates a zero delay. The third sub-figure in Fig. 11.3 gives an example of CS where the ramp-up time of the second and third chirps is shifted by a delay  $\delta$ .

The working principle of CS mode is similar to that of frequency hopping. The mixed signal consists of desired signals and interference. The desired signals come from reflectors. For a desired signal, frequency is unvaried for all chirps, but phase varies between delayed chirps and non-delayed chirps. For an interference signal, however, the frequency in misaligned chirps will be beyond the cutoff frequency of LPF before ADC, so the interference signal will not be visible to baseband.

In CS mode, the phase difference between delayed chirps and normal chirps is given by:

$$\theta_{CS} = 2\pi \frac{2B}{T_u} \frac{R}{c} \delta \quad (11.3)$$

Just like the case of FH, CS compensation is also done between RNG-FFT and VEL-FFT. So the compensation can be further given by:

$$\theta_{CS} = 2\pi \frac{F_s \cdot \delta \cdot k}{N_{rng}} \quad (11.4)$$

### 11.2.1.5 Coexistence of Frequency Hopping and Chirp Shifting

Frequency hopping mode and chirp shifting mode can be activated simultaneously. When this is the case, there would be altogether four types of chirps. As illustrated by the fourth sub-figure in Fig. 11.3, the first chirp is frequency hopped and the second one is chirp shifted. The third one is both frequency hopped and chirp shifted, while the fourth chirp is neither frequency hopped nor chirp shifted. The phase to be compensated for each type of chirp is different. The following table lists the compensation phases for all 4 combinations.

Index	FH	CS	Compensation Phase between RNG-FFT and VEL-FFT
0	0	0	0
1	0	1	$2\pi \frac{F_s \cdot \delta \cdot k}{N_{rng}}$
2	1	0	$-2\pi \frac{T_u}{B} \frac{F_s \cdot f_{shift} \cdot k}{N_{rng}}$
3	1	1	$2\pi \left( \frac{\delta}{N_{rng}} - \frac{T_u}{B} \frac{f_{shift}}{N_{rng}} \right) F_s \cdot k$



The compensation of phases is implemented as multiplying the result of RNG-FFT by a unit amplitude complex value  $e^{j\theta}$ , where  $\theta$  is the phase to be compensated. The procedure is illustrated as Fig. 11.4.

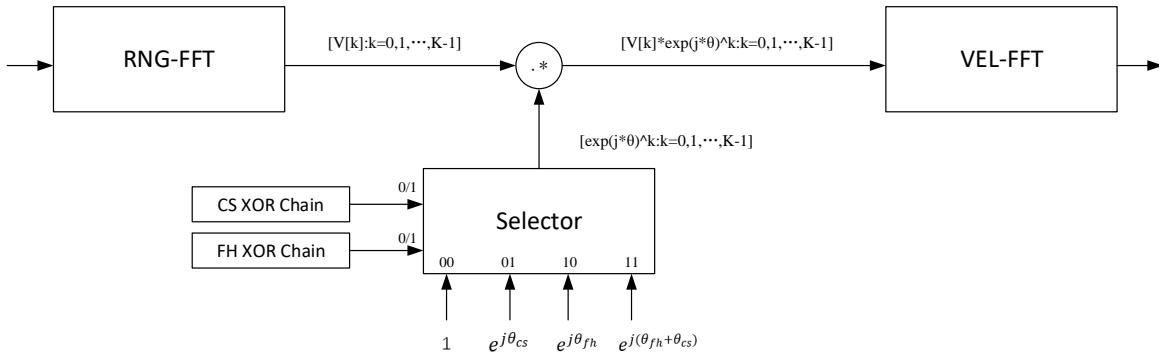


Fig. 11.4: Frequency Hopping & Chirp Shifting Compensation Procedure

## 11.2.2 Interference Mitigation

### 11.2.2.1 Interference Generation Analysis

Interference is caused by other radar systems working within the same bandwidth. Fig. 11.5 shows the frequency modulation triangle waveforms for transmitted, received, and interference chirps.

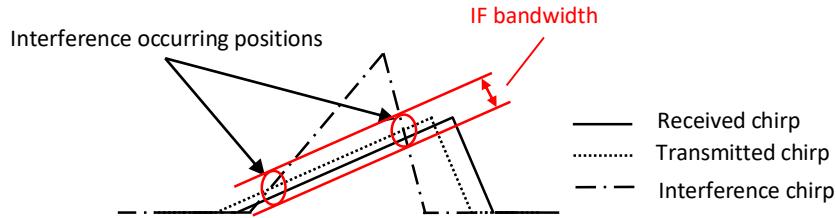


Fig. 11.5: Mechanism of Mutual Interference

Generated by modulation slope differences between interference chirp and transmitted chirp, an interference waveform theoretically looks like a linear frequency modulated (LFM) signal that is filtered by a IF low-pass filter (LPF). Fig. 11.6 is the corresponding waveform.



## 11.2. Functional Description

---

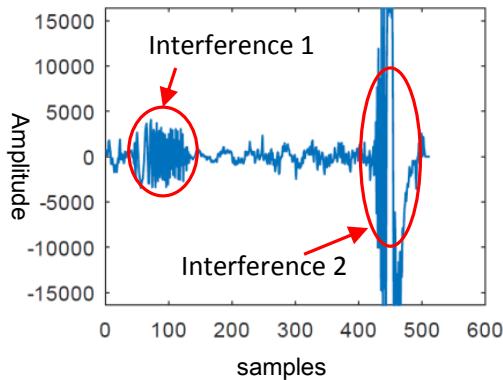


Fig. 11.6: Waveform of Chirp Samples with Interference

Sometimes, due to the multi-path effect and narrow IF LPF passband, the interference looks like a spike, as shown in Fig. 11.7.

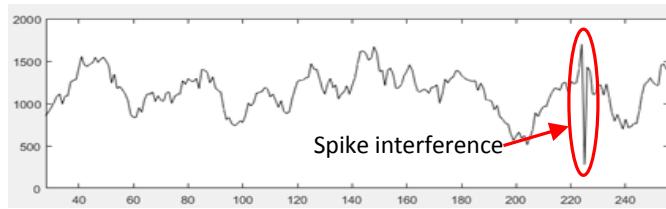


Fig. 11.7: Waveform of Spike Interference

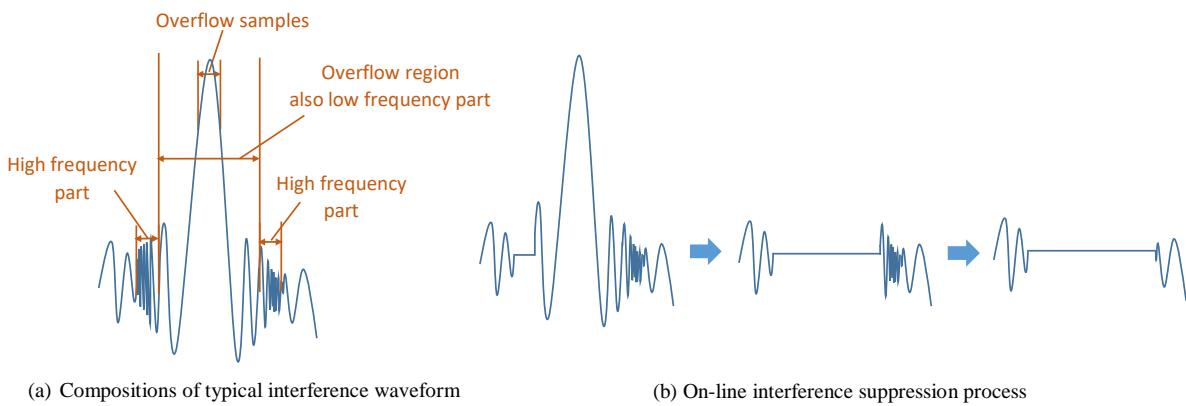


Fig. 11.8: Interference Compositions and On-Line Suppression Process

As shown in Fig. 11.8, a typical interference waveform can be roughly divided into three parts in sequence: the high frequency (HF) part at the head and low frequency (LF) part in the middle and HF part at the tail.

### 11.2.2.2 Mitigation of High Frequency Part in Interference Waveform

To mitigate interference, interference should first be detected and then be suppressed.

#### 11.2.2.2.1 Detecting HF Part of Interference

An intuitive and effective way to detect the HF part of interference is by exploiting the amplitude difference between neighboring samples.

For LFMCW radar, the HF part in the received waveform usually stands for faraway objects and is usually with relatively lower power. However, the HF part of interference from nearby radar can have large power, which means the amplitude difference between neighboring interference samples can be very large. So the amplitude difference can be chosen as a criterion to detect interference in time domain.

Given ADC samples  $S(n)$ ,  $D(n)$  denotes the absolute difference between neighboring samples:

$$D(n) = |S(n) - S(n - 1)|$$

Denote  $D_{max} = \max\{D(n)\}$ , where  $n$  is sampling index in a chirp. For cases without interference,  $D_{max}$  is determined by objects' radar cross section (RCS), distance, direction, and other effects such as noise. The  $D_{max}$ 's values for different chirps in one frame are normally very close, so it is an ideal threshold for detecting the HF part of interference.

In one chirp, the periodicity of sine-waves with different frequencies representing objects of various distances implies that  $D_{max}$  can be estimated by searching the maximum  $D(n)$  within a certain length of samples:  $S(N_1)$ ,  $S(N_1 + 1)$ , ...,  $S(N_2)$ , i.e.  $D_{max} = \max\{D(n)\}, n = N_1, N_1 + 1, \dots, N_2$ .

With  $D_{max}$ ,  $D(n)$  is then to be compared with  $D_{max}$  when deciding whether  $S(n)$  is an HF interference sample. To avoid misjudgment, in addition to  $D(n)$ , its following  $L$  consecutive sample differences  $D(n + 1)$ ,  $D(n + 2)$ , ...,  $D(n + L)$  also need to be considered. A simple way is to take a sum over them:

$$D_{sum} = D(n) + D(n + 1) + D(n + 2) + \dots + D(N + L)$$

Only if both of the following conditions are satisfied,  $S(n)$  is judged to be an HF part sample of interference:

$$D(n) > D_{max} \quad (11.5)$$

$$D_{sum} > c \cdot L \cdot D_{max} \quad (11.6)$$

Where  $c$  is a configurable adjusting coefficient.

#### 11.2.2.2.2 Suppressing HF Part of Interference

If  $S(n)$  is judged to be an interference sample, two options are provided to suppress the interference:

- Option 1: Replace  $S(n)$  by  $S_{new}(n) = 0$ ;
- Option 2: Replace  $S(n)$  by  $S_{new}(n) = S(n - 1) + D_{max}$  or  $S_{new}(n) = S(n - 1) - D_{max}$ , according to the sign of  $S(n) - S(n - 1)$ .

Either option is chosen by setting 1 or 0 to bit register CFG\_SAM\_SPK\_SET\_ZERO.

---

**Note:** When judging the next sample  $S(n + 1)$ ,  $D(n + 1)$  in Expression (11.5) should be updated by  $|S(n + 1) - S_{new}(n)|$ .

---



### 11.2.2.3 Mitigation of Low Frequency Part in Interference Waveform

#### 11.2.2.3.1 Detecting LF Part of Interference

The LF part in the middle of an interference waveform has a higher amplitude due to IF LPF frequency response and multi-path effect. To detect the LF part of strong interference, the overflow criteria is introduced.

In the overflow region, as illustrated in Fig. 11.8, the frequencies of interference samples can be very low. So Equation (11.6) for detecting high frequency may no longer be satisfied. So the criterion for deciding  $S(n)$  to be an overflow interference is reduced to only Expression (11.5).

Sample  $S(n)$  is in overflow region when there are at least  $M$  overflow samples in  $S(n)$ 's  $K$  neighboring samples:  $S(n - K), S(n - K + 1), \dots, S(n - 1), S(n), S(n + 1), \dots, S(n + K - 1), S(n + K)$ .

To simplify the usage,  $K$  is chosen to equal the buffer length  $L$  in Expression (11.6), which is specified by the register `CFG_SAM_SPK_CFM_SIZE`.  $M$  is specified by the upper 2 bits in `CFG_SAM_SPK_OVER_NUM`. The lower 2 bits of `CFG_SAM_SPK_OVER_NUM` specify the overflow threshold for judging whether  $S(n)$  is an overflow sample.

#### 11.2.2.3.2 Suppressing LF Part of Interference

The detected overflow interference samples can be suppressed also using the 2 options described in *Suppressing HF Part of Interference*.

---

**Note:** Overflow detection is not always effective. For interference without strong power, even the LF part can hardly overflow.

---

## 11.3 Programming Interfaces

### 11.3.1 Interference Avoidance Modes

#### 11.3.1.1 Enabling Interference Avoidance Modes

To enable interference avoidance modes, the register `CFG_FFT_DINT_ENA` (0xC0\_1556) should be set. As discussed in Section 11.2.1, Alps supports 3 modes of interference avoidance, each of which can work independently. Different modes correspond to different bits in `CFG_FFT_DINT_ENA` as listed in Table 11.2.

Table 11.2: `CFG_FFT_DINT_ENA` description

Bits	Description
<i>bit0</i>	This bit enables frequency hopping mode
<i>bit1</i>	This bit enables chirp shifting mode
<i>bit2</i>	This bit enables phase scrambling mode



### 11.3.1.2 Programming XOR Chain

As mentioned in [Section 11.2.1.1](#), the random pattern of each mode is controlled by an individual XOR chain, which can be programmed separately. Each XOR chain is associated with two 32-bit registers. See [Table 11.3](#).

Table 11.3: Registers of XOR chain

Register Name	Address	Description
CFG_FFT_DINT_DAT_FH	0xC0_1568	Initial data of FH XOR chain
CFG_FFT_DINT_MSK_FH	0xC0_1580	Tapper for FH XOR chain
CFG_FFT_DINT_DAT_CS	0xC0_1572	Initial data of CS XOR chain
CFG_FFT_DINT_MSK_CS	0xC0_1584	Tapper for CS XOR chain
CFG_FFT_DINT_DAT_PS	0xC0_1576	Initial data of PS XOR chain
CFG_FFT_DINT_MSK_PS	0xC0_1588	Tapper for PS XOR chain

For different modes, pick up different polynomials for m-sequence and program the taps of XOR chains by setting the register `CFG_FFT_DINT_MSK_FH/CS/PS` to the value of  $[g_31 g_30 \dots g_0]$  (See [Fig. 11.1](#)). Select different initial states of XOR chains and set the register `CFG_FFT_DINT_DAT_FH/CS/PS`.

### 11.3.1.3 Programming Frequency Hopping and Chirp Shifting

As mentioned in [Section 11.2.1.5](#), the compensation of phases is implemented as multiplying the result of RNG-FFT by a unit amplitude complex value  $e^{j\theta}$ , where  $\theta$  is the phase to be compensated. Since there are maximum 4 different combinations when talking about enabling or disabling FH and CS features, 4 different unit amplitude complex values corresponding to 4 different phases need to be configured. When neither FH nor CS is enabled,  $e^{j\theta}$  is the fixed value of 1.

The other 3 unit amplitude complex values are configured through registers `CFG_FFT_DINT_COE_FH` (0xC0\_1592), `CFG_FFT_DINT_COE_CS` (0xC0\_1596) and `CFG_FFT_DINT_COE_FC` (0xC0\_1600), respectively. These 3 registers have an identical 28-bit format, with 14 of them storing the real part and the other 14 storing the imaginary part. Every 14 bits is in the format of  $FXI(14, 1, S)$ .

The detailed configuration is shown in [Table 11.4](#).

Table 11.4: Register Settings for Phase Compensation in FH & CS

Registers	Numeric Value
<code>CFG_FFT_DINT_COE_FH</code>	$e^{-j \cdot 2\pi \cdot \frac{T_u}{B} \cdot \frac{F_s \cdot f_{shift}}{\text{DecFlt} \cdot N_{rng}}}$
<code>CFG_FFT_DINT_COE_CS</code>	$e^{j2\pi \frac{F_s \cdot \delta}{\text{DecFlt} \cdot N_{rng}}}$
<code>CFG_FFT_DINT_COE_FC</code>	$e^{j \cdot 2\pi \cdot F_s \left( \frac{\delta}{N_{rng}} - \frac{T_u}{B} \cdot \frac{f_{shift}}{N_{rng}} \right) / \text{DecFlt}}$

### 11.3.1.4 Programming Phase Scrambling

As mentioned in [Section 11.2.1.2](#), besides simply inverting the sign of the signal where all TX virtual group are compensated by a phase shift of  $180^\circ$ , a more sophisticated and refined compensation for phase scrambling is also available, enabling users to compensate different TX virtual groups for different phases. From the point of view of implementation, just like things in FH or CS cases, phase compensation is done by baseband through multiplying the result of RNG-FFT by a unit amplitude complex value. Since there are maximum 4 TX virtual groups, the unit amplitude complex values corresponding to 4 TX virtual groups should be configured in 4 different registers as listed in [Table 11.5](#). When virtual array is configured with  $N_{va}$  virtual groups, only the configuration of first  $N_{va}$  registers matter.



Table 11.5: Phase Compensation for PS

Registers	Description
CFG_FFT_DINT_COE_PS_0	phase combination for virtual array group 0
CFG_FFT_DINT_COE_PS_1	phase combination for virtual array group 1
CFG_FFT_DINT_COE_PS_2	phase combination for virtual array group 2
CFG_FFT_DINT_COE_PS_3	phase combination for virtual array group 3

The register as mentioned in Table 11.5 are of 28 bits, with 14 bits storing the real part and the other 14 for the imaginary part. The format for both the real and imaginary parts is  $FXI(14, 1, S)$ .

#### 11.3.2 Interference Mitigation

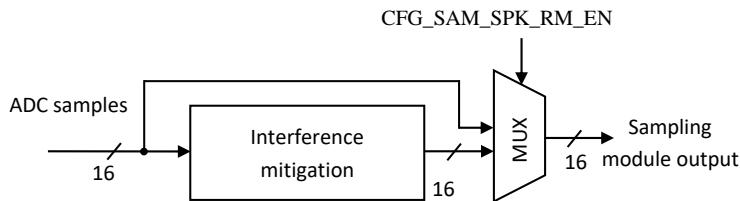


Fig. 11.9: Enabling Interference Mitigation

- To enable interference mitigation function, set the bit register `CFG_SAM_SPK_RM_EN` to 1. Otherwise, interference mitigation is bypassed as shown in Fig. 11.9.
- `CFG_SAM_SPK_CFM_SIZE` specifies the buffer length  $L$  in Equation (11.6) for detecting the HF part of interference.
- `CFG_SAM_SPK_SET_ZERO` is a bit register. When it is set to 1, detected interference samples are set to zero, which means Option 1 described in [Suppressing HF Part of Interference](#) is used to mitigate the interference. When it is set to 0, Option 2 described in [Suppressing HF Part of Interference](#) is used.
- `CFG_SAM_SPK_OVER_NUM` determines the parameters for overflow detection. Its upper 2 bits represent the least number of overflow samples in the overflow region. Its lower 2 bits represent the threshold for deciding whether a sample is an overflowed interference sample. for details refer to register table.
- `CFG_SAM_SPK_THRES_DB` is a bit register. If it is set to 1,  $c$  in Equation (11.6) is 2, i.e., doubling the threshold for detecting the HF part of interference.
- In implemented algorithm,  $D_{max}$  is estimated twice by searching the maximum  $D_{max}$  in the two different sample segments within a chirp. If interference occurs in the estimation sample segment, then estimated  $D_{max}$  can be very large due to the HF part of interference. In this case, it is recommended that the smaller  $D_{max}$  be chosen. To achieve this, set `CFG_SAM_SPK_MIN_MAX_SEL` to 1 .

## 11.4 Limitation and Constraints

### 11.4.1 Interference Avoidance Modes

The three interference avoidance modes can be activated independently and working together with other features simultaneously.

### 11.4.2 Interference Mitigation

If the interference power is not large enough, the low frequency part of interference is hard to detect.

If the chirp length is too short (such as 64 sample points in a chirp), it may be hard to estimate  $D_{max}$  correctly and interference is more likely to occur in the  $D_{max}$  estimation segments.

When RNG-FFT is 2048, interference mitigation should not be enabled (CFG\_SAM\_SPK\_RM\_EN should be set to 0).

## 11.5 Software and Configuration Suggestions

### 11.5.1 Interference Avoidance Modes

#### 11.5.1.1 XOR chain

The initial states and taps of three interference avoidance modes should be configured differently to avoid pattern alignments among these modes.

#### 11.5.1.2 Frequency Hopping

As can be seen from Equation (11.1), the phase to be compensated is proportional to the range of the reflector denoted by  $R$ . In real life,  $R$  is a continuous value. However, the estimated range of a reflector in baseband is mapped from the result of RNG-FFT, and as a result, the  $R$  in Equation (11.1) can only be discrete. The accuracy of  $R$  is dominated by the discrete error, which can be given by Equation (11.7).

$$\Delta R = \frac{T_u \cdot c \cdot F_S}{4 \cdot B \cdot N_{rng} \cdot DecFlt} \quad (11.7)$$

The discrete error of the estimated range propagates to the compensation as:

$$\Delta\theta_{fh} = 2\pi \frac{T_u \cdot f_{shift} \cdot F_S}{2 \cdot B \cdot N_{rng} \cdot DecFlt} \quad (11.8)$$

To eliminate the error, the restriction  $\frac{\Delta\theta_{fh}}{2\pi} \ll 1$  should be satisfied. A suggested maximum value of  $f_{shift}$ , denoted by  $f_{shift,max}$ , is:

$$f_{shift,max} = \frac{B \cdot N_{rng} \cdot DecFlt}{10 \cdot T_u \cdot F_S} \quad (11.9)$$



## 11.5. Software and Configuration Suggestions

---

### 11.5.1.3 Chirp Shifting

Similar to in FH mode as described in [Section 11.5.1.2](#), configuration for chirp shifting should also eliminate the compensation phase error. And the suggested maximum value of  $\delta$ , denoted by  $\delta_{\max}$ , is:

$$\delta_{\max} = \frac{N_{rng} \cdot \text{DecFlt}}{10 \cdot F_S} \quad (11.10)$$

## 11.5.2 Interference Mitigation

In outdoor scenarios, the DC level of a received signal is close to zero. Setting detected interference samples to 0 will help to suppress the interference.

For indoor applications, however, the heavy indoor multi-path effect will lead to a raised DC level, and setting detected interference samples to 0 will probably introduce positive or negative pulse.

Meanwhile, the interference detection sensitivity can be adjusted by configuring the following registers:

- The buffer length used for detecting HF interference is specified by `CFG_SAM_SPK_CFM_SIZE`. When buffer length is smaller, fewer samples need to be considered, and the sensitivity is higher.
- Setting `CFG_SAM_SPK_THRES_DBL` to 1 will double the threshold when deciding HF part of interference, thus decreasing the interference detection sensitivity.
- The minimum number of overflow samples in the overflow region is specified by the upper 2 bits of `CFG_SAM_SPK_OVER_NUM`. A smaller value means that more samples can lie in the overflow region, which equivalently means that the overflow region becomes larger.
- The overflow threshold specified by the lower 2 bits of `CFG_SAM_SPK_OVER_NUM` determines the overflow threshold. A higher threshold indicates lower sensitivity for detecting overflow samples.

The principles for adjusting sensitivity are summarized in [Table 11.6](#). Higher sensitivity will also lead to higher interference misjudgment probability.

Table 11.6: Interference Detection Sensitivity Adjusting Principle

Register Name	Value Settings to Achieve Higher Detection Sensitivity
<code>CFG_SAM_SPK_CFM_SIZE</code>	lower values
<code>CFG_SAM_SPK_THRES_DBL</code>	lower values (0)
<code>CFG_SAM_SPK_OVER_NUM</code> (Bits[3:2])	lower values
<code>CFG_SAM_SPK_OVER_NUM</code> (Bits[1:0])	lower values

[Table 11.7](#) summarizes the performances of suppressing different types of interference.

Table 11.7: Mitigation Performances for Different Types of Interference

Interference Type	HF Part	LF Part
spike	enough suppression	enough suppression
strong LFM interference	enough suppression	enough suppression
weak LFM interference	enough suppression	not enough suppression



## 11.6 Examples

### 11.6.1 Interference Avoidance Modes

This section gives an example of how to configure and activate interference avoidance modes.

In this example, initial system parameters are as follows:

Table 11.8: Initial System Configuration

System Configuration	Value	System Configuration	Value
$B$ (system bandwidth)	675 Mhz	$N_{rng}$ (RNG-FFT Size)	512
$T_u$ (ramp-up time)	40 $\mu s$	$F_s$ (sampling frequency)	25 Mhz
$DecFlt$ (decimation filter factor)	1		

$N_{va} = 2$  is configured and the phase compensation for Virtual Group 0 and Virtual Group 1 are  $181^\circ$  and  $179^\circ$  respectively.

We want to configure the interference avoidance functionality to meet the following requirements:

- All three interference avoidance modes are enabled.
- XOR chains for different modes are configured differently, so that the random pattern used by each mode is different.
- The states of three XOR chains are reset manually by setting a trigger.
- The delay time  $\delta$  for CS is restricted as described in Section 11.5.1.2.
- The hopping frequency  $f_{shift}$  for FH is restricted as described in Section 11.5.1.3.

Configuration steps are as follows:

1. Enable all three interference avoidance modes by setting `CFG_FFT_DINT_ENA`.
2. Set three different sets of initial states and taps for XOR chains.

Table 11.9: Register Settings for XOR Chains

Register Name	Addr	Description
<code>CFG_FFT_DINT_DAT_FH</code>	0xC0_1568	0x12348765
<code>CFG_FFT_DINT_MSK_FH</code>	0xC0_1580	0x00000407
<code>CFG_FFT_DINT_DAT_CS</code>	0xC0_1572	0x87654321
<code>CFG_FFT_DINT_MSK_CS</code>	0xC0_1584	0x00000409
<code>CFG_FFT_DINT_DAT_PS</code>	0xC0_1576	0xdeadbeaf
<code>CFG_FFT_DINT_MSK_PS</code>	0xC0_1588	0x00000401

3. To minimize the possibility of alignment among different radar systems, manually reset XOR chains by setting all bits of `CFG_FFT_DINT_MOD` to 1. And to trigger the loading of initial states and taps only during system initialization, set corresponding bits of `CTL_FFT_DINT_SET` to 1.
4. Determine the value of  $f_{shift}$ .

Calculated from Equation (11.9),  $f_{shift, \max}$  equals 34.56 Mhz, so we choose  $f_{shift} = 15MHz$ .

5. Determine the value of  $\delta$ .

Calculating from Equation (11.10), we choose  $\delta = 1 \mu s$ .



## 11.6. Examples

---

6. Calculate the numeric values to configure registers for phase compensation in FH and CS, as shown in:

Table 11.10: Phase Compensation for FH & CS

Registers	Numeric Value
CFG_FFT_DINT_COE_FH	$0.9630 - 0.2693i$
CFG_FFT_DINT_COE_CS	$0.9533 + 0.3020i$
CFG_FFT_DINT_COE_FC	$0.9994 + 0.0341i$

7. Configure the phase compensation for the two virtual groups as:

Table 11.11: Phase Compensation for PS

Registers	Description
CFG_FFT_DINT_COE_PS_0	$e^{\frac{j\pi \cdot 181}{180}} = -0.9998 - 0.0175i$
CFG_FFT_DINT_COE_PS_1	$e^{\frac{j\pi \cdot 179}{180}} = -0.9998 + 0.0175i$

The sample code for initialization is shown below. This function should be called during system initialization.

```
void init_INT_PSFHCS() {
    int va      = 2; /* virtual array with 2 TX */
    int Tu      = 40; /* unit us, 1e-6s */
    int Fs      = 25; /* sample frequency, unit is MHz */
    int DecFltr = 1; /* decimate filter rate */
    int B       = 675; /* bandwidth, unit is MHz */

    /* enabler, user can disable any mode by switching corresponding 1 to 0 */
    unsigned int PS_en = 1; /* enable Phase Scrambling */
    unsigned int FH_en = 1; /* enable Frequency Hopping */
    unsigned int CS_en = 1; /* enable Chirp Shifting */

    unsigned int en_bitmask = ((FH_en & 0x01)
                               | ((CS_en & 0x01) << 1)
                               | ((PS_en & 0x01) << 2));

    /* programming XOR chain for FH */
    write_reg(CFG_FFT_DINT_DAT_FH, 0x12348765);
    write_reg(CFG_FFT_DINT_MSK_FH, 0x000000407);

    /* programming XOR chain for CS */
    write_reg(CFG_FFT_DINT_DAT_CS, 0x87654321);
    write_reg(CFG_FFT_DINT_MSK_CS, 0x000000409);

    /* programming XOR chain for PS */
    write_reg(CFG_FFT_DINT_DAT_PS, 0xdeadbeaf);
    write_reg(CFG_FFT_DINT_MSK_PS, 0x000000401);

    /* programming phase compensation for FH */
    complex_t Cplx_FH = complex_t(0.9630, -0.2693);
    /* programming phase compensation for CS */
    complex_t Cplx_CS = complex_t(0.9533, 0.3020);
    /* programming phase compensation for CS & FH */
    complex_t Cplx_FC = complex_t(0.9994, 0.0341);

    /* format of CFG_FFT_DINT_COE_FH is CFX(14,1,S) */
    write_reg(CFG_FFT_DINT_COE_FH, complex_to_cfx(&Cplx_FH, 14, 1, true));
}
```

(continues on next page)



(continued from previous page)

```

/* format of CFG_FFT_DINT_COE_CS is CFX(14,1,S) */
write_reg(CFG_FFT_DINT_COE_CS, complex_to_cfx(&Cplx_CS, 14, 1, true));

/* format of CFG_FFT_DINT_COE_FC is CFX(14,1,S) */
write_reg(CFG_FFT_DINT_COE_FC, complex_to_cfx(&Cplx_FC, 14, 1, true));

/* programming phase compensation for PS */
complex_t Cplx_PS_VA0 = complex_t(-0.9998, -0.0175);
complex_t Cplx_PS_VA1 = complex_t(-0.9998, +0.0175);

/* format of CFG_FFT_DINT_COE_PS_0 is CFX(14,1,S) */
write_reg(CFG_FFT_DINT_COE_PS_0, complex_to_cfx(&Cplx_PS_VA0, 14, 1, true));
write_reg(CFG_FFT_DINT_COE_PS_1, complex_to_cfx(&Cplx_PS_VA1, 14, 1, true));

/* enable Interference avoidance mode */
write_reg(CFG_FFT_DINT_ENA, en_bitmask);

/* the reset of Xor-chain only triggered by setting 7 to
   register CTL_FFT_DINT_SET */
write_reg(CFG_FFT_DINT_MOD, 7);

/* trigger loading of initial states and taps for PS/CS/FH */
write_reg(CTL_FFT_DINT_SET, 7);
}

```

## 11.6.2 Interference Mitigation

An example configuration table and corresponding suppression results are shown in Table 11.12 and Fig. 11.10

Table 11.12: Example Settings for Interference Mitigation

Register name	Register value	parameters
CFG_SAM_SPK_RM_EN	1	Enable interference mitigation function
CFG_SAM_SPK_CFM_SIZE	0x4	Buffer length for detecting HF interference is set to 4
CFG_SAM_SPK_THRES_DBL	1	Threshold for detecting HF interference is doubled
CFG_SAM_SPK_OVER_NUM (Bits[3:2])	0x0	The minimum number of overflow samples in overflow region is 1
CFG_SAM_SPK_OVER_NUM (Bits[1:0])	0x0	Threshold for detecting overflow samples is 8192
CFG_SAM_SPK_MIN_MAX_SEL	1	Select the smaller $D_{max}$ estimated in a chirp
CFG_SAM_SPK_SET_ZERO	1	Set detected interference samples to zero



## 11.6. Examples

---

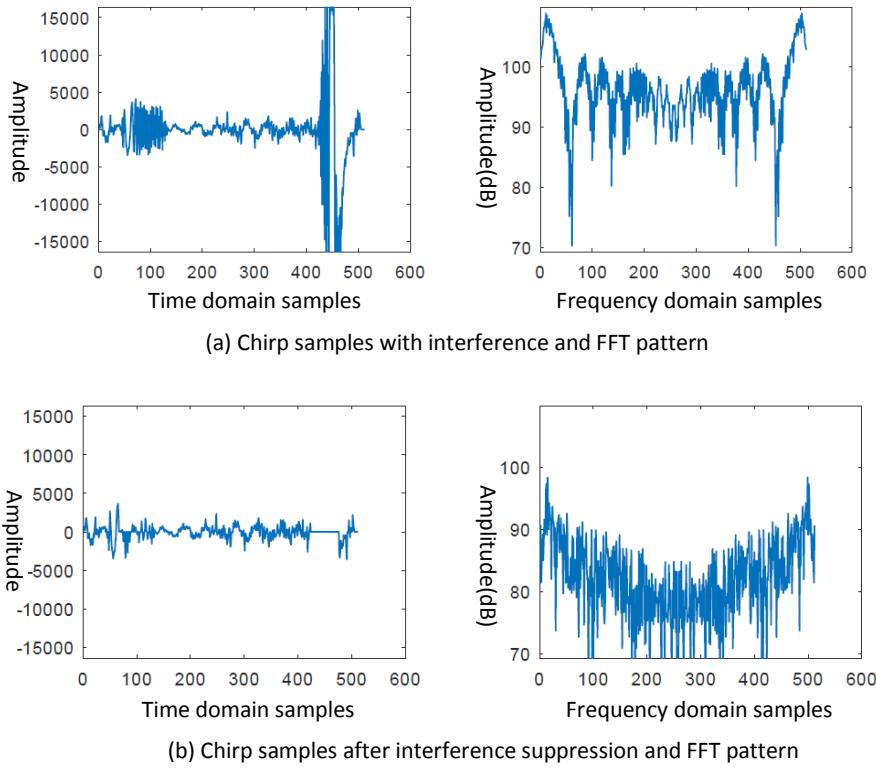


Fig. 11.10: An Example of Interference Mitigation

## FRAME INTERLEAVING

### 12.1 Overview

Frame interleaving is designed for improving radar performance. For example, with the frame interleaving feature, a single radar sensor can detect objects both in the long range and the short range, only by switching the frame bank. This feature enables multiple frame banks to avoid reprogramming. See Fig. 12.1.

Different frame banks can be selected flexibly to adapt to different scenarios.

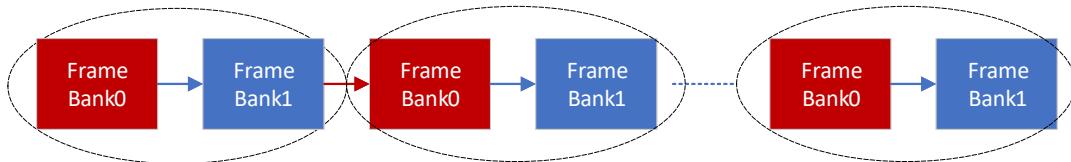


Fig. 12.1: Frame Interleaving Example of Two Banks

#### 12.1.1 Features

- **Single Mode:** In this mode, only a single bank of frame configurations is run.
- **Rotate Mode:** In this mode, multiple banks of frame configurations are run cyclically.

### 12.2 Functional Description

Frame interleaving supports maximal four different banks of frame configurations, including Bank0, Bank1, Bank2, and Bank3. To use the frame interleaving feature, once Alps is powered on, all banks of frame configurations should be programmed to be initialized. In each bank, one can configure baseband and radio parameters, such as FFT number, chirp number, scan frequencies, virtual array control, AGC enable/disable, frequency hopping on/off, phase scrambling on/off, and DoA methods.

After initialization, 2 modes are designed to launch different banks of frame configurations, including:

- single mode
- rotate mode

### 12.2.1 Single Mode

*Single mode* is the basic mode of frame interleaving in Alps.

In *single mode*, baseband and FMCW always work on one single bank that is selected from four banks of frame configurations, as shown in Fig. 12.2. Even if multiple banks are initialized, only one bank will be launched. So in *single mode*, it is not necessary to initialize multiple banks. It is suggested that users should only initialize one bank when using *single mode*, because obviously, initializing one bank is faster than initializing multiple banks.

Besides, *single mode* can use more hardware resources to achieve higher performance, because some RAM in hardware is shared for multiple banks, such as MEM\_COE.

### 12.2.2 Rotate Mode

*Rotate mode* is used for more complicated scenarios.

As shown in Fig. 12.3, in *rotate mode*, baseband and FMCW can switch from one mode to another mode automatically, by switching configurations from one bank to another bank. This is convenient and time-saving for adapting to multiple different system requirements without reprogramming the system settings, because the settings of different modes are all initialized to corresponding banks when the device is powered on.

The switch order of *rotate mode* is also configurable. To use *rotate mode*, designers should know well about the requirements to set the configurations in multiple banks.

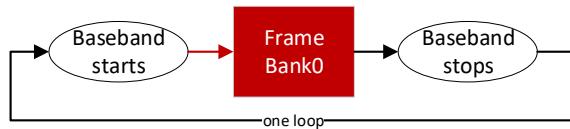


Fig. 12.2: Single Mode Using Bank0

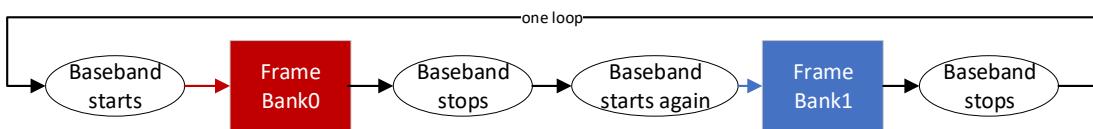


Fig. 12.3: Rotate Mode Using Bank0 and Bank1

## 12.3 Programming Interfaces

Frame interleaving related configurations involve baseband registers and radio registers. The radio register configuration for frame interleaving is described in Section 4. For more details about radio registers, refer to *Calterah Alps Reference Manual*.



### 12.3.1 Single Mode

#### 12.3.1.1 Baseband Configuration

To enter *single mode*, `CFG_SYS_BNK_MODE` and `CFG_SYS_BNK_ACT` should be configured.

- `CFG_SYS_BNK_MODE` is the working mode of baseband. Value 0 means single mode and value 2 means rotate mode.
- `CFG_SYS_BNK_ACT` is the current active bank, it has 5 values from 0 to 4. It is the working bank in *single mode*. In *single mode*, the bank specified by the register `CFG_SYS_BNK_ACT` will be launched for each frame.

### 12.3.2 Rotate Mode

#### 12.3.2.1 Baseband Configuration

To enter *rotate mode*, `CFG_SYS_BNK_MODE` and `CFG_SYS_BNK_QUE` should be configured. In *rotate mode*, all the banks specified by the register `CFG_SYS_BNK_QUE` will be launched cyclically for each frame.

- `CFG_SYS_BNK_MODE` is the working mode of baseband. Value 0 means single mode and value 2 means rotate mode.
- `CFG_SYS_BNK_QUE` is the enabled banks in rotate mode. It has 5 bits from 0 to 4. Each bit mask with 1 means that the corresponding bank is enabled. In *rotate mode*, when baseband starts, all enabled banks will be launched one by one. The loop order is also determined by this register.

For example, in Fig. 12.3, Bank0 and Bank1 are used in loop. The value of `CFG_SYS_BNK_QUE` in this example is 0011 in binary. Then, while baseband is running, 2 frame banks will be triggered and the frame bank order is Bank0 → Bank1. That means, the first frame will be launched using Bank0 configurations, and then the second frame will be launched using Bank1 configurations.

### 12.3.3 Interaction Between CPU and Baseband

There are maximum 5 working banks in baseband, but the CPU can only access one bank at a time. So the bank should be switched for CPU accessing. `FDB_SYS_BNK_ACT` and `CTL_SYS_BNK_RST` handle the interaction between the CPU and baseband.

- `FDB_SYS_BNK_ACT` is the feedback value of the current active bank, it is used for the interaction between CPU and baseband. The current active bank is actually indicated by `FDB_SYS_BNK_ACT`, which is a read-only register designed. The CPU performs data interaction with baseband according to this register. In *single mode*, the value of `FDB_SYS_BNK_ACT` is the same with the value of `CFG_SYS_BNK_ACT`. In *rotate mode*, the value of `FDB_SYS_BNK_ACT` is looped and determined by `CFG_SYS_BNK_QUE`.
- `CTL_SYS_BNK_RST` is the reset for bank selection. It is designed to initialize or terminate the loop and return to the first frame bank. After the baseband registers have been configured, `CTL_SYS_BNK_RST` should be asserted to reset `FDB_SYS_BNK_ACT` to be the initial state. In *single mode*, `CTL_SYS_BNK_RST` makes `FDB_SYS_BNK_ACT` equal to `CFG_SYS_BNK_ACT`. In *rotate mode*, `CTL_SYS_BNK_RST` resets `FDB_SYS_BNK_ACT` to the first frame bank in `CFG_SYS_BNK_QUE`.



# 12.4 Limitation and Constraints

## 12.4.1 Registers for Rotate Mode

There are 2 kinds of baseband registers, banked registers and non-banked registers.

- Non-banked registers

Non-banked registers need only to be configured once during initialization. Their values are effective across all banks for all frames.

- Banked registers

When using banked registers for a frame, a bank must first be specified through `CFG_SYS_MEM_ACT`. See [Bank Shifting](#) for detailed information. If, for example, Bank 0 is selected in a frame, then banked registers are all used for Bank 0 in this frame. To select different banks for different frames, users need to configure banked registers multiple times during initialization, each time corresponding to a specific bank.

Refer to the Bank column of [Baseband Register Tables](#).

## 12.4.2 RAMs for Rotate Mode

There are 4 kinds of baseband RAMs:

- *Non-Banked RAMs*
- *Banked RAMs*
- *MEM\_RLT*
- *RAMs with Address Pointers*

Refer to [Accessing Memories](#).

### 12.4.2.1 Non-Banked RAMs

A non-banked RAM is a shared RAM for all banks. `MEM_COD`, `MEM_SAM`, `MEM_BUFS` are non-banked RAMs and only `MEM_COD` is configurable. During initialization, non-banked RAMs only need to be configured once.

### 12.4.2.2 Banked RAMs

A banked RAM is split into 4 individual memories of an equal size, corresponding to 4 banks. The 4 memories are independent and cannot be shared among different banks. During initialization, the start address of a banked RAM should be set corresponding to the bank number. Refer to [Bank Shifting](#). Only one bank of RAM space can be selected and be active in a frame.

`MEM_WIN`, `MEM_NVE`, `MEM_MAC`, `MEM_AMB`, and `MEM_DML` are banked RAMs, which can be configured multiple times corresponding to multiple banks. So to read data from or write data to these memories for all banks, multiple reads/writes must be done, with each read/write corresponds to one specific bank.



### 12.4.2.3 MEM\_RLT

MEM\_RLT is a special kind of RAM, which is used to store the baseband CFAR and DoA results and is accessed by the CPU.

- When  $\text{CFG\_CFR\_SIZE\_OBJ} < 256$ , MEM\_RLT can be split into be 4 banks.
- When  $\text{CFG\_CFR\_SIZE\_OBJ} > 256$ , MEM\_RLT is used as one whole memory. Banks are ignored.

### 12.4.2.4 RAMs with Address Pointers

A RAM with address pointers can be split into multiple separate memories corresponding to multiple banks, through address pointer registers. The separate memories for banks composes the whole memory space.

MEM\_COE is the only one RAM with address pointers and it stores 3 types of parameters, whose data width is 28 bits.

- CFAR MIMO coefficients (See *Combination Engines*)
- DBPM coefficients before DoA (See *Virtual Array*)
- DoA coefficients of steering vectors (See *Programming Steering Vectors*)

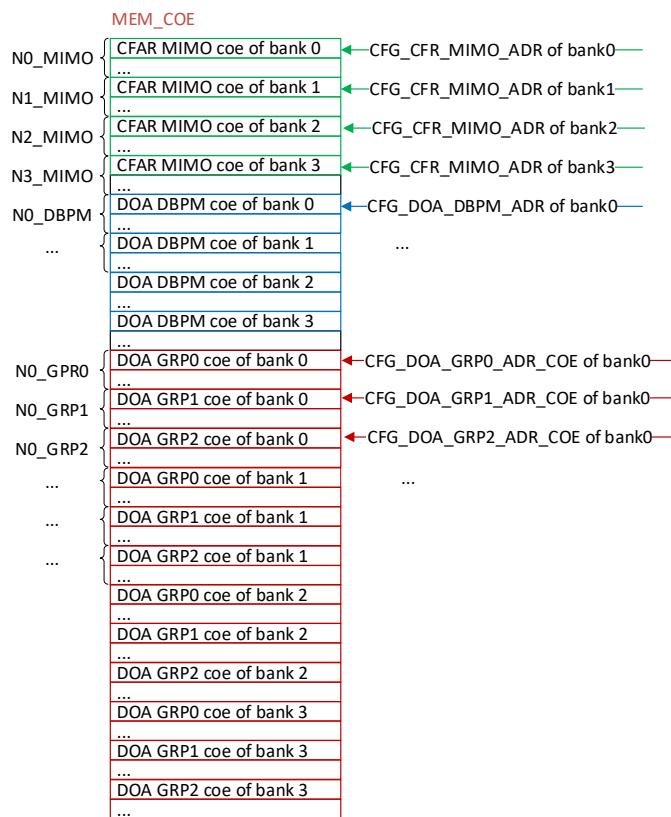


Fig. 12.4: Memory Space in MEM\_COE

Suppose that MEM\_COE is split into 4 banks as illustrated by Fig. 12.4. The sizes of these 3 parameters are  $s_c$ ,  $s_b$ ,

## 12.4. Limitation and Constraints

---

$s_d$ , respectively. Then the following conditions must be met:

$$s_c + s_b + s_d \leq 3872 \times 4$$

In other words, the total size of these 3 parameters in all banks (see Table 12.1) should not be greater than the memory size, which is 3872 x 4 (data width is 28 bits).

---

**Note:** CFAR MIMO coefficients and DoA DBPM coefficients are not always required, and the multiple banks are also not always required. So some variables in [Table 12.1](#) could be invalid and MEM\_COE should be programmed flexibly to adapt to the specific application.

---

The address pointers of these 3 parameters all support multiple banks.

- CFG\_CFR\_MIMO\_ADR is the address pointer of CFAR MIMO coefficients. See [Combination Engines](#).
- CFG\_DoA\_DBPM\_ADR is the address pointer of DoA DBPM coefficients. See [Virtual Array](#).
- CFG\_DoA\_GRP0\_ADR\_COE, CFG\_DoA\_GRP1\_ADR\_COE and CFG\_DoA\_GRP2\_ADR\_COE are the address pointers of DoA coefficients of steering vectors, as DoA can supports multiple groups (See [Programming Steering Vectors](#)).



Table 12.1: MEM\_COE Size

Variable	Description	Computation Method
N0_MIMO	coefficients number of CFAR MIMO in bank 0	see <i>Combination Engines</i>
N1_MIMO	coefficients number of CFAR MIMO in bank 1	see <i>Combination Engines</i>
N2_MIMO	coefficients number of CFAR MIMO in bank 2	see <i>Combination Engines</i>
N3_MIMO	coefficients number of CFAR MIMO in bank 3	see <i>Combination Engines</i>
N0_DBPM	coefficients number of DoA DBPM in bank 0	see <i>Virtual Array</i>
N1_DBPM	coefficients number of DoA DBPM in bank 1	see <i>Virtual Array</i>
N2_DBPM	coefficients number of DoA DBPM in bank 2	see <i>Virtual Array</i>
N3_DBPM	coefficients number of DoA DBPM in bank 3	see <i>Virtual Array</i>
N0_GRP0	coefficients number of DoA GRP0 in bank 0	see <i>Programming Steering Vectors</i>
N0_GRP0	coefficients number of DoA GRP1 in bank 0	see <i>Programming Steering Vectors</i>
N0_GRP0	coefficients number of DoA GRP2 in bank 0	see <i>Programming Steering Vectors</i>
N0_GRP1	coefficients number of DoA GRP0 in bank 1	see <i>Programming Steering Vectors</i>
N0_GRP1	coefficients number of DoA GRP1 in bank 1	see <i>Programming Steering Vectors</i>
N0_GRP1	coefficients number of DoA GRP2 in bank 1	see <i>Programming Steering Vectors</i>
N0_GRP2	coefficients number of DoA GRP0 in bank 2	see <i>Programming Steering Vectors</i>
N0_GRP2	coefficients number of DoA GRP1 in bank 2	see <i>Programming Steering Vectors</i>
N0_GRP2	coefficients number of DoA GRP2 in bank 2	see <i>Programming Steering Vectors</i>
N0_GRP3	coefficients number of DoA GRP0 in bank 3	see <i>Programming Steering Vectors</i>
N0_GRP3	coefficients number of DoA GRP1 in bank 3	see <i>Programming Steering Vectors</i>
N0_GRP3	coefficients number of DoA GRP2 in bank 3	see <i>Programming Steering Vectors</i>

The values of address pointers should be computed carefully, which is relative to previous address pointers.

- The value of the first address pointer should be 0.
- The value of the next address pointer (denoted by  $A1$ ) should be equal to the previous pointer value (denoted by  $A0$ ) plus the number ( $N0$ ) of the related parameters associated with the previous pointer, which is  $A1 = A0 + N0$ . See **Table 12.2**.

## 12.5. Software and Configuration Suggestions

---

Table 12.2: MEM\_COE Address Pointers

Address	Definition	Value	Description
A0_MIMO	CFG_CFMAR_MIMO_ADR in bank 0	0	memory address of CFAR MIMO in bank 0
A1_MIMO	CFG_CFMAR_MIMO_ADR in bank 1	A0_MIMO + N0_MIMO	memory address of CFAR MIMO in bank 1
A2_MIMO	CFG_CFMAR_MIMO_ADR in bank 2	A1_MIMO + N1_MIMO	memory address of CFAR MIMO in bank 2
A3_MIMO	CFG_CFMAR_MIMO_ADR in bank 3	A2_MIMO + N2_MIMO	memory address of CFAR MIMO in bank 3
A0_DBPM	CFG_DoA_DBPM_ADR in bank 0	A3_MIMO + N3_MIMO	memory address of DoA DBPM in bank 0
A1_DBPM	CFG_DoA_DBPM_ADR in bank 1	A0_DBPM + N0_DBPM	memory address of DoA DBPM in bank 1
A2_DBPM	CFG_DoA_DBPM_ADR in bank 2	A1_DBPM + N1_DBPM	memory address of DoA DBPM in bank 2
A3_DBPM	CFG_DoA_DBPM_ADR in bank 3	A2_DBPM + N2_DBPM	memory address of DoA DBPM in bank 3
A0_GRP0	CFG_DoA_GRP0_ADR_COE in bank 0	A3_DBPM + N3_DBPM	memory address of DoA GRP0 in bank 0
...	...	...	...

### 12.4.3 Aligning Baseband and Radio

The challenge about frame interleaving is that the configuration in baseband and radio should be aligned. The most important registers are `CFG_SYS_BNK_QUE` in baseband and `VECS` in radio. Configuration of these two registers should match. Otherwise, the radar will go wrong.

For example, if Bank2 and Bank3 are to be used in *rotate mode*, Bank2 and Bank3 should be enabled in `CFG_SYS_BNK_QUE` and the value should be 1100 in binary. And in `VEC` (see frame interleaving in [Section 4.4.3](#)), binary 11\_10 should be repeated 8 times and the final value should be 11\_10\_11\_10\_11\_10\_11\_10\_11\_10\_11\_10\_11\_10\_11\_10\_11\_10.

## 12.5 Software and Configuration Suggestions

### 12.5.1 Initialization

For the frame interleaving feature, the most complicated work is how to initialize the baseband.

#### 12.5.1.1 Initialization Flow

The main steps to initialize baseband for frame interleaving include:

1. Set the `CFG_SYS_BNK_ACT` register to a specific bank number.
2. Initialize corresponding banked registers controlled by `CFG_SYS_BNK_ACT`.
3. Initialize corresponding RAMs controlled by `CFG_SYS_BNK_ACT`.

The start address of each bank should be set corresponding to the bank number and bank size.

4. Initialize other banks. For each bank, loop Steps 1 to 3.

5. After all banks are initialized, reset the CTL\_SYS\_BNK\_RST register to ensure that FDB\_SYS\_BNK\_ACT is set with the correct initial state.

### 12.5.1.2 Example

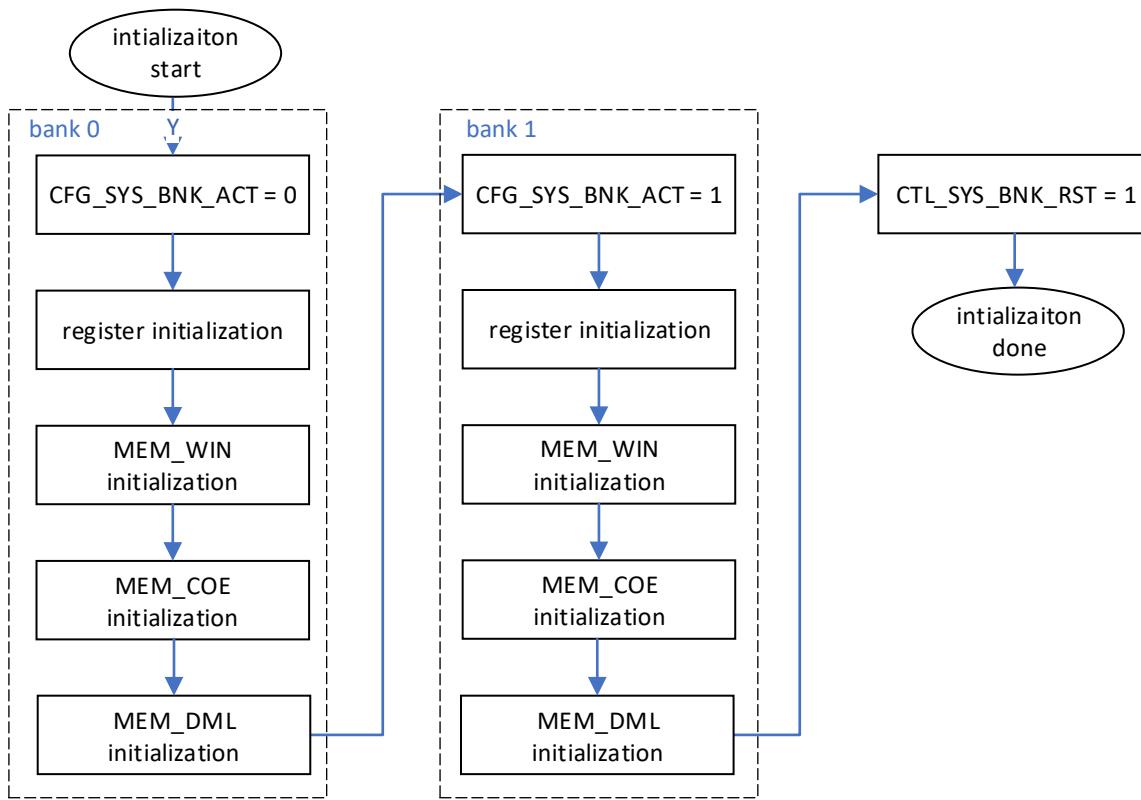


Fig. 12.5: Baseband Initialization Flow for Frame Interleaving

Fig. 12.5 illustrates an example, where:

- Bank 0 and Bank 1 are initialized for the rotate mode of frame interleaving.
- The following 3 RAMs are initialized for each bank.
  - MEM\_WIN for FFT window coefficients  
Refer to [FFT Windows](#) for details about how to initialize FFT window.
  - MEM\_COE for DML steering vectors
  - MEM\_DML for DML coefficients  
Refer to [Programming Steering Vectors](#) for details about how to initialize DML coefficients.

## 12.5. Software and Configuration Suggestions

### 12.5.1.3 Pseudo-Code for Rotate Mode Initialization

The following example pseudo-code shows how to complete the initialization for frame interleaving in the rotate mode.

```
1 #define BANK0 0
2 #define BANK1 1
3 #define QUEUE_BANK1_BANK0 4'b0011
4 #define ROTATE_MODE 2
5 #define SINGLE_MODE 0
6 #define CFG_SYS_BNK_ACT 0xC0_0008
7
8 int init_rotate_mode()
{
9
10    // initialize BANK0
11    reg_write(CFG_SYS_BNK_ACT, BANK0); // activate BANK0
12    init_register(); // initialize registers in SAMPLE, FFT, CFAR and DoA of BANK0
13    init_MEM_WIN(); // initialize MEM_WIN of BANK0
14    init_MEM_COE(); // initialize MEM_COE with address points of BANK0
15    init_MEM_DML(); // initialize MEM_DML of BANK0
16
17    // initialize BANK1
18    reg_write(CFG_SYS_BNK_ACT, BANK1); // activate BANK1
19    init_register(); // initialize registers in SAMPLE, FFT, CFAR and DoA of BANK1
20    init_MEM_WIN(); // initialize MEM_WIN of BANK1
21    init_MEM_COE(); // initialize MEM_COE with address points of BANK1
22    init_MEM_DML(); // initialize MEM_DML of BANK1
23
24    // rotate mode
25    reg_write(CFG_SYS_BNK_QUE, QUEUE_BANK1_BANK0); // enable BANK0 and BANK1
26    reg_write(CFG_SYS_BNK_MODE, ROTATE_MODE); // enable rotate mode
27    reg_write(CTL_SYS_BNK_RST, 1); // reset bank selection to BANK0
28
29    return 0;
30}
31 }
```

### 12.5.1.4 Pseudo-Code for Single Mode Initialization

The following example pseudo-code shows how to complete the initialization for frame interleaving in the single mode, which is relatively simple, compared to the rotate mode.

```
1 #define BANK0 0
2 #define ROTATE_MODE 2
3 #define SINGLE_MODE 0
4 #define CFG_SYS_BNK_ACT 0xC0_0008
5
6 int init_single_mode()
{
7
8    // initialize BANK0
9    reg_write(CFG_SYS_BNK_ACT, BANK0); // activate BANK0
10   init_register(); // initialize registers in SAMPLE, FFT, CFAR and DoA of BANK0
11   init_MEM_WIN(); // initialize MEM_WIN of BANK0
12   init_MEM_COE(); // initialize MEM_COE with address points of BANK0
13   init_MEM_DML(); // initialize MEM_DML of BANK0
14
15 }
```

(continues on next page)



(continued from previous page)

```

16 // single mode
17 reg_write(CFG_SYS_BNK_MODE, SINGLE_MODE);           // enable single mode
18 reg_write(CTL_SYS_BNK_RST, 1);                      // reset bank selection to BANK0
19
20     return 0;
21 }
```

## 12.5.2 Configuration Suggestions

- MEM\_RLT

In *rotate mode*, for simplicity, MEM\_RLT should be considered as a non-banked RAM, by setting CFG\_CFR\_SIZE\_OBJ to a value greater than 256. In this case, MEM\_RLT is wholly used in each frame and will be overwritten by next frame. This is convenient for designers to read back the content in MEM\_RLT.

## 12.6 Examples

In automotive radar application scenarios, medium range radar (MRR) and short range radar (SRR) are both needed in a single chip. So *rotate mode* should be enabled with 2 banks (bank 0 and bank 1), corresponding to MRR and SRR respectively. MRR and SRR in the 2 banks will be switched and looped automatically one frame by one frame, as shown in Fig. 12.1.

### 12.6.1 MRR and SRR Settings

- MRR is focused on the range within 150 meters, and it puts high requirements on angular accuracy and angular resolution. So virtual array is chosen for MRR.
- SRR is focused on the range within 50 meters, and it puts high requirements on range accuracy. So larger FFT points in the range dimension will be used for SRR.

The main settings for MRR and SRR are shown in Table 12.3 and Table 12.4. For other settings, such as CFAR and DoA related settings, refer to the related chapters.

#### 12.6.1.1 MRR Settings in Bank 0

MRR Settings will be initialized in bank 0.



## 12.6. Examples

---

Table 12.3: MRR Settings

Registers Name	Example Value	Description
CFG_SYS_SIZE_RNG_PRD	999	Chirp period is $40\mu s$ ( $F_{ADC} = 25MHz$ )
CFG_SYS_SIZE_FLT	0	No down sampling
CFG_SYS_SIZE_RNG_SKP	49	Skipped sample length from chirp start is $2\mu s$ ( $F_{ADC} = 25MHz$ )
CFG_SYS_SIZE_RNG_BUF	499	Used sample length after skipping for range FFT is $20\mu s$ ( $F_{ADC} = 25MHz$ )
CFG_SYS_SIZE_RNG_FFT	511	Range FFT size is 512
CFG_SYS_SIZE_BPM	1	Virtual array number is 2
CFG_SYS_SIZE_VEL_BUF	127	Chirp number in a frame is 128
CFG_SYS_SIZE_VEL_FFT	127	Velocity FFT size is 128
CFG_FFT_SHFT_RNG	511	Enable 9-stage shifter, stage 1~9
CFG_FFT_SHFT_VEL	127	Enable 8-stage shifter, stage 1~7
CFG_DoA_DBPM_ENA	0	TDM MIMO is selected, so disable demodulation of BPM
CFG_DoA_GRP0_TYP_SCH	2	DML method is selected
CFG_DoA_GRP0_SIZ_ONE_ANG	511	DML angle size is 512
CFG_DoA_GRP0_SIZ_CMB	7	Combined number of RX channels is 8

### 12.6.1.2 SRR Settings in Bank 1

SRR Settings will be initialized in bank 1.

Table 12.4: SRR Settings

Registers Name	Example Value	Description
CFG_SYS_SIZE_RNG_PRD	1499	Chirp period is $60\mu s$ ( $F_{ADC} = 25MHz$ )
CFG_SYS_SIZE_FLT	0	No down sampling
CFG_SYS_SIZE_RNG_SKP	49	Skipped sample length from chirp start is $2\mu s$ ( $F_{ADC} = 25MHz$ )
CFG_SYS_SIZE_RNG_BUF	999	Used sample length after skipping for range FFT is $40\mu s$ ( $F_{ADC} = 25MHz$ )
CFG_SYS_SIZE_RNG_FFT	1023	Range FFT size is 1024
CFG_SYS_SIZE_BPM	0	Virtual array mode is off
CFG_SYS_SIZE_VEL_BUF	127	Chirp number in a frame is 128
CFG_SYS_SIZE_VEL_FFT	127	Velocity FFT size is 128
CFG_FFT_SHFT_RNG	1023	Enable 10-stage shifter, stage 1~10
CFG_FFT_SHFT_VEL	127	Enable 8-stage shifter, stage 1~7
CFG_DoA_GRP0_TYP_SCH	2	DML method is selected
CFG_DoA_GRP0_SIZ_ONE_ANG	511	DML angle size is 512
CFG_DoA_GRP0_SIZ_CMB	3	Combined number of RX channels is 4



## 12.6.2 Frame Interleaving Settings

### 12.6.2.1 Baseband Switch

Table 12.5: baseband switch in frame interleaving

Registers Name	Example Value	Description
CFG_SYS_BNK_MODE	1	rotate mode
CFG_SYS_BNK_QUE	3	rotate bank 0 and bank 1

### 12.6.2.2 Radio Switch

Radio parameters have to be configured to match baseband settings. Follow these steps:

1. Configure FMCW related parameters in related banks, including  $F_{start}$ ,  $F_{stop}$ ,  $T_u$ ,  $T_d$ , and  $T_{idle}$ .
2. Configure FMCW for rotate mode and set the frame interleaving period to 2 to match baseband settings, as shown in the following table:

Table 12.6: FMCW Frame Interleaving Registers Settings

ADDR	Example Value	Bits	Bit Name	Bit Description
0x00	8'b0000_0011	7:4	Reserved	Not used
		3:0	fmcw_bank_sel[3:0]	Select bank 3, which is FMCW controlling bank
0x1C	8'b0000_0100	7:0	fmcw_fil_vecs[7:0]	Set the first frame to type 0 and the second to type 1
0x1D	8'b0000_0000	7:0	fmcw_fil_vecs[15:8]	Not used
0x1E	8'b0000_0000	7:0	fmcw_fil_vecs[23:16]	Not used
0x1F	8'b0000_0000	7:0	fmcw_fil_vecs[31:24]	Not used
0x25	8'b0000_0001	7:4	Reserved	Not used
		3:0	fmcw_fil_prd[3:0]	Set frame interleaving period to 2
0x27	8'b0000_0001	7:1	Reserved	Not used
		0	fmcw_fil_en	Enable frame interleaving
0x2C	8'b0000_0010	7:3	Reserved	Not used
		2:0	fmcw_mode_sel[2:0]	Set to predefined mode



## **12.6. Examples**

---



## ANTI VELOCITY AMBIGUITY (ANTI-VELAMB)

This section describes the anti velocity ambiguity (VELAMB) functionality adopted by Alps chip.

### 13.1 Overview

As mentioned in [Section 1.3](#), baseband uses 2D-FFT to extract a reflector's range and velocity information. From velocity FFT's point of view, the range of the Doppler frequency  $f_D$  that can be accepted by the FFT engine is  $[-\frac{F_D}{2}, \frac{F_D}{2})$ , where  $F_D$  is the sampling frequency of FFT input. Using  $T_r$  to denote the length of a chirp,  $F_D$  equals  $1/(N_{va} \cdot T_r)$  when virtual array with  $N_{va}$  virtual groups is applied and  $1/T_r$  otherwise.

If the moving object's velocity is so fast that  $f_D$  is outside the range of  $[-\frac{F_D}{2}, \frac{F_D}{2})$ , velocity ambiguity happens.  $f_D$  can be expressed as:

$$f_D = f_{rd} + q \cdot F_D, q \in \mathbb{Z} \quad (13.1)$$

where  $f_{rd} \in [-\frac{F_D}{2}, \frac{F_D}{2})$ . The Anti-VELAMB function is for estimating Ambiguity Factor  $q$  and using it to restore a correct Doppler frequency and then a correct velocity of the moving object.

The Alps chip supports two anti velocity ambiguity mechanisms.

- Anti velocity ambiguity with chirp delay (Anti-VELAMB CD)
- Anti velocity ambiguity with multi-frame (Anti-VELAMB MF)

The Anti-VELAMB CD mechanism modifies a radio waveform by inserting an extra chirp and a time delay before a normal chirp in TX Group 0. Due to the existence of the Doppler frequency of a moving object, the phase of the normal chirp is drifted from the phase of the extra chirp by an amount linear to the multiplication of  $f_D$  and time interval of the extra chirp plus the delay. Baseband checks the phase difference and estimates the  $q$  value behind that.

The Anti-VELAMB MF mechanism, on the other hand, uses *Chinese remainder theorem* to estimate the  $q$  value by applying different sets of frame-wise system parameters to interleaved frames. The parameters are so designed that the sampling frequencies of VEL-FFT input are different and, for a given reflector's Doppler frequency  $f_D$ , the resulting  $q$  values are different. One assumption is held that for a physical reflector, its range and velocity cannot change dramatically between two consecutive frames. By trying different pairs of  $q$  values which result in the same  $f_D$  into CFAR outputs of adjacent frames, whose range can be considered close enough, baseband detects the reflector's range and solves velocity ambiguity at the same time.

Both mechanisms have their own pros and cons. The Anti-VELAMB CD mechanism does not involve pairing operations, but decreases the sampling frequency of VEL-FFT, which makes maximum non-ambiguous velocity even smaller. Anti-VELAMB MF, however, only involves software changes but the associated pairing operations are much more vulnerable to the surrounding interference and system thermal noise.

### 13.1.1 Features

Both mechanisms can work simultaneously with other features such as cascade, virtual array etc.

## 13.2 Functional Description

### 13.2.1 Anti Velocity Ambiguity with Chirp Delay

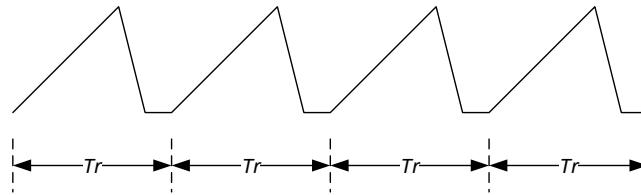


Fig. 13.1: Conventional Chirp Sequence w/o Virtual Array

To overcome the problem of velocity ambiguity, the conventional chirp sequence as illustrated by Fig. 13.1 is modified. In case of no virtual array, the Alps chip inserts a time delay in front of every second chirp in the sequence as illustrated by Fig. 13.2. The delay amount is denoted by  $a$ . The modified chirp sequence has a new period  $T_D = 2T_r + a$ .

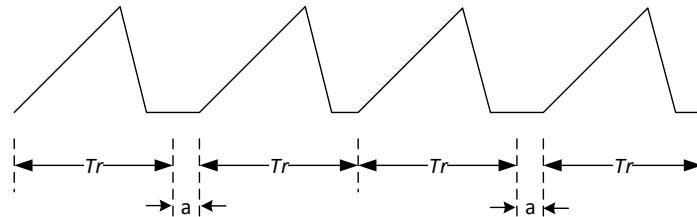


Fig. 13.2: Anti Velocity Ambiguity Chirp Sequence w/o Virtual Array

In case of virtual array with  $N_{va}$  virtual groups, the extra chirp and the delay are inserted in front of the normal chirp in Virtual Group 0 as illustrated by Fig. 13.3. One can argue that a case without virtual array can be considered as a special virtual array case with  $N_{va}$  equal to 1. In that sense, we can unify our saying that the extra chirp and the delay is always inserted before the normal chirp in Virtual Goup 0. The new period of virtual array, or equivalently speaking, the sampling period of VEL-FFT input is changed from  $T_D = N_{va} \cdot T_r$  to  $T_D = (N_{va} + 1) \cdot T_r + a$ . The sampling frequency of VEL-FFT input  $F_D$  becomes  $F_D = 1 / ((N_{va} + 1) \cdot T_r + a)$ .

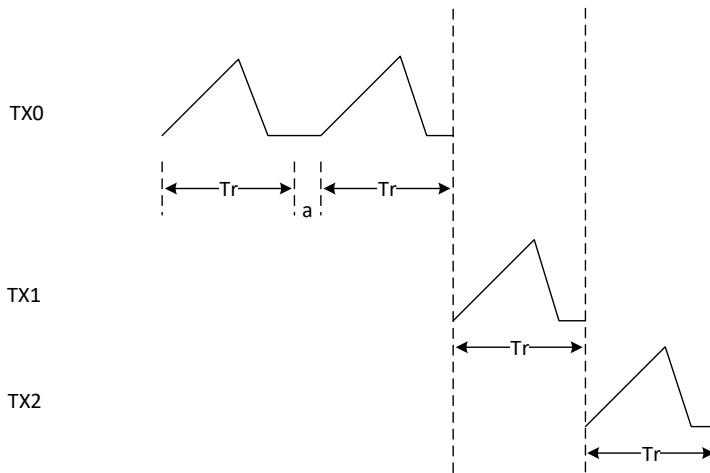


Fig. 13.3: Anti Velocity Ambiguity Chirp Sequence With Virtual Array

The new chirp sequence for Virtual Group 0 comprises two intertwined sub-sequences. The first one is comprised of chirps without preceding delays, or chirps whose indices are even numbers, starting from 0. The other one is comprised of chirps with the preceding time delay  $a$ . Let's use  $s_1(n, l)$  and  $s_2(n, l)$  to express these two sub-sequences, where  $n$  is the sample index within a chirp, and  $l$  is the index of a virtual array period. Performing 2D-FFT over  $s_1(n, l)$  and  $s_2(n, l)$  gets  $S_1(k, p)$  and  $S_2(k, p)$ , where  $k$  and  $p$  are the frequency indexes of 2D-FFT output.

Let's use  $(k_{\text{peak}}, p_{\text{peak}})$  to denote the location of a tone of CFAR output and  $S_1(k_{\text{peak}}, p_{\text{peak}})$  and  $S_2(k_{\text{peak}}, p_{\text{peak}})$  are the 2D-FFT outputs corresponding to that specific tone. Define  $\varphi_d(k_{\text{peak}}, p_{\text{peak}})$  as:

$$\varphi_d(k_{\text{peak}}, p_{\text{peak}}) \triangleq \arg(S_2(k_{\text{peak}}, p_{\text{peak}})) - \arg(S_1(k_{\text{peak}}, p_{\text{peak}})) - f_{rd}(a + T_r) \quad (13.2)$$

Where  $\arg(x)$  is the function to calculate the angle of the complex value  $x$ . Then we will have:

$$\frac{\varphi_d(k_{\text{peak}}, p_{\text{peak}})}{2\pi} - \text{round}\left(\frac{\varphi_d(k_{\text{peak}}, p_{\text{peak}})}{2\pi}\right) = q \frac{T_r + a}{T_D} - \text{round}\left(q \frac{T_r + a}{T_D}\right) \quad (13.3)$$

Since  $T_r$ ,  $a$ , and  $T_D$  are known for a specific system configuration, by evaluating Equation (13.3) we can extract the value of  $q$ .

#### 13.2.1.1 Number Of $q$ and Unambiguous Range of Doppler Frequency

$N_q$  is the number of different  $q$  values, which can result in different values of  $\frac{\varphi_d(k_{\text{peak}}, p_{\text{peak}})}{2\pi} - \text{round}\left(\frac{\varphi_d(k_{\text{peak}}, p_{\text{peak}})}{2\pi}\right)$  in Equation (13.3). It is configurable to the system.

The set of all possible  $q$  values is defined as:  $q \in \{q_{\min} + 0, q_{\min} + 1, \dots, q_{\min} + N_q - 1\}$ , where the minimum  $q$  value  $q_{\min}$  is also configurable. The unambiguous Doppler frequency range changes from  $[-\frac{F_D}{2}, \frac{F_D}{2}]$  to  $[-\frac{F_D}{2} + F_D \cdot q_{\min}, \frac{F_D}{2} + F_D \cdot (q_{\min} + N_q - 1)]$ . As can be seen, it is enlarged by  $N_q$  times. Users can shift unambiguous Doppler frequency range by configuring  $q_{\min}$  to different values.

Refer to Section 13.3 for how to configure  $a$  and other parameters from the point of view of system registers. However, better performance of the feature can be achieved by applying a good set of configurations. Refer to Section 13.5.1 for general rules of good configurations.

## 13.2. Functional Description

---

### 13.2.1.2 Estimating Ambiguity Factor $q$

Estimating the  $q$  value is a very essential task of de-velocity ambiguity. Equation (13.3) gives us an ideal model of the relationship between  $\frac{\varphi_d(k_{peak}, p_{peak})}{2\pi} - \text{round}\left(\frac{\varphi_d(k_{peak}, p_{peak})}{2\pi}\right)$  and the  $q$  behind. In reality, the received signal is always contaminated somehow by thermal noise or interference, so the realistic model for the signal should be:

$$\varphi = q \frac{T_r + a}{T_D} + \varphi_n - \text{round}\left(q \frac{T_r + a}{T_D} + \varphi_n\right) \quad (13.4)$$

Where:

- $\varphi$  is defined as  $\varphi \triangleq \frac{\varphi_d(k_{peak}, p_{peak})}{2\pi} - \text{round}\left(\frac{\varphi_d(k_{peak}, p_{peak})}{2\pi}\right)$ , representing the angle difference of VEL-FFT output of two sub-sequences from contaminated received signals.
- $\varphi_n$  is the phase caused by noise or interference.

Baseband uses maximum likelihood algorithm to estimate  $q$ . The output of the estimated  $q$  is:

$$\hat{q} = \underset{q}{\operatorname{argmin}} \left\{ \text{abs} \left( \varphi - q \frac{T_r + a}{T_D} - \text{round} \left( q \frac{T_r + a}{T_D} \right) \right) \right\} \quad (13.5)$$

Where  $\text{abs}$  is the function of calculating the *absolute value*. Basically, the resulting  $\varphi$  is compared with  $N_q$  anchor phases defined as  $(q_{\min} + i) \frac{T_r + a}{T_D} - \text{round}\left((q_{\min} + i) \frac{T_r + a}{T_D}\right)$ ,  $i = 0, 1, \dots, N_q - 1$ . The absolute distance between  $\varphi$  and each individual anchor phase is calculated. The  $i$  associated with the smallest absolute distance is selected and  $(q_{\min} + i)$  is output as the estimated result of  $q$ .

Anchor phases should be configured during system initialization. Refer to Section 13.3 for more details.

### 13.2.1.3 Phase Compensation for Virtual Array

The virtual array technology virtualizes more receiving antennas by signal modulation at TX antennas. For TDM mode of virtual array, physical TX antennas transmitting at the same time slot are regarded as the same virtual group. As shown in Fig. 13.3, consecutive TX groups are transmitting normal chirps continuously with an interval of  $T_r$ . Since the main purpose of virtual array is to determine the angle of a received signal by examining the phase differences of received signals of all virtualized antennas. These phase differences are supposed to be only subject to the spacial position of the reflector. However, virtualized receiving antennas are associated with multiple virtual TX groups, which transmit signals in different time slots. So the Doppler frequency of the returning waveform will contribute part of the phase differences in the received signals. That is the reason why we should compensate for this part of phase differences associated with different virtual TX groups before doing DoA estimation.

Let's assume that an object is moving with a Doppler frequency shift equal to  $f_D$ . Then the phase to be compensated for the virtual array group with the index  $v$  is:  $\text{Ph}(v) = \exp(j \cdot 2\pi \cdot f_D \cdot T_r \cdot v)$ .

The peak location of VEL-FFT is denoted by  $p_{peak}$ .

For  $p_{peak} \leq \frac{L_z}{2}$ ,  $p_{peak} = f_{rd} \cdot T_D \cdot L_z$ , so  $f_{rd} = \frac{p_{peak}}{T_D \cdot L_z}$ .

$$\text{Ph}(v) = \exp \left( j \cdot 2\pi \cdot \left( \frac{p_{peak} + q \cdot L_z}{T_D \cdot L_z} \right) \cdot T_r \cdot v \right) \quad (13.6)$$

For  $p_{peak} > \frac{L_z}{2}$ ,  $p_{peak} = f_{rd} \cdot T_D \cdot L_z + L_z$ , so  $f_{rd} = \frac{p_{peak} - L_z}{T_D \cdot L_z}$ .

$$\text{Ph}(v) = \exp \left( j \cdot 2\pi \cdot \left( \frac{p_{peak} - L_z + q \cdot L_z}{T_D \cdot L_z} \right) \cdot T_r \cdot v \right) \quad (13.7)$$

### 13.2.1.4 High Speed Compensation for $p_{peak}$

$p_{peak}$  is the frequency index of VEL-FFT, whose value is a function of  $f_{rd}$  as:

$$p_{peak} = f_{rd} \cdot T_D \cdot L_z, \quad \text{when } f_{rd} \geq 0 \quad (13.8)$$

$$p_{peak} = f_{rd} \cdot T_D \cdot L_z + L_z, \quad \text{when } f_{rd} < 0 \quad (13.9)$$

When the reflector's velocity is slow, the phase compensation for virtual array as defined in Equation (13.6) and Equation (13.7) is valid. However, as velocity increases, or in other words, as the absolute value of the associated  $q$  becomes bigger, the frequency index of VEL-FFT  $p_{peak}$  is not only subject to  $f_{rd}$  as indicated in Equation (13.8) and Equation (13.9), but also subject to the value of  $\frac{2B}{T_u \cdot c} \cdot v \cdot T_s \cdot n$ .

Where:

Table 13.1: Parameter Descriptions

Item	Explanation
$\frac{B}{T_u}$	slop of frequency change within the chirp
$v$	speed of the reflector
$T_s$	sampling period for RNG-FFT
$n$	integer factor that has the same oder of magnitude as the size RNG-FFT

When  $v$  is relatively small, the impact to  $p_{peak}$  can be ignored. But when  $v$  gets big,  $p_{peak}$  should be compensated. The compensation amount is derived from  $\frac{2B}{T_u \cdot c} \cdot v \cdot T_s \cdot n$  and  $n$  is set as half of the window size added before RNG-FFT. The compensation is:

$$\Delta p_{peak} = -\frac{B}{2T_u} \cdot \frac{win1sz \cdot L_z}{f_c \cdot F_s} \cdot q \quad (13.10)$$

### 13.2.1.5 Coexistence with Interference Avoidance and AGC

The Anti-VELAMB CD mechanism can be working simultaneously with 3 interference avoidance modes as described in Section 11.2.1. The extra chirp is treated as normal chirp from the point of view of interference avoidance modes. It can be phase scrambled or frequency hopped or chirp shifted, the compensation of FH/CS/PS will be performed before any Anti-VELAMB CD related operation in baseband to erase the effect of Interference Avoidance. So the interference avoidance procedure is not visible to the Anti-VELAMB CD mechanism.

For AGC, it is similar to what happened for Interference Avoidance. The extra chirp of the Anti-VELAMB CD mechanism is treated by AGC as normal working chirp.

## 13.2.2 Anti Velocity Ambiguity with Multi-Frame

### 13.2.2.1 Mechanism

Another method to solve velocity ambiguity is using multiple frames with different chirp periods. By comparing Doppler frequencies generated after 2D-FFT in different frames, the true velocity can be estimated. For simplicity, only two types of frames with different chirp periods are used to show how to solve velocity ambiguity in following discussion.

Fig. 13.4 shows the principle of how to find the true velocity using two different chirp periods. In Frame 1, the chirp period is  $T_{c1}$ , and the estimated Doppler frequency is  $f_{d1}$ . In Frame 2, the chirp period is  $T_{c2}$ , and the estimated Doppler frequency is  $f_{d2}$ . Due to the ambiguity,  $f_{d1}$  is not equal to  $f_{d2}$ . However, the following equation is valid:

$$f_{d1} + \frac{n1}{T_{c1}} = f_{d2} + \frac{n2}{T_{c2}} = f_D \quad (13.11)$$



## 13.2. Functional Description

---

$f_D$  is the true Doppler frequency, and  $n_1$  and  $n_2$  are integers. By searching for  $n_1$  and  $n_2$  that meets (or approximate) Equation (13.11) in a reasonable range, the true Doppler frequency  $f_D$  (which is proportional to true velocity) can be rightly estimated. Since from 2D-FFT output of different frames, many targets can be detected. It is not certain which target's velocity is too large, and therefore results in velocity ambiguity. So it is critical to match targets from 2 different frames using available information, such as distance, velocity (ambiguity may occur), direction of arrival, power, and so on. Based on that matching relationship and difference of doppler frequencies, it can be concluded whether velocity ambiguity exists for a particular target and true velocity can be estimated.

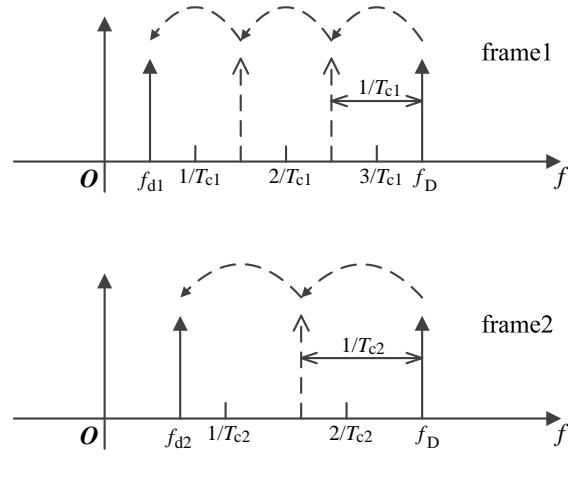


Fig. 13.4: Mechanism of Solving Velocity Ambiguity by Two frames with Different Chirp Periods  $T_c$

The baseband integrated into the Alps chip is easy to be configured in a frame rotate mode (where two types of frames can be interleaved) to solve velocity ambiguity. The two types of frames can be configured with all the same parameters only except chirp periods. See [Section 12](#) for configuration details about frame rotate mode. In both frames, the baseband state machines are set to work until DoA stage is over (see [Section 3](#) for baseband state machines details) to generate two sets of targets information which are to be matched. After the matching process, true velocity can be calculated. The typical processing flow is shown in [Fig. 13.5](#).

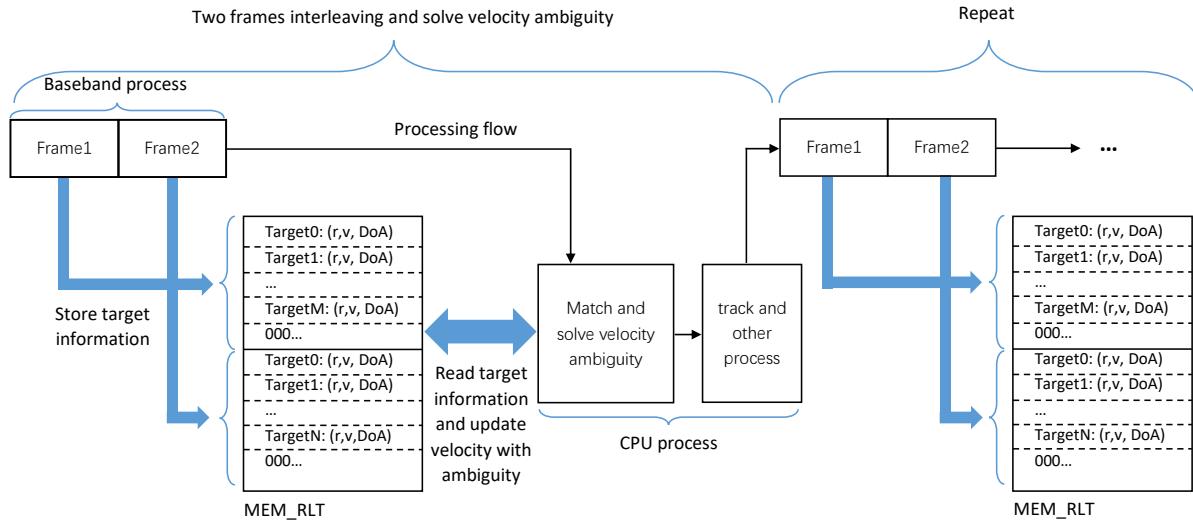


Fig. 13.5: Typical Processing Flow of Alps Chip for Solving Velocity Ambiguity in Frame Interleaving Mode

## 13.3 Programming Interfaces

### 13.3.1 Anti Velocity Ambiguity with Chirp Delay

#### 13.3.1.1 Enabling Anti-VELAMB CD

To activate the anti velocity ambiguity with chirp delay (Anti-VELAMB CD) feature, one needs to configure both radio and baseband. For radio configuration, refer to [Anti Velocity Ambiguity with Chirp Delay \(Anti-VELAMB CD\)](#). Here the focus is on baseband setup.

To enable extra chirp handling in baseband, the register `CFG_SYS_TYPE_FMCW` (0xC0\_001C) should be set.

The total length of the extra chirp and delay, which is  $T_r + a$  when denoted in terms of time in the unit of  $\mu s$ , must be configured in the register `CFG_SAM_DAMB_PRD` (0xC0\_0460) in terms of the number of ADC samples minus 1.

#### 13.3.1.2 Configuring Unambiguous Range of Doppler frequency

As described in [Section 13.2.1.1](#), users can change the unambiguous range of Doppler frequency by configuring a different combination of  $N_q$  and  $q_{min}$ .

$N_q$  is configured in the register `CFG_DoA_DAMB_IDX_LEN` (0xC0\_0A08) with the value  $N_q - 1$ , and  $q_{min}$  is configured in register `CFG_DoA_DAMB_IDX_BGN` (0xC0\_0A04). The supported  $q$  range is defined as  $\{q_{min}, q_{min} + 1, \dots, q_{min} + N_q - 1\}$ . Since the format of the  $q$  value in baseband is FXI(5,  $S$ ),  $q_{min}$  must be so configured that all possible  $q$  values in the range are accommodated by the format FXI(5,  $S$ ), or within the range of [-16, 15].

#### 13.3.1.3 Phase Compensation for Virtual Array

Phase compensation for virtual array is controlled by the register `CFG_DoA_DAMB_TYP` (0xC0\_0A0C). This register contains two bits, so 4 options are valid.

- When the register is set to 0, phase compensation is disabled.
- Setting it to 1 selects the  $q$  value from the Anti-VELAMB CD mechanism as defined in [Section 13.2.1](#).
- Setting it to 2 selects the  $q$  value pre-stored in RAM `MEM_RLT`, whose storage address is shown in [Fig. 3.6](#) with the name `amb_idx`.
- Optionally, users can choose to compensate for the phase by a default  $q$  value configured in the register `CFG_DoA_DAMB_DEFQ` (0xC0\_0A10). In this case, the register `CFG_DoA_DAMB_TYP` (0xC0\_0A0C) should be set to 3.

Table 13.2: `CFG_DoA_DAMB_TYP` Configuration

Value	Behavior
0	bypass phase compensation
1	compensate with $q$ calculated by Anti-VELAMB CD ( <a href="#">Section 13.2.1</a> )
2	compensate with $q$ pre-stored in RAM <code>MEM_RLT</code> , whose storage address is shown in <a href="#">Fig. 3.6</a>
3	compensate with a default $q$ value

The phase compensating in baseband is implemented as multiplying a complex value in unit circle in complex plain with the angle equal to the desired phase. There is a LUT (look-up table) storing the mapping between phases in range  $[0, 2\pi)$  and the corresponding sine values. The size of this mapping table is 2048, so the phase in  $[0, 2\pi)$  is discretized as  $[0, 2047]$ .

The phase to be compensated is shown in [Equation \(13.6\)](#) and [Equation \(13.7\)](#), which involve many multiplication operations. To speed up, the multiplication involving pre-defined system parameters are calculated beforehand and the result is stored in the register `CFG_DoA_DAMB_FDTR` (0xC0\_0A14).

The register `CFG_DoA_DAMB_FDTR` (0xC0\_0A14) should be configured to the **Numeric Value** of  $f_D T_r = \frac{2048 \cdot T_r}{T_D \cdot L_Z}$ .

As discussed in [Section 13.2.2](#), if users want to calculate the  $q$  value in the CPU first, and use that for virtual array phase compensation, the register `CFG_DoA_DAMB_TYP` needs to be set to 2. Then the  $q$  value needs to be written to the exact bit positions with the name `amb_idx` as shown in [Fig. 3.6](#). When DoA is run afterwards, `amb_idx` of each detected object will be fetched by DBF engine for phase compensation.

The  $q$  value is stored in `MEM_RLT` with the name `amb_idx` as a signed integer and located in the second word of a CFAR object memory block. This doubleword can be read or written by the CPU.

---

**Note:** If the register `CFG_DoA_DAMB_TYP` is set to a non-zero value as in [Table 13.2](#), to disable phase compensation conducted in FFT module (see [Section 6.3.5](#)), the register `CFG_FFT_DBPM_ENA` should be set to zero. Otherwise, the phase compensation for virtual array will be done **twice**.

---



### 13.3.1.4 Estimating Ambiguity Factor $q$

As shown in Equation (13.2), calculating  $\varphi_d(k_{\text{peak}}, p_{\text{peak}})$  involves an operation of subtracting  $f_{rd}(a + T_r)$ , where  $f_{rd}(a + T_r)$  involves an operation of calculating  $\frac{2(T_r+a)}{T_D \cdot L_z}$ , which can be calculated beforehand. The pre-calculated result should be set in the register CFG\_DoA\_DAMB\_FRDTRA (0xC0\_0A00).

The anchor phases corresponding to  $\{q_{\min} + 0, q_{\min} + 1, \dots, q_{\min} + N_q - 1\}$  as defined in Section 13.2.1.2 should be configured in the memory defined in Section 3.2.1.2 with CFG\_SYS\_MEM\_ACT (0xC0\_0014) set to 8. The memory address starts from 0x80\_0000 with a length of 128 and is capable of storing 4 banks of anchor phases. For more information about banks, please refer to Section 12. Each bank can store 32 phases, where 32 is the maximum of  $N_q$ .

The format of each anchor phase is FXR(15, 2, S). Each is stored in a memory space of 4 bytes long in little-endian. The memory layout is shown in Fig. 13.6.

bank0	0x000	$(q_{\min}+0)((Tr_0+ao)/TD_0)\text{-round}((q_{\min}+0)((Tr_0+ao)/TD_0))$
	0x004	$(q_{\min}+1)((Tr_0+ao)/TD_0)\text{-round}((q_{\min}+1)((Tr_0+ao)/TD_0))$
	...	...
	$0x04 * (q_{\text{num}} - 1)$	$(q_{\min}+q_{\text{num}}-1)((Tr_0+ao)/TD_0)\text{-round}((q_{\min}+q_{\text{num}}-1)((Tr_0+ao)/TD_0))$
		...
bank1	0x080	$(q_{\min}+0)((Tr_1+a_1)/TD_1)\text{-round}((q_{\min}+0)((Tr_1+a_1)/TD_1))$
		.
		.

Fig. 13.6: Memory Layout of Anchor Phases

### 13.3.1.5 High Speed Compensation for $p_{\text{peak}}$

Register CFG\_DoA\_DAMB\_VELCOM (0xC0\_0A18) should be configured to the **Numeric Value** of velcomp =  $\frac{B \cdot win1sz \cdot L_z \cdot DecFlt}{2 \cdot T_u \cdot f_c \cdot F_S}$ , where  $win1sz$  is the length of the window added in the input of RNG-FFT and  $DecFlt$  is the decimation filter factor.

In baseband, this numeric value will be multiplied by  $q$  to get  $\Delta p_{\text{peak}}$  and compensated to  $p_{\text{peak}}$  before virtual array phase compensation as described in Section 13.2.1.3.

## 13.3.2 Anti Velocity Ambiguity with Multi-Frame

To solve velocity ambiguity using multiple types frames, the baseband needs to be configured to work in 2 frames interleaving mode. The only difference between the interleaving frames should be chirp periods. For configuration details, see programming interfaces for rotate mode in Section 12. And the details about chirp period configurations is in Section 5.

## 13.4 Limitation and Constraints

As discussed in [Section 13.3.1.3](#), one common constraint for both anti velocity ambiguity methods is that, in virtual array mode, if the register `CFG_DoA_DAMB_TYP` is set to a non-zero value as in [Table 13.2](#), the register `CFG_FFT_DBPM_ENA` should be set to zero to avoid duplicated phase compensation in the FFT module .

In addition, for Anti-VELAMB CD mechanism:

- The Anti-VELAMB CD mechanism does not support BPM scheme.
- When virtual array and the Anti-VELAMB CD mechanism coexist, maximum 3 virtual groups are supported.

And for Anti-VELAMB MF mechanism:

- CPU run time cost

The matching process may cost a large amount of CPU process time, so the enough time interval should be reserved before starting the next pair of frames with different chirp periods.

- Mismatch

Due to the power fluctuation, the power of the target power may not be constant in adjacent frames. And obstacles or ground reflection can also generate false targets. So mismatch may happen. Then the estimated velocity may also be incorrect. But one common feature of false targets is that their lifetime across frames is quite short. So the mismatch effect can be mitigated by tracking or clustering method.

- Memory cost

In the provided matching algorithm, candidate chains are constructed for each target, which requires memory space to store them. And the more information is utilized for matching, the more memory space is required.

## 13.5 Software and Configuration Suggestions

### 13.5.1 Anti Velocity Ambiguity with Chirp Delay

Theoretically, parameters of Anti-VELAMB CD configured by the registers as described in [Section 13.3.1](#) can be set arbitrarily. However, if the configurations are coherent to each other, better performance can be achieved. Here are some tips on how to make coherent configurations.

The chirp delay  $a$  is the most fundamental configuration. When  $a$  is settled, a coherent  $N_q$  should be derived from:

$$N_q = \min \left( 32, \frac{T_D}{\gcd(T_r + a, T_D)} \right) \quad (13.12)$$

Where  $\gcd$  is the function of calculating *greatest common divisors* and 32 is the maximum q number the Alps chip can support.

The minimum q value  $q_{min}$  defines the lower bound of unambiguous Doppler frequency. Users can shift this configuration depending on scenarios. If one wants the unambiguous Doppler frequency range to be centered close to zero Doppler,  $q_{min}$  should be configured as  $q_{min} = -1 \cdot \frac{N_q - N_q \% 2}{2}$ . Both  $q_{min}$  or  $q_{min} + N_q - 1$  should be so configured to be within the range of [-16, 15].

Usually, configuration of  $a$  is an iterative procedure. First of all, the value of  $a$  affects the maximum velocity that VEL-FFT can recognize as  $V_{max} = \frac{c}{4f_c((va+1)\cdot T_r+a)}$ .

Together with the configuration of  $q_{min}$  and  $N_q$ , the range of unambiguous velocity is  $[(2q_{min} - 1)V_{max}, (2(q_{min} + N_q - 1) + 1)V_{max}]$ .

If the range cannot satisfy the system requirement, a second round or more rounds of parameter adjustment are needed.

An example of coherent configuration would be like:

Table 13.3: Anti-VELAMB CD Configuration for  $va = 2$

Item	Value	Item	Value
$T_r$	$30 \mu s$	$N_q$	9
$a$	$18 \mu s$	$q_{min}$	-4

Under this set of configuration, there will be 9 evenly separated anchor phases as shown in Fig. 13.7.

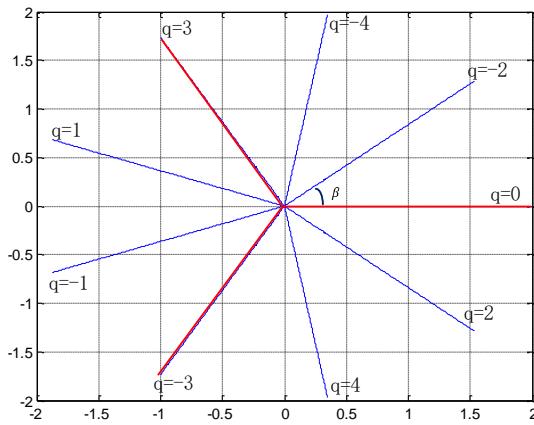


Fig. 13.7: Coherent Configuration:  $va=2 / Tr=30 / a=18 / N_q = 9$

As an incoherent configuration example, if we change  $a$  from  $18 \mu s$  to  $11 \mu s$  and do not change other remaining parameters, the distribution of anchor phases will look like Fig. 13.8.

### 13.5. Software and Configuration Suggestions

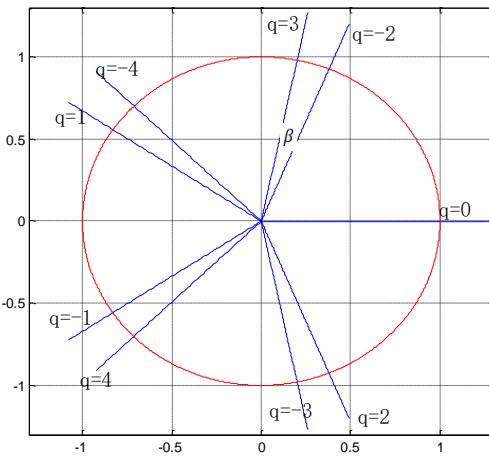


Fig. 13.8: Incoherent Configuration:  $va=2$  /  $Tr=30$  /  $a=11$  /  $N_q = 9$

If we use  $\beta$  to denote the smallest anchor phase separation, it can be seen that  $\beta$  of the incoherent configuration is much smaller than that of the coherent one. Since the SNR requirement of the feature is dominated by  $\beta$ , the SNR performance of the coherent one is much better than that of the incoherent one. The SNR requirement can be derived as:

$$snr = -20 \log_{10} \left( \frac{\beta}{2} \right) \quad (13.13)$$

The firmware of Anti-VELAMB CD involves initialization of corresponding registers as described in Section 13.3.1. Example code is as follows:

```
void init_velamb_CD() {
    int va      = 2;      /* virtual array with 2 TX */
    int Tr      = 30;     /* unit us, 1e-6s */
    int Tu      = 25;     /* unit us, 1e-6s */
    int delay   = 18;     /* unit us, 1e-6s */
    int Fs      = 25;     /* sample frequency, unit is MHz */
    int DecFltr = 1;     /* decimate filter rate */
    int B       = 675;    /* bandwidth, unit is MHz */

    /* half of window length added in the input of RNG-FFT */
    /* Win1Sz / 2 */
    int n_half = 512 / 2; /* half of window length added in the input of RNG-FFT */
    int fc      = 77;      /* frequency of wWave, unit in GHz */

    /* virtual array period, sampling interval for input of vel-FFT */
    int TD = (va + 1) * Tr + delay;

    int q_num = min(32, TD / gcd(Tr + delay, TD)); /* Nq, should be 9 here */
    int q_min = -1 * (q_num - q_num % 2) / 2;        /* qmin */

    int nfft2 = 256;
    float frd_tr_a = 2.0 * (Tr + delay) / TD / nfft2;
    float frd_tr   = 2048.0 * Tr / (TD * nfft2);
}
```

(continues on next page)

(continued from previous page)

```

/* 1000 is used to make the unit of numerator is identical to that
   of denominator */
float vel_comp = 1.0 * B * n_half * nfft2 * DecFltr / (Tu * Fs * fc * 1000);

/* enable velamb-CD feature */
write_reg(CFG_SYS_TYPE_FMCW, 1);

/* compensate for MIMO phases using q calculated by velamb-CD */
write_reg(CFG_DoA_DAMB_TYP, 1);

/* samples of extra chirp plus delay */
write_reg(CFG_SAM_DAMB_PRD, (Tr + delay) * Fs / DecFltr - 1);
write_reg(CFG_DoA_DAMB_IDX_LEN, q_num - 1); /* q number */
write_reg(CFG_DoA_DAMB_IDX_BGN, q_min); /* minimum q */
write_reg(CFG_DoA_DAMB_FRDTRA, float_to_fx(frd_tr_a, 15, -2, false));
write_reg(CFG_DoA_DAMB_FDTR, float_to_fx(frd_tr, 16, 4, false));
write_reg(CFG_DoA_DAMB_VELCOM, float_to_fx(vel_comp, 16, 1, false));

/* Anchor Phase setting */
write_reg(CFG_SYS_MEM_ACT, 8);
int bank_idx = 0;
int bank_offset = bank_idx * 32 * 4;

for (int idx = 0; idx < q_num; idx++) {
    int q = q_min + idx;
    float anchor_phase = 2 * q * (Tr + delay) / TD;
    while (anchor_phase >= 1) {
        anchor_phase -= 2;
    }
    while (anchor_phase < -1) {
        anchor_phase += 2;
    }
    int addr = 0xE00000 + bank_offset + 4 * idx;
    write_reg(addr, float_to_fx(anchor_phase, 15, 2, true));
}
}
}

```

### 13.5.2 Anti Velocity Ambiguity with Multi-Frame

- Choosing proper chirp periods

Given 2 successfully matched targets with Doppler frequencies  $f_{d1}$  and  $f_{d2}$ , searching for integers  $n1$  and  $n2$  to approximate Equation (13.11) equals minimizing the following function  $D(n1, n2)$ :

$$D(n1, n2) = |(f_{d1} - f_{d2}) \cdot T_{c1} \cdot T_{c2} + n1 \cdot T_{c2} - n2 \cdot T_{c1}| \quad (13.14)$$

To ensure only the correct  $(n1, n2)$  corresponds to the minimum of  $D(n1, n2)$ , do not choose  $T_{c1}$  and  $T_{c2}$  that let  $n1 \cdot T_{c2} - n2 \cdot T_{c1}$  have the same value given different pairs of  $(n1, n2)$ . A bad example is to choose  $T_{c2} : T_{c1} = 4 : 5$ , thus  $(n1, n2) = (-3, -3)$  and  $(n1, n2) = (2, 1)$  will result in the same  $D(n1, n2)$  value, so if  $(n1, n2) = (-3, -3)$  is the correct pair to solve velocity ambiguity,  $(n1, n2) = (2, 1)$  will also be.

- Working with TDM virtual array mode

When baseband is configured in virtual array mode, velocity will introduce additional phase shift among TX groups. Set CFG\_DoA\_DAMB\_TYP to 2 and update target velocity ambiguity result in MEM\_RLT, and base-



## 13.6. Examples

---

band will automatically compensate for the additional phase shift when running DBF. The configuration details are shown in [Section 13.3.1.3](#).

---

**Note:** The register `CFG_DoA_DAMB_FDTR` needs to be configured with a correct Doppler chirp period, i.e.  $T_D = N_{va} \cdot T_r$ , which is different from what in the chirp delay scheme.

---

## 13.6 Examples

### 13.6.1 Anti Velocity Ambiguity with Chirp Delay

While [Section 13.5.1](#) illustrates a coherent configuration of Anti-VELAMB CD, the current section will use that as an example to further explain the general procedure of creating a proper configuration for Anti-VELAMB CD. The most essential part of configuring Anti-VELAMB CD is to select a pair of matching  $T_r$  and  $a$ . All the remaining parameters can be derived from these two and other basic system parameters.

First of all, let's assume that some basic system configuration is settled as:

Table 13.4: Pre-Configuration for Anti-VELAMB CD

Item	Value	Item	Value
$N_{va}$	2	$T_r$	$28 \mu s$
$T_u$	$25 \mu s$	$T_d$	$2 \mu s$
$F_s$	25 Mhz	Decimate Filter Factor	1
Rng-FFT size	512	VEL-FFT size	128
$win1Sz$	512	$f_c$	76 Ghz

Let's assume that we need a radar system that can detect moving objects whose velocity range is from -80 m/s to +80m/s and the working SNR is higher than 13 dB.

From Anti-VELAMB CD's point of view, the most fundamental configuration is the delay  $a$ . However, since  $T_r$  will directly impact the range of unambiguous velocity, which is the ultimate goal of the anti velocity ambiguity feature, we might adjust  $a$  and  $T_r$  at the same time.

[Table 13.5](#) summarizes the impact of different configuration of  $a$  and  $T_r$ .

Table 13.5: Impact of  $a$  and Other Parameters

In-index	Parameter	Description
1	$V_{max} = \frac{c}{4f_c((va+1) \cdot Tr + a)}$	The maximum velocity can be recognized by VEL-FFT.
2	$N_q$ , as defined in <a href="#">(13.12)</a>	$N_q$ should not be too big. Otherwise, a very high working SNR is required.
3	$q_{min} = -1 \cdot \frac{N_q - N_q \% 2}{2}$	The required velocity range is $[-80, 80]$ , with the center being zero.
4	Maximum Negative Velocity (MNV) $(2q_{min} - 1) V_{max}$	This value must be less than or equal to -80.
5	Maximum Positive Velocity (MPV) $(2(q_{min} + N_q - 1) + 1) V_{max}$	This value must be greater than or equal to 80.

By iteratively searching different combinations of  $a$  and  $T_r$ , we will get different  $N_q$ ,  $MNV$  and  $MPV$ . And we will verify whether  $MNV$  and  $MPV$  can satisfy the required range of  $[-80, 80]$ , and check whether SNR requirement is

a doable condition as defined in (13.13).

The script for finding a set of proper configuration is shown as below:

```

fc = 76e9;
c = 3.0e8;
va = 2;

snr = 13; % db
vMin = -80;
vMax = 80;

for Trd = [27:35]
    for ad = [1:30]
        Tr = Trd * 1e-6;
        a = ad * 1e-6;

        TD = (va+1)*Tr + a;
        TDD = (va+1)*Trd + ad;

        VMax = c / (4 * fc * TD);

        Nq = min(32, TDD / gcd(Trd+ad, TDD));
        q_min = -1 * (Nq - mod(Nq, 2)) / 2;

        MNV = (2 * q_min - 1) * VMax;
        MPV = (2*(q_min+Nq-1)+1) * VMax;

        if -20*log10(pi/Nq) > snr
            continue;
        end

        if MNV > vMin
            continue;
        end

        if MPV < vMax
            continue;
        end

        fprintf('\nTr= %f, a = %f', Trd, ad);
        fprintf('\nNq = %d', Nq);
        fprintf('\nMNV = %f', MNV);
        fprintf('\nMPV = %f', MPV);
    end
end

```

As can been seen, 13 combinations of  $a$  and  $T_r$  pass the script. However, if we decrease the working SNR requirement from 13 dB down to 10 dB, only one combination passes, which is that  $a$  equals  $18 \mu s$  and  $T_r$  equals  $30 \mu s$ . That is exactly the example we used in [Section 13.5.1](#).

So we update  $T_r$  from  $28 \mu s$  to  $30 \mu s$  and configure other parameters (such as  $N_q$ ). The programming procedure is described in [Section 13.3.1](#).



## 13.6. Examples

---

### 13.6.2 Anti Velocity Ambiguity with Multi-Frame

#### 13.6.2.1 Multi-Frame Interleaving Settings

In this example,  $T_{c1}$ , the chirp period of Frame 1 is set to  $40\mu\text{s}$ .  $T_{c2}$ , the chirp period of Frame 2 is set to  $50\mu\text{s}$ . The center frequency is  $f_C = 76 \text{ GHz}$ . See examples in [Section 12](#) and [Section 5](#) for frame interleaving configuration and chirp period configuration. The relationship between the target velocity  $v$  and Doppler frequency  $f_d$  is

$$f_d = \frac{-2 \cdot v \cdot f_C}{c} \quad (13.15)$$

where  $c$  is the light speed.

The maximum Doppler frequency can be measured is  $f_d = \pm 1/(2 \cdot T_c)$ .

Therefore, it can be calculated that velocity ranges without ambiguity for Frame 1 and Frame 2 are  $\pm 24.7 \text{ m/s}$  and  $\pm 19.7 \text{ m/s}$ , respectively.

#### 13.6.2.2 Example Matching Algorithm

After obtaining targets information by CFAR, we need to match the targets from two interleaved frames. There are many kinds of algorithms for targets matching. An example matching algorithm that is already verified is presented here.

Use  $T_A$  to denote the set of targets from Frame A and  $T_B$  to denote the set of targets from Frame B, and  $M$  and  $N$  are the numbers of targets detected in Frame A and Frame B respectively. Then:

$$\begin{aligned} T_A &= A_1, A_2, \dots, A_M \\ T_B &= B_1, B_2, \dots, B_N \end{aligned}$$

For any two targets  $A_m$  and  $B_n$  from different frames, define a function  $D(A_m, B_n)$  to measure the degree of their information difference. So the difference function  $D(A_m, B_n)$  is negatively correlated with their likelihood of matching.

The purpose of matching is to find as many pair of targets  $(A_m, B_n)$  from target sets  $T_A$  and  $T_B$  as possible. And after all the matching relationships are established, the sum of difference functions of all matched target pairs  $(A_m, B_n)$ , which expressed in Expression [\(13.16\)](#), should be the smallest.

$$\sum_{(m,n)} D(A_m, B_n) \quad (13.16)$$

The following describes the matching steps:

1. Select the type of target information to use for matching.

For the target  $A_m$  detected in Frame A,  $R(A, m)$  denotes distance,  $Pow(A, m)$  denotes power, and  $Ang(A, m)$  denotes direction of arrival.

For the target  $B_n$ , the corresponding denotations are  $R(B, n)$ ,  $Pow(B, n)$ , and  $Ang(B, n)$ .

Based on this, the difference function  $D(A_m, B_n)$  can be defined as:

$$D(A_m, B_n) = w_1 \cdot |R(A, m) - R(B, n)| + w_2 \cdot |Pow(A, m) - Pow(B, n)| + w_3 \cdot |Ang(A, m) - Ang(B, n)| \quad (13.17)$$

Where  $||$  means taking the absolute value, and  $w_1$ ,  $w_2$ , and  $w_3$  are adjustable weight coefficients for various types of information difference.

## 2. Construct candidate targets chains.

For each target  $A_m$ , select  $K$  candidate matching targets  $B_{n1}, B_{n2}, \dots, B_{nk}, \dots, B_{nK}$  from Frame B.

Each candidate matching target  $B_n$  should satisfy the following conditions:

$$\begin{cases} |R(A, m) - R(B, n)| < R_{\text{th}} \\ |Pow(A, m) - Pow(B, n)| < Pow_{\text{th}} \\ |Ang(A, m) - Ang(B, n)| < Ang_{\text{th}} \end{cases} \quad (13.18)$$

Where  $R_{\text{th}}$ ,  $Pow_{\text{th}}$ , and  $Ang_{\text{th}}$  are thresholds to narrow down the search range during matching process.

$K$  is the length of the chain and is adjustable.

- If less than  $K$  targets in Frame B satisfy the above threshold conditions, select as many candidate targets as possible.
- If more than  $K$  targets in Frame B satisfy the conditions, select  $K$  candidate targets with the smallest values of difference functions  $D(A_m, B_n)$ , or with maximum matching likelihood.

Then arrange the  $K$  candidate targets  $B_{n1}, B_{n2}, \dots, B_{nk}, \dots, B_{nK}$  in an ascent order in respect of the value of the difference function  $D(A_m, B_{nk})$ . For example, if

$$D(A_m, B_{n1}) < D(A_m, B_{n2}) < \dots < D(A_m, B_{nk}) < \dots < D(A_m, B_{nK})$$

then arrange  $A_m$ 's candidate targets as

$$B_{n1}, B_{n2}, \dots, B_{nk}, \dots, B_{nK}$$

which is called  $A_m$ 's candidate target chain. Fig. 13.9 shows an example of how a candidate target chain is constructed.

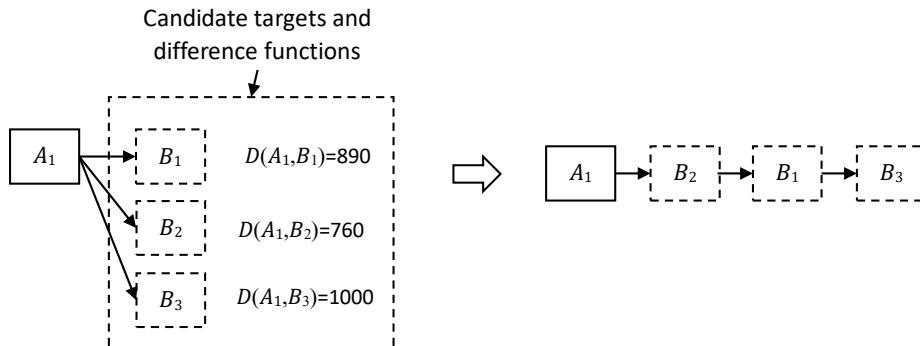


Fig. 13.9: Candidate Target Chain for a Target in Frame A

Similarly, for each target  $B_n$  from Frame B, construct the candidate targets chain in the same way.

After selecting candidate matching targets for all targets, the chain array is as shown in Fig. 13.10.

### 13.6. Examples

---

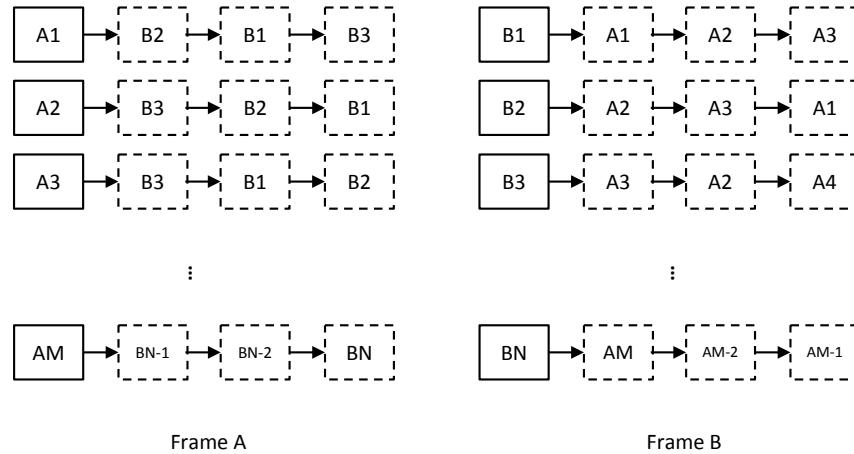


Fig. 13.10: Candidate Target Chains for All Targets in Both Frames

#### 3. Define 4 functions before matching process

- nearest()

$\text{nearest}(A_m)$  returns the candidate matching target with the smallest value of the difference function  $D(A_m, B_n)$ .

For example, if  $D(A_m, B_{n1}) < D(A_m, B_{n2}) < \dots < D(A_m, B_{nk}) < \dots < D(A_m, B_{nK})$ , then  $\text{nearest}(A_m) = B_{n1}$ . If  $B_{n1}$  is removed from the candidate target chain, then  $\text{nearest}(A_m) = B_{n2}$ . If no targets from Frame B satisfy the threshold conditions in Equation (13.18), or all targets in the candidate target chain are removed, then  $\text{nearest}(A_m) == \text{NULL}$ , which means that no targets from Frame B can match  $A_m$ .

For any two targets  $A_m$  and  $B_n$  from Frame A and Frame B, the conditions they are matched with each other are:

$$\begin{cases} \text{nearest}(A_m) = B_n \\ \text{nearest}(B_n) = A_m \end{cases} \quad (13.19)$$

- match()

$\text{match}(A_m)$  returns a bool value indicating whether  $A_m$  is successfully matched with any targets from Frame B. If  $A_m$  is successfully matched,  $\text{match}(A_m) = 1$ . Otherwise,  $\text{match}(A_m) = 0$ .

$\text{match}(B_n)$  returns a bool value indicating  $B_n$ 's matching state in the same way.

- pairA()

$\text{pairA}(A_m)$  is to find a target from Frame B to match  $A_m$ . The program flow diagram is shown in Fig. 13.11.

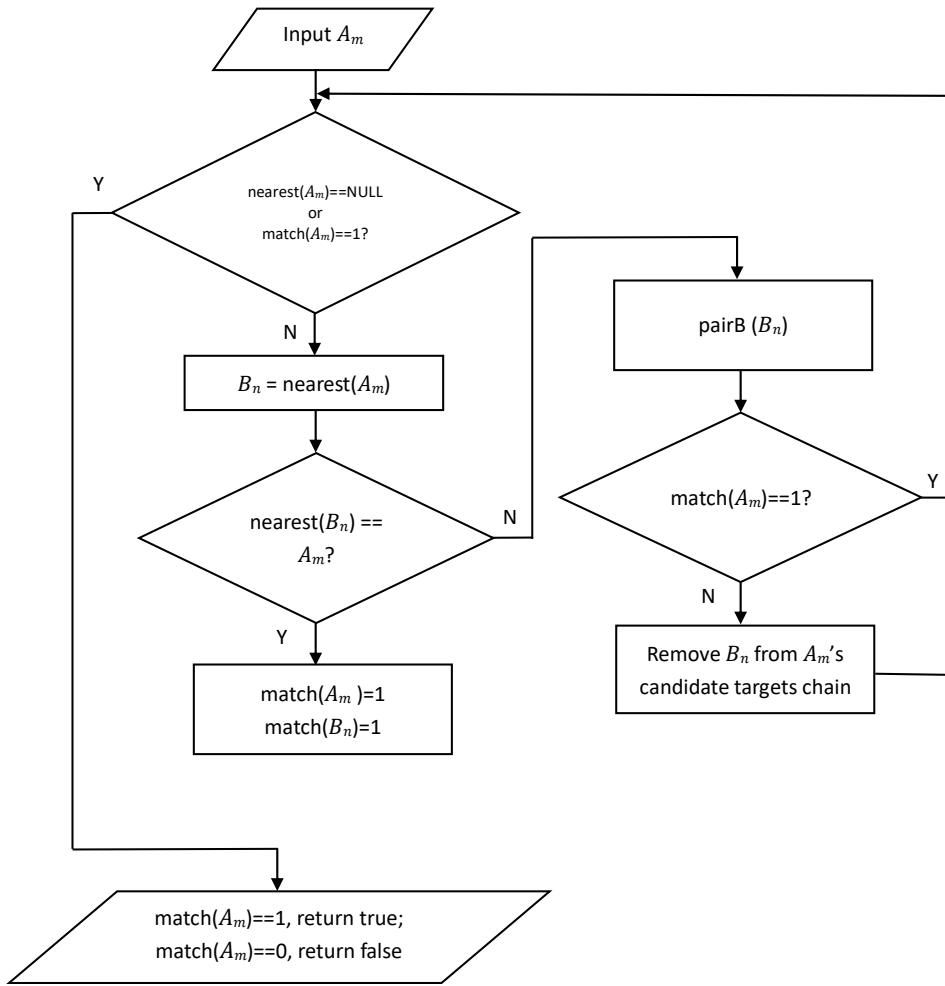


Fig. 13.11: Program Flow Diagram of Function pairA()

- pairB()

pairB(B<sub>n</sub>) is to find a target from Frame A to match B<sub>n</sub>. The program flow diagram is shown in Fig. 13.12.

### 13.6. Examples

---

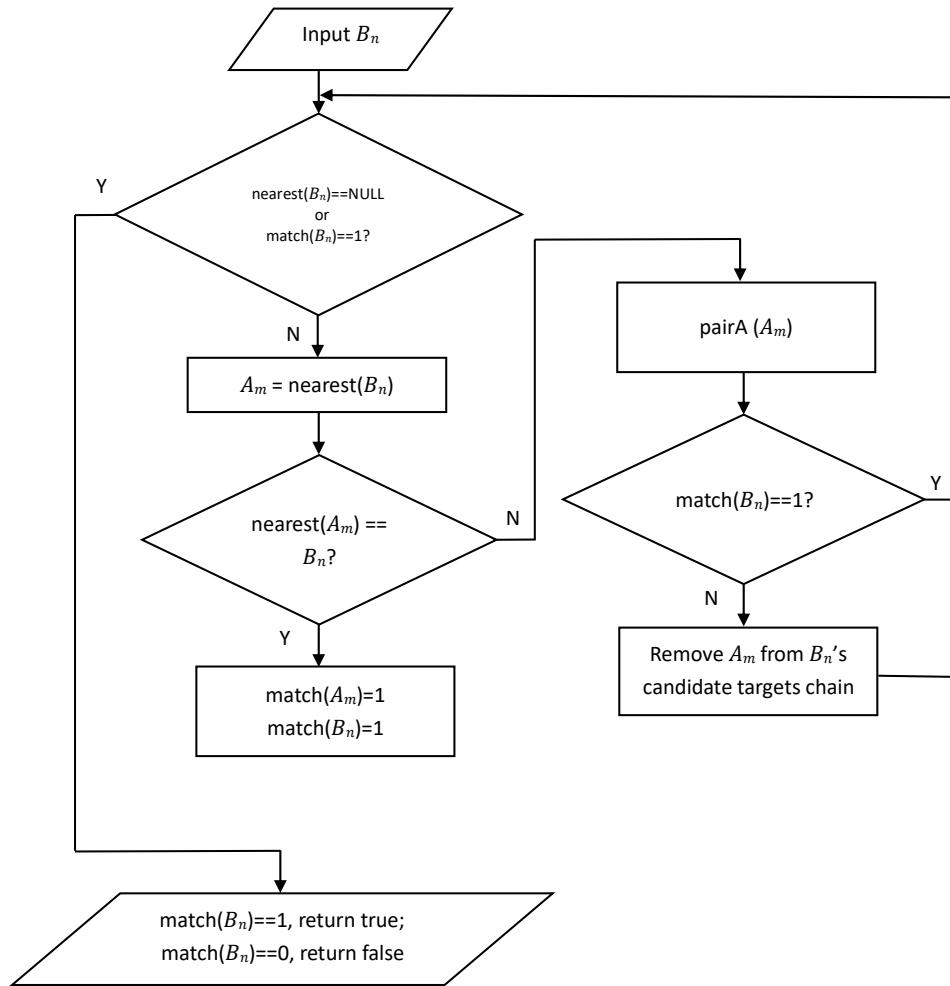


Fig. 13.12: Program Flow Diagram of Function pairB()

Fig. 13.13 shows how to remove one candidate target from candidate target chain which involved in pairA() and pairB().

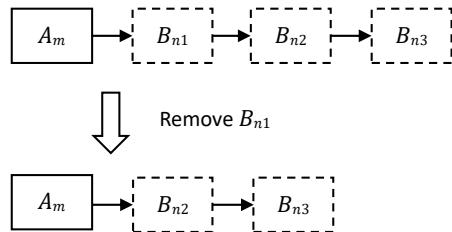


Fig. 13.13: Remove One Candidate Target from  $A_m$ 's Candidate Targets Chain

## 4. Matching process.

The program flow diagram of the matching process is shown in Fig. 13.14 .

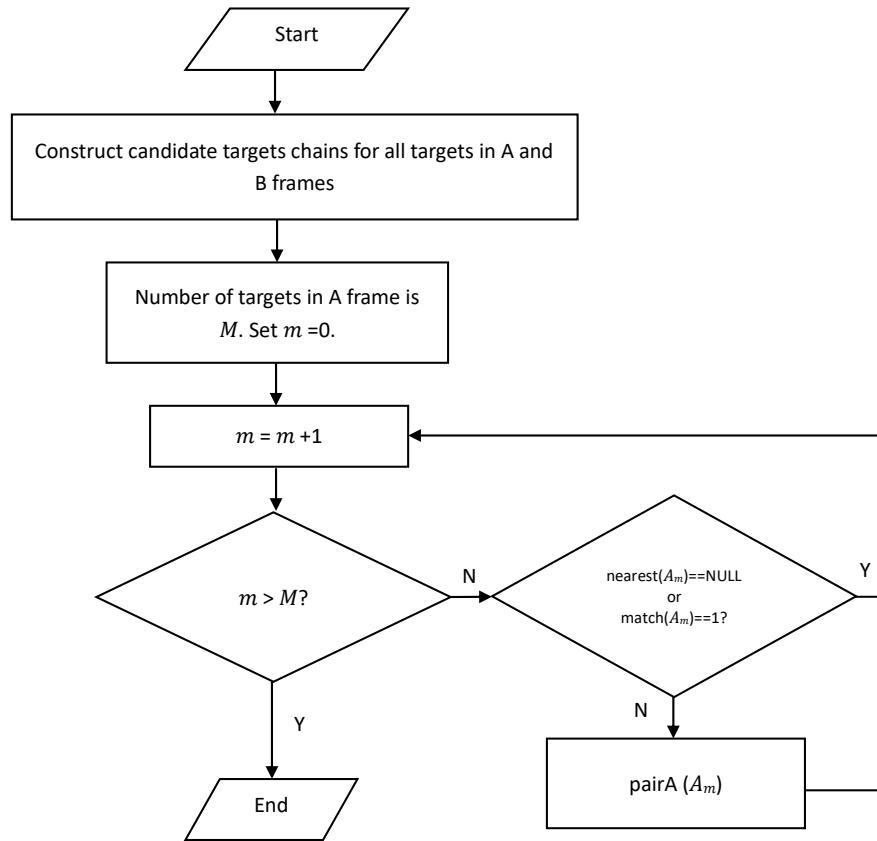


Fig. 13.14: Program Flow Diagram of Matching Process

### 13.6.2.3 Solving Velocity Ambiguity

Assume that the estimated Doppler frequencies of one pair of matched targets from Frame 1 and Frame 2 are  $f_{d1} = 8\text{kHz}$  and  $f_{d2} = 12.5\text{kHz}$ . Then velocity is solved by searching proper integers  $n_1$  and  $n_2$  to satisfy Equation (13.11). The searching range for choosing  $n_1$  and  $n_2$  is  $[-2, 2]$ .

According to parameters described in Section 13.6.2.1, the best choice to meet Equation (13.11) is  $n_1 = 1$  and  $n_2 = 1$ , and the true Doppler frequency is approximately  $f_D = 33\text{kHz}$ . The corresponding true velocity is 65.1 m/s.

### **13.6. Examples**

---



## NOISE ESTIMATOR

Alps provides an estimate of noise power for each range gate during each FMCW frame. The noise estimation results are used in the CFAR module (Section 7).

### 14.1 Features

- Global noise estimate per range gate
- Flexible combination of RX channels

#### 14.1.1 Functional Description

Noise estimation is done after 1D-FFT and parallel with Doppler-gate FFT. So it does not increase the latency of processing time.

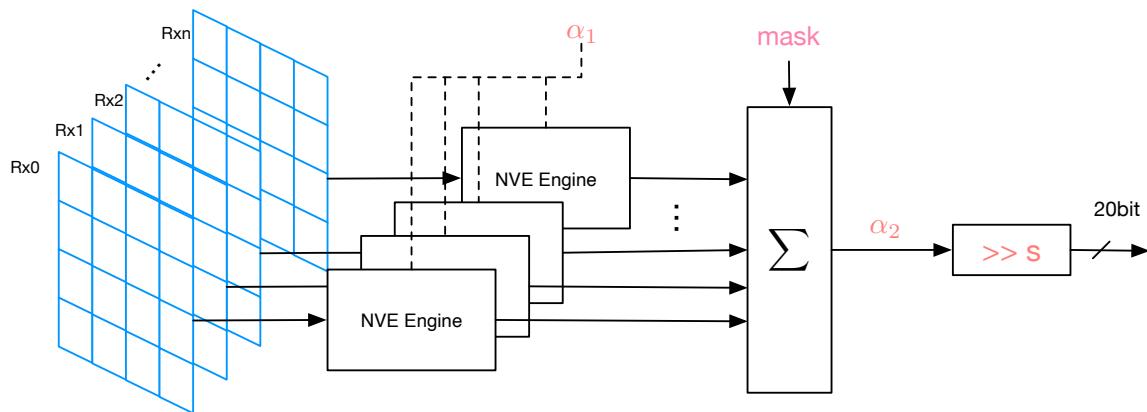


Fig. 14.1: Noise Estimator Diagram

Fig. 14.1 briefly shows the internal structure of the noise estimator. For each RX channel, including virtualized channels, a noise estimate is computed for each range gate. The final results are generated by summing across different channels, multiplying a programming scalar  $\alpha_2$ , and right-shifting a programmable shifter  $s$ :

- Scalar  $\alpha_1$  should be given to NVE engines. It should be always the inverse of the number of chirps per frame per RX channel. For example, the total number of chirps in a frame is 256 and the system uses 2-TX virtual array. Then  $\alpha_1 = \frac{1}{256 \div 2} = \frac{1}{128}$ .

## 14.1. Features

---

- The mask in Fig. 14.1 is programmable to give flexibility in choosing subset of channels for summation.
- Scalar  $\alpha_2$  and shifter  $s$  are also programmable to scale the noise level according to applications.

### 14.1.2 Programming Interfaces

The control of Noise Estimator is limited to following registers:

- CFG\_FFT\_NVE\_SCL\_0: It corresponds to  $\alpha_1$  in Fig. 14.1 and is in the format of FLR(6, 0,  $U$ , 3,  $U$ ).
- CFG\_FFT\_NVE\_SCL\_1: It corresponds to  $\alpha_2$  in Fig. 14.1 and is in the format of FLR(6, 0,  $U$ , 3,  $U$ ).
- CFG\_FFT\_NVE\_SFT: It corresponds to  $s$  in Fig. 14.1 and is in the format of FXI3,  $U$ .
- CFG\_FFT\_NVE\_CH\_MSK: It corresponds to mask in Fig. 14.1 and is in the format of FXI16,  $U$ . Its LSB indicates RX 0.

Noise estimates in the format of FLR(15, 1,  $U$ , 5,  $U$ ) can be accessed from CPU.

### 14.1.3 Limitation and Constraints

- Support up to 16 RX Channels.

### 14.1.4 Software and Configuration Suggestions

- Although one can choose an arbitrary value for scalar  $\alpha_2$ , it is recommended to do following so that the noise estimates have the same level as CFAR combination outputs.
  - $\alpha_2$  should be given by a product of two factors, where  $\alpha_2 = \alpha'_2 * \alpha''_2$ .
  - $\alpha'_2$  should be the inverse of number of 1's in *mask* in order to convert summation to average.
  - $\alpha''_2$  should be the power of the window used before Doppler-gate FFT. The power of  $win[n]$  is  $\frac{1}{N} \sum_{i=1}^N win[i]^2$ , where  $n = 1, \dots, N$ .
- Shifter  $s$  is recommended to be zero unless the estimates are significantly above the CFAR noise floor.
- If one needs noise estimation in the post processing at CPU level, one should note that the current noise estimator may output some overshooting spikes (Fig. 14.2) due to large moving objects. Therefore, it is advised to exclude these large values in one way or another.



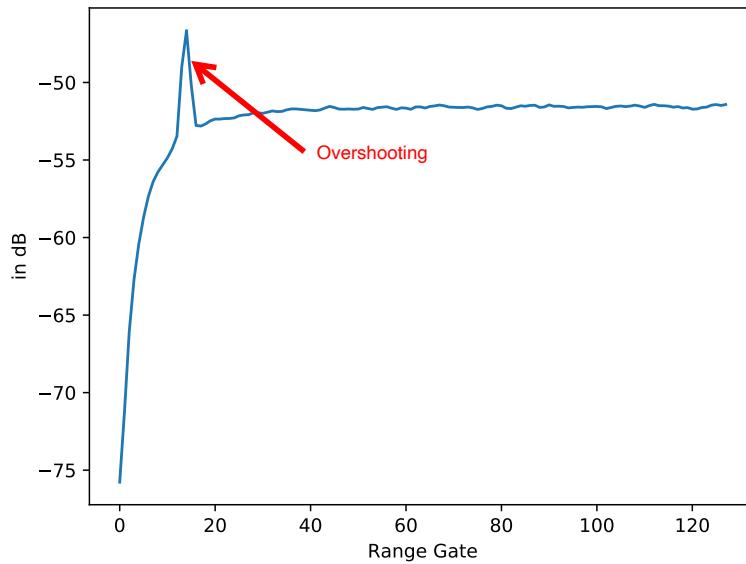


Fig. 14.2: Overshooting of Noise Estimator

### 14.1.5 Examples

Suppose that a radar system has 8 RX channels, the number of chirps per channel is 128, and the power of window ( $\frac{1}{N} \sum_{i=1}^N \text{win}[i]^2$ ) used in the Doppler gate is 0.68. The corresponding register values are:

Table 14.1: Example Settings for DBF

Register Name	Address	Value	Comments
CFG_FFT_NVE_SCL_0	0xC0065C	0x7F	1/128 in format of FLR(6, 0, U, 3, U)
CFG_FFT_NVE_SCL_1	0xC00660	0xB7	0.68/128 in format of FLR(6, 0, U, 3, U)
CFG_FFT_NVE_SFT	0xC00664	0x0	No additional shifting
CFG_FFT_NVE_CH_MSK	0xC00668	0xFF	Select 8 RX Channels

## **14.1. Features**

---



## DATA COLLECTION AND HARDWARE-IN-THE-LOOP (HIL)

### 15.1 Overview

This chapter is focused on data debugging, which includes:

- data collection
- hardware-in-the-loop (HIL)

Data collection is to dump baseband processing data to memory or GPIO or LVDS, which can help the designers to debug the baseband processing. There are 2 types of data collection, which are:

- standard output data collection
- debug data collection

The difference between these 2 types is that the standard output data will be buffered directly and the debug data will not.

Hardware-in-the-loop, or HIL, is a technique that is used in the development and testing of complex real-time embedded systems. HIL simulation provides an effective platform by adding the complexity of the plant under control to the test platform. HIL is to input data to baseband and verify whether the baseband IP on the chip is correct or not. HIL can be used for sample playback and this will also help the designers to debug the baseband processing.

#### 15.1.1 Features

- **Standard Output Data Collection:** Collects the SAMPLE or 1D-FFT or 2D-FFT standard output data to GPIO or LVDS;
- **Extra Debug Data Collection:** Collects SAMPLE, CFAR or DoA debug data to GPIO, LVDS, or MEM\_BUF;
- **HIL:** Supports 2 ways to input data to baseband, CPU AHB\_BUS input and GPIO input.

### 15.2 Functional Description

#### 15.2.1 Standard Output Data Collection

As shown in the following, there are 3 sources of standard output data, corresponding to 3 baseband processing states.

- SAMPLE output data
- 1D-FFT output data
- 2D-FFT output data

## 15.2. Functional Description

---

The standard output data is first buffered to baseband memory and then dumped to GPIO or LVDS interface.

### 15.2.1.1 Data Collection Block Diagram

Data collection goes through two main steps, as shown in Fig. 15.1.

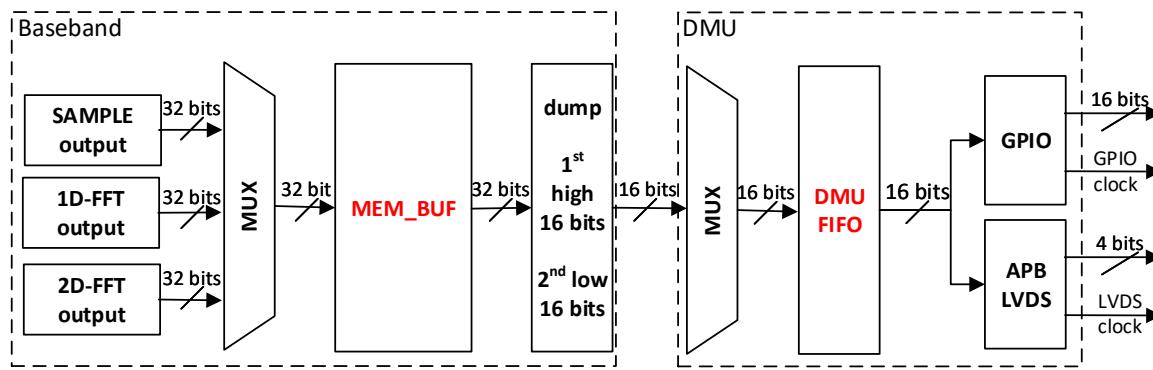


Fig. 15.1: Data Collection of Standard Output Flow

1. Buffering the data to MEM\_BUF.

The 3 data sources (SAMPLE, 1D-FFT and 2D-FFT) are all buffered to MEM\_BUF, so 1D-FFT output data will overwrite SAMPLE output data, and 2D-FFT output data will overwrite 1D-FFT output data.

2. Dumping the MEM\_BUF data to GPIO or LVDS.

When data is ready in MEM\_BUF, the next action is to dump the data to GPIO or LVDS. Either GPIO or LVDS can be selected at a time. The states DMP\_MID and DMP\_FNL are dedicated to this action.

### 15.2.1.2 SAMPLE Output Data

#### 15.2.1.2.1 SAMPLE Output Data Format

SAMPLE data width is 16 bits. Considering the data width of MEM\_BUF is 32 bits, 2 SAMPLE outputs are saved in one MEM\_BUF unit. The data format is shown in Table 15.1

Table 15.1: Standard Output Data Format

Data Source	Bit Width	Data Format	Memory Unit (32 Bits)
SAMPLE	16	FXR(16, 2, S)	the 1st output in the upper 16 bits; the 2nd output in the lower 16 bits
1D-FFT	32	CFL(14, 1, S, 4, U)	1 set of data in a 32-bit unit
2D-FFT	32	CFL(14, 1, S, 4, U)	1 set of data in a 32-bit unit

### 15.2.1.2.2 SAMPLE Output Data Size

The size of SAMPLE output data is determined by the registers in Table 15.2.

Table 15.2: Registers for Data Collection of SAMPLE Output Data

Register Name	Example Value
CFG_SYS_SIZE_RNG_FFT	511
CFG_SYS_SIZE_BPM	0
CFG_SYS_TYPE_FMCW	1
CFG_SYS_SIZE_VEL_FFT	255
CFG_SYS_SIZE_RNG_BUF	511
CFG_SYS_SIZE_VEL_BUF	255

The maximum size is 2 MB, which is limited by the size of MEM\_BUF (2 MB). The size is computed by  $V \times T \times A \times R \times W$ . The definitions are in Table 15.3.

Table 15.3: Data Size of SAMPLE Output Data

Abbr	Name	Definition	Description
V	SYS_SIZE_VEL_FFT	CFG_SYS_SIZE_VEL_FFT + 1	size of 2D-FFT in Doppler dimension
T	SYS_SIZE_VIRTUAL	CFG_SYS_SIZE_BPM + 1 + CFG_SYS_TYPE_FMCW	size of virtual array
A	SYS_SIZE_ANT	4	default antenna size
R	SYS_SIZE_RNG_FFT	CFG_SYS_SIZE_RNG_FFT + 1	size of 1D-FFT in range dimension
W	DATA WIDTH	16 bits	width of SAMPLE data

Table 15.4 shows an example of SAMPLE output data size, using the example values in Table 15.2.

Table 15.4: SAMPLE Standard Output Data Size Example

Data Source	Computation	Data Size
SAMPLE	$256 \times 2 \times 4 \times 512 \times 16 \text{ bits}$	2 MB

The registers CFG\_SYS\_SIZE\_RNG\_BUF and CFG\_SYS\_SIZE\_VEL\_BUF do not change the output data size, but affect the data value of SAMPLE output.

When dumping SAMPLE output data, it is suggested that CFG\_SYS\_SIZE\_VEL\_BUF should be configured to be the same value with CFG\_SYS\_SIZE\_VEL\_FFT. Otherwise, the SAMPLE output data will go wrong.

It is also suggested that CFG\_SYS\_SIZE\_RNG\_BUF should be configured to a value equivalent to or approximately equivalent to CFG\_SYS\_SIZE\_RNG\_FFT. As shown in Fig. 15.2, if  $\text{CFG\_SYS\_SIZE\_RNG\_BUF} < \text{CFG\_SYS\_SIZE\_RNG\_FFT}$ , zero padding will be performed to fill up CFG\_SYS\_SIZE\_RNG\_FFT. If  $\text{CFG\_SYS\_SIZE\_RNG\_BUF} > \text{CFG\_SYS\_SIZE\_RNG\_FFT}$ , data cutting will be performed to adapt to CFG\_SYS\_SIZE\_RNG\_FFT.



## 15.2. Functional Description

---

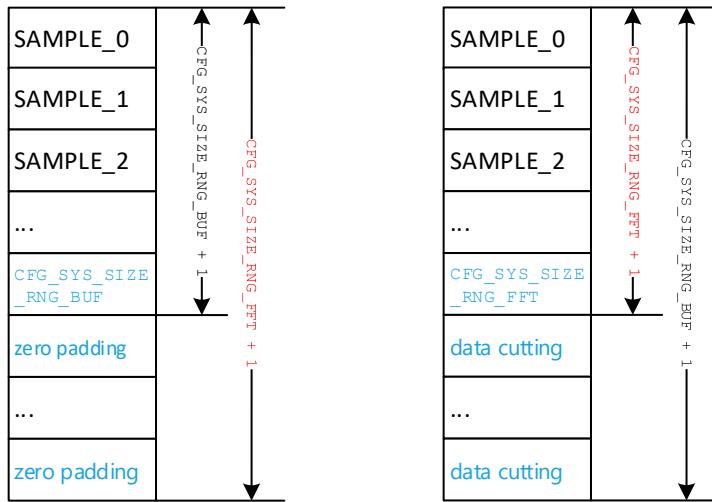


Fig. 15.2: SAMPLE Output Data Collection

**Note:** Both DMP\_MID and DMP\_FNL can be used for dumping SAMPLE data from MEM\_BUF to the interface of the Alps device. The difference between DMP\_MID and DMP\_FNL is that the data size can be different when CFG\_SYS\_SIZE\_VEL\_BUF is not equal to CFG\_SYS\_SIZE\_VEL\_FFT.

- For DMP\_MID, CFG\_SYS\_SIZE\_VEL\_BUF is used as V in  $V \times T \times A \times R \times W$ .
  - For DMP\_FNL, CFG\_SYS\_SIZE\_VEL\_FFT is used as V in  $V \times T \times A \times R \times W$ .
- 

### 15.2.1.2.3 SAMPLE Output Data Order

As the data width of MEM\_BUF is 32 bits and that of SAMPLE data is only 16 bits, 2 SAMPLE outputs will be in one memory unit. The 1st SAMPLE output will be in the upper 16 bits and the 2nd SAMPLE output will be in the lower 16 bits, as shown in [Table 15.5](#).

Table 15.5: SAMPLE Standard Output Data in MEM\_BUF

Memory Address	Memory Bits [31:16]	Memory bits [15:0]
0x00000	SAMPLE output[0]	SAMPLE output[1]
0x00001	SAMPLE output[2]	SAMPLE output[3]
0x00002	SAMPLE output[4]	SAMPLE output[5]
...	SAMPLE output[i]	SAMPLE output[i+1]

Here we use V-T-A-R to describe the output data order in MEM\_BUF. The output data can be considered as a 4D array [V][T][A][R]. This is a C memory order. With the most rapidly changing index being the last, optimum cache performance can be achieved. The data order following V-T-A-R means that the data is first output along the range dimension, then antenna dimension, virtual array dimension, and finally the Doppler dimension.

**Note:** [V][T][A][R] is the output data order when collecting data on Alps, which is more convenient for later data

processing. But physically, the data is stored in MEM\_BUF in the order of [T][V][A][R], which means that the CPU should follow [T][V][A][R] to access SAMPLE, 1D-FFT, and 2D-FFT data.

SAMPLE, 1D-FFT and 2D-FFT have the same output data order in MEM\_BUF, as shown in Fig. 15.3.

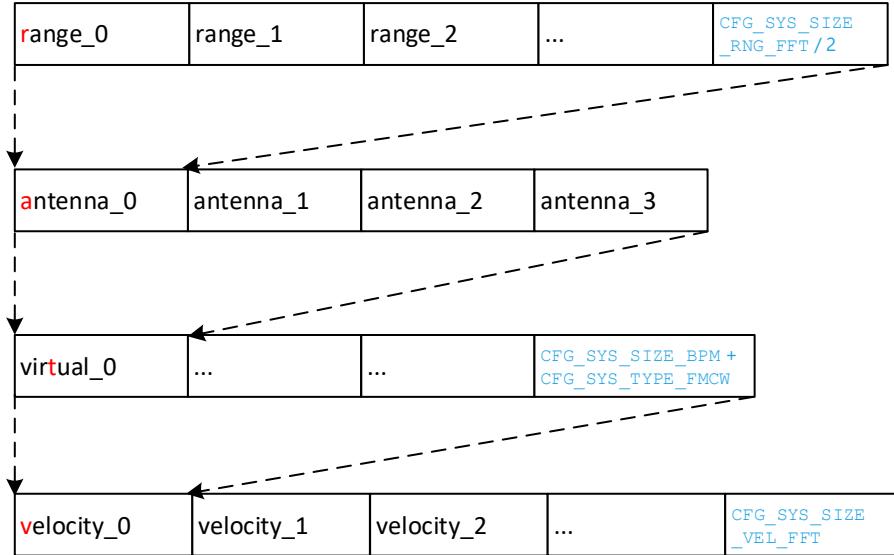


Fig. 15.3: MEM\_BUF Output Data Order

The example program iterates through the indexing space in V-T-A-R order.

```

1 #define SYS_SIZE_VEL_FFT 512
2 #define SYS_SIZE_BPM 0
3 #define SYS_TYPE_FMCW 1
4 #define SYS_SIZE_ANT 4
5 #define SYS_SIZE_RNG_FFT 256
6 int main()
7 {
8     int index = 0;
9     int V, T, A, R;
10    int T_SIZE = SYS_SIZE_BPM + SYS_TYPE_FMCW;
11
12    int values[V][T][A][R];
13    for(V = 0; V < SYS_SIZE_VEL_FFT/2; V++)
14        for(T = 0; T < T_SIZE; T++)
15            for(A = 0; A < SYS_SIZE_ANT; A++)
16                for(R = 0; R < SYS_SIZE_RNG_FFT; R++)
17                    ...
18
19    return 0;
}

```

## 15.2. Functional Description

---

### 15.2.1.3 1D-FFT and 2D-FFT Output Data

1D-FFT and 2D-FFT share the same data format, data size, and data order.

#### 15.2.1.3.1 FFT Output Data Format

FFT output data width is 32 bits. The data format is shown in [Table 15.6](#)

Table 15.6: FFT Standard Output Data Format

Data Source	Bit Width	Data Format	Memory Unit (32 Bits)
1D-FFT	32	CFL (14, 1, S, 4, U)	1 set of data in a 32-bit unit
2D-FFT	32	CFL (14, 1, S, 4, U)	1 set of data in a 32-bit unit

#### 15.2.1.3.2 FFT Output Data Size

Data size of FFT output is controlled by the registers in [Table 15.7](#).

Table 15.7: Registers for Data Collection of FFT Output Data

Register Name	Example Value
CFG_SYS_SIZE RNG_FFT	511
CFG_SYS_SIZE BPM	0
CFG_SYS_TYPE FMCW	1
CFG_SYS_SIZE VEL_FFT	255

The maximum size is 2 MB, which is limited by the size of MEM\_BUF (2 MB). The size is computed by  $V \times T \times A \times (R/2) \times W$ . The definitions are in [Table 15.8](#).

Table 15.8: Data Size of FFT Output Data

Abbr	Name	Definition	Description
V	SYS_SIZE_VEL_FFT	CFG_SYS_SIZE_VEL_FFT + 1	size of 2D-FFT in Doppler dimension
T	SYS_SIZE_VIRTUAL	CFG_SYS_SIZE_BPM + 1 + CFG_SYS_TYPE_FMCW	size of virtual array
A	SYS_SIZE_ANT	4	default antenna size
R	SYS_SIZE RNG_FFT	CFG_SYS_SIZE RNG_FFT + 1	size of 1D-FFT in range dimension
W	DATA WIDTH	32 bits	width of FFT data

Considering the symmetry of FFT, only half of the FFT output is saved to MEM\_BUF. [Table 15.9](#) shows an example of FFT output data size, using the register values in [Table 15.7](#).

Table 15.9: FFT Data Size Example

Data Source	Computation	Data Size
1D-FFT	$256 \times 2 \times 4 \times (512/2) \times 32 \text{ bits}$	2 MB
2D-FFT	$256 \times 2 \times 4 \times (512/2) \times 32 \text{ bits}$	2 MB

---

**Note:** Both DMP\_MID and DMP\_FNL can be used for dumping FFT data from MEM\_BUF to the interface of the Alps device. The difference between DMP\_MID and DMP\_FNL is that the data size can be different when CFG\_



SYS\_SIZE\_VEL\_BUF is not equal to CFG\_SYS\_SIZE\_VEL\_FFT.

- For DMP\_MID, CFG\_SYS\_SIZE\_VEL\_BUF is used as V in  $V \times T \times A \times R \times W$ .
- For DMP\_FNL, CFG\_SYS\_SIZE\_VEL\_FFT is used as V in  $V \times T \times A \times R \times W$ .

### 15.2.1.3.3 FFT Output Data Order

1D-FFT output data and 2D-FFT output data have the same data order with the SAMPLE output data. Refer to *SAMPLE Output Data Order*.

### 15.2.1.4 Clock Sequence

When dumping the standard output data, there is no need to change baseband clock frequency, because the data is buffered. So this type of data collection can be considered as real-time streaming while baseband is running at the default clock frequency of 200 MHz.

The clock sequence of GPIO data is shown in Fig. 15.4.

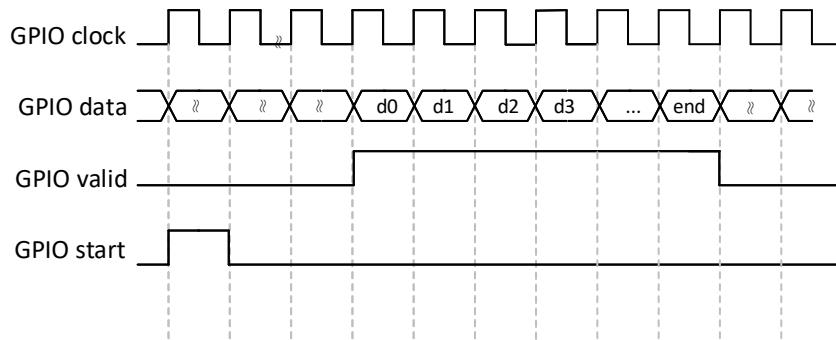


Fig. 15.4: Clock Sequence of GPIO Data

The clock sequence of LVDS data is shown in Fig. 15.5.

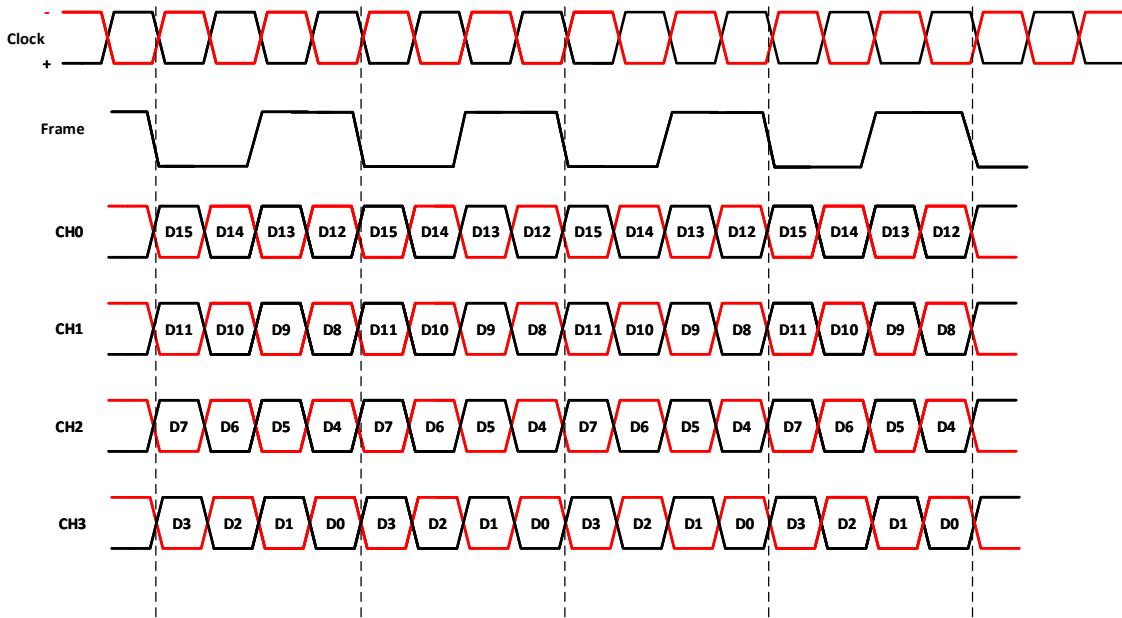


Fig. 15.5: Clock Sequence of LVDS Data

### 15.2.2 Debug Data Collection

As shown in the following, there are 3 sources of debug data, corresponding to 3 baseband processing states.

- SAMPLE debug data, which consists of SAMPLE raw data and SAMPLE data after sample decimation filter
- CFAR debug data, specifically CFAR power data
- DoA debug data, specifically DoA power data

The SAMPLE debug data is dumped to either GPIO or LVDS interface. It cannot be dumped to memory.

The CFAR and DoA debug data can be dumped not only to GPIO or LVDS interface but also to memory as the data size is small. Specifically, the second 1-MB space in MEM\_BUFS can be used to store the CFAR and DoA debug data only if it is not used by FFT.

### 15.2.2.1 Data Collection Block Diagram

The debug data will directly go to GPIO, LVDS, or memory when related baseband processing is enabled, as shown in Fig. 15.6. The related baseband registers and DMU registers should be configured to enable the output of debug data.

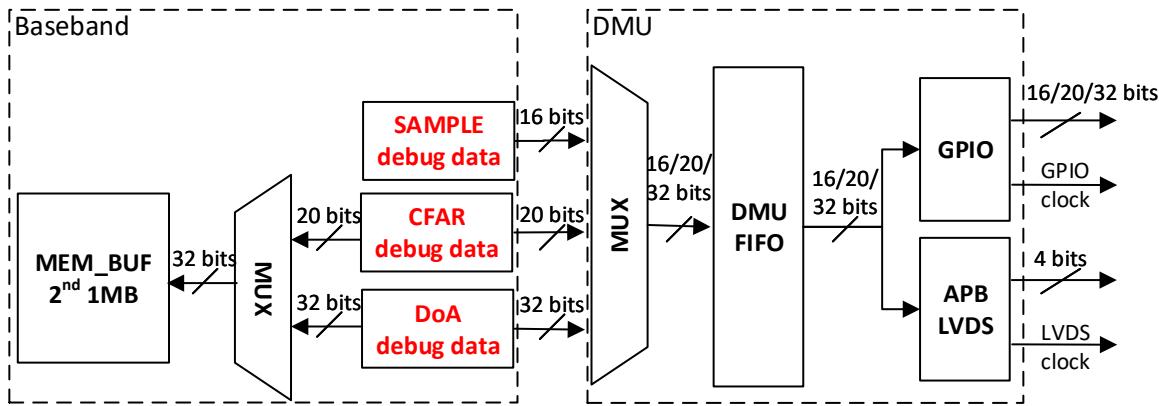


Fig. 15.6: Flow of Debug Data Collection

### 15.2.2.2 SAMPLE Debug Data

#### 15.2.2.2.1 Debug Data Format

Data formats of SAMPLE, CFAR and DoA debug data are different, as shown in Table 15.10.

Table 15.10: Debug Data Formats

Data Source	Bit Width	Data Format	Memory Unit (32 Bits)
SAMPLE	16	FXR(16, 2, S)	not to memory
CFAR	20	FLR(15, 1, U, 5, U)	1 set of data in a 32-bit unit; the upper 12 bits are 0
DoA-DBF	20	FLR(15, 1, U, 5, U)	1 set of data in a 32-bit unit; the upper 12 bits are 0

#### 15.2.2.2.2 SAMPLE Debug Data Size

The size of SAMPLE debug data is controlled by the registers in Table 15.11.

Table 15.11: Registers for Data Collection of SAMPLE Debug Data

Register Name	Example Value
CFG_SAM_SIZE_DBG_BGN	21
CFG_SAM_SIZE_DBG_END	532
CFG_SAM_SIZE_BPM	0
CFG_SYS_TYPE_FMCW	1
CFG_SYS_SIZE_VEL_BUF	255

The size is computed by  $V \times T \times A \times R \times W$ . The definitions are in Table 15.12.

## 15.2. Functional Description

---

Table 15.12: Data Size of SAMPLE Debug Data

Abbr	Name	Definition	Description
V	SYS_SIZE_VEL_BUF	CFG_SYS_SIZE_VEL_BUF + 1	size of chirp number in a frame
T	SYS_SIZE_VIRTUAL	CFG_SYS_SIZE_BPM + 1 + CFG_SYS_TYPE_FMCW	size of virtual array
A	SYS_SIZE_ANT	4	default antenna size
R	SYS_SIZE RNG DBG	CFG_SAM_SIZE_DBG_END - CFG_SAM_SIZE_DBG_BGN + 1	size of ADC data in a chirp
W	DATA WIDTH	16 bits	width of SAMPLE debug data

Table 15.13 shows an example of SAMPLE debug data size, using the register values in Table 15.11.

Table 15.13: SAMPLE Debug Data Size Example

Data Source	Computation	Data Size
SAMPLE	$256 \times 2 \times 4 \times (532 - 21 + 1) \times 16 \text{ bits}$	2 MB

### 15.2.2.3 SAMPLE Debug Data Order

The SAMPLE debug data order follows V-T-R-A, which means the data will be first saved along antenna dimension, then the range dimension, then BPM or TDM dimension, and at last Doppler dimension. See Fig. 15.7.

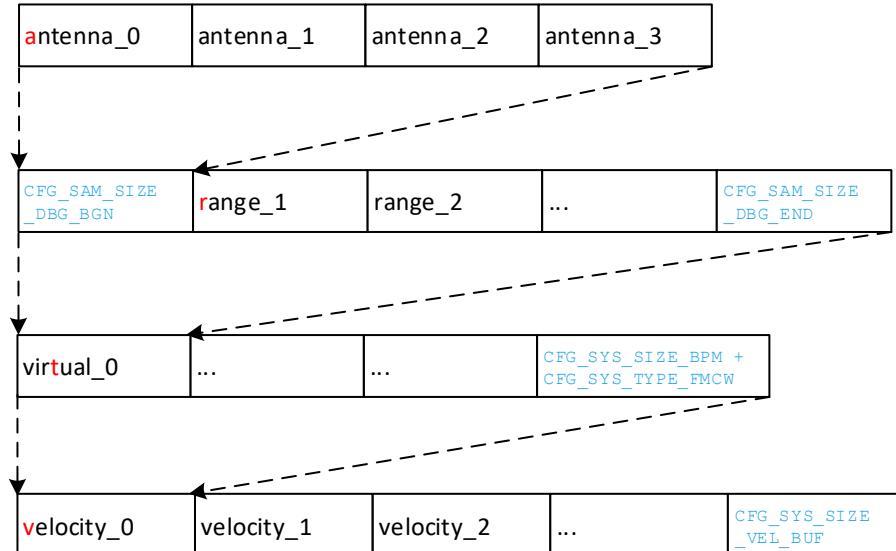


Fig. 15.7: SAMPLE Debug Data Order

### 15.2.2.3 CFAR Debug Data

#### 15.2.2.3.1 CFAR Debug Data Format

CFAR debug data format is shown in [Table 15.14](#).

Table 15.14: CFAR Debug Data Format

Data Source	Bit Width	Data Format	Memory Unit(32 Bits)
CFAR	20	FLR(15, 1, U, 5, U)	1 set of data in a 32-bit unit; the upper 12 bits are 0

#### 15.2.2.3.2 CFAR Debug Data Size

The size of CFAR debug data is controlled by the registers in [Table 15.15](#).

Table 15.15: Registers for Data Collection of CFAR Debug Data

Register Name	Example Value
CFG_SYS_SIZE_RNG_FFT	511
CFG_SYS_SIZE_VEL_FFT	255

The CFAR debug data size of one frame is  $V \times (R/2) \times W$ . The definitions are in [Table 15.16](#).

Table 15.16: Data Size of CFAR Debug Data

Abbr	Name	Definition	Description
V	SYS_SIZE_VEL_FFT	CFG_SYS_SIZE_VEL_FFT + 1	size of 2D-FFT in Doppler dimension
R	SYS_SIZE_RNG_FFT	CFG_SYS_SIZE_RNG_FFT + 1	size of 1D-FFT in range dimension
W	DATA WIDTH	20 bits	width of CFAR debug data

[Table 15.17](#) shows an example of CFAR debug data size, using the register values in [Table 15.15](#).

Table 15.17: CFAR Debug Data Size Example

Data Source	Computation
CFAR debug	$256 \times (512/2) \times 20 \text{ bits}$

#### 15.2.2.3.3 CFAR Debug Data Order

The CFAR debug data order follows R-V.Because CFAR searches first along the Doppler dimension and then along the range dimension, CFAR debug data is generated along the Doppler dimension firstly and then the range dimension.

### 15.2.2.4 DoA Debug Data

DoA debug data means DBF power spectrum data. Only DBF supports debug data, but DML does not.



## 15.2. Functional Description

---

### 15.2.2.4.1 DoA Debug Data Format

DoA debug data format is shown in [Table 15.18](#).

Table 15.18: DoA Debug Data Format

Data Source	Bit Width	Data Format	Memory Unit (32 Bits)
DoA-DBF	20	FLR(15, 1, U, 5, U)	1 set of data in a 32-bit unit; the upper 12 bits are 0

### 15.2.2.4.2 DoA Debug Data Size

The size of DoA debug data is controlled by the registers in [Table 15.19](#). During DBF 2D-DoA, there will be one copy of power spectrum for each group. In each group, if multiple objects (up to 4) within the same range and Doppler gates are searched out, only the power spectrum of the 1st object can be dumped. The number of the groups depends on CFG\_DoA\_NUMB\_GRP.

- If CFG\_DoA\_NUMB\_GRP is 0, 1 copy of power spectrum can be dumped.
- If CFG\_DoA\_NUMB\_GRP is 1, 2 copies of power spectrum can be dumped.
- If CFG\_DoA\_NUMB\_GRP is 2, 3 copies of power spectrum can be dumped.

Table 15.19: Registers for Data Collection of DoA Debug Data

Register Name	Example Value
FDB_CFAR_NUMB_OBJ	100
CFG_DoA_NUMB_GRP	2
CFG_DoA_GRP0_SIZ RNG_PKS CRS	359
CFG_DoA_GRP1_SIZ RNG_PKS CRS	359
CFG_DoA_GRP2_SIZ RNG_PKS CRS	359

The DoA debug data size of one frame is  $n_o \times n_a \times W$ , and  $n_a$  is dependent on CFG\_DoA\_NUMB\_GRP. The definitions are in [Table 15.20](#).

- If CFG\_DoA\_NUMB\_GRP is 0,  $n_a = (n_{a0})$
- If CFG\_DoA\_NUMB\_GRP is 1,  $n_a = (n_{a0} + n_{a1})$
- If CFG\_DoA\_NUMB\_GRP is 2,  $n_a = (n_{a0} + n_{a1} + n_{a2})$

Table 15.20: Data Size of DoA Debug Data

Abbr	Definition	Description
$n_o$	FDB_CFAR_NUMB_OBJ	number of cfar objects
$n_a$	total angle number in used group	number of angles of DoA
$n_{a0}$	CFG_DoA_GRP0_SIZ RNG_PKS CRS + 1	number of angles in group 0 of DoA
$n_{a1}$	CFG_DoA_GRP1_SIZ RNG_PKS CRS + 1	number of angles in group 1 of DoA
$n_{a2}$	CFG_DoA_GRP2_SIZ RNG_PKS CRS + 1	number of angles in group 2 of DoA
$W$	20 bits	bit width of DoA debug data

[Table 15.21](#) shows an example of DoA debug data size, using the register values in [Table 15.19](#).

Table 15.21: DoA Debug Data Size Example

Data Source	Computation
DoA debug	$100 \times (360 + 360 + 360) \times 20 \text{ bits}$



### 15.2.2.4.3 DoA Debug Data Order

The DoA debug data order follows the DBF angle searching order, which means the data is generated along the angle searching in sequence.

### 15.2.2.5 Clock Sequence

When dumping the 3 sources of debug data to GPIO or LVDS interface, the baseband clock frequency (200 MHz by default) should be reduced to GPIO clock frequency (100 MHz maximum). Otherwise, the debug data will be missing, because baseband frequency is faster than GPIO frequency and it has no buffer.

When dumping the CFAR and DoA debug data to MEM\_BUF, there is no need to switch clock frequency, because baseband memory use the same clock with baseband processing. Once it is in memory, the data cannot be dumped to GPIO or LVDS interface. CPU can access the memory to check the debug data.

It is not suggested that CFAR and DoA debug data be dumped to GPIO, because their data width is 20 bits, which is not efficient for hardware application.

The clock sequence of GPIO data is shown in Fig. 15.4. And the clock sequence of LVDS data is shown in Fig. 15.5.

## 15.2.3 HIL

### 15.2.3.1 HIL Block Diagram

As shown in Fig. 15.8, input data of HIL has 2 sources, including:

- CPU AHB\_BUS
- GPIO

In HIL mode, FMCW is not needed. CPU or GPIO output data will replace ADC data. Generally, a certain data pattern can be used for the baseband input, and the output of baseband can be collected to do further analysis.

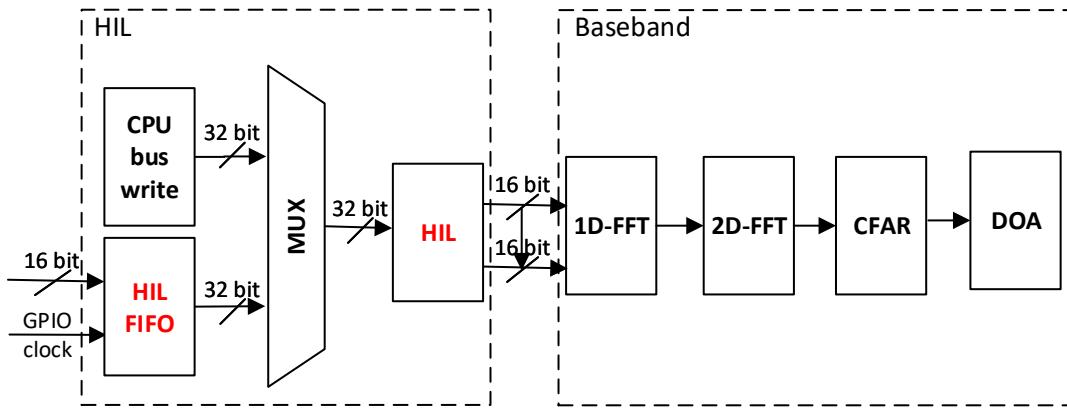


Fig. 15.8: HIL Data Flow

## 15.2. Functional Description

---

### 15.2.3.2 HIL Input Data

Data format, data size and data order will be introduced.

#### 15.2.3.2.1 HIL Input Data Format

HIL input data should be considered as SAMPLE output data, because it is used for 1D-FFT directly. So the data format of HIL input data should be same as SAMPLE output data, as shown in [Table 15.22](#).

Table 15.22: HIL Input Data Format

Data Source	Bit Width	Data Format	Memory Unit (32 Bits)
HIL	16	FXR(16, 2, S)	not to memory

#### 15.2.3.2.2 HIL Input Data Size

The size of HIL input data is controlled by the registers in [Table 15.23](#).

Table 15.23: Registers for HIL Input Data

Register Name	Example Value
CFG_SYS_SIZE_RNG_FFT	511
CFG_SYS_SIZE_BPM	0
CFG_SYS_TYPE_FMCW	1
CFG_SYS_SIZE_VEL_FFT	255

The maximum size is 2 MB, which is limited by the size of MEM\_BUF (2 MB). The size is computed by  $V \times T \times A \times R \times W$ . The definitions are in [Table 15.3](#).

[Table 15.24](#) shows an example of HIL input data size, using the register values in [Table 15.23](#).

Table 15.24: HIL Input Data Size Example

Data Source	Computation	Data Size
HIL	$256 \times 2 \times 4 \times 512 \times 16 \text{ bits}$	2 MB

#### 15.2.3.2.3 HIL Input Data Order

HIL input data follows the same data order with MEM\_BUF output data, which is V-T-A-R. In other words, the data is input along the range dimension firstly, then antenna dimension, and then BPM or TDM dimension, and at last the Doppler dimension. Refer to [Fig. 15.3](#).



### 15.2.3.3 Clock Sequence

The data widths of CPU AHB\_BUS is 32 bits. Data is stored into the upper 16 bits and then the lower 16 bits in sequence.

The data width of GPIO is 16 bits.

The clock sequence of CPU AHB is shown in Fig. 15.9. And the clock sequence of GPIO is shown in Fig. 15.10.

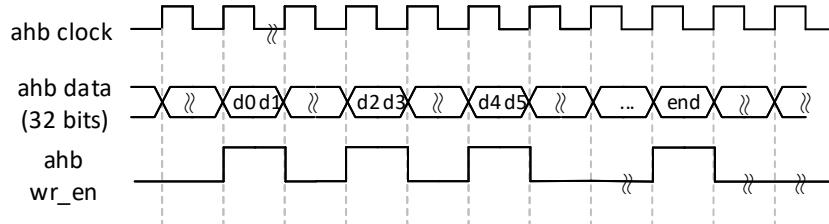


Fig. 15.9: Clock Sequence of HIL from CPU AHB

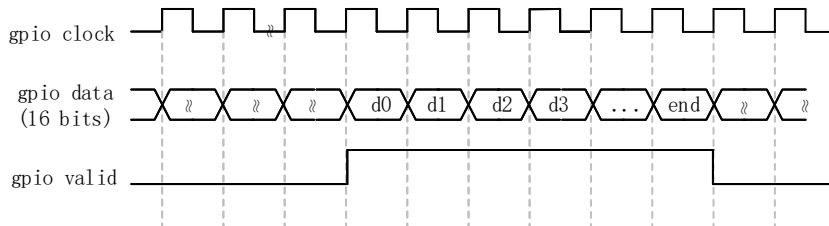


Fig. 15.10: Clock Sequence of HIL from GPIO

## 15.3 Programming Interfaces

Both baseband registers and DMU registers need to be set for data collection or HIL.

### 15.3.1 Baseband Registers

There are 2 types of baseband registers to be set, data source selection and data size.

## 15.3. Programming Interfaces

---

### 15.3.1.1 Data Source Selection Settings

- CFG\_SAM\_SINKER

The register CFG\_SAM\_SINKER determines whether ADC data will go into the 1D-FFT engine directly or not, which will take effect when the *SAMPLE state* is enabled. The value can be set to 0 or 1.

- If CFG\_SAM\_SINKER is set to 1, ADC data will go into the 1D-FFT engine while ADC data is streaming in, which is time-saving because 1D-FFT calculation time is mostly shared with chirp sequence time.
- If CFG\_SAM\_SINKER is set to 0, ADC data will go into MEM\_BUF while ADC data is streaming in, which is used for data debugging. After all ADC data is collected, MEM\_BUF data will then be output for 1D-FFT calculation.

- CFG\_SYS\_ENABLE

The register CFG\_SYS\_ENABLE determines which states will be enabled. This register has 9 bits, each bit corresponding to a baseband processing state.

- Bit value of 1 means enabling the corresponding state.
- Bit value of 0 means disabling the corresponding state.

- CFG\_SAM\_SIZE\_DBG\_SRC

The register CFG\_SAM\_SIZE\_DBG\_SRC is used to select the data source when collecting SAMPLE debug data.

The value of 0 means collecting ADC data before decimation filter, and 1 means collecting ADC data after decimation filter.

- CFG\_DBG\_BUF\_TAR

The register CFG\_DBG\_BUF\_TAR is used to select the data source when collecting CFAR and DoA debug data to MEM\_BUF.

The value of 6'b00\_1000 means buffering CFAR debug data, and 6'b01\_0000 means buffering DoA debug data.

### 15.3.1.2 Data Size Settings

Different data sources use different data size settings. Refer to *Functional Description* for more details.

- CFG\_SYS\_SIZE\_VEL\_FFT and CFG\_SYS\_SIZE\_RNG\_FFT determine the size of FFT points. Suppose the size of FFT is  $2^{n_1} \times 2^{n_2}$ . Then program the two registers with the values of  $2^{n_1} - 1$  and  $2^{n_2} - 1$ , respectively.
- CFG\_SYS\_SIZE\_VEL\_BUF is the effective chirp number in a frame (Doppler dimension). Suppose the effective chirp number in the Doppler dimension is  $2^{n_2}$ . Then program the register with the value of  $2^{n_2} - 1$ .
- CFG\_SYS\_SIZE\_RNG\_BUF is the effective ADC points in a chirp (range dimension). Suppose the effective ADC points in the range dimension is  $2^{n_1}$ . Then program the register with the value of  $2^{n_1} - 1$ .
- CFG\_SAM\_SIZE\_DBG\_BGN and CFG\_SAM\_SIZE\_DBG\_END are used to set the data size for SAMPLE debugging. Suppose the beginning point is  $N_1$  and the end point is  $N_2$ , then program the two registers with the values of  $N_1$  and  $N_2$ , respectively.
- CFG\_SYS\_TYPE\_FMCW is the type of FMCW chirps. Value 1 means enabling anti velocity ambiguity.
- CFG\_SYS\_SIZE\_BPM is the size of virtual array. Suppose the virtual array number is  $N$ . Then program the register with the value of  $N - 1$ .
- CFG\_DoA\_GRP2\_SIZ\_RNG\_PKS\_CRS determines the angle size of Group 2 in DoA. Suppose the angle size is  $N$ . Then program the register with the value of  $N - 1$ .



### 15.3.2 DMU Registers

The DMU module controls the data selection and input or output direction. The register `DMU_DBG_SRC` is used to select the output data source, either standard output data or debug data. The register `DMU_HIL_ENA` is used to select the HIL input data source, either GPIO or CPU AHB. For more information on DMU, refer to the DMU section in *Calterah Alps Reference Manual*.

There are 2 interfaces in the Alps device for data output. One is GPIO and the other is LVDS.

- For the configuration of GPIO interface, clock, and enable, refer to the DMU section in *Calterah Alps Reference Manual*.
- For the configuration of LVDS interface, clock, and enable, refer to the LVDS section in *Calterah Alps Reference Manual*.

## 15.4 Limitation and Constraints

### 15.4.1 Standard Output Data Size

The size of standard output data (SAMPLE, 1D-FFT or 2D-FFT) is limited by the memory size of `MEM_BUF`. And the size will also be influenced by anti velocity ambiguity mode, which will generate one extra chirp of data, compared to cases where anti velocity ambiguity is disabled.

#### 15.4.1.1 2MB Limited

The maximum size of standard output data (SAMPLE, 1D-FFT or 2D-FFT) is 2M bytes, which is limited by the memory size of `MEM_BUF`.

`MEM_BUF` can be shared with CPU, which means baseband will only use the remaining space, which is not shared. Suppose that the shared memory size (in byte) is  $S$ . Then the remaining memory space for baseband is  $(2M - S)$ . Refer to [Memory Sharing](#) for detailed information.

---

**Note:** The size of `MEM_BUF` is 2M bytes without memory sharing, and  $(2M - S)$  bytes when memory sharing is enabled.

---

The sizes of SAMPLE, 1D-FFT and 2D-FFT are controlled by the registers in [Table 15.7](#). The size is computed by  $V \times T \times A \times (R/2) \times W$ . The definitions are in [Table 15.25](#). The data width of SAMPLE is 16 bits, which is half of the FFT data width. But considering the symmetry of FFT, only half of the range FFT output is saved to `MEM_BUF`. So SAMPLE and FFT have the same data size.

Table 15.25: Data Size Settings

Abbr	Name	Definition	Description
V	SYS_SIZE_VEL_FFT	<code>CFG_SYS_SIZE_VEL_FFT + 1</code>	size of 2D-FFT in Doppler dimension
T	SYS_SIZE_VIRTUAL	<code>CFG_SYS_SIZE_BPM + 1 + CFG_SYS_TYPE_FMCW</code>	size of virtual array
A	SYS_SIZE_ANT	4	default antenna size
R	SYS_SIZE_RNG_FFT	<code>CFG_SYS_SIZE_RNG_FFT + 1</code>	size of 1D-FFT in range dimension
W	DATA WIDTH	32 bits	width of FFT data



## 15.4. Limitation and Constraints

---

Table 15.26 shows an example of the output data size.

Table 15.26: Data Size Example

Data Source	Computation	Data Size
SAMPLE	$256 \times 2 \times 4 \times 512 \times 16 \text{ bits}$	2 MB
1D-FFT	$256 \times 2 \times 4 \times (512/2) \times 32 \text{ bits}$	2 MB
2D-FFT	$256 \times 2 \times 4 \times (512/2) \times 32 \text{ bits}$	2 MB

### 15.4.1.2 Memory Sharing

MEM\_BUF can be shared between baseband hardware and CPU, which means CPU can expand storage space by occupying part of MEM\_BUF.

When MEM\_BUF is not shared with the CPU, 2MB will all be used for baseband processing, including SAMPLE, 1D-FFT and 2D-FFT. But when MEM\_BUF is shared with the CPU, only the second 1MB can be shared. See [Table 15.27](#).

To enable memory sharing, there are 2 registers to be set and 2 other prerequisites to be prepared. The following shows the register settings for memory sharing.

- Set CFG\_SYS\_TYP\_ARB (Address 0xC00058) to 0
- Set CFG\_DBG\_BUF\_TAR (Address 0xC00C00) to 0x20

And the following shows the 2 prerequisites for memory sharing. Without these 2 prerequisites, memory sharing will not work.

- The size of range FFT (Register CFG\_SYS\_SIZE\_RNG\_FFT, Address 0xC00030) should be set, depending on the real application.
- The clock of baseband memory should be enabled, that is, the register (Address 0xB2020C) should be set to 1.

---

**Note:** The specific shared size is not configured by a specific register. The allocation should be designed in firmware. And then set the baseband related registers listed in [Table 15.25](#).

When MEM\_BUF is shared, debug data for CFAR and DoA cannot be enabled to be saved to MEM\_BUF, because the second 1MB memory is not available for baseband.

---

Table 15.27: 1MB Memory Sharing

Address	Description	BB Access	CPU Access
0x000000 ~ 0x0FFFFF	1st 1MB for baseband	always	see <a href="#">Accessing Memories</a>
0x100000 ~ 0x1FFFFFF	2nd 1MB for CPU	no	always, via base address 0x800000

---

**Note:** When the 2nd 1MB memory is shared with CPU:

- CPU should access the 2nd 1MB memory via the base address 0x800000. See [Section A](#).
  - And CPU should access the 1st 1MB of MEM\_BUF and other memories via the base address 0xE00000. See [Accessing Memories](#).
  - CPU cannot access these 2 different addresses simultaneously.
- 



### 15.4.1.3 Anti Velocity Ambiguity

When the anti velocity ambiguity feature is enabled, one extra chirp is generated, as shown in [Table 15.28](#). The extra chirp can be considered as a virtual array. So when anti velocity ambiguity is enabled, T in V-T-A-R should equal `CFG_SYS_SIZE_BPM + 1 + CFG_SYS_TYPE_FMCW`.

Table 15.28: Anti Velocity Ambiguity

Register Name	Values	Notes
<code>CFG_SYS_TYPE_FMCW</code>	1 or 0	high active; 1 means anti velocity ambiguity is enabled

### 15.4.2 SAMPLE Debug Data Size

The size of SAMPLE debug data( $(end - bgn + 1)$ ) in a chirp should be smaller than the size of `CFG_SYS_SIZE_RNG_PRD`. See [Table 15.29](#).

The debug points should follow  $(end - bgn + 1) < (prd - 1)$ , and  $(prd - 1)$  is a limitation from RTL design.

Table 15.29: Data Size Settings

Abbr	Name	Definition	Description
bgn	<code>SAM_SIZE_DBG_BGN</code>	<code>CFG_SAM_SIZE_DBG_BGN</code>	begin point in a chirp
end	<code>SAM_SIZE_DBG_END</code>	<code>CFG_SAM_SIZE_DBG_END</code>	end point in a chirp
prd	<code>SYS_SIZE_RNG_PRD</code>	<code>CFG_SYS_SIZE_RNG_PRD + 1</code>	total points in a chirp

### 15.4.3 CFG\_SYS\_ENABLE

When the register `CFG_SAM_SINKER` is set to 1, make sure that Bit 4 (*1D-FFT state*) of `CFG_SYS_ENABLE` is set to 0, because SAMPLE and 1D-FFT are running parallel in the *SAMPLE state* to save the time consumption. Setting *1D-FFT state* to 0 ensures that 1D-FFT state shall not be triggered.

To enter HIL mode, however, the states AGC, SAMPLE and 1D-FFT should all be set to 0 to ensure that they will not be triggered, because HIL input data is directly sent for 1D-FFT processing. Otherwise, the baseband will go wrong.

### 15.4.4 CFG\_SAM\_FORCE

The SAMPLE module starts to collect input data when a start signal is received. The start signal can be sent by either FMCW or `CFG_SAM_FORCE`. As shown in [Table 15.30](#), `CFG_SAM_FORCE` is specially designed for sending a force start signal when FMCW is disabled. So if FMCW is disabled, make sure that the `CFG_SAM_FORCE` register is set to 1. When FMCW is enabled, `CFG_SAM_FORCE` cannot be set to 1 at the same time. Otherwise, the SAMPLE data goes disordered.

Table 15.30: Sample Force Start

Register Name	Values	Notes
<code>CFG_SAM_FORCE</code>	1 or 0	high active, 1 means start signal is enabled



### 15.4.5 Streaming out

When baseband is running, the standard output data can be directly streamed out to GPIO or LVDS interface. There is no need to change the clock frequency of baseband , because the standard output data is buffered to MEM\_BUF. So it is suggested that while baseband is running at the default clock frequency of 200 MHz, standard output data collection should be used for streaming out data.

When baseband is running, the debug data also can be directly streamed out to GPIO or LVDS interface. But make sure that the baseband clock frequency (200 MHz by default) is reduced to the clock frequency of the GPIO (100 MHz maximum). Otherwise, the debug data will be missing, because the debug data has no buffer and its clock frequency is faster than the clock frequency of GPIO.

## 15.5 Software and Configuration Suggestions

### 15.5.1 Register Suggestions

To run data collection or HIL, there are 2 types of registers to be configured, baseband registers and DMU registers. The register suggestions are shown in [Table 15.31](#).

Table 15.31: Configuration Suggestions of Data Collection and HIL

Data Source	Register Name	Function Description	Suggested Value
SAMPLE output	CFG_SAM_SINKER	buffer ADC data to MEM_BUF	0
	CFG_SYS_ENABLE	enable SAMPLE and DMP_MID	9'b0_0000_1100
1D-FFT output	CFG_SAM_SINKER	not buffer ADC data to MEM_BUF	1
	CFG_SYS_ENABLE	enable SAMPLE and DMP_MID	9'b0_0000_1100
2D-FFT output	CFG_SAM_SINKER	not buffer ADC data to MEM_BUF	1
	CFG_SYS_ENABLE	enable SAMPLE, 2D-FFT and DMP_FNL	9'b1_0010_0100
SAMPLE debug	CFG_SAM_SINKER	not buffer ADC data to MEM_BUF	1
	CFG_SYS_ENABLE	enable SAMPLE	9'b0_0000_1100
	CFG_SAM_SIZE_DBG_SRC	ADC data before decimation filter	0
CFAR debug	CFG_SAM_SINKER	not buffer ADC data to MEM_BUF	1
	CFG_SYS_ENABLE	enable SAMPLE, 2D-FFT and CFAR	9'b0_0110_1100
	CFG_DBG_BUF_TAR	enable CFAR debug data to MEM_BUF	6'b00_1000
DoA debug	CFG_SAM_SINKER	not buffer ADC data to MEM_BUF	1
	CFG_SYS_ENABLE	enable SAMPLE, 2D-FFT, CFAR and DoA	9'b0_1110_1100
	CFG_DBG_BUF_TAR	enable DoA debug data to MEM_BUF	6'b01_0000
HIL	CFG_SAM_SINKER	not buffer ADC data to MEM_BUF	1
	CFG_SYS_ENABLE	enable HIL, 2D-FFT, CFAR and DoA	9'b0_1110_1010



## 15.5.2 Standard Output Data Collection

Baseband registers and DMU registers are both needed to be configured, as shown in [Table 15.31](#).

### 15.5.2.1 Baseband Configuration

There are 3 data sources in this feature to be collected, corresponding to different configurations.

#### 15.5.2.1.1 SAMPLE Output Data Collection

By default, SAMPLE output data is not buffered to MEM\_BUF, because SAMPLE and 1D-FFT are running parallel to save the time consumption. To dump SAMPLE output data, SAMPLE output data must firstly be buffered to MEM\_BUF. So the register CFG\_SAM\_SINKER is should be set to 0. Then in the state DMP\_MID, SAMPLE output data in MEM\_BUF is transmitted to GPIO or LVDS. And if neither 1D-FFT nor 2D-FFT is enabled, SAMPLE output data can be also dumped in the state DMP\_FNL.

See [Fig. 15.11](#)



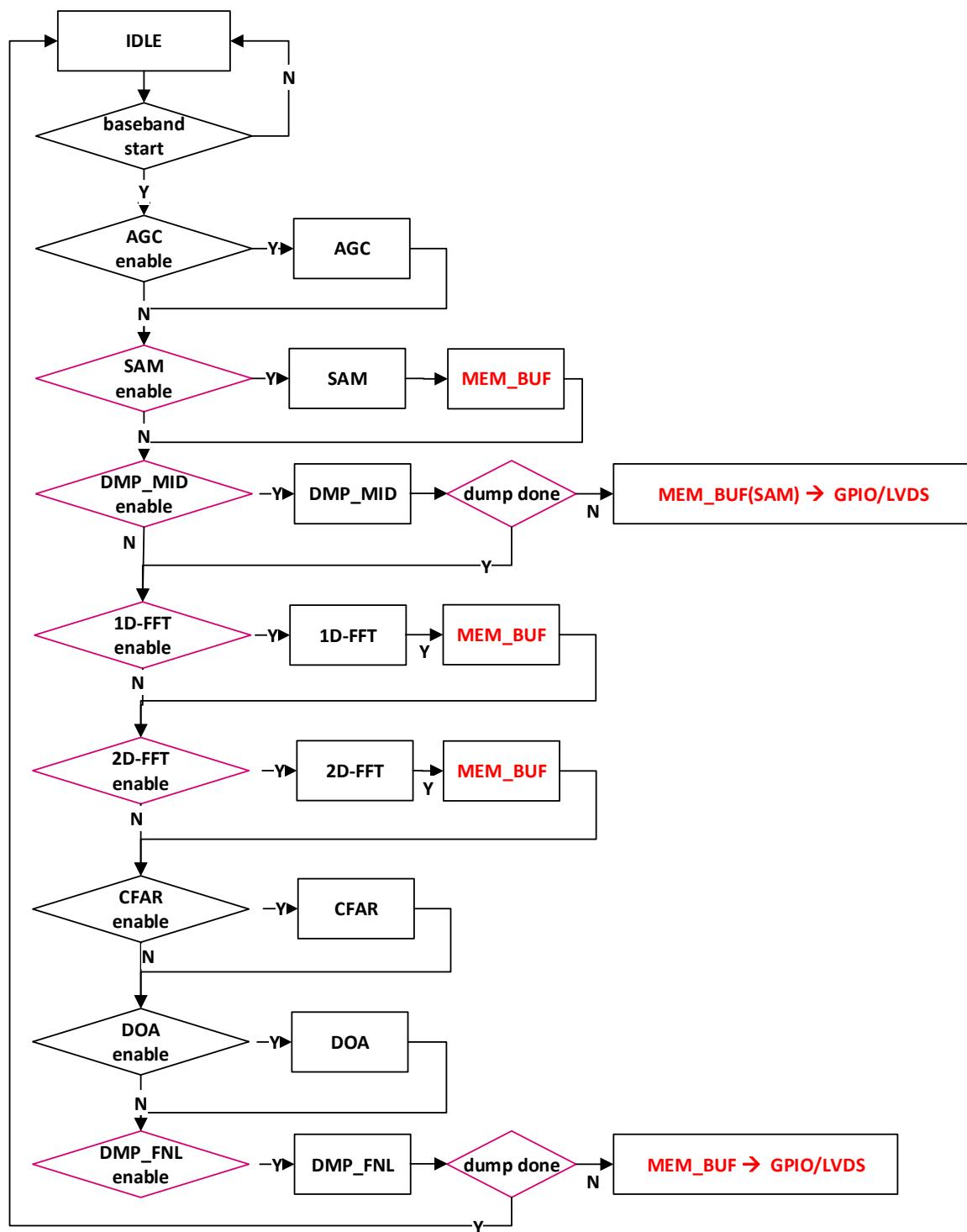


Fig. 15.11: Data Collection of Standard Output Data when Buffering SAMPLE

### 15.5.2.1.2 1D-FFT Output Data Collection

1D-FFT output and 2D-FFT output are always buffered to MEM\_BUF. When dumping 1D-FFT output data without buffering SAMPLE output data, the state DMP\_MID can be used to start the dumping. And if 2D-FFT is not enabled, 1D-FFT output data can optionally be dumped in the state DMP\_FNL. So to dump 1D-FFT output data, at least 2 states should be enabled. The first is SAMPLE, and the second can be DMP\_MID or DMP\_FNL. Other states, such as 1D-FFT and 2D-FFT, are not necessary. See Fig. 15.12.



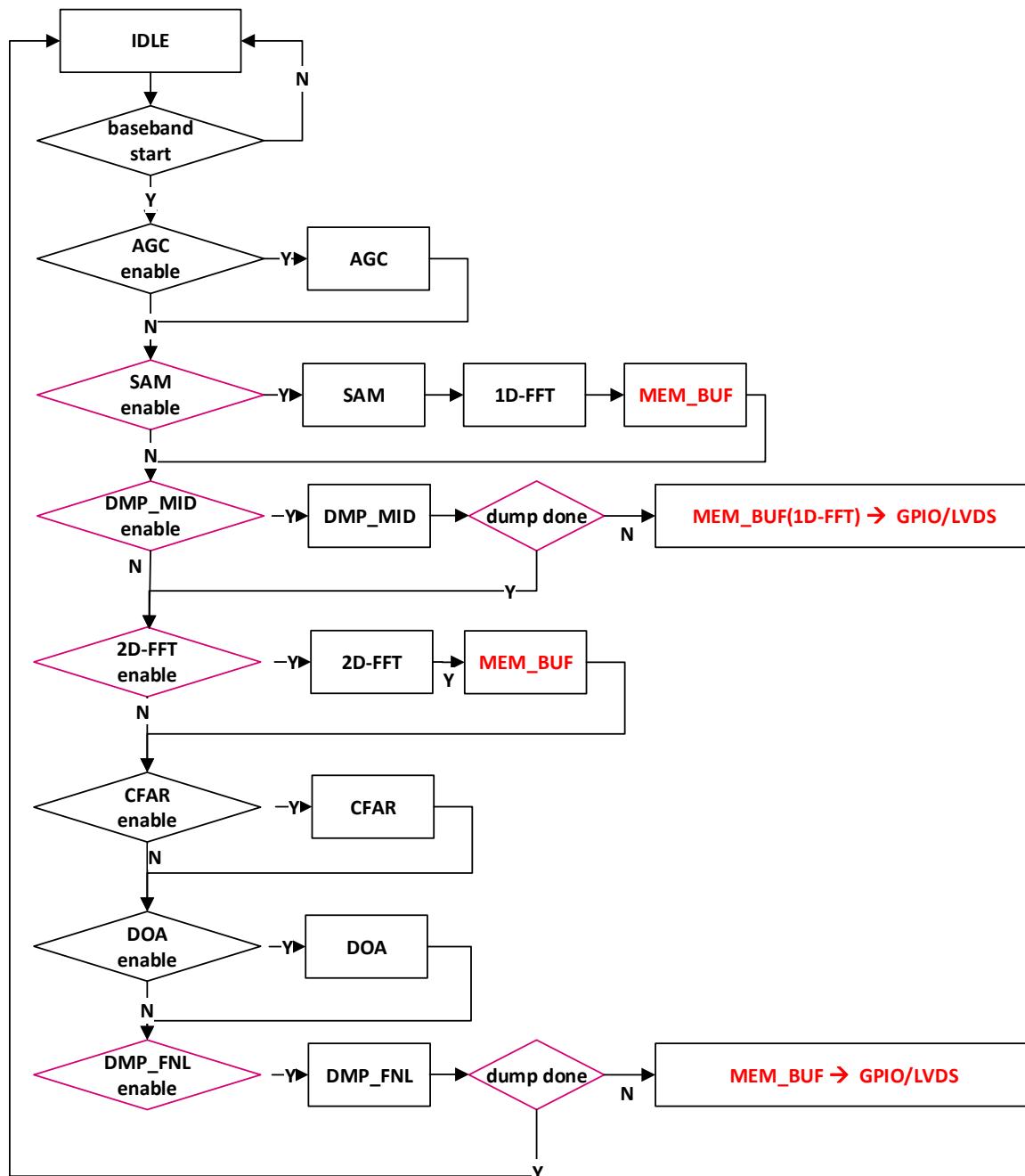


Fig. 15.12: Data Collection of Standard Output Data Without Buffering SAMPLE

### 15.5.2.1.3 2D-FFT Output Data Collection

1D-FFT output and 2D-FFT output are always buffered to MEM\_BUF. When dumping 2D-FFT output data, only the state DMP\_FNL can be used to start the dumping. So to dump 2D-FFT output data, at least 3 states should be enabled, SAMPLE, FFT2D, and DMP\_FNL. Other states, such as CFAR and DoA, are not necessary. See Fig. 15.12.

### 15.5.2.2 DMU Configuration

The DMU module controls the data selection. The register DMU\_DBG\_SRC is used to select the data source.

- 6'b10\_0000 → dumping MEM\_BUF data, including SAMPLE, 1D-FFT, or 2D-FFT output data.

There are 2 interfaces in Alps to output the data. One is GPIO, and the other is LVDS.

- For the configuration of GPIO interface, clock, and enable, refer to the DMU section in *Calterah Alps Reference Manual*.
- For the configuration of LVDS interface, clock, and enable, refer to the LVDS section in *Calterah Alps Reference Manual*.

## 15.5.3 Debug Data Collection

Baseband registers and DMU registers are both needed to be configured, as shown in Table 15.31.

### 15.5.3.1 Baseband Configuration

- To dump SAMPLE debug data to GPIO/LVDS, the register CFG\_SAM\_SIZE\_DBG\_SRC should be configured to select data source.
- To dump CFAR and DoA debug data to MEM\_BUF, only the register CFG\_DBG\_BUF\_TAR should be configured.
- To dump CFAR and DoA debug data to GPIO/LVDS, no register in baseband needs to be configured.

The real-time debug data is streamed out while the related baseband state is running, so there is no other extra configuration in baseband. Refer to [CFG\\_SYS\\_ENABLE](#).

### 15.5.3.2 DMU Configuration

The DMU module controls the data selection. The register DMU\_DBG\_SRC is used to select the data source.

- 6'b00\_0010 → dumping SAMPLE debug data
- 6'b00\_0100 → dumping CFAR debug data
- 6'b00\_1000 → dumping DoA debug data

There are 2 interfaces in Alps to output the data. One is GPIO, and the other is LVDS.

- For more information on the configuration of GPIO interface, clock, and enable, refer to the DMU section in *Calterah Alps Reference Manual*.
- For more information on the configuration of LVDS interface, clock, and enable, refer to the LVDS section in *Calterah Alps Reference Manual*.



### **15.5.4 HIL**

Both baseband registers and DMU registers need to be configured, as shown in [Table 15.31](#).

#### **15.5.4.1 Baseband Configuration**

HIL input data is directly sent for 1D-FFT processing, so the states AGC, SAMPLE and 1D-FFT are not used when running HIL. The state AGC, SAMPLE and 1D-FFT should not be enabled. Otherwise, the baseband will go wrong. To enable HIL, the register `CFG_SYS_ENABLE` in baseband should be set to `9'b0_1110_0010`.

To run HIL, the states HIL, 2D-FFT, CFAR, and DoA should be enabled, as shown in [Fig. 15.13](#).



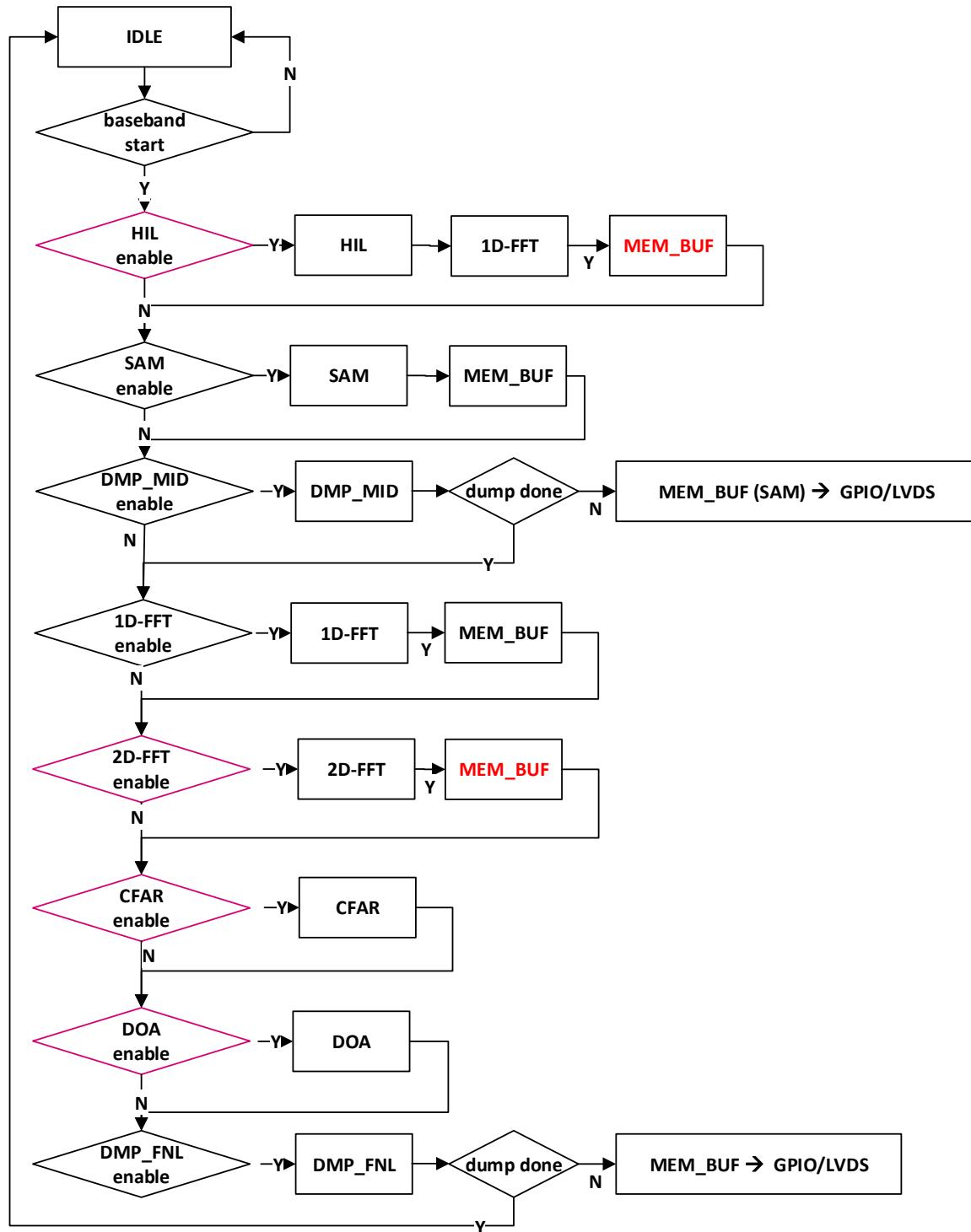


Fig. 15.13: HIL State Flow

## 15.6. Examples

---

### 15.5.4.2 DMU Configuration

Since there are 2 input data sources, which are input from CPU AHB\_BUS and input from GPIO, the input data source should be set. The register `DMU_HIL_ENA` in the DMU module is used to select the data source. If the HIL data is from CPU AHB, `DMU_HIL_ENA` should be set to 0. If the HIL data is from GPIO, `DMU_HIL_ENA` should be set to 1.

For more information on the configuration of GPIO interface, clock, and enable, refer to the DMU section in *Calterah Alps Reference Manual*.

## 15.6 Examples

The steps are similar when running data collection and HIL, so take data collection of SAMPLE standard output data for example. 4 types of settings should be set to complete the data collection in [Table 15.32](#).

1. Set data size.
2. Set baseband states to select data source.
3. Set DMU registers including data source, interface, clock and enable. Refer to the DMU section in *Calterah Alps Reference Manual*.
4. Set baseband start.

In this example, 2-MB data will be collected for each frame, as shown in [Table 15.4](#). The states SAMPLE and DMP\_MID are enabled, which are the basic states to complete this example. Other states, such as 1D-FFT and 2D-FFT, are not necessary.

Table 15.32: SAMPLE Standard Output Data Collection Example

Register Name	Example Value	Description
<code>CFG_SYS_SIZE RNG_PRD</code>	999 (0x3E7)	Chirp period is $50\mu s$ ( $F_{ADC} = 20\text{MHz}$ )
<code>CFG_SYS_SIZE FLT</code>	1	Down sampling rate is 2
<code>CFG_SYS_SIZE RNG_SKP</code>	19 (0x13)	Skipped sample length from chirp start is $2\mu s$ ( $F_{ADC} = 20\text{MHz}$ )
<code>CFG_SYS_SIZE RNG_BUF</code>	399 (0x18F)	Used sample length after skipping for range FFT is $40\mu s$ ( $F_{ADC} = 20\text{MHz}$ )
<code>CFG_SYS_SIZE RNG_FFT</code>	511 (0x1FF)	Range FFT size is 512
<code>CFG_SYS_SIZE BPM</code>	0	Virtual array mode is off
<code>CFG_SYS_SIZE VEL_BUF</code>	255 (0xFF)	Chirp number in a frame is 256
<code>CFG_SYS_SIZE VEL_FFT</code>	255 (0xFF)	Velocity FFT size is 256
<code>CFG_SYS_TYPE FMCW</code>	1	enable FMCW anti velocity ambiguity
<code>CFG_SAM_SINKER</code>	0	buffer ADC data to MEM_BUF
<code>CFG_SYS_ENABLE</code>	9'b0_0000_1100	enable SAMPLE and DMP_MID
<code>CFG_SYS_START</code>	1	start baseband

---

CHAPTER  
**SIXTEEN**

---

## **FUNCTIONAL SAFETY**

### **16.1 Overview**

Functional safety is very important and required in automotive design, which is specified by the ISO 26262 series of standards. This series of standards ensures a closely watched automotive safety life cycle and ensures that products meet the requirements of certain Automotive Safety Integrity Levels (ASIL). Alps baseband provides a series of functional safety features to ensure that baseband can operate safely and consistently, and is poised to minimize the effects of failures.

#### **16.1.1 Features**

- **Logic BIST:** Baseband supports real-time Logic BIST to monitor the working status of its digital logic circuits.
- **Analog BIST with a shadow bank:** Baseband can trigger a shadow bank to start analog BIST to monitor the working status of analog circuits.
- **ECC:** Baseband supports Error Correction Code (ECC) to monitor the working status of its memory circuit.

## 16.2 Functional Description

### 16.2.1 Logic BIST

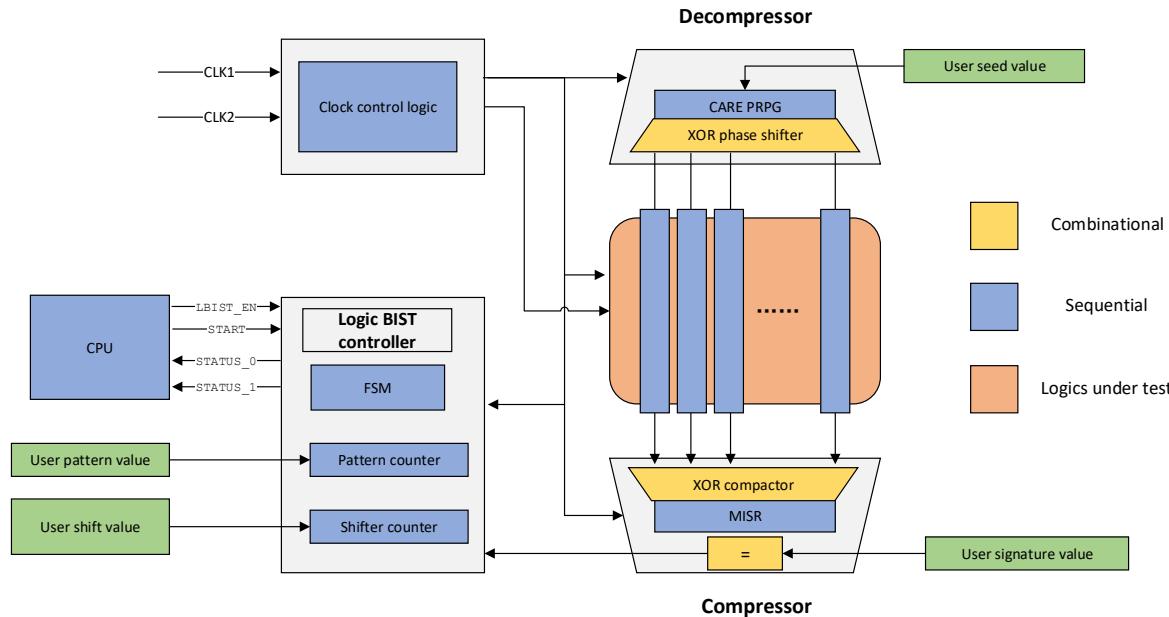


Fig. 16.1: Diagram of Logic BIST

Logic built-in self-test (Logic BIST) helps to reduce the testing complexity by order of magnitude. Logic BIST is circuitry embedded in the chip that performs scan-based structural tests of the design, as shown in Fig. 16.1. This feature gives measurement of fault coverage with minimum vectors and helps to overcome drawbacks of the earlier discussed techniques.

### 16.2.2 Analog BIST with Shadow Bank

Analog BIST is used to test the working status of analog modules, including TIA, VGA, and HPF (high pass filter). Normally, TIA, VGA, and HPF have stable gain for the input signal. If the gain changes obviously, the chip will be considered abnormal. The process flow is shown in Fig. 16.2.

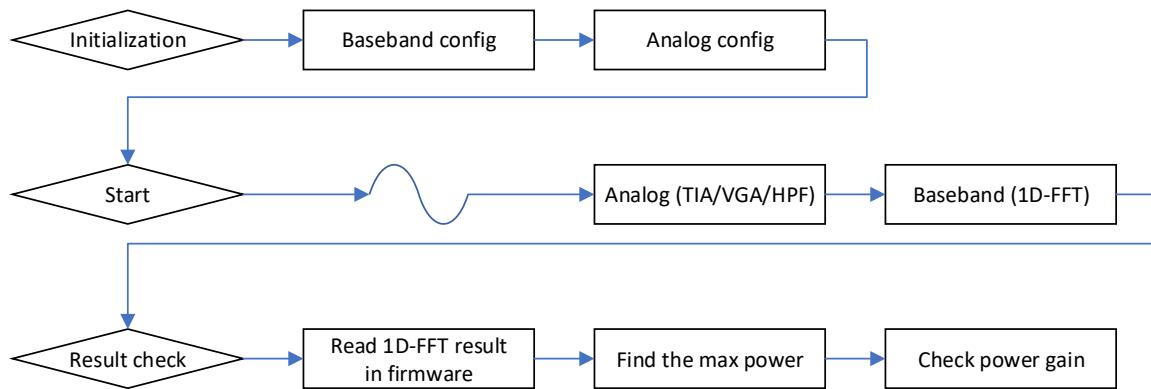


Fig. 16.2: Analog BIST Process Flow

The testing is processed through analog, baseband, and firmware. Baseband shadow bank is dedicated to configuring the related registers for analog BIST. When the shadow bank is enabled, an inner sine wave is triggered and used as the radar signal input. The sine wave goes through TIA, VGA, and HPF and then goes into ADC. Then baseband handles the ADC data and do 1D-FFT processing. Other processing including 2D-FFT, CFAR, and DoA is not needed. The 1D-FFT results is read in firmware and the maximum power is found corresponding to the input sine wave. The maximum power is checked to evaluate whether the analog (TIA/VGA/HPF) gain is correct or not.

### 16.2.3 ECC

All baseband memories support ECC, except AGC code LUT, which is implemented with registers, as shown in Table 16.1. There are altogether 15 RAM or ROM memory cells in baseband, so there are 15 bits corresponding to each RAM or ROM memory cell.

ECC is a method of detecting and then correcting single-bit memory errors, but it cannot correct double-bit memory errors. When writing data to memory, ECC memory uses parity bits to store encrypted code, and the ECC code is stored at the same time. When data is read, the stored ECC code is compared to the ECC code that was generated when the data was read. If the code that was read does not match the stored code, it is decrypted by the parity bits to determine which bit was in error, then this single-bit is immediately corrected. As data is processed, ECC memory is constantly scanning code with a special algorithm to detect and correct single-bit memory errors.

Table 16.1: ECC and Memory

Memory Name	ECC Bit Mask	Description	CPU Access
COD	NULL	AGC code LUT based on registers	Y
SAM	0	ADC sample buffer	Y
WIN	1	FFT windowing LUT	Y
NVE	2	NVE results	Y
BUF	3	FFT output memory	Y
COE	4	CFAR and DoA coefficients LUT	Y
MAC	5	Cascade memory	Y
RLT	6	CFAR or DoA results	Y
AMB	7	anti velocity ambiguity parameters	Y
DML	8	DML setting parameters	Y
FFT_CMP	9	FFT phase compensation LUT	N
FFT_BUF	10	FFT core processing buffer	N
FFT_COE	11	FFT rotate coefficients LUT	N
FFT_NVE	12	NVE processing buffer	N
CFR_BUF	13	CFAR processing buffer	N
DML_BUF	14	DML processing buffer	N

## 16.3 Programming Interfaces

### 16.3.1 Logic BIST

To program for Logic BIST:

- Enable clock by setting `EMU_BB_LBIST_CLK_ENA` (0xB00208) to 1 before using this feature.
- Enable Logic BIST by setting `EMU_BB_LBIST_ENA` (0xB00210) to 1.
- Choose a mode by setting `EMU_BB_LBIST_MODE` (0xB0020C). 0 means normal mode, and 1 means speed mode.

To check the test result, read Bit 0 of `EMU_DIG_ERR_STA` (0xB0042C). To clear the test result, set `EMU_BB_LBIST_CLR` (0xB00214) to 1 before next test.

---

**Note:** For more Logic BIST details, refer to the EMU section in *Calterah Alps Reference Manual*.

---

### 16.3.2 Analog BIST with Shadow Bank

Analog registers and baseband registers should both be initialized.

### 16.3.2.1 Baseband Registers

Besides several common registers that do not fall into any bank, most baseband registers for analog BIST are in the shadow bank, which is designed to just handle analog BIST. These registers in the shadow bank can be activated together (see Section 3.2.2) to simplify the testing process.

- Enable the shadow bank by setting CFG\_SYS\_BNK\_ACT to 4.
- Enable 1D-FFT shared with SAMPLE by setting CFG\_SAM\_SINKER to 1.
- Enable SAMPLE by setting CFG\_SYS\_ENABLE to 9'b0\_0000\_0100.

For the other registers in the shadow bank, refer to [Section C](#). For initialization of those registers, refer to related chapters.

### 16.3.2.2 Analog Registers

Analog BIST can be classified into RF loop test and IF loop test, but these two tests share the same registers, which include the one controlling the enable of the sine wave generator and the one for switching on the testing path. Detailed settings are shown in the following table.



### 16.3. Programming Interfaces

---

Table 16.2: Register Settings for Analog BIST

ADDR	Working Value	Bits	Bit Name	Memory Access	Bit Description
0x00	8'b0000_1001	7:4	Reserved	R/W	Not used
		3:0	fmcw_bank_sel[3:0]		bank selection for FMCW related registers
0x02	8'b0000_0000	7	Reserved	R/W	Not used
		6:4	spi_rxrf_bist_0_en	R/W	Power Enable for BIST 000 = disable 111 = enable
		3:0	spi_rxrf_bist_1_en	R/W	RXRF BIST enable 00000 = IF loop test enable 11111 = RX loop test enable
0x03	8'b0000_0000	7:0	spi_rxrf_bist_2_en	R/W	RXRF BIST enable 0011_1111 = RF loop test enable 1100_0000 = IF loop test enable
0x05	8'b0000_0001	7:4	spi_rxbb_ch1_tstmux_sel_lp_tst[3:0]	R/W	Input test selection for channel 1
		3:0	spi_rxbb_ch1_outmux_sel_lp_tst[3:0]	R/W	Output test selection for channel 1
0x06	8'b0000_0001	7:4	spi_rxbb_ch2_tstmux_sel_lp_tst[3:0]	R/W	Input test selection for channel 2
		3:0	spi_rxbb_ch2_outmux_sel_lp_tst[3:0]	R/W	Output test selection for channel 2
0x07	8'b0000_0001	7:4	spi_rxbb_ch3_tstmux_sel_lp_tst[3:0]	R/W	Input test selection for channel 3
		3:0	spi_rxbb_ch3_outmux_sel_lp_tst[3:0]	R/W	Output test selection for channel 3
0x08	8'b0000_0001	7:4	spi_rxbb_ch4_tstmux_sel_lp_tst[3:0]	R/W	Input test selection for channel 4
		3:0	spi_rxbb_ch4_outmux_sel_lp_tst[3:0]	R/W	Output test selection for channel 4
0x09	8'b0001_0001	7:5	Reserved	R/W	Not used
		4:0	spi_dac_en	R/W	enable for sine wave generator 0 = disable 1 = enable
0x0A	8'b0000_0000	7:0	fmcw_start_freq_lp_tst[7:0]	R/W	32bits FMCW fixed frequency
0x0B	8'b0000_0000	7:0	fmcw_start_freq_lp_tst[15:8]	R/W	
0x0C	8'b0000_0000	7:0	fmcw_start_freq_lp_tst[23:16]	R/W	
0x0D	8'b1000_0000	7:0	fmcw_start_freq_lp_tst[31:24]	R/W	
0x18	8'b0000_0001	7:1	Reserved	R/W	Not used
		0	fmcw_lp_tst_en	R/W	FMCW enable for loop test 0 = disable 1 = enable



### 16.3.3 ECC

The following describes the registers about ECC.

- To enable ECC, in the register `CFG_SYS_ECC_ENA`, set the related bit to 1.
- To check ECC single-bit status, in the register `FDB_SYS_ECC_SB_STATUS`, read the related bit.
- To check ECC double-bit status, in the register `FDB_SYS_ECC_DB_STATUS`, read the related bit.
- To clear ECC single-bit error, in the register `CTL_SYS_ECC_SB_CLR`, set the related bit to 1.
- To clear ECC double-bit error, in the register `CTL_SYS_ECC_DB_CLR`, set the related bit to 1.

## 16.4 Limitation and Constraints

### 16.4.1 Logic BIST

Baseband supports running Logic BIST in frame intervals, as illustrated by Fig. 16.3. This running mechanism will guarantee maximum safety and monitor the logic status in every frame. So this will be a fast response when an abnormal state happens in a frame.

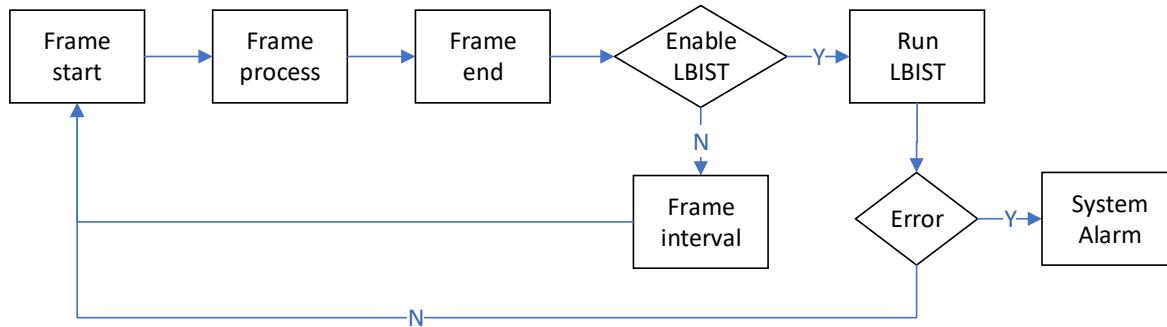


Fig. 16.3: Baseband Logic BIST Running Flow

### 16.4.2 Analog BIST with Shadow Bank

#### 16.4.2.1 `CFG_SYS_SIZE_VEL_FFT`

The analog BIST feature concerns time consumption, so the chirp number should be small to save time consumption in the SAMPLE state. And because only SAMPLE and 1D-FFT processing is needed for analog BIST, `CFG_SYS_SIZE_VEL_FFT` can be set to 0, which means only 1 chirp in a frame. With this setting, baseband can complete the SAMPLE and 1D-FFT processing in the least amount of time.

## 16.5. Examples

---

### 16.4.2.2 CFG\_SAM\_FORCE

Normally, the start of baseband SAMPLE relies on FMCW, which sends a start signal to let SAMPLE begin collecting input data. However, when analog BIST is working, FMCW is not running, so no start signal will be sent out. So another start signal CFG\_SAM\_FORCE is designed to handle this situation.

To start SAMPLE when analog BIST is working, set ``CFG\_SAM\_FORCE`` to 1 after all initialization is ready.

## 16.4.3 ECC

ECC bits and its corresponding memory are shown in [Table 16.1](#).

### 16.4.3.1 Single-Bit Error to IRQ

All 15 bits of ECC single-error will perform *OR* operation and then will be connected to a CPU IRQ\_23. So if any one bit is a single-bit error, the CPU IRQ\_23 will be raised. Besides, there is a feedback register FDB\_SYS\_ECC\_SB\_STATUS to indicate the single-error for CPU.

### 16.4.3.2 Double-Bit Error to EMU

Double-bit ECC errors are not connected to CPU IRQ. Most of them are connected to EMU, except Bits [14:10]. Their corresponding memories are shown in [Table 16.1](#).

- Bits [4:0] of ECC double-error will perform *OR* operation and then will be connected to EMU.
- Bits [9:5] of ECC double-error will perform *OR* operation and then will be connected to EMU.
- Bits [14:10] of ECC double-error are not connected to EMU, but there is a feedback register FDB\_SYS\_ECC\_DB\_STATUS to indicate the double-error for CPU.

## 16.5 Examples

[Table 16.3](#) gives an example of how to set Baseband for analog BIST.



### 16.5.1 Baseband Settings

Table 16.3: Example Settings for Analog BIST

Registers Name	Example Value	Description
CFG_SAM_SINKER	1	Enable 1D-FFT shared with SAMPLE by setting CFG_SAM_SINKER to 1
CFG_SYS_BNK_ACT	4	switch to bank 4(shadow bank)
CFG_SYS_BNK_MOD	0	single bank mode
CFG_SYS_ENABLE	9'b0_0000_0100	enable SAMPLE state
CFG_SYS_SIZE_RNG_PRD	1500	Chirp period is $60\mu s$ ( $F_{ADC} = 25MHz$ )
CFG_SYS_SIZE_FLT	0	no down sampling
CFG_SYS_SIZE_RNG_SKP	49	Skipped sample length from chirp start is $2\mu s$ ( $F_{ADC} = 25MHz$ )
CFG_SYS_SIZE_RNG_BUF	999	Used sample length after skipping for range FFT is $40\mu s$ ( $F_{ADC} = 25MHz$ )
CFG_SYS_SIZE_RNG_FFT	1023	Range FFT size is 1024
CFG_SYS_SIZE_BPM	0	Virtual array mode is off
CFG_SYS_SIZE_VEL_BUF	0	Chirp number in a frame is 1
CFG_SYS_SIZE_VEL_FFT	0	Velocity FFT size is 1
CFG_FFT_SHFT_RNG	1023	enable 10 stage shifter, stage 1~10
CFG_FFT_SHFT_VEL	1	enable 8 stage shifer, stage 1
CFG_FFT_NO_WIN	1	disable windowing in FFT

### 16.5.2 Firmware Flow

The following pseudo-code shows the processing flow in firmware.

```

1 #define BANK4 4
2 #define QUEUE_BANK1_BANK0 4'b00011
3 #define ROTATE_MODE 1
4 #define SINGLE_MODE 0
5 #define MEM_BUF 4
6 #define CFG_SYS_BNK_MODE 0xC0_0004
7 #define CFG_SYS_BNK_ACT 0xC0_0008
8 #define CFG_SYS_BNK_ACT 0xC0_0464
9 #define CFG_SYS_MEM_ACT 0xC0_0014
10
11 int analog_bist()
12 {
13
14     // initialize baseband registers
15     reg_write(CFG_SYS_BNK_ACT, BANK4);           // activate BANK4
16     init_register_bank4();                      // initialize registers in SAMPLE, FFT
17     reg_write(CFG_SYS_BNK_MODE, SINGLE_MODE);    // enable single mode
18     reg_write(CTL_SYS_BNK_RST, 1);              // reset bank selection to BANK0
19
20     // initialize analog registers
21     init_register_analog();
22
23     // start
24     reg_write(CFG_SAM_FORCE, 1);                // force start signal
25
26     while(baseband_is_running() == true) // wait baseband

```

(continues on next page)



## 16.5. Examples

---

(continued from previous page)

```
27 ;  
28  
29 // check 1D-FFT max power  
30 reg_write(CFG_SYS_MEM_ACT, MEM_BUF); // switch to FFT result RAM  
31 check_max_fft_power(); // find and check power gain  
32  
33 return 0;  
34 }
```



## CASCADE MODE

In some applications, the larger number of RX channels is preferred. As a response to this type of requirement, Alps CAL77S244-AE provides a way of cascading at the system level.

---

**Note:** Other chips in the Alps family, including CAL77S224-AE, CAL77S244-AB, and CAL77S244-IB, do not support cascade mode.

---

### 17.1 Overview

By cascading two Alps CAL77S244-AE chips, one can build a radar system with 4 TX channels and 8 RX channels, as roughly illustrated by Fig. 17.1. In particular, the 4 TX channels and 8 RX channels are shared at the RF level, and post-processing data is exchanged through on-chip interfaces.

The challenge of such a system is the synchronization of the two chips. To ensure that the system works properly, synchronization of the two chips must be achieved at RF, ADC, data, and CPU levels. Alps CAL77S244-AE features many built-in synchronization mechanisms to ease the system design.

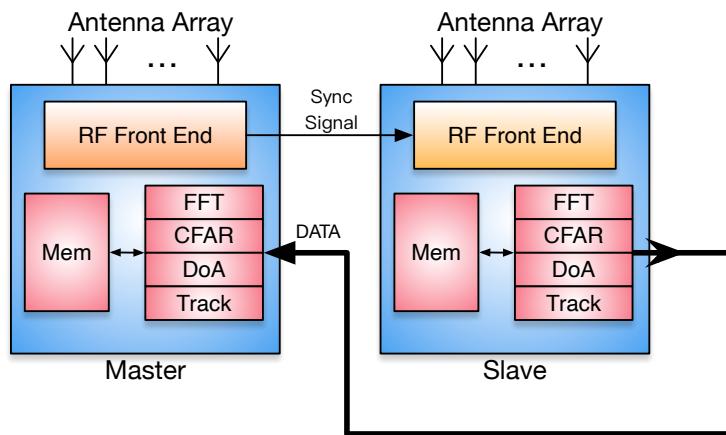


Fig. 17.1: Cascade System with Two Alps CAL77S244-AE ICs

## 17.2. Functional Description

### 17.1.1 Features

- **Sync-up mechanisms:** Enable the system to be synchronized at RF, ADC, and data levels through designed hardware.
- **Full-feature support:** Almost all features of a single Alps chip can be extended to a cascade system.

## 17.2 Functional Description

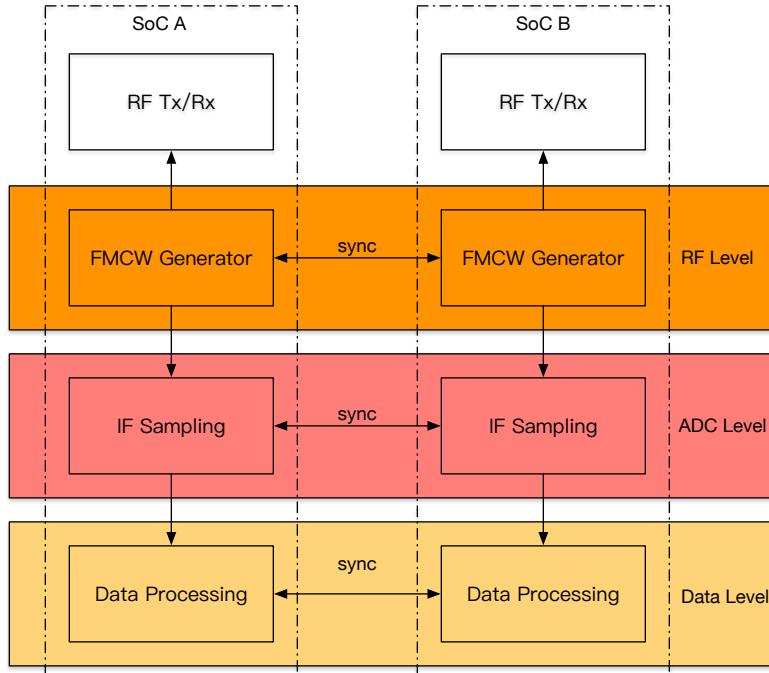


Fig. 17.2: High-level Block Diagram of a Cascade System

A high-level block diagram of 2-Alps cascade system is shown in Fig. 17.2. From the perspectives of structure and function, one chip is a *master* (SoC A) and the other is a *slave* (SoC B). The synchronization is achieved at the following levels:

- At the RF level, the *slave* and the *master* share signals such as LO to make sure that both receivers work coherently;
- At the IF level, the synchronization of ADC is critical. Otherwise, frequency-dependent phase difference between two receivers will occur, which affects the DoA performance.
- At the data level, partial results of the *slave* (results after CFAR) are transmitted to the *master* and the *master* then can take advantages of joint processing.

### 17.2.1 Synchronization

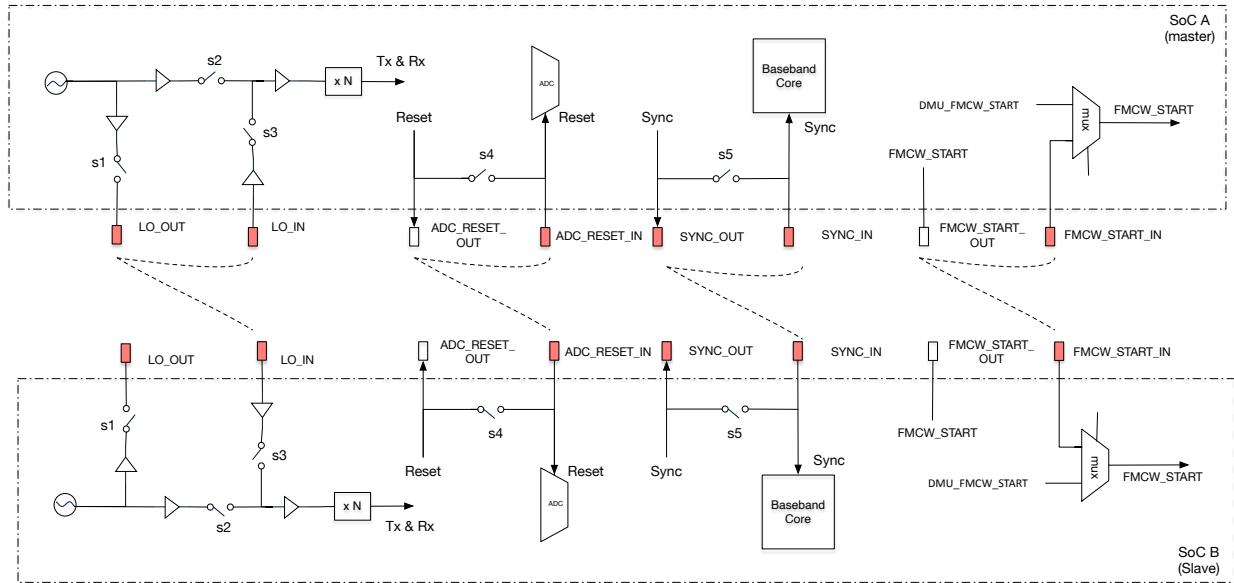


Fig. 17.3: Hardware Synchronization Diagram

Fig. 17.3 shows the hardware-level synchronization schemes used by Alps. In particular, RF, ADC, baseband core, and FMCW generator need to be in sync:

- RF: The FMCW PLL in *slave* should be turned off, and the *slave* should use LO from the *master*.
- ADC: ADCs of the *master* and *slave* should be reset simultaneously to ensure that ADC sampling starts at the same time. The reset should be done each time the system is powered on.
- Baseband core: Baseband cores of the two receivers should start running at the same time.
- FMCW generator: The FMCW generator in the *slave* should start running at the same time with that in the *master*. However, FMCW generator of the *slave* **should not** drive its own PLL, for the PLL is not in use. The purpose of keeping both FMCW generators running at the same time is to keep radio control at two RX sides in sync.

Conceptually, the *master* and the *slave* should be connected as the dashed lines shown in Fig. 17.3<sup>1</sup>. Internal configuration of the *master* and *slave* are listed as follows:

- RF: In the *master*, s1 and s3 should be closed and s2 should be open; In the *slave*, the s1 should be open and s3 should be closed.
- ADC: In both the *master* and the *slave*, s4 should be open. Reset signals should come from the *master*'s CPU.
- Baseband core: In both the *master* and the *slave*, s5 should be open. Sync signals should come from the *master*'s CPU (by forcing *master*'s baseband manually) or from the *master*'s FMCW generator.
- FMCW generator: Both the *master* and the *slave* should switch the mux to take the signal from the pad. The FMCW start signal should come from the *master*'s CPU.

<sup>1</sup> For details about pin mux constraints and layout requirement, refer to [calterah2020].

## 17.3. Limitation and Constraints

---

### 17.2.2 Data Exchange and Process Flow

For each frame, a general data exchange and process flow is as follows:

1. Both the *master* and *slave* run independently till CFAR and then stops.
2. After CFAR, the *slave* passes its CFAR results together with *corresponding* FFT results to the *master* through SPI or QSPI. CPU will be involved in this process.
3. The *master* merges the *slave*'s CFAR results with its own results and writes the *slave*'s FFT results to the Cascade Memory (listed in [Table 3.3](#)).
4. The *master* forces its own baseband core to run DoA to get the final results.

The procedure is also illustrated by Fig. 17.4, where blocks marked in red is mainly done by hardware and other blocks are software-driven.

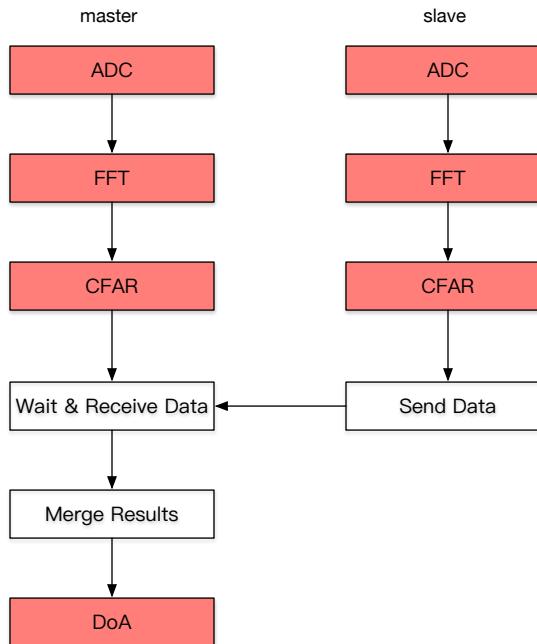


Fig. 17.4: Data Exchange and Process Diagram

## 17.3 Limitation and Constraints

- All pads marked as pink in [Fig. 17.3](#) are dedicated. For other pads, one can use any GPIO pads.
- When the anti velocity ambiguity with chirp shifting feature is on, at most 3 TX groups are supported instead of 4.
- When the anti velocity ambiguity with chirp shifting feature is on, only the master does anti velocity ambiguity.

## 17.4 Software and Config Suggestions

Ultimately, the whole cascade system is triggered by the *master*'s FMCW\_START, which is controlled by CPU. So before raising the start signal, one should make sure baseband cores and FMCW generators of both the *master* and the *slave* are ready to run. Since the *slave* and the *master* are running with different CPUs, it is critical to keep them in sync. Otherwise, out of synchronization will occur at the signal level.

To ensure the synchronization between the two CPUs, it is suggested that one should use GPIO pins to connect the *slave* (output) to the *master* (input). Configure the *master*'s GPIO port to generate edge-triggered IRQ (both positive and negative). Then the firmware of the *master* should do the following:

1. Install an IRQ handler for the GPIO, which sets a software flag to `true` whenever getting the IRQ;
2. Get Baseband core and FMCW ready to run;
3. Check the software flag until it becomes `true`;
4. Set the flag to `false`;
5. Trigger the whole system to run by raising the signal FMCW\_START;
6. Go to Step 3 for the next frame.

The firmware of the *slave* should do the following:

1. Get Baseband core and FMCW ready to run;
2. Flip the GPIO's level;
3. Do certain processing;
4. Go to Step 2 when it is ready for the next frame;

In other words, firmware needs to make sure that the two CPUs are in sync before each frame or before baseband runs.



#### **17.4. Software and Config Suggestions**

---



**MEMORY MAPPING TABLE**

Table A.1: Memory Mapping

Name	Mapping Space	Valid Size (Byte)	Bus	Cacheable	Description
ROM	0x000000-0x003FFF	16 K	AHB	Yes	Boot ROM
ICCM	0x100000-0x107FFF	32 K		Yes	Instruction closely coupled memory
XIP (mem)	0x300000-0x6FFFFFF	4 M/16 K	AHB	Yes	Memory interface of XIP; 4 M when XIP is on; 16 K when XIP is off, starting from 0x6FC000
RAM	0x770000-0x7FFFFFF	576 K	AHB	Yes	RAM
RAM (shared)	0x800000-0x8FFFFFF	1 M	AHB	Yes	Baseband shared memory
DCCM	0xA00000-0xA07FFF	32 K		No	Data closely coupled memory
EMU	0xB00000-0xB00FFF	4 K	APB	No	Register interface of EMU (Error Management Unit)
Timer	0xB10000-0xB100FF	256	APB	No	Register interface of Timer
CLKGEN	0xB20000-0xB20FFF	4 K	APB	No	Register interface of clock and reset management modules
UART0	0xB30000-0xB300FF	256	APB	No	Register interface of UART 0
UART1	0xB40000-0xB400FF	256	APB	No	Register interface of UART 1
I2C_M	0xB50000-0xB500FF	256	APB	No	Register interface of I <sup>2</sup> C master
SPI_M0	0xB60000-0xB600FF	256	APB	No	Register interface of SPI master 0
SPI_M1	0xB70000-0xB700FF	256	APB	No	Register interface of SPI master 1
SPI_S	0xB80000-0xB800FF	256	APB	No	Register interface of SPI slave
QSPI_M	0xB90000-0xB900FF	256	APB	No	Register interface of QSPI master
DMU (radio_ctrl)	0xBA0000-0xBA00FF	256	APB	No	Register interface of radio configuration
DMU (debug_bus)	0xBA0100-0xBA01FF	256	APB	No	Register interface of Debug Bus
DMU (io_mux)	0xBA0200-0xBA02FF	256	APB	No	Register interface of IO mux
DMU (sys_config)	0xBA0300-0xBA03FF	256	APB	No	Register interface of system configuration
DMU (sw_interrupt)	0xBA0400-0xBA04FF	256	APB	No	Register interface of software interrupt
DMU (cpu_irq_mask)	0xBA0500-0xBA05FF	256	APB	No	Register interface of CPU interrupt mask
CAN_0 <sup>1</sup>	0xBB0000-0xBB3FFF	16 K	APB	No	Register interface of CAN 0
CAN_1 <sup>1</sup>	0xBC0000-0xBC3FFF	16 K	APB	No	Register interface of CAN 1
PWM	0xBD0000-0xBD00FF	256	APB	No	Register interface of PWM
GPIO	0xBE0000-0xBE00FF	256	APB	No	Register interface of GPIO
OTP	0xBF0000-0xBF01FF	512	APB	No	OTP memory
BB (reg)	0xC00000-0xC00FFF	4 K	AHB	No	Register interface of baseband
CRC (reg)	0xC10000-0xC100FF	256	AHB	No	Hardware CRC
DMA	0xC20000-0xC203FF	1 K	AHB	No	Register interface of DMA
XIP (reg)	0xD00000-0xD001FF	512	APB	No	Register interface of XIP
LVDS	0xD10000-0xD100FF	256	APB	No	Register interface of LVDS
BB (mem)	0xE00000-0xFFFFFFF	2 M	AHB	No	Memory interface of baseband

<sup>1</sup> Reserved for Alps CAL77S244-IB, because CAL77S244-IB does not support CAN functions.



## BASEBAND REGISTER TABLES

Table B.1: Baseband Register Table

Name	Addr	Access	BW	Bank	DV	Description	Function Description
CTL_SYS_START	0x0	W/SC	1	N	0	start bit for baseband	high active: write 1 to start baseband (this bit is self-cleared, no need to write 0)
CFG_SYS_BNK_MODE	0x4	R/W	2	N	0	working mode of banks	0 -> "single" mode, run the bank specified by register CFG_SYS_BNK_ACT for each start 1 -> reserved 2 -> "rotate" mode, run the banks specified by register CFG_SYS_BNK_QUE, one bank for each start
CFG_SYS_BNK_ACT	0x8	R/W	3	N	0	active bank for APB access enabled bank in "single" mode	x -> bank x is enabled for APB access and in "single" mode (0<=x<=4)
CFG_SYS_BNK_QUE	0xC	R/W	5	N	0	enabled banks in "rotate" mode	high active: bit mask with 1 means enabled in "rotate" modes and 0 means disabled bit 4: bank 4 bit 3: bank 3 bit 2: bank 2 bit 1: bank 1 bit 0: bank 0
CTL_SYS_BNK_RST	0x10	W/SC	1	N	0	reset for bank mode	high active: write 1 to reset band mode (this bit is self-cleared, no need to write 0)
CFG_SYS_MEM_ACT	0x14	R/W	4	N	0	active memory for CPU access	memory index for CPU access 9: MEM_DML 8: MEM_AMB 7: MEM_RLT 6: MEM_MAC 5: MEM_COE 4: MEM_BUF 3: MEM_NVE 2: MEM_WIN 1: MEM_SAM 0: MEM_COD

continues on next page

Table B.1 – continued from previous page

Name	Addr	Access	BW	Bank	DV	Description	Function Description
CFG_SYS_ENABLE	0x18	R/W	9	Y	0	enable bits for sub-function	high active: bit value with 1 means enabled bit 8 -> DMP_FNL bit 7 -> DoA bit 6 -> CFAR bit 5 -> 2D-FFT bit 4 -> 1D-FFT bit 3 -> DMP_MID bit 2 -> SAM bit 1 -> HIL bit 0 -> AGC
CFG_SYS_TYPE_FMCW	0x1C	R/W	1	Y	0		high active, 1 means anti velocity ambiguity is enabled
CFG_SYS_SIZE RNG_PRD	0x20	R/W	32	Y	0	chirp length	Format: FXI(32, U) x -> each chirp has x+1 points
CFG_SYS_SIZE FLT	0x24	R/W	4	Y	0	down sample rate	Format: FXI(4, U) x -> sample 1 data out of x+1 data
CFG_SYS_SIZE RNG_SKP	0x28	R/W	32	Y	0	skip length	Format: FXI(32, U) x -> for each chirp, skip first x+1 points (x+1 must be even)
CFG_SYS_SIZE RNG_BUF	0x2C	R/W	11	Y	0	buffer length	Format: FXI(11, U) x -> for each chirp, buffers x+1 points after skip and send them to the sinker (x+1 must be even)
CFG_SYS_SIZE RNG_FFT	0x30	R/W	11	Y	0	1DFFT length	Format: FXI(11, U) $2^x - 1 \rightarrow$ 1DFFT has $2^x$ points ( $6 \leq x \leq 11$ )
CFG_SYS_SIZE BPM	0x34	R/W	2	Y	0	BPM size	Format: FXI(2, U) x -> each BPM group has x+1 chirps
CFG_SYS_SIZE VEL_BUF	0x38	R/W	10	Y	0	chirp number	Format: FXI(10, U) x -> each frame has x+1 chirps
CFG_SYS_SIZE VEL_FFT	0x3C	R/W	10	Y	0	2DFFT length	Format: FXI(10, U) $2^x - 1 \rightarrow$ 2DFFT has $2^x$ points ( $6 \leq x \leq 10$ )
CFG_SYS_FRMT ADC	0x40	R/W	1	Y	0	selection bit for data format of ADC data	0 -> 4 valid data in every 4 samples 1 -> 4 valid data in every 5 samples

continues on next page



Table B.1 – continued from previous page

Name	Addr	Access	BW	Bank	DV	Description	Function Description
CFG_SYS_IRQ_ENA	0x44	R/W	3	N	0	enable bits for IRQ	high active: bit mask with 1 means enabled and 0 means disabled bit 2: IRQ indicate that sample of ADC data is done bit 1: IRQ indicate that one bank is done bit 0: IRQ indicate that baseband is done
CTL_SYS_IRQ_CLR	0x48	W/SC	3	N	0	clear bits for IRQ	high active: write 1 to clear the specific IRQ corresponding to register CFG_SYS_IRQ_ENA (this bit is self-cleared, no need to write 0)
CFG_SYS_ECC_ENA	0x4C	R/W	15	N	0	enable bits for ECC	high active: bit mask with 1 means enabled and 0 means disabled bit 14: ECC error detector for BUF_DML bit 13: ECC error detector for BUF_CFR bit 12: ECC error detector for BUF_FFT_NVE bit 11: ECC error detector for BUF_FFT_COE bit 10: ECC error detector for BUF_FFT_CORE bit 9: ECC error detector for BUF_FFT_COMPEN_COE bit 8: ECC error detector for MEM_DML bit 7: ECC error detector for MEM_AMB bit 6: ECC error detector for MEM_RLT bit 5: ECC error detector for MEM_MAC bit 4: ECC error detector for MEM_COE bit 3: ECC error detector for MEM_BUF bit 2: ECC error detector for MEM_NVE bit 1: ECC error detector for MEM_WIN bit 0: ECC error detector for MEM_SAM ((AGC) code memory (mem_cod) is implemented with registers)
CTL_SYS_ECC_SB_CLR	0x50	W/SC	15	N	0	clear bits for ECC (single-bit error)	high active: write 1 to clear the specific ECC (single-bit error) corresponding to register CFG_SYS_ECC_ENA (this bit is self-cleared, no need to write 0)
CTL_SYS_ECC_DB_CLR	0x54	W/SC	15	N	0	clear bits for ECC (double-bit error)	high active: write 1 to clear the specific ECC (double-bit error) corresponding to register CFG_SYS_ECC_ENA (this bit is self-cleared, no need to write 0)
CFG_SYS_TYP_ARB	0x58	R/W	1	N	0	memory share mode	0 means static mode and 1 means dynamic mode

continues on next page



Table B.1 – continued from previous page

Name	Addr	Access	BW	Bank	DV	Description	Function Description
FDB_SYS_STATUS	0x5C	RO	4			current status of baseband	0 -> idle; 1 -> AGC is busy; 2 -> HIL and 1DFFT is busy; 3 -> Sample busy; 4 -> Middle data dump is busy; 5 -> 1DFFT is busy; 6 -> 2DFFT is busy; 7 -> CFAR is busy; 8 -> DoA is busy; 9 -> Final data dump is busy; 10-> Reserved
FDB_SYS_BNK_ACT	0x60	RO	3			current active bank of baseband	x -> bank x is active for baseband process (0<=x<=3)
FDB_SYS_IRQ_STATUS	0x64	RO	3			current status of IRQ	high active: bit value with 1 means the specific IRQ corresponding to register CFG_SYS_IRQ_ENA is raised
FDB_SYS_ECC_SB_STATUS	0x68	RO	15			current status of ECC (single-bit error)	high active: bit value with 1 means the specific ECC (single-bit error) corresponding to register CFG_SYS_ECC_ENA is raised
FDB_SYS_ECC_DB_STATUS	0x6C	RO	15			current status of ECC (double-bit error)	high active: bit value with 1 means the specific ECC (double-bit error) corresponding to register CFG_SYS_ECC_ENA is raised
CFG_AGC_SAT_THR_TIA	0x200	R/W	24	Y	0	threshold for saturation decision of TIA	Format: FXI(16, U) x -> if active periods of saturation indicator in one chirp is more than x, the corresponding part is considered as saturated
CFG_AGC_SAT_THR_VGA1	0x204	R/W	24	Y	0	threshold for saturation decision of VGA1	This register shares the same format with CFG_AGC_SAT_THR_TIA
CFG_AGC_SAT_THR_VGA2	0x208	R/W	24	Y	0	threshold for saturation decision of VGA2	This register shares the same format with CFG_AGC_SAT_THR_TIA
CTL_AGC_SAT_CNT_CLR_FRA	0x20C	W/SC	1	N	0	clear frame counter for saturation indicator	high active: bit value with 1 means clearing frame counter
CFG_AGC_DAT_MAX_SEL	0x210	R/W	2	Y	0	selection bits for ADC data used to estimate ADC power	2 -> 3rd maximum absolute value of ADC data 1 -> 2nd maximum absolute value of ADC data 0 -> 1st maximum absolute value of ADC data
CFG_AGC_CODE_LNA_0	0x214	R/W	4	N	0	RF bits for level 0 of LNA	RF bits for level 0 of LNA
CFG_AGC_CODE_LNA_1	0x218	R/W	4	N	0	RF bits for level 1 of LNA	RF bits for level 1 of LNA
CFG_AGC_CODE_TIA_0	0x21C	R/W	4	N	0	RF bits for level 0 of TIA	RF bits for level 0 of TIA
CFG_AGC_CODE_TIA_1	0x220	R/W	4	N	0	RF bits for level 1 of TIA	RF bits for level 1 of TIA

continues on next page



Table B.1 – continued from previous page

Name	Addr	Access	BW	Bank	DV	Description	Function Description
CFG_AGC_CODE_TIA_2	0x224	R/W	4	N	0	RF bits for level 2 of TIA	RF bits for level 2 of TIA
CFG_AGC_CODE_TIA_3	0x228	R/W	4	N	0	RF bits for level 3 of TIA	RF bits for level 3 of TIA
CFG_AGC_GAIN_MIN	0x22C	R/W	10	N	0	minimum gain of LNA, TIA, VGA1, VGA2	Format: FXI(9, S) x -> the corresponding gain of the above settings is x DB
CFG_AGC_CDGN_INIT	0x230	R/W	17	N	0	initial setting and the corresponding gain by “setting”, it refers to a specific level group of LNA, TIA, VGA1 & VGA2	bit 16: x -> LNA is set to level x (0<=x<=1) bit 15-14: x -> TIA is set to level x (0<=x<=3) bit 13-11: x -> VGA1 is set to level x (0<=x<=7) bit 10-8: x -> VGA2 is set to level x (0<=x<=7) bit 7-0: Format: FXI(8, U) x -> the corresponding gain of the above settings is x DB
CFG_AGC_CDGN_C0_0	0x234	R/W	17	N	0	setting 0-0	This register shares the same format with register CFG_AGC_CDGN_INIT
CFG_AGC_CDGN_C0_1	0x238	R/W	17	N	0	setting 0-0	This register shares the same format with register CFG_AGC_CDGN_INIT
CFG_AGC_CDGN_C0_2	0x23C	R/W	17	N	0	setting 0-1	This register shares the same format with register CFG_AGC_CDGN_INIT
CFG_AGC_CDGN_C1_0	0x240	R/W	17	N	0	setting 1-0	This register shares the same format with register CFG_AGC_CDGN_INIT
CFG_AGC_CDGN_C1_1	0x244	R/W	17	N	0	setting 1-1	This register shares the same format with register CFG_AGC_CDGN_INIT
CFG_AGC_CDGN_C1_2	0x248	R/W	17	N	0	setting 1-2	This register shares the same format with register CFG_AGC_CDGN_INIT
CFG_AGC_CDGN_C1_3	0x24C	R/W	17	N	0	setting 1-3	This register shares the same format with register CFG_AGC_CDGN_INIT
CFG_AGC_CDGN_C1_4	0x250	R/W	17	N	0	setting 1-4	This register shares the same format with register CFG_AGC_CDGN_INIT
CFG_AGC_CDGN_C1_5	0x254	R/W	17	N	0	setting 1-5	This register shares the same format with register CFG_AGC_CDGN_INIT
CFG_AGC_CDGN_C1_6	0x258	R/W	17	N	0	setting 1-6	This register shares the same format with register CFG_AGC_CDGN_INIT
CFG_AGC_CDGN_C1_7	0x25C	R/W	17	N	0	setting 1-7	This register shares the same format with register CFG_AGC_CDGN_INIT
CFG_AGC_CDGN_C1_8	0x260	R/W	17	N	0	setting 1-8	This register shares the same format with register CFG_AGC_CDGN_INIT
CFG_AGC_CHCK_ENA	0x264	R/W	1	Y	0	enable bit for check adc buffer saturation	high active: bit value with 1 means adc buffer saturation check is enabled
CFG_AGC_ALIGN_EN	0x268	R/W	1	Y	0	enable bit for aligning AGC of 4 channels	high active: bit value with 1 means AGC alignment is enabled
CFG_AGC_CMPN_EN	0x26C	R/W	1	Y	0	enable bit for ADC compensation	high active: bit value with 1 means ADC compensation is enabled
CFG_AGC_CMPN_ALIGN_EN	0x270	R/W	1	Y	0	enable bit for aligning ADC compensation of 4 channels	high active: bit value with 1 means ADC compensation alignment is enabled
CFG_AGC_CMPN_LVL	0x274	R/W	4	Y	0	ADC compensation reference level	Format: FXI(4, U)
CFG_AGC_DB_TARGET	0x278	R/W	8	Y	0	AGC target power value(DB)	Format: FXR(8, 7, U)

continues on next page



Table B.1 – continued from previous page

Name	Addr	Access	BW	Bank	DV	Description	Function Description
CFG_AGC_IRQ_ENA	0x27C	R/W	12	N	0	enable bits for IRQ	high active: bit mask with 1 means enabled and 0 means disabled bit 11: IRQ of TIA in channel 0 bit 10: IRQ of TIA in channel 1 bit 9: IRQ of TIA in channel 2 bit 8: IRQ of TIA in channel 3 bit 7: IRQ of VGA1 in channel 0 bit 6: IRQ of VGA1 in channel 1 bit 5: IRQ of VGA1 in channel 2 bit 4: IRQ of VGA1 in channel 3 bit 3: IRQ of VGA2 in channel 0 bit 2: IRQ of VGA2 in channel 1 bit 1: IRQ of VGA2 in channel 2 bit 0: IRQ of VGA2 in channel 3
CTL_AGC_IRQ_CLR	0x280	W/SC	12	N	0	clear IRQ status	high active: bit mask with 1 means enabled and 0 means disabled bit 11: IRQ of TIA in channel 0 bit 10: IRQ of VGA1 in channel 0 bit 9: IRQ of VGA2 in channel 0 bit 8: IRQ of TIA in channel 1 bit 7: IRQ of VGA1 in channel 1 bit 6: IRQ of VGA2 in channel 1 bit 5: IRQ of TIA in channel 2 bit 4: IRQ of VGA1 in channel 2 bit 3: IRQ of VGA2 in channel 2 bit 2: IRQ of TIA in channel 3 bit 1: IRQ of VGA1 in channel 3 bit 0: IRQ of VGA2 in channel 3
FDB_AGC_COD_C0	0x284	RO	9			current setting of LNA, TIA, VGA1 & VGA2 in channel 0	bit 8: x -> LNA is set to level x ( $0 \leq x \leq 1$ ) bit 7-6: x -> TIA is set to level x ( $0 \leq x \leq 3$ ) bit 5-3: x -> VGA1 is set to level x ( $0 \leq x \leq 7$ ) bit 2-0: x -> VGA2 is set to level x ( $0 \leq x \leq 7$ )
FDB_AGC_COD_C1	0x288	RO	9			current setting of LNA, TIA, VGA1 & VGA2 in channel 1	This register shares the same format with register FDB_AGC_COD_C0
FDB_AGC_COD_C2	0x28C	RO	9			current setting of LNA, TIA, VGA1 & VGA2 in channel 2	This register shares the same format with register FDB_AGC_COD_C0
FDB_AGC_COD_C3	0x290	RO	9			current setting of LNA, TIA, VGA1 & VGA2 in channel 3	This register shares the same format with register FDB_AGC_COD_C0
FDB_AGC_SAT_CNT_TIA_C0	0x294	RO	24			counter for saturation indicator of TIA in channel 0	Format: FXI(8, U) Bits[7:0]: x -> Saturation counters for TIA block in a chirp in channel 0 is x. Bits[15:8]: x -> Saturation counters for TIA block in a frame in channel 0 is x. Bits[23:16]: x -> The number of saturated frames for TIA block in channel 0 is x.
FDB_AGC_SAT_CNT_TIA_C1	0x298	RO	24			counter for saturation indicator of TIA in channel 1	This register shares the same format with register FDB_AGC_SAT_CNT_TIA_C0

continues on next page



Table B.1 – continued from previous page

Name	Addr	Access	BW	Bank	DV	Description	Function Description
FDB_AGC_SAT_CNT_TIA_C2	0x29C	RO	24			counter for saturation indicator of TIA in channel 2	This register shares the same format with register FDB_AGC_SAT_CNT_TIA_C0
FDB_AGC_SAT_CNT_TIA_C3	0x2A0	RO	24			counter for saturation indicator of TIA in channel 3	This register shares the same format with register FDB_AGC_SAT_CNT_TIA_C0
FDB_AGC_SAT_CNT_VGA1_C0	0x2A4	RO	24			counter for saturation indicator of VGA1 in channel 0	This register shares the same format with register FDB_AGC_SAT_CNT_TIA_C0
FDB_AGC_SAT_CNT_VGA1_C1	0x2A8	RO	24			counter for saturation indicator of VGA1 in channel 1	This register shares the same format with register FDB_AGC_SAT_CNT_TIA_C0
FDB_AGC_SAT_CNT_VGA1_C2	0x2AC	RO	24			counter for saturation indicator of VGA1 in channel 2	This register shares the same format with register FDB_AGC_SAT_CNT_TIA_C0
FDB_AGC_SAT_CNT_VGA1_C3	0x2B0	RO	24			counter for saturation indicator of VGA1 in channel 3	This register shares the same format with register FDB_AGC_SAT_CNT_TIA_C0
FDB_AGC_SAT_CNT_VGA2_C0	0x2B4	RO	24			counter for saturation indicator of VGA2 in channel 0	This register shares the same format with register FDB_AGC_SAT_CNT_TIA_C0
FDB_AGC_SAT_CNT_VGA2_C1	0x2B8	RO	24			counter for saturation indicator of VGA2 in channel 1	This register shares the same format with register FDB_AGC_SAT_CNT_TIA_C0
FDB_AGC_SAT_CNT_VGA2_C2	0x2BC	RO	24			counter for saturation indicator of VGA2 in channel 2	This register shares the same format with register FDB_AGC_SAT_CNT_TIA_C0
FDB_AGC_SAT_CNT_VGA2_C3	0x2C0	RO	24			counter for saturation indicator of VGA2 in channel 3	This register shares the same format with register FDB_AGC_SAT_CNT_TIA_C0
FDB_AGC_DAT_MAX_1ST_C0	0x2C4	RO	12			1st maximum absolute value of ADC data in channel 0	Format: FXI(12, U) x -> maximum absolute value of ADC data is x
FDB_AGC_DAT_MAX_1ST_C1	0x2C8	RO	12			1st maximum absolute value of ADC data in channel 1	This register shares the same format with register FDB_AGC_DAT_MAX_1ST_C0
FDB_AGC_DAT_MAX_1ST_C2	0x2CC	RO	12			1st maximum absolute value of ADC data in channel 2	This register shares the same format with register FDB_AGC_DAT_MAX_1ST_C0
FDB_AGC_DAT_MAX_1ST_C3	0x2D0	RO	12			1st maximum absolute value of ADC data in channel 3	This register shares the same format with register FDB_AGC_DAT_MAX_1ST_C0
FDB_AGC_DAT_MAX_2ND_C0	0x2D4	RO	12			2nd maximum absolute value of ADC data in channel 0	This register shares the same format with register FDB_AGC_DAT_MAX_1ST_C0
FDB_AGC_DAT_MAX_2ND_C1	0x2D8	RO	12			2nd maximum absolute value of ADC data in channel 1	This register shares the same format with register FDB_AGC_DAT_MAX_1ST_C0
FDB_AGC_DAT_MAX_2ND_C2	0x2DC	RO	12			2nd maximum absolute value of ADC data in channel 2	This register shares the same format with register FDB_AGC_DAT_MAX_1ST_C0
FDB_AGC_DAT_MAX_2ND_C3	0x2E0	RO	12			2nd maximum absolute value of ADC data in channel 3	This register shares the same format with register FDB_AGC_DAT_MAX_1ST_C0
FDB_AGC_DAT_MAX_3RD_C0	0x2E4	RO	12			3rd maximum absolute value of ADC data in channel 0	This register shares the same format with register FDB_AGC_DAT_MAX_1ST_C0
FDB_AGC_DAT_MAX_3RD_C1	0x2E8	RO	12			3rd maximum absolute value of ADC data in channel 1	This register shares the same format with register FDB_AGC_DAT_MAX_1ST_C0
FDB_AGC_DAT_MAX_3RD_C2	0x2EC	RO	12			3rd maximum absolute value of ADC data in channel 2	This register shares the same format with register FDB_AGC_DAT_MAX_1ST_C0

continues on next page



Table B.1 – continued from previous page

Name	Addr	Access	BW	Bank	DV	Description	Function Description
FDB_AGC_DAT_MAX_3RD_C3	0x2F0	RO	12			3rd maximum absolute value of ADC data in channel 3	This register shares the same format with register FDB_AGC_DAT_MAX_1ST_C0
FDB_AGC_IRQ_STATUS	0x2F4	RO	12			AGC IRQ status	high active: bit mask with 1 means asserted and 0 means de-asserted bit 11: IRQ of TIA in channel 0 bit 10: IRQ of VGA1 in channel 0 bit 9: IRQ of VGA2 in channel 0 bit 8: IRQ of TIA in channel 1 bit 7: IRQ of VGA1 in channel 1 bit 6: IRQ of VGA2 in channel 1 bit 5: IRQ of TIA in channel 2 bit 4: IRQ of VGA1 in channel 2 bit 3: IRQ of VGA2 in channel 2 bit 2: IRQ of TIA in channel 3 bit 1: IRQ of VGA1 in channel 3 bit 0: IRQ of VGA2 in channel 3
CFG_SAM_SINKER	0x400	R/W	1	N	0	selection bit for sinker of module SAM	0 -> ADC data will be sent to FFT data memory (mem_buf) first 1 -> ADC data will be directly sent to module FFT
CFG_SAM_F_0_S1	0x404	R/W	2	Y	0	shifter in stage 0	Format: FXI(2, U) 0 -> arithmetic right shift 0 bits : >>>0 ; 1 -> arithmetic right shift 1 bits : >>>1 ; 2 -> arithmetic right shift 2 bits : >>>2
CFG_SAM_F_0_B1	0x408	R/W	8	Y	0	coefficient b1 in stage 0	Format: FXR(8, 2, S)
CFG_SAM_F_0_A1	0x40C	R/W	8	Y	0	coefficient a1 in stage 0	Format: FXR(8, 2, S)
CFG_SAM_F_0_A2	0x410	R/W	8	Y	0	coefficient a2 in stage 0	Format: FXR(8, 0, U)
CFG_SAM_F_1_S1	0x414	R/W	2	Y	0	shifter in stage 1	Format: FXI(2, U) 0 -> arithmetic right shift 0 bits : >>>0 ; 1 -> arithmetic right shift 1 bits : >>>1 ; 2 -> arithmetic right shift 2 bits : >>>2
CFG_SAM_F_1_B1	0x418	R/W	8	Y	0	coefficient b1 in stage 1	Format: FXR(8, 2, S)
CFG_SAM_F_1_A1	0x41C	R/W	8	Y	0	coefficient a1 in stage 1	Format: FXR(8, 2, S)
CFG_SAM_F_1_A2	0x420	R/W	8	Y	0	coefficient a2 in stage 1	Format: FXR(8, 0, U)
CFG_SAM_F_2_S1	0x424	R/W	2	Y	0	shifter in stage 2	Format: FXI(2, U) 0 -> arithmetic right shift 0 bits : >>>0 ; 1 -> arithmetic right shift 1 bits : >>>1 ; 2 -> arithmetic right shift 2 bits : >>>2
CFG_SAM_F_2_B1	0x428	R/W	8	Y	0	coefficient b1 in stage 2	Format: FXR(8, 2, S)
CFG_SAM_F_2_A1	0x42C	R/W	8	Y	0	coefficient a1 in stage 2	Format: FXR(8, 2, S)
CFG_SAM_F_2_A2	0x430	R/W	8	Y	0	coefficient a2 in stage 2	Format: FXR(8, 0, U)

continues on next page



Table B.1 – continued from previous page

Name	Addr	Access	BW	Bank	DV	Description	Function Description
CFG_SAM_F_3_S1	0x434	R/W	2	Y	0	shifter in stage 3	Format: FXI(2, U) 0 -> arithmetic right shift 0 bits : >>>0 ; 1 -> arithmetic right shift 1 bits : >>>1 ; 2 -> arithmetic right shift 2 bits : >>>2
CFG_SAM_F_3_B1	0x438	R/W	8	Y	0	coefficient b1 in stage 3	Format: FXR(8, 2, S)
CFG_SAM_F_3_A1	0x43C	R/W	8	Y	0	coefficient a1 in stage 3	Format: FXR(8, 2, S)
CFG_SAM_F_3_A2	0x440	R/W	8	Y	0	coefficient a2 in stage 3	Format: FXR(8, 0, U)
CFG_SAM_FNL_SHF	0x444	R/W	1	Y	0	shifter in final stage	Format: FXI(1, U) 0 -> arithmetic left shift 0 bits : <<<0 ; 1 -> arithmetic left shift 1 bits : <<<1
CFG_SAM_FNL_SCL	0x448	R/W	10	Y	0	coefficient in final stage	Format: FXR(10, 2, S)
CFG_SAM_DINT_ENA	0x44C	R/W	1	Y	0	enable bit for de-interference	high active: bit value with 1 means de-interference is enabled
CFG_SAM_DINT_MOD	0x450	R/W	1	Y	0	host control or manual control	0 -> host control 1 -> manual control
CTL_SAM_DINT_SET	0x454	W/SC	1	N	0	reinitialize CFG_SAM_DINT_DAT for de-interference XOR chain	high active: bit value with 1 means reinitialization is enabled
CFG_SAM_DINT_DAT	0x458	R/W	32	Y	0	initial data for de-interference XOR chain	Format: FXI(32, U)
CFG_SAM_DINT_MSK	0x45C	R/W	32	Y	0	tapper for de-interference XOR chain	Format: FXI(32, U)
CFG_SAM_DAMB_PRD	0x460	R/W	32	Y	0	chirp length of anti velocity ambiguity	Format: FXI(16, U) x -> each chirp has x+1 points
CFG_SAM_FORCE	0x464	R/W	1	Y	0	force start bit for module SAM	high active: write 1 to start module SAM without waiting for the synchronization signal from RF
CFG_SAM_DBG_SRC	0x468	R/W	1	Y	0	selection bits for debug data in module SAM	0 -> data before down sampling 1 -> data after down sampling
CFG_SAM_SIZE_DBG_BGN	0x46C	R/W	32	Y	0	start position to dump debug data in each chirp	Format: FXI(16, U) x -> start dumping debug data from x points (the xth point is dumped)
CFG_SAM_SIZE_DBG_END	0x470	R/W	32	Y	0	stop position to dump debug data in each chirp	Format: FXI(16, U) x -> stop dumping debug data from x points (the xth point is dumped)
CFG_SAM_SPK_RM_EN	0x474	R/W	1	Y	0	enable mitigating ADC interference samples caused by interference radar	0 -> disable; 1 -> enable

continues on next page



Table B.1 – continued from previous page

Name	Addr	Access	BW	Bank	DV	Description	Function Description
CFG_SAM_SPK_CFM_SIZE	0x478	R/W	4	Y	0	buffer length for internal algorithm used to detect high frequency interference	Format: FXI(4, U) 4 -> buffer length is 4; 5 -> buffer length is 5; 6 -> buffer length is 6; 7 -> buffer length is 7; other values -> automatically set to 4 by internal logic
CFG_SAM_SPK_SET_ZERO	0x47C	R/W	1	Y	0	enable setting interference samples to zero	0 -> disable; 1 -> enable
CFG_SAM_SPK_OVER_NUM	0x480	R/W	4	Y	0	Upper 2 bits determines the least number of overflow samples in overflow region; Lower 2 bits determines the threshold for detecting overflow interference samples	x(3:2) -> Format: FXI(2, U) least number of overflow samples in overflow region is x(3:2) + 1, x(3:2) can be set as 0, 1, 2, 3 x(1:0) -> Format: FXI(2, U) overflow threshold = 8192 + x(1:0) 2048, x(1:0) can be set as 0, 1, 2, 3
CFG_SAM_SPK_THRES_DBL	0x484	R/W	1	Y	0	enable doubling the threshold value for detecting high frequency interference	0 -> disable; 1 -> enable
CFG_SAM_SPK_MIN_MAX_SEL	0x488	R/W	1	Y	0	enable choosing the smaller “maximum sample difference” estimated in a chirp	0 -> disable; 1 -> enable
FDB_SAM_DINT_DAT	0x48C	RO	32			initial data for de-interference XOR chain	Format: FXI(32, U)
CFG_FFT_SHFT RNG	0x600	R/W	11	Y	0	range level fft core mask	1D FFT mask
CFG_FFT_SHFT VEL	0x604	R/W	10	Y	0	velocity level fft core mask	2D FFT mask
CFG_FFT_DBPM_ENA	0x608	R/W	1	Y	0	enable bits for enable phase compensation in virtual array	high active: bit value with 1 means enabled
CFG_FFT_DBPM_DIR	0x60C	R/W	1	N	0	phase compensation direction	0 -> positive compensation; 1 -> negative compensation
CFG_FFT_DAMB_ENA	0x610	R/W	1	Y	0	enable bits for enable anti velocity ambiguity	high active: bit value with 1 means enabled
CFG_FFT_DINT_ENA	0x614	R/W	3	Y	0	enable bits for corresponding feature	high active: bit value with 1 means enabled bit 2 -> phase de-scramble bit 1 -> chirp shifting bit 0 -> frequency hopping
CFG_FFT_DINT_MOD	0x618	R/W	3	Y	0	host control or manual control for corresponding feature	bit value 0 -> host control; 1 -> manual control bit 2 -> phase de-scramble bit 1 -> chirp shifting bit 0 -> frequency hopping

continues on next page



Table B.1 – continued from previous page

Name	Addr	Access	BW	Bank	DV	Description	Function Description
CTL_FFT_DINT_SET	0x61C	R/W	3	N	0	initialize CFG_FFT_DINT_DAT_** for corresponding feature	high active: bit value 1 means initialization enabled bit 2 → phase de-scramble bit 1 → chirp shifting bit 0 → frequency hopping
CFG_FFT_DINT_DAT_FH	0x620	R/W	32	Y	0	initial data for frequency hopping	Format: FXI(32, U)
CFG_FFT_DINT_DAT_CS	0x624	R/W	32	Y	0	initial data for chirp shifting	Format: FXI(32, U)
CFG_FFT_DINT_DAT_PS	0x628	R/W	32	Y	0	initial data for de-scramble XOR chain	Format: FXI(32, U)
CFG_FFT_DINT_MSK_FH	0x62C	R/W	32	Y	0	tapper for frequency hopping	Format: FXI(32, U)
CFG_FFT_DINT_MSK_CS	0x630	R/W	32	Y	0	tapper for chirp shifting	Format: FXI(32, U)
CFG_FFT_DINT_MSK_PS	0x634	R/W	32	Y	0	tapper for de-scramble XOR chain	Format: FXI(32, U)
CFG_FFT_DINT_COE_FH	0x638	R/W	28	Y	0	phase compensation coefficient for frequency hopping	Format: FXI(14, 1, S)
CFG_FFT_DINT_COE_CS	0x63C	R/W	28	Y	0	initial data for chirp shifting	Format: FXI(14, 1, S)
CFG_FFT_DINT_COE_FC	0x640	R/W	28	Y	0	phase compensation coefficient when frequency hopping and chirp shifting both enabled	Format: FXI(14, 1, S)
CFG_FFT_DINT_COE_PS_0	0x644	R/W	28	Y	0	phase de-scramble coefficient of virtual array 0	Format: FXI(14, 1, S)
CFG_FFT_DINT_COE_PS_1	0x648	R/W	28	Y	0	phase de-scramble coefficient of virtual array 1	Format: FXI(14, 1, S)
CFG_FFT_DINT_COE_PS_2	0x64C	R/W	28	Y	0	phase de-scramble coefficient of virtual array 2	Format: FXI(14, 1, S)
CFG_FFT_DINT_COE_PS_3	0x650	R/W	28	Y	0	phase de-scramble coefficient of virtual array 3	Format: FXI(14, 1, S)
CFG_FFT_NO_WIN	0x654	R/W	1	Y	0	enable bits for windowing	low active: bit value with 1 means windowing is disabled and 0 means enabled
CFG_FFT_NVE_MODE	0x658	R/W	1	Y	0	NVE mode register	1:bypass NVE, fill the default nve value to memory; 0:enable NVE, fill the real nve value to memory
CFG_FFT_NVE_SCL_0	0x65C	R/W	9	Y	0	Zero doppler Scale parameter	velocity level average scale parameter bit 2:0, exp bit 3:8, mattisa
CFG_FFT_NVE_SCL_1	0x660	R/W	9	Y	0	NVE Scale parameter	NVE parameter bit 2:0, exp bit 3:8, mattisa
CFG_FFT_NVE_SFT	0x664	R/W	3	Y	0	NVE shift parameter	NVE shift value

continues on next page



Table B.1 – continued from previous page

Name	Addr	Access	BW	Bank	DV	Description	Function Description
CFG_FFT_NVE_CH_MSK	0x668	R/W	16	Y	0	NVE channel mask	NVE ant and bpm mask 0:ant0 bpm0 1:ant1 bpm0 2:ant2 bpm0 3:ant3 bpm0 4:ant0 bpm1 and so on
CFG_FFT_NVE_DFT_VALUE	0x66C	R/W	20	Y	0	NVE default value	NVE default value nve mode bypass bits[4:0] exp bits[19:5] mattisa
CFG_FFT_ZER_DPL_ENB	0x670	R/W	1	Y	0	Zero doppler enable register	1:zero doppler enable 0:zero doppler disable
FDB_FFT_DINT_DAT_FH	0x674	RO	32			feedback for frequency hopping	Format: FXI(32, U)
FDB_FFT_DINT_DAT_CS	0x678	RO	32			feedback for chirp shifting	Format: FXI(32, U)
FDB_FFT_DINT_DAT_PS	0x67C	RO	32			feedback for phase scramble	Format: FXI(32, U)
CFG_CFR_SIZE_OBJ	0x800	R/W	10	Y	0	Max obj size register, 0base	Maximum CFAR detect target number threshold,0 to 1023
CFG_CFR_BACK RNG	0x804	R/W	11	Y	0	bc range boundary	CFAR detect target distance boundary
CFG_CFR_TYPE_CMB	0x808	R/W	3	Y	0	CFAR combine type register	0: SISO 1: MIMO 2: cascade 3: acc all 4: acc on fix 5: acc one flt
CFG_CFR_MIMO_NUM	0x80C	R/W	4	Y	0	MIMO number register	maximum support 16 mimo number
CFG_CFR_MIMO_ADR	0x810	R/W	12	Y	0	MIMO coe start address	start point of fetch coe memory address
CFG_CFR_PRT_VEL_00	0x814	R/W	10	Y	0	Region dependent boundary value, Vel0	Velocity boundary 00 set <= CFG_CFR_PRT_VEL_01
CFG_CFR_PRT_VEL_01	0x818	R/W	10	Y	0	Region dependent boundary value, Vel1	Velocity boundary 01 can be set 0 to 1023
CFG_CFR_PRT_VEL_10	0x81C	R/W	10	Y	0	Region dependent boundary value, Vel2	Velocity boundary 10 set <= CFG_CFR_PRT_VEL_11
CFG_CFR_PRT_VEL_11	0x820	R/W	10	Y	0	Region dependent boundary value, Vel3	Velocity boundary 11 can be set 0 to 1023
CFG_CFR_PRT_VEL_20	0x824	R/W	10	Y	0	Region dependent boundary value, Vel4	Velocity boundary 20 set <= CFG_CFR_PRT_VEL_21
CFG_CFR_PRT_VEL_21	0x828	R/W	10	Y	0	Region dependent boundary value, Vel5	Velocity boundary 21 can be set 0 to 1023
CFG_CFR_PRT_VEL_30	0x82C	R/W	10	Y	0	Region dependent boundary value, Vel6	Velocity boundary 30 set <= CFG_CFR_PRT_VEL_31
CFG_CFR_PRT_VEL_31	0x830	R/W	10	Y	0	Region dependent boundary value, Vel7	Velocity boundary 31 can be set 0 to 1023
CFG_CFR_PRT RNG_00	0x834	R/W	10	Y	0	Region dependent boundary value, Ran0	Range boundary 00 set <= CFG_CFR_PRT RNG_01
CFG_CFR_PRT RNG_01	0x838	R/W	10	Y	0	Region dependent boundary value, Ran1	Range boundary 01 set <= CFG_CFR_PRT RNG_02

continues on next page



Table B.1 – continued from previous page

Name	Addr	Access	BW	Bank	DV	Description	Function Description
CFG_CFR_PRT_RNG_02	0x83C	R/W	10	Y	0	Region dependent boundary value, Ran2	Range boundary 02 can be set 0 to 1023
CFG_CFR_TYP_AL	0x840	R/W	16	Y	0	8 Regions CFAR algorithm scheme	CFAR algorithm definition coded in two bits : 00b: CA CFAR; 01b: OS CFAR; 10b: SOGO CFAR; 11b: NR CFAR bits[1:0] Region 0 CFAR algorithm bits[3:2] Region 1 CFAR algorithm bits[5:4] Region 2 CFAR algorithm bits[7:6] Region 3 CFAR algorithm bits[9:8] Region 4 CFAR algorithm bits[11:10] Region 5 CFAR algorithm bits[13:12] Region 6 CFAR algorithm bits[15:14] Region 7 CFAR algorithm
CFG_CFR_TYP_DS	0x844	R/W	16	Y	0	8 Rectangular win decimation scheme	Decimation scheme coded in two bits. 00: no decimation; 10: decimating along the Doppler gate; 01: decimating along the range gate; 11: decimating along both bits[15:14] Region 7 bits[13:12] Region 6 bits[11:10] Region 5 bits[9:8] Region 4 bits[7:6] Region 3 bits[5:4] Region 2 bits[3:2] Region 1 bits[1:0] Region 0
CFG_CFR_PRM_0_0	0x848	R/W	28	Y	0	Parameter 0_0 register, value shared in different CFAR algorithm	28 bits covering 4 regions. Each region has 7 bits for different CFAR algorithm definition bits[27:21] Region 7 bits[20:14] Region 6 bits[13:7] Region 5 bits[6:0] Region 4
CFG_CFR_PRM_0_1	0x84C	R/W	28	Y	0	Parameter 0_1 register, value shared in different CFAR algorithm	28 bits covering 4 regions. Each region has 7 bits for different CFAR algorithm definition bits[27:21] Region 3 bits[20:14] Region 2 bits[13:7] Region 1 bits[6:0] Region 0
CFG_CFR_PRM_1_0	0x850	R/W	30	Y	0	Parameter 1_0 register, value shared in different CFAR algorithm	30 bits covering 3 regions. Each region has 10 bits for different CFAR algorithm definition bits[29:20] Region 7 bits[19:10] Region 6 bits[9:0] Region 5
CFG_CFR_PRM_1_1	0x854	R/W	30	Y	0	Parameter 1_1 register, value shared in different CFAR algorithm	30 bits covering 3 regions. Each region has 10 bits for different CFAR algorithm definition bits[29:20] Region 4 bits[19:10] Region 3 bits[9:0] Region 2

continues on next page



Table B.1 – continued from previous page

Name	Addr	Access	BW	Bank	DV	Description	Function Description
CFG_CFR_PRM_1_2	0x858	R/W	20	Y	0	Parameter 1_2 register, value shared in different CFAR algorithm	20 bits covering 2 regions. Each region has 10 bits for different CFAR algorithm definition bits[19:10] Region 1 bits[9:0] Region 0
CFG_CFR_PRM_2_0	0x85C	R/W	30	Y	0	Parameter 2_0 register, value shared in different CFAR algorithm	30 bits covering 3 regions. Each region has 10 bits for different CFAR algorithm definition bits[29:20] Region 7 bits[19:10] Region 6 bits[9:0] Region 5
CFG_CFR_PRM_2_1	0x860	R/W	30	Y	0	Parameter 2_1 register, value shared in different CFAR algorithm	30 bits covering 3 regions. Each region has 10 bits for different CFAR algorithm definition bits[29:20] Region 4 bits[19:10] Region 3 bits[9:0] Region 2
CFG_CFR_PRM_2_2	0x864	R/W	20	Y	0	Parameter 2_2 register, value shared in different CFAR algorithm	20 bits covering 2 regions. Each region has 10 bits for different CFAR algorithm definition bits[19:10] Region 1 bits[9:0] Region 0
CFG_CFR_PRM_3_0	0x868	R/W	24	Y	0	Parameter 3_0 register, value shared in different CFAR algorithm	24 bits covering 2 regions. Each region has 12 bits for different CFAR algorithm definition bits[23:12] Region 7 bits[11:0] Region 6
CFG_CFR_PRM_3_1	0x86C	R/W	24	Y	0	Parameter 3_1 register, value shared in different CFAR algorithm	24 bits covering 2 regions. Each region has 12 bits for different CFAR algorithm definition bits[23:12] Region 5 bits[11:0] Region 4
CFG_CFR_PRM_3_2	0x870	R/W	24	Y	0	Parameter 3_2 register, value shared in different CFAR algorithm	24 bits covering 2 regions. Each region has 12 bits for different CFAR algorithm definition bits[23:12] Region 3 bits[11:0] Region 2
CFG_CFR_PRM_3_3	0x874	R/W	24	Y	0	Parameter 3_3 register, value shared in different CFAR algorithm	24 bits covering 2 regions. Each region has 12 bits for different CFAR algorithm definition bits[23:12] Region 1 bits[11:0] Region 0
CFG_CFR_PRM_4_0	0x878	R/W	28	Y	0	Parameter 4_0 register, value shared in different CFAR algorithm	28 bits covering 4 regions. Each region has 7 bits for different CFAR algorithm definition bits[27:21] Region 7 bits[20:14] Region 6 bits[13: 7] Region 5 bits[6:0] Region 4

continues on next page



Table B.1 – continued from previous page

Name	Addr	Access	BW	Bank	DV	Description	Function Description
CFG_CFR_PRM_4_1	0x87C	R/W	28	Y	0	Parameter 4_1 register, value shared in different CFAR algorithm	28 bits covering 4 regions. Each region has 7 bits for different CFAR algorithm definition bits[27:21] Region 3 bits[20:14] Region 2 bits[13: 7] Region 1 bits[6:0] Region 0
CFG_CFR_PRM_5_0	0x880	R/W	28	Y	0	Parameter 5_0 register, value shared in different CFAR algorithm	28 bits covering 4 regions. Each region has 7 bits for different CFAR algorithm definition bits[27:21] Region 7 bits[20:14] Region 6 bits[13: 7] Region 5 bits[6:0] Region 4
CFG_CFR_PRM_5_1	0x884	R/W	28	Y	0	Parameter 5_1 register, value shared in different CFAR algorithm	28 bits covering 4 regions. Each region has 7 bits for different CFAR algorithm definition bits[27:21] Region 3 bits[20:14] Region 2 bits[13: 7] Region 1 bits[6:0] Region 0
CFG_CFR_PRM_6_0	0x888	R/W	24	Y	0	Parameter 6_0 register, value shared in different CFAR algorithm	24 bits covering 2 regions. Each region has 12 bits for different CFAR algorithm definition bits[23:12] Region 7 bits[11:0] Region 6
CFG_CFR_PRM_6_1	0x88C	R/W	24	Y	0	Parameter 6_1 register, value shared in different CFAR algorithm	24 bits covering 2 regions. Each region has 12 bits for different CFAR algorithm definition bits[23:12] Region 5 bits[11:0] Region 4
CFG_CFR_PRM_6_2	0x890	R/W	24	Y	0	Parameter 6_2 register, value shared in different CFAR algorithm	24 bits covering 2 regions. Each region has 12 bits for different CFAR algorithm definition bits[23:12] Region 3 bits[11:0] Region 2
CFG_CFR_PRM_6_3	0x894	R/W	24	Y	0	Parameter 6_3 register, value shared in different CFAR algorithm	24 bits covering 2 regions. Each region has 12 bits for different CFAR algorithm definition bits[23:12] Region 1 bits[11:0] Region 0
CFG_CFR_PRM_7_0	0x898	R/W	8	Y	0	Parameter 7_0 register, value shared in different CFAR algorithm	8 bits covering 8 regions. Each region has 1 bit for different CFAR algorithm definition bit x corresponds to region x.

continues on next page



Table B.1 – continued from previous page

Name	Addr	Access	BW	Bank	DV	Description	Function Description
CFG_CFR_MSK_DS_00	0x89C	R/W	32	Y	0	CFAR window mask register0	Bit mask with 1 means the corresponding data is used in CFAR and 0 means not used which is for region 7; bit x: xth data in this line bits[31:21] Doppler gate 0, range gate 0 to 10 mask bits[20:10] Doppler gate 1, range gate 0 to 10 mask bits[09:00] Doppler gate 2, range gate 0 to 9 mask
CFG_CFR_MSK_DS_01	0x8A0	R/W	32	Y	0	CFAR window mask register1	Bit mask with 1 means the corresponding data is used in CFAR and 0 means not used which is for region 7; bit x: xth data in this line bits[31:31] Doppler gate 2, range gate 10 to 10 mask bits[30:20] Doppler gate 3, range gate 0 to 10 mask bits[19:09] Doppler gate 4, range gate 0 to 10 mask bits[08:00] Doppler gate 5, range gate 0 to 8 mask
CFG_CFR_MSK_DS_02	0x8A4	R/W	32	Y	0	CFAR window mask register2	Bit mask with 1 means the corresponding data is used in CFAR and 0 means not used which is for region 7; bit x: xth data in this line bits[31:30] Doppler gate 5, range gate 9 to 10 mask bits[29:19] Doppler gate 6, range gate 0 to 10 mask bits[18:08] Doppler gate 7, range gate 0 to 10 mask bits[07:00] Doppler gate 8, range gate 0 to 7 mask
CFG_CFR_MSK_DS_03	0x8A8	R/W	25	Y	0	CFAR window mask register3	Bit mask with 1 means the corresponding data is used in CFAR and 0 means not used which is for region 7; bit x: xth data in this line bits[24:22] Doppler gate 8, range gate 8 to 10 mask bits[21:11] Doppler gate 9, range gate 0 to 10 mask bits[10:00] Doppler gate 10, range gate 0 to 10 mask
CFG_CFR_MSK_DS_10	0x8AC	R/W	32	Y	0	CFAR window mask register4	Bit mask with 1 means the corresponding data is used in CFAR and 0 means not used which is for region 6; bit x: xth data in this line Share the same format with CFG_CFR_MSK_DS_00
CFG_CFR_MSK_DS_11	0x8B0	R/W	32	Y	0	CFAR window mask register5	Bit mask with 1 means the corresponding data is used in CFAR and 0 means not used which is for region 6; bit x: xth data in this line Share the same format with CFG_CFR_MSK_DS_01

continues on next page



Table B.1 – continued from previous page

Name	Addr	Access	BW	Bank	DV	Description	Function Description
CFG_CFR_MSK_DS_12	0x8B4	R/W	32	Y	0	CFAR window mask register6	Bit mask with 1 means the corresponding data is used in CFAR and 0 means not used which is for region 6; bit x: xth data in this line Share the same format with CFG_CFR_MSK_DS_02
CFG_CFR_MSK_DS_13	0x8B8	R/W	25	Y	0	CFAR window mask register7	Bit mask with 1 means the corresponding data is used in CFAR and 0 means not used which is for region 6; bit x: xth data in this line Share the same format with CFG_CFR_MSK_DS_03
CFG_CFR_MSK_DS_20	0x8BC	R/W	32	Y	0	CFAR window mask register8	Bit mask with 1 means the corresponding data is used in CFAR and 0 means not used which is for region 5; bit x: xth data in this line Share the same format with CFG_CFR_MSK_DS_00
CFG_CFR_MSK_DS_21	0x8C0	R/W	32	Y	0	CFAR window mask register9	Bit mask with 1 means the corresponding data is used in CFAR and 0 means not used which is for region 5; bit x: xth data in this line Share the same format with CFG_CFR_MSK_DS_01
CFG_CFR_MSK_DS_22	0x8C4	R/W	32	Y	0	CFAR window mask register10	Bit mask with 1 means the corresponding data is used in CFAR and 0 means not used which is for region 5; bit x: xth data in this line Share the same format with CFG_CFR_MSK_DS_02
CFG_CFR_MSK_DS_23	0x8C8	R/W	25	Y	0	CFAR window mask register11	Bit mask with 1 means the corresponding data is used in CFAR and 0 means not used which is for region 5; bit x: xth data in this line Share the same format with CFG_CFR_MSK_DS_03
CFG_CFR_MSK_DS_30	0x8CC	R/W	32	Y	0	CFAR window mask register12	Bit mask with 1 means the corresponding data is used in CFAR and 0 means not used which is for region 4; bit x: xth data in this line Share the same format with CFG_CFR_MSK_DS_00
CFG_CFR_MSK_DS_31	0x8D0	R/W	32	Y	0	CFAR window mask register13	Bit mask with 1 means the corresponding data is used in CFAR and 0 means not used which is for region 4; bit x: xth data in this line Share the same format with CFG_CFR_MSK_DS_01
CFG_CFR_MSK_DS_32	0x8D4	R/W	32	Y	0	CFAR window mask register14	Bit mask with 1 means the corresponding data is used in CFAR and 0 means not used which is for region 4; bit x: xth data in this line Share the same format with CFG_CFR_MSK_DS_02

continues on next page



Table B.1 – continued from previous page

Name	Addr	Access	BW	Bank	DV	Description	Function Description
CFG_CFR_MSK_DS_33	0x8D8	R/W	25	Y	0	CFAR window mask register15	Bit mask with 1 means the corresponding data is used in CFAR and 0 means not used which is for region 4; bit x: xth data in this line Share the same format with CFG_CFR_MSK_DS_03
CFG_CFR_MSK_DS_40	0x8DC	R/W	32	Y	0	CFAR window mask register16	Bit mask with 1 means the corresponding data is used in CFAR and 0 means not used which is for region 3; bit x: xth data in this line Share the same format with CFG_CFR_MSK_DS_00
CFG_CFR_MSK_DS_41	0x8E0	R/W	32	Y	0	CFAR window mask register17	Bit mask with 1 means the corresponding data is used in CFAR and 0 means not used which is for region 3; bit x: xth data in this line Share the same format with CFG_CFR_MSK_DS_01
CFG_CFR_MSK_DS_42	0x8E4	R/W	32	Y	0	CFAR window mask register18	Bit mask with 1 means the corresponding data is used in CFAR and 0 means not used which is for region 3; bit x: xth data in this line Share the same format with CFG_CFR_MSK_DS_02
CFG_CFR_MSK_DS_43	0x8E8	R/W	25	Y	0	CFAR window mask register19	Bit mask with 1 means the corresponding data is used in CFAR and 0 means not used which is for region 3; bit x: xth data in this line Share the same format with CFG_CFR_MSK_DS_03
CFG_CFR_MSK_DS_50	0x8EC	R/W	32	Y	0	CFAR window mask register20	Bit mask with 1 means the corresponding data is used in CFAR and 0 means not used which is for region 2; bit x: xth data in this line Share the same format with CFG_CFR_MSK_DS_00
CFG_CFR_MSK_DS_51	0x8F0	R/W	32	Y	0	CFAR window mask register21	Bit mask with 1 means the corresponding data is used in CFAR and 0 means not used which is for region 2; bit x: xth data in this line Share the same format with CFG_CFR_MSK_DS_01
CFG_CFR_MSK_DS_52	0x8F4	R/W	32	Y	0	CFAR window mask register22	Bit mask with 1 means the corresponding data is used in CFAR and 0 means not used which is for region 2; bit x: xth data in this line Share the same format with CFG_CFR_MSK_DS_02
CFG_CFR_MSK_DS_53	0x8F8	R/W	25	Y	0	CFAR window mask register23	Bit mask with 1 means the corresponding data is used in CFAR and 0 means not used which is for region 2; bit x: xth data in this line Share the same format with CFG_CFR_MSK_DS_03

continues on next page



Table B.1 – continued from previous page

Name	Addr	Access	BW	Bank	DV	Description	Function Description
CFG_CFR_MSK_DS_60	0x8FC	R/W	32	Y	0	CFAR window mask register24	Bit mask with 1 means the corresponding data is used in CFAR and 0 means not used which is for region 1; bit x: xth data in this line Share the same format with CFG_CFR_MSK_DS_00
CFG_CFR_MSK_DS_61	0x900	R/W	32	Y	0	CFAR window mask register25	Bit mask with 1 means the corresponding data is used in CFAR and 0 means not used which is for region 1; bit x: xth data in this line Share the same format with CFG_CFR_MSK_DS_01
CFG_CFR_MSK_DS_62	0x904	R/W	32	Y	0	CFAR window mask register26	Bit mask with 1 means the corresponding data is used in CFAR and 0 means not used which is for region 1; bit x: xth data in this line Share the same format with CFG_CFR_MSK_DS_02
CFG_CFR_MSK_DS_63	0x908	R/W	25	Y	0	CFAR window mask register27	Bit mask with 1 means the corresponding data is used in CFAR and 0 means not used which is for region 1; bit x: xth data in this line Share the same format with CFG_CFR_MSK_DS_03
CFG_CFR_MSK_DS_70	0x90C	R/W	32	Y	0	CFAR window mask register28	Bit mask with 1 means the corresponding data is used in CFAR and 0 means not used which is for region 0; bit x: xth data in this line Share the same format with CFG_CFR_MSK_DS_00
CFG_CFR_MSK_DS_71	0x910	R/W	32	Y	0	CFAR window mask register29	Bit mask with 1 means the corresponding data is used in CFAR and 0 means not used which is for region 0; bit x: xth data in this line Share the same format with CFG_CFR_MSK_DS_01
CFG_CFR_MSK_DS_72	0x914	R/W	32	Y	0	CFAR window mask register30	Bit mask with 1 means the corresponding data is used in CFAR and 0 means not used which is for region 0; bit x: xth data in this line Share the same format with CFG_CFR_MSK_DS_02
CFG_CFR_MSK_DS_73	0x918	R/W	25	Y	0	CFAR window mask register31	Bit mask with 1 means the corresponding data is used in CFAR and 0 means not used which is for region 0; bit x: xth data in this line Share the same format with CFG_CFR_MSK_DS_03

continues on next page



Table B.1 – continued from previous page

Name	Addr	Access	BW	Bank	DV	Description	Function Description
CFG_CFR_MSK_PK_00	0x91C	R/W	25	Y	0	CFAR peak mask register0	Bit mask with 1 means the corresponding data is used in peak and 0 means not used which is for region 0. bit x: xth data in this line bits[24:20]: range gate 0 Doppler gate 0 to 4 mask bits[19:15]: range gate 1 Doppler gate 0 to 4 mask bits[14:10]: range gate 2 Doppler gate 0 to 4 mask bits[9:5]: range gate 3 Doppler gate 0 to 4 mask bits[4:0]: range gate 4 Doppler gate 0 to 4 mask
CFG_CFR_MSK_PK_01	0x920	R/W	25	Y	0	CFAR peak mask register1	Bit mask with 1 means the corresponding data is used in peak and 0 means not used which is for region 1. bit x: xth data in this line Share same format with CFG_CFR_MSK_PK_00
CFG_CFR_MSK_PK_02	0x924	R/W	25	Y	0	CFAR peak mask register2	Bit mask with 1 means the corresponding data is used in peak and 0 means not used which is for region 2. bit x: xth data in this line Share same format with CFG_CFR_MSK_PK_00
CFG_CFR_MSK_PK_03	0x928	R/W	25	Y	0	CFAR peak mask register3	Bit mask with 1 means the corresponding data is used in peak and 0 means not used which is for region 3. bit x: xth data in this line Share same format with CFG_CFR_MSK_PK_00
CFG_CFR_MSK_PK_04	0x92C	R/W	25	Y	0	CFAR peak mask register4	Bit mask with 1 means the corresponding data is used in peak and 0 means not used which is for region 4. bit x: xth data in this line Share same format with CFG_CFR_MSK_PK_00
CFG_CFR_MSK_PK_05	0x930	R/W	25	Y	0	CFAR peak mask register5	Bit mask with 1 means the corresponding data is used in peak and 0 means not used which is for region 5. bit x: xth data in this line Share same format with CFG_CFR_MSK_PK_00
CFG_CFR_MSK_PK_06	0x934	R/W	25	Y	0	CFAR peak mask register6	Bit mask with 1 means the corresponding data is used in peak and 0 means not used which is for region 6. bit x: xth data in this line Share same format with CFG_CFR_MSK_PK_00

continues on next page



Table B.1 – continued from previous page

Name	Addr	Access	BW	Bank	DV	Description	Function Description
CFG_CFR_MSK_PK_07	0x938	R/W	25	Y	0	CFAR peak mask register7	Bit mask with 1 means the corresponding data is used in peak and 0 means not used which is for region 7. bit x: xth data in this line Share same format with CFG_CFR_MSK_PK_00
CFG_CFR_PK_ENB	0x93C	R/W	8	Y	0	8 region pk function enable	Peak enable signal: 1:enable, 0:disable bit0: Region 0 bit1: Region 1 bit2: Region 2 bit3: Region 3 bit4: Region 4 bit5: Region 5 bit6: Region 6 bit7: Region 7
CFG_CFR_PK_THR_0	0x940	R/W	20	Y	0	region0 pk threshold	Low 5bit active: region 0 peak threshold
CFG_CFR_PK_THR_1	0x944	R/W	20	Y	0	region1 pk threshold	Low 5bit active: region 1 peak threshold
CFG_CFR_PK_THR_2	0x948	R/W	20	Y	0	region2 pk threshold	Low 5bit active: region 2 peak threshold
CFG_CFR_PK_THR_3	0x94C	R/W	20	Y	0	region3 pk threshold	Low 5bit active: region 3 peak threshold
CFG_CFR_PK_THR_4	0x950	R/W	20	Y	0	region4 pk threshold	Low 5bit active: region 4 peak threshold
CFG_CFR_PK_THR_5	0x954	R/W	20	Y	0	region5 pk threshold	Low 5bit active: region 5 peak threshold
CFG_CFR_PK_THR_6	0x958	R/W	20	Y	0	region6 pk threshold	Low 5bit active: region 6 peak threshold
CFG_CFR_PK_THR_7	0x95C	R/W	20	Y	0	region7 pk threshold	Low 5bit active: region 7 peak threshold
CFG_CFR_CS_ENA	0x960	R/W	1	Y	0	Cross window enable	window mode enable register 0:rectangular window 1:cross window
CFG_CFR_CS_SKP_VEL	0x964	R/W	4	Y	0	velocity level skip size register	cross window guard cell size of Doppler gate support 0 to 15
CFG_CFR_CS_SKP_RNG	0x968	R/W	4	Y	0	range level skip size register	cross window guard cell size of range-gate support 0 to 15
CFG_CFR_CS_SIZ_VEL	0x96C	R/W	4	Y	0	velocity level cross size register	cross window Doppler gate length support 1 to 9
CFG_CFR_CS_SIZ_RNG	0x970	R/W	4	Y	0	range level cross size register	cross window range-gate length support 1 to 9

continues on next page



Table B.1 – continued from previous page

Name	Addr	Access	BW	Bank	DV	Description	Function Description
CFG_CFR_TYP_NOI	0x974	R/W	8	Y	0	8 region noise level output select register	CFAR output noise mode:1:local noise;0:outside noise bit0: Region0 bit1: Region1 bit2: Region2 bit3: Region3 bit4: Region4 bit5: Region5 bit6: Region6 bit7: Region7
FDB_CFR_NUMB_OBJ	0x978	RO	11			CFAR detect target number	CFAR detect target number,I base, maximum support 1024 targets
CFG_DoA_DAMB_FRDTRA	0xA00	R/W	15	Y	0	VELAMB phase offset factor register (used to compensate for the phase difference when detecting ambiguity factor)	Format: FXR(15, -2, U)
CFG_DoA_DAMB_IDX_BGN	0xA04	R/W	5	Y	0	the smallest Q value to be tested (for anti velocity ambiguity)	Format: FXI(5, S) x -> value is x
CFG_DoA_DAMB_IDX_LEN	0xA08	R/W	5	Y	0	total number of the Q value to be tested (for anti velocity ambiguity)	Format: FXI(5, U) x -> number is x
CFG_DoA_DAMB_TYP	0xA0C	R/W	2	Y	0	behaviour register for the de-AMB compensation (for anti velocity ambiguity)	0 -> bypassed 1 -> compensate with hardware calculated Q value 2 -> compensate with firmware configured Q value 3 -> compensate with default Q value
CFG_DoA_DAMB_DEFQ	0xA10	R/W	5	Y	0	the default Q value (for anti velocity ambiguity)	Format: FXI(5, S) x -> value is x
CFG_DoA_DAMB_FDTR	0xA14	R/W	16	Y	0	virtual array phase compensation factor register	Format: FXR(16, 4, U)
CFG_DoA_DAMB_VELCOM	0xA18	R/W	16	Y	0	virtual array more refined phase compensation factor register	Format: FXR(16, 1, U)
CFG_DoA_DBPM_ENA	0xA1C	R/W	1	Y	0	enable bit for the DBPM compensation	Format: FXI(12, 1) 0 -> disabled 1 -> enabled
CFG_DoA_DBPM_ADR	0xA20	R/W	12	Y	0	offset address for the DBPM coefficients	Format: FXI(12, U) x -> coefficients are stored from base_address_of_mem_coe + x * 16 (in bytes and in terms of APB access)

continues on next page



Table B.1 – continued from previous page

Name	Addr	Access	BW	Bank	DV	Description	Function Description
CFG_DoA_MODE_RUN	0xA24	R/W	2	Y	0	selection bits for the run mode	Format: FXI(2, U) 0 -> normal mode 1 -> cascade mode 2 -> manual mode 3 -> reserved mode
CFG_DoA_NUMB_OBJ	0xA28	R/W	11	Y	0	total number of the merged objects under cascade mode	Format: FXI(11, U) x -> number is x
CFG_DoA_MODE_GRP	0xA2C	R/W	2	Y	0	selection bits for the group mode	Format: FXI(2, U) 0 -> normal mode 1 -> 2-d combined mode 2 -> one-hot mode
CFG_DoA_NUMB_GRP	0xA30	R/W	2	Y	0	total number of the enabled groups	Format: FXI(2, U) if group mode is set to normal mode or 2-d combined mode x -> number is x + 1 if group mode is set to one-hot mode x -> group x is enabled
CFG_DoA_GRP0_TYP_SCH	0xA34	R/W	2	Y	0	selection bits for the search method in group 0	Format: FXI(2, U) 0 -> DBF 1 -> reserved type 2 -> DML
CFG_DoA_GRP0_MOD_SCH	0xA38	R/W	3	Y	0	selection bits for the search mode in group 0 (active only when search mode is set to DBF)	Format: FXI(2, U) 0 -> test the raw peaks which pass the power threshold 1 -> test all of the raw peaks directly 2 -> search and test peaks iteratively until reaching the number of raw peaks or noise threshold 3 -> search and test peaks iteratively until reaching the noise threshold
CFG_DoA_GRP0_NUM_SCH	0xA3C	R/W	2	Y	0	maximum number of the bins to be searched in group 0 (active only when search mode is set to DBF)	Format: FXI(2, U) x -> x + 1 bins are searched
CFG_DoA_GRP0_ADR_COE	0xA40	R/W	12	Y	0	offset address for the DBF/DML coefficients in group 0	Format: FXI(12, U) x -> coefficients are stored from base_address_of_mem_coe + x * 16 (in bytes and in terms of APB access)
CFG_DoA_GRP0_SIZ_ONE_ANG	0xA44	R/W	1	Y	0	reserved	reserved

continues on next page



Table B.1 – continued from previous page

Name	Addr	Access	BW	Bank	DV	Description	Function Description
CFG_DoA_GRP0_SIZ_RNG_PKS_CRS	0xA48	R/W	10	Y	0	range of the coarse search in group 0	Format: FXI(10, U) x -> x + 1 angles are tested (if the search type is set to DBF, x must be less than or equal to 511)
CFG_DoA_GRP0_SIZ_STP_PKS_CRS	0xA4C	R/W	4	Y	0	step of the coarse search in group 0	Format: FXI(4, U) x -> step size is x + 1 angles
CFG_DoA_GRP0_SIZ_RNG_PKS_RFD	0xA50	R/W	6	Y	0	range of the refined search in group 0	Format: FXI(6, U) x -> search all the angles in [-x, x-1]
CFG_DoA_GRP0_SIZ_CMB	0xA54	R/W	5	Y	0	total number of antennas to be used in group 0	Format: FXI(5, U) x -> number is x + 1
CFG_DoA_GRP0_DAT_IDX_0	0xA58	R/W	20	Y	0	antenna index 0~3 to be combined in round 0 in group 0	bit 19-15: antenna index for the 0th antenna to be combined bit 14-10: antenna index for the 1st antenna to be combined bit 09-05: antenna index for the 2nd antenna to be combined bit 04-00: antenna index for the 3rd antenna to be combined antennas are denoted in the following way: (1) for antennas of chip in non-cascade mode or master chip in cascade mode: 00: m-tx0-rx0, 01: m-tx0-rx1, 02: m-tx0-rx2, 03: m-tx0-rx3, 04: m-tx1-rx0, 05: m-tx1-rx1, 06: m-tx1-rx2, 07: m-tx1-rx3, 08: m-tx2-rx0, 09: m-tx2-rx1, 10: m-tx2-rx2, 11: m-tx2-rx3, 12: m-tx3-rx0, 13: m-tx3-rx1, 14: m-tx3-rx2, 15: m-tx3-rx3, (2) for antennas of slave chip in cascade mode: 16: s-tx0-rx0, 17: s-tx0-rx1, 18: s-tx0-rx2, 19: s-tx0-rx3; 20: s-tx1-rx0, 21: s-tx1-rx1, 22: s-tx1-rx2, 23: s-tx1-rx3; 24: s-tx2-rx0, 25: s-tx2-rx1, 26: s-tx2-rx2, 27: s-tx2-rx3; 28: s-tx3-rx0, 29: s-tx3-rx1, 30: s-tx3-rx2, 31: s-tx3-rx3;
CFG_DoA_GRP0_DAT_IDX_1	0xA5C	R/W	20	Y	0	antenna index 0~3 to be combined in round 1 in group 0	This register shares the same format with register CFG_DoA_GRP0_DAT_IDX_0
CFG_DoA_GRP0_DAT_IDX_2	0xA60	R/W	20	Y	0	antenna index 0~3 to be combined in round 2 in group 0	This register shares the same format with register CFG_DoA_GRP0_DAT_IDX_0
CFG_DoA_GRP0_DAT_IDX_3	0xA64	R/W	20	Y	0	antenna index 0~3 to be combined in round 3 in group 0	This register shares the same format with register CFG_DoA_GRP0_DAT_IDX_0
CFG_DoA_GRP0_DAT_IDX_4	0xA68	R/W	20	Y	0	antenna index 0~3 to be combined in round 4 in group 0	This register shares the same format with register CFG_DoA_GRP0_DAT_IDX_0

continues on next page



Table B.1 – continued from previous page

Name	Addr	Access	BW	Bank	DV	Description	Function Description
CFG_DoA_GRP0_DAT_IDX_5	0xA6C	R/W	20	Y	0	antenna index 0~3 to be combined in round 5 in group 0	This register shares the same format with register CFG_DoA_GRP0_DAT_IDX_0
CFG_DoA_GRP0_DAT_IDX_6	0xA70	R/W	20	Y	0	antenna index 0~3 to be combined in round 6 in group 0	This register shares the same format with register CFG_DoA_GRP0_DAT_IDX_0
CFG_DoA_GRP0_DAT_IDX_7	0xA74	R/W	20	Y	0	antenna index 0~3 to be combined in round 7 in group 0	This register shares the same format with register CFG_DoA_GRP0_DAT_IDX_0
CFG_DoA_GRP0_SIZ_WIN	0xA78	R/W	1	Y	0	selection bit for the window size during search in group 0	Format: FXI(1, U) 0 -> size is 3 1 -> size is 5
CFG_DoA_GRP0_THR_SNR_0	0xA7C	R/W	9	Y	0	SNR threshold 0 in group 0	Format: FXI(7, U) x -> threshold is x
CFG_DoA_GRP0_THR_SNR_1	0xA80	R/W	9	Y	0	SNR threshold 1 in group 1	Format: FXI(7, U) x -> threshold is x
CFG_DoA_GRP0_SCL_POW	0xA84	R/W	10	Y	0	scaler for the power in group 0	Format W6_I6_S0_E4_S0_P0 x -> scaler is x
CFG_DoA_GRP0_SCL NOI_0	0xA88	R/W	10	Y	0	scaler 0 for the noise in group 0	Format W6_I6_S0_E4_S0_P0 x -> scaler is x
CFG_DoA_GRP0_SCL NOI_1	0xA8C	R/W	10	Y	0	scaler 1 for the noise in group 0	This register shares the same format with register CFG_DoA_GRP0_SCL NOI_0
CFG_DoA_GRP0_SCL NOI_2	0xA90	R/W	10	Y	0	scaler 2 for the noise in group 0	This register shares the same format with register CFG_DoA_GRP0_SCL NOI_0
CFG_DoA_GRP0_TST_POW	0xA94	R/W	1	Y	0	enable bit of the power testing (testing power before searching the next bin)	Format: FXI(1, U) 0 -> disabled 1 -> enabled
CFG_DoA_GRP0_SIZ RNG AML	0xA98	R/W	6	Y	0	reserved	
CFG_DoA_GRP0_SIZ STP AML CRS	0xA9C	R/W	4	Y	0	reserved	
CFG_DoA_GRP0_SIZ STP AML RFD	0xAA0	R/W	4	Y	0	reserved	
CFG_DoA_GRP0_SCL REM_0	0xAA4	R/W	10	Y	0	reserved	
CFG_DoA_GRP0_SCL REM_1	0xAA8	R/W	10	Y	0	reserved	
CFG_DoA_GRP0_SCL REM_2	0xAAC	R/W	10	Y	0	reserved	
CFG_DoA_GRP0_ENA_NEI	0xAB0	R/W	1	Y	0	reserved	
CFG_DoA_GRP0_SIZ SUB	0xAB4	R/W	2	Y	0	reserved	
CFG_DoA_GRP0_TYP SMO	0xAB8	R/W	1	Y	0	reserved	
CFG_DoA_GRP1_TYP SCH	0 ABC	R/W	2	Y	0	selection bits for the search method in group 1	This register shares the same format with register CFG_DoA_GRP0_TYP SCH

continues on next page



Table B.1 – continued from previous page

Name	Addr	Access	BW	Bank	DV	Description	Function Description
CFG_DoA_GRP1_MOD_SCH	0xAC0	R/W	3	Y	0	selection bits for the search mode in group 1 (active only when search mode is set to DBF)	This register shares the same format with register CFG_DoA_GRP0_MOD_SCH
CFG_DoA_GRP1_NUM_SCH	0xAC4	R/W	2	Y	0	maximum number of the bins to be searched in group 1 (active only when search mode is set to DBF)	This register shares the same format with register CFG_DoA_GRP0_NUM_SCH
CFG_DoA_GRP1_ADR_COE	0xAC8	R/W	12	Y	0	offset address for the DBF/DML coefficients in group 1	This register shares the same format with register CFG_DoA_GRP0_ADR_COE
CFG_DoA_GRP1_SIZ_ONE_ANG	0xACC	R/W	1	Y	0	reserved	
CFG_DoA_GRP1_SIZ_RNG_PKS_CRS	0xAD0	R/W	10	Y	0	range of the coarse search in group 1	This register shares the same format with register CFG_DoA_GRP0_SIZ_RNG_PKS_CRS
CFG_DoA_GRP1_SIZ_STP_PKS_CRS	0xAD4	R/W	4	Y	0	step of the coarse search in group 1	This register shares the same format with register CFG_DoA_GRP0_SIZ_STP_PKS_CRS
CFG_DoA_GRP1_SIZ_RNG_PKS_RFD	0xAD8	R/W	6	Y	0	range of the refined search in group 1	This register shares the same format with register CFG_DoA_GRP0_SIZ_RNG_PKS_RFD
CFG_DoA_GRP1_SIZ_CMB	0xADC	R/W	5	Y	0	total number of antennas to be used in group 1	This register shares the same format with register CFG_DoA_GRP0_SIZ_CMB
CFG_DoA_GRP1_DAT_IDX_0	0xAE0	R/W	20	Y	0	antenna index 0~3 to be combined in round 0 in group 1	This register shares the same format with register CFG_DoA_GRP0_DAT_IDX_0
CFG_DoA_GRP1_DAT_IDX_1	0xAE4	R/W	20	Y	0	antenna index 0~3 to be combined in round 1 in group 1	This register shares the same format with register CFG_DoA_GRP0_DAT_IDX_1
CFG_DoA_GRP1_DAT_IDX_2	0xAE8	R/W	20	Y	0	antenna index 0~3 to be combined in round 2 in group 1	This register shares the same format with register CFG_DoA_GRP0_DAT_IDX_2
CFG_DoA_GRP1_DAT_IDX_3	0xAEC	R/W	20	Y	0	antenna index 0~3 to be combined in round 3 in group 1	This register shares the same format with register CFG_DoA_GRP0_DAT_IDX_3
CFG_DoA_GRP1_DAT_IDX_4	0xAF0	R/W	20	Y	0	antenna index 0~3 to be combined in round 4 in group 1	This register shares the same format with register CFG_DoA_GRP0_DAT_IDX_4
CFG_DoA_GRP1_DAT_IDX_5	0xAF4	R/W	20	Y	0	antenna index 0~3 to be combined in round 5 in group 1	This register shares the same format with register CFG_DoA_GRP0_DAT_IDX_5
CFG_DoA_GRP1_DAT_IDX_6	0xAF8	R/W	20	Y	0	antenna index 0~3 to be combined in round 6 in group 1	This register shares the same format with register CFG_DoA_GRP0_DAT_IDX_6
CFG_DoA_GRP1_DAT_IDX_7	0xAFc	R/W	20	Y	0	antenna index 0~3 to be combined in round 7 in group 1	This register shares the same format with register CFG_DoA_GRP0_DAT_IDX_7
CFG_DoA_GRP1_SIZ_WIN	0xB00	R/W	1	Y	0	selection bit for the window size during search in group 1	This register shares the same format with register CFG_DoA_GRP0_SIZ_WIN
CFG_DoA_GRP1_THR_SNR_0	0xB04	R/W	9	Y	0	SNR threshold 0 in group 1	This register shares the same format with register CFG_DoA_GRP0_THR_SNR_0
CFG_DoA_GRP1_THR_SNR_1	0xB08	R/W	9	Y	0	SNR threshold 1 in group 1	This register shares the same format with register CFG_DoA_GRP0_THR_SNR_1
CFG_DoA_GRP1_SCL_POW	0xB0C	R/W	10	Y	0	scaler for the power in group 1	This register shares the same format with register CFG_DoA_GRP0_SCL_POW
CFG_DoA_GRP1_SCL NOI_0	0xB10	R/W	10	Y	0	scaler 0 for the noise in group 1	This register shares the same format with register CFG_DoA_GRP0_SCL NOI_0

continues on next page



Table B.1 – continued from previous page

Name	Addr	Access	BW	Bank	DV	Description	Function Description
CFG_DoA_GRP1_SCL NOI_1	0xB14	R/W	10	Y	0	scaler 1 for the noise in group 1	This register shares the same format with register CFG_DoA_GRP0_SCL NOI_1
CFG_DoA_GRP1_SCL NOI_2	0xB18	R/W	10	Y	0	scaler 2 for the noise in group 1	This register shares the same format with register CFG_DoA_GRP0_SCL NOI_2
CFG_DoA_GRP1_TST_POW	0xB1C	R/W	1	Y	0	enable bit of the power testing	This register shares the same format with register CFG_DoA_GRP0_TST_POW
CFG_DoA_GRP1_SIZ RNG AML	0xB20	R/W	6	Y	0	reserved	reserved
CFG_DoA_GRP1_SIZ STP AML CRS	0xB24	R/W	4	Y	0	reserved	reserved
CFG_DoA_GRP1_SIZ STP AML RFD	0xB28	R/W	4	Y	0	reserved	reserved
CFG_DoA_GRP1_SCL REM_0	0xB2C	R/W	10	Y	0	reserved	reserved
CFG_DoA_GRP1_SCL REM_1	0xB30	R/W	10	Y	0	reserved	reserved
CFG_DoA_GRP1_SCL REM_2	0xB34	R/W	10	Y	0	reserved	reserved
CFG_DoA_GRP1_ENA_NEI	0xB38	R/W	1	Y	0	reserved	reserved
CFG_DoA_GRP1_SIZ SUB	0xB3C	R/W	2	Y	0	reserved	reserved
CFG_DoA_GRP1_TYP SMO	0xB40	R/W	1	Y	0	reserved	reserved
CFG_DoA_GRP2_TYP SCH	0xB44	R/W	2	Y	0	selection bits for the search method in group 2	This register shares the same format with register CFG_DoA_GRP0_TYP SCH
CFG_DoA_GRP2_MOD SCH	0xB48	R/W	3	Y	0	selection bits for the search mode in group 2 (active only when search mode is set to DBF)	This register shares the same format with register CFG_DoA_GRP0_MOD SCH
CFG_DoA_GRP2_NUM SCH	0xB4C	R/W	2	Y	0	maximum number of the bins to be searched in group 2 (active only when search mode is set to DBF)	This register shares the same format with register CFG_DoA_GRP0_NUM SCH
CFG_DoA_GRP2_ADR COE	0xB50	R/W	12	Y	0	offset address for the DBF/DML coefficients in group 2	This register shares the same format with register CFG_DoA_GRP0_ADR COE
CFG_DoA_GRP2_SIZ ONE ANG	0xB54	R/W	1	Y	0	reserved	reserved
CFG_DoA_GRP2_SIZ RNG PKS CRS	0xB58	R/W	10	Y	0	range of the coarse search in group 2	This register shares the same format with register CFG_DoA_GRP0_SIZ RNG PKS CRS
CFG_DoA_GRP2_SIZ STP PKS CRS	0xB5C	R/W	4	Y	0	step of the coarse search in group 2	This register shares the same format with register CFG_DoA_GRP0_SIZ STP PKS CRS
CFG_DoA_GRP2_SIZ RNG PKS RFD	0xB60	R/W	6	Y	0	range of the refined search in group 2	This register shares the same format with register CFG_DoA_GRP0_SIZ RNG PKS RFD
CFG_DoA_GRP2_SIZ CMB	0xB64	R/W	5	Y	0	total number of antennas to be used in group 2	This register shares the same format with register CFG_DoA_GRP0_SIZ CMB
CFG_DoA_GRP2_DAT IDX_0	0xB68	R/W	20	Y	0	antenna index 0~3 to be combined in round 0 in group 2	This register shares the same format with register CFG_DoA_GRP0_DAT IDX_0
CFG_DoA_GRP2_DAT IDX_1	0xB6C	R/W	20	Y	0	antenna index 0~3 to be combined in round 1 in group 2	This register shares the same format with register CFG_DoA_GRP0_DAT IDX_1

continues on next page



Table B.1 – continued from previous page

Name	Addr	Access	BW	Bank	DV	Description	Function Description
CFG_DoA_GRP2_DAT_IDX_2	0xB70	R/W	20	Y	0	antenna index 0~3 to be combined in round 2 in group 2	This register shares the same format with register CFG_DoA_GRP0_DAT_IDX_2
CFG_DoA_GRP2_DAT_IDX_3	0xB74	R/W	20	Y	0	antenna index 0~3 to be combined in round 3 in group 2	This register shares the same format with register CFG_DoA_GRP0_DAT_IDX_3
CFG_DoA_GRP2_DAT_IDX_4	0xB78	R/W	20	Y	0	antenna index 0~3 to be combined in round 4 in group 2	This register shares the same format with register CFG_DoA_GRP0_DAT_IDX_4
CFG_DoA_GRP2_DAT_IDX_5	0xB7C	R/W	20	Y	0	antenna index 0~3 to be combined in round 5 in group 2	This register shares the same format with register CFG_DoA_GRP0_DAT_IDX_5
CFG_DoA_GRP2_DAT_IDX_6	0xB80	R/W	20	Y	0	antenna index 0~3 to be combined in round 6 in group 2	This register shares the same format with register CFG_DoA_GRP0_DAT_IDX_6
CFG_DoA_GRP2_DAT_IDX_7	0xB84	R/W	20	Y	0	antenna index 0~3 to be combined in round 7 in group 2	This register shares the same format with register CFG_DoA_GRP0_DAT_IDX_7
CFG_DoA_GRP2_SIZ_WIN	0xB88	R/W	1	Y	0	selection bit for the window size during search in group 2	This register shares the same format with register CFG_DoA_GRP0_SIZ_WIN
CFG_DoA_GRP2_THR_SNR_0	0xB8C	R/W	9	Y	0	SNR threshold 0 in group 2	This register shares the same format with register CFG_DoA_GRP0_THR_SNR_0
CFG_DoA_GRP2_THR_SNR_1	0xB90	R/W	9	Y	0	SNR threshold 1 in group 1	This register shares the same format with register CFG_DoA_GRP0_THR_SNR_1
CFG_DoA_GRP2_SCL_POW	0xB94	R/W	10	Y	0	scaler for the power in group 2	This register shares the same format with register CFG_DoA_GRP0_SCL_POW
CFG_DoA_GRP2_SCL_NOI_0	0xB98	R/W	10	Y	0	scaler 0 for the noise in group 2	This register shares the same format with register CFG_DoA_GRP0_SCL_NOI_0
CFG_DoA_GRP2_SCL_NOI_1	0xB9C	R/W	10	Y	0	scaler 1 for the noise in group 2	This register shares the same format with register CFG_DoA_GRP0_SCL_NOI_1
CFG_DoA_GRP2_SCL_NOI_2	0xBA0	R/W	10	Y	0	scaler 2 for the noise in group 2	This register shares the same format with register CFG_DoA_GRP0_SCL_NOI_2
CFG_DoA_GRP2_TST_POW	0xBA4	R/W	1	Y	0	enable bit of the power testing	This register shares the same format with register CFG_DoA_GRP0_TST_POW
CFG_DoA_GRP2_SIZ_RNG_AML	0xBA8	R/W	6	Y	0	reserved	reserved
CFG_DoA_GRP2_SIZ_STP_AML_CRS	0xBAE	R/W	4	Y	0	reserved	reserved
CFG_DoA_GRP2_SIZ_STP_AML_RFD	0xBB0	R/W	4	Y	0	reserved	reserved
CFG_DoA_GRP2_SCL_REM_0	0xBB4	R/W	10	Y	0	reserved	reserved
CFG_DoA_GRP2_SCL_REM_1	0xBB8	R/W	10	Y	0	reserved	reserved
CFG_DoA_GRP2_SCL_REM_2	0xBC0	R/W	10	Y	0	reserved	reserved
CFG_DoA_GRP2_ENA_NEI	0xBC4	R/W	1	Y	0	reserved	reserved
CFG_DoA_GRP2_SIZ_SUB	0xBC8	R/W	2	Y	0	reserved	reserved
CFG_DoA_GRP2_TYP_SMO	0xBD0	R/W	1	Y	0	reserved	reserved
CFG_DoA_C2D1_ADR_COE	0xBCD	R/W	12	Y	0	offset address for the DBF coefficients used for 2-d combine in c2d group 1	Format: FXI(12, U) x -> coefficients are stored from base_address_of_mem_coe + x * 16 (in bytes and in terms of APB access)

continues on next page



Table B.1 – continued from previous page

Name	Addr	Access	BW	Bank	DV	Description	Function Description
CFG_DoA_C2D1_SIZ_CMB	0xBD0	R/W	5	Y	0	total number of antennas to be used for 2-d combine in c2d group 1	This register shares the same format with register CFG_DoA_GRP0_DAT_IDX_0
CFG_DoA_C2D2_ADR_COE	0xBD4	R/W	12	Y	0	offset address for the DBF coefficients used for 2-d combine in c2d group 2	This register shares the same format with register CFG_DoA_C2D1_ADR_COE
CFG_DoA_C2D2_SIZ_CMB	0xBD8	R/W	5	Y	0	total number of antennas to be used for 2-d combine in c2d group 2	This register shares the same format with register CFG_DoA_C2D1_SIZ_CMB
CFG_DoA_C2D3_ADR_COE	0xBDc	R/W	12	Y	0	offset address for the DBF coefficients used for 2-d combine in c2d group 3	This register shares the same format with register CFG_DoA_C2D1_ADR_COE
CFG_DoA_C2D3_SIZ_CMB	0xBE0	R/W	5	Y	0	total number of antennas to be used for 2-d combine in c2d group 3	This register shares the same format with register CFG_DoA_C2D1_SIZ_CMB
CFG_DBG_BUF_TAR	0xC00	R/W	6	N	0	selection bits for debug data dumped to FFT data memory (mem_buf) when it is not fully used	high active: bit mask with 1 means the corresponding data is saved to the 2nd 1MB of mem_buf and 0 means not saved. bit 5: CPU data bit 4: combined signal from DBF bit 3: combined signal from CFAR bit 2: reserved bit 1: reserved bit 0: reserved
CFG_DBG_MAP_RLT	0xC04	R/W	1	N	0	selection bits for data mapping of result memory (mem_rlt)	0 -> BOP order 1 -> PBO order
CTL_DBG_RFRSH_START	0xC08	W/SC	32	N	0	enable bits for the reflash operation in 32 pieces of MEM_BUF	write 1 to start, it is a self-clear register, so no need to write 0
CTL_DBG_RFRSH_CLR	0xC0C	W/SC	32	N	0	clear bits for the status of reflash operation in 32 pieces of MEM_BUF	write 1 to clear, it is a self-clear register, so no need to write 0
FDB_DBG_RFRSH_STATUS	0xC10	RO	32	N		indicate the status of reflash operation	high active for 32 pieces of MEM_BUF
CFG_DML_GRP0_SV_STP	0xE00	R/W	6	Y	0	step size of DML coarse search in group 0	Format: FXI(6, U) x -> step size is x, x should be larger than 1
CFG_DML_GRP0_SV_START	0xE04	R/W	9	Y	0	start position of DML coefficients in group 0	Format: FXI(9, U) x -> the xth coefficients is used for the start position
CFG_DML_GRP0_SV_END	0xE08	R/W	10	Y	0	stop position of DML coefficients in group 0	Format: FXI(10, U) x -> the xth coefficients is used for the stop position
CFG_DML_GRP0_DC_COE_0	0xE0C	R/W	14	Y	0	decision function coefficient 0 in group 0	Format: FXR(14, 1, S)

continues on next page



Table B.1 – continued from previous page

Name	Addr	Access	BW	Bank	DV	Description	Function Description
CFG_DML_GRP0_DC_COE_1	0xE10	R/W	14	Y	0	decision function coefficient 1 in group 0	Format: FXR(14, 1, S)
CFG_DML_GRP0_DC_COE_2	0xE14	R/W	14	Y	0	decision function coefficient 2 in group 0	Format: FXR(14, 1, S)
CFG_DML_GRP0_DC_COE_3	0xE18	R/W	14	Y	0	decision function coefficient 3 in group 0	Format: FXR(14, 1, S)
CFG_DML_GRP0_DC_COE_4	0xE1C	R/W	14	Y	0	decision function coefficient 4 in group 0	Format: FXR(14, 1, S)
CFG_DML_GRP0_EXTRA_EN	0xE20	R/W	1	Y	0	enable bit for DML extra search in group 0	high active: bit value with 1 means DML extra search is enabled
CFG_DML_GRP0_DC_COE_2_EN	0xE24	R/W	1	Y	0	enable bit for 2nd part in decision function in group 0	high active: bit value with 1 means the 2nd part in decision function is enabled
CFG_DML_GRP1_SV_STP	0xE28	R/W	6	Y	0	step size of DML coarse search in group 1	Format: FXI(6, U) x -> step size is x, x should be larger than 1
CFG_DML_GRP1_SV_START	0xE2C	R/W	9	Y	0	start position of DML coefficients in group 1	Format: FXI(9, U) x -> the xth coefficients is used for the start position
CFG_DML_GRP1_SV_END	0xE30	R/W	10	Y	0	stop position of DML coefficients in group 1	Format: FXI(10, U) x -> the xth coefficients is used for the stop position
CFG_DML_GRP1_DC_COE_0	0xE34	R/W	14	Y	0	decision function coefficient 0 in group 1	Format: FXR(14, 1, S)
CFG_DML_GRP1_DC_COE_1	0xE38	R/W	14	Y	0	decision function coefficient 1 in group 1	Format: FXR(14, 1, S)
CFG_DML_GRP1_DC_COE_2	0xE3C	R/W	14	Y	0	decision function coefficient 2 in group 1	Format: FXR(14, 1, S)
CFG_DML_GRP1_DC_COE_3	0xE40	R/W	14	Y	0	decision function coefficient 3 in group 1	Format: FXR(14, 1, S)
CFG_DML_GRP1_DC_COE_4	0xE44	R/W	14	Y	0	decision function coefficient 4 in group 1	Format: FXR(14, 1, S)
CFG_DML_GRP1_EXTRA_EN	0xE48	R/W	1	Y	0	enable bit for DML extra search in group 1	high active: bit value with 1 means DML extra search is enabled
CFG_DML_GRP1_DC_COE_2_EN	0xE4C	R/W	1	Y	0	enable bit for 2nd part in decision function in group 1	high active: bit value with 1 means the 2nd part in decision function is enabled
CFG_DML_MEM_BASE_ADR	0xE50	R/W	10	Y	0	address offset for group 1 in DML Memory(D,K coefficient)	Format: FXI(10, U)



## BASEBAND SHADOW BANK (BANK 5) REGISTERS

Table C.1: Registers in Shadow Bank (Bank 5)

RegisterName	Address	Access	BW	Shadow Bank (Bank 5)
CFG_SYS_ENABLE	0x0018	R/W	9	Y
CFG_SYS_SIZE RNG_PRD	0x0020	R/W	32	Y
CFG_SYS_SIZE FLT	0x0024	R/W	4	Y
CFG_SYS_SIZE RNG_SKP	0x0028	R/W	32	Y
CFG_SYS_SIZE RNG_BUF	0x002C	R/W	11	Y
CFG_SYS_SIZE RNG_FFT	0x0030	R/W	11	Y
CFG_SYS_SIZE BPM	0x0034	R/W	2	Y
CFG_SYS_SIZE VEL_BUF	0x0038	R/W	10	Y
CFG_SYS_SIZE VEL_FFT	0x003C	R/W	10	Y
CFG_AGC_SAT_THR_TIA	0x0200	R/W	24	Y
CFG_AGC_SAT_THR_VGA1	0x0204	R/W	24	Y
CFG_AGC_SAT_THR_VGA2	0x0208	R/W	24	Y
CFG_AGC_DAT_MAX_SEL	0x0210	R/W	2	Y
CFG_AGC_CHCK_ENA	0x0264	R/W	1	Y
CFG_FFT_SHFT_RNG	0x0600	R/W	11	Y
CFG_FFT_SHFT_VEL	0x0604	R/W	10	Y
CFG_FFT_DBPM_ENA	0x0608	R/W	1	Y
CFG_FFT_DAMB_ENA	0x0610	R/W	1	Y
CFG_FFT_DINT_ENA	0x0614	R/W	3	Y
CFG_FFT_DINT_MOD	0x0618	R/W	3	Y
CFG_FFT_DINT_DAT_FH	0x0620	R/W	32	Y
CFG_FFT_DINT_DAT_CS	0x0624	R/W	32	Y
CFG_FFT_DINT_DAT_PS	0x0628	R/W	32	Y
CFG_FFT_DINT_MSK_FH	0x062C	R/W	32	Y
CFG_FFT_DINT_MSK_CS	0x0630	R/W	32	Y
CFG_FFT_DINT_MSK_PS	0x0634	R/W	32	Y
CFG_FFT_DINT_COE_FH	0x0638	R/W	28	Y
CFG_FFT_DINT_COE_CS	0x063C	R/W	28	Y
CFG_FFT_DINT_COE_FC	0x0640	R/W	28	Y
CFG_FFT_DINT_COE_PS_0	0x0644	R/W	28	Y
CFG_FFT_DINT_COE_PS_1	0x0648	R/W	28	Y
CFG_FFT_DINT_COE_PS_2	0x064C	R/W	28	Y
CFG_FFT_DINT_COE_PS_3	0x0650	R/W	28	Y
CFG_FFT_NO_WIN	0x0654	R/W	1	Y
CFG_FFT_NVE_MODE	0x0658	R/W	1	Y
CFG_FFT_NVE_SCL_0	0x065C	R/W	9	Y
CFG_FFT_NVE_SCL_1	0x0660	R/W	9	Y
CFG_FFT_NVE_SFT	0x0664	R/W	3	Y
CFG_FFT_NVE_CH_MSK	0x0668	R/W	16	Y
CFG_FFT_NVE_DFT_VALUE	0x066C	R/W	20	Y
CFG_FFT_ZER_DPL_ENB	0x0670	R/W	1	Y



---

**APPENDIX****D**

---

**REVISION HISTORY**

Table D.1: Revision History

Revision	Description	Author
0.1/ June 2018	Initial release for Alps A sample.	yzhu
0.2/ April 2019	<p>Added the Auto Gain Control section to the Advanced Features chapter.</p> <p>Modified register addresses throughout the book.</p> <p>Updated the Memory Mapping table.</p> <p>Changed maximal 1D-FFT size from 1024 to 2048.</p> <p>Changed CFAR maximal 2D-window size from <math>5 \times 5</math> to <math>7 \times 9</math>.</p> <p>Updated the Rx Beamforming chapter.</p> <p>Updated the Virtual Array (MIMO) section in the Advanced Features chapter.</p> <p>Updated register tables throughout the book.</p>	pwang
0.3/ July 2019	<p>Added the 2D Support in Digital Beamforming, Frame Interleaving, Multiple Signal Classification (MUSIC) sections to the Advanced Features chapter.</p> <p>Updated the CFAR Output Rule figure.</p> <p>Updated the Rx Beamforming chapter.</p> <p>Updated the process time in both typical and tough scenarios.</p> <p>Other minor changes.</p>	pwang
0.9/ February 2020	Major updates throughout the document. Preliminary release for the mass production version.	pwang
1.0/ September 2020	Official release for the mass production version.	pwang
1.0.1/ December 2020	<p>Added the section Programmable Commands for Chirps in the FMCW Waveform Generator chapter.</p> <p>Switched the order of the scalar and shifter in the Data Flow in Decimation Filter figure and Post-Process at Output of Last (Fourth) Second-Order Section figure.</p> <p>Changed the number of steering vectors for DML from 1024 to 760.</p> <p>Updated the format of SAMPLE output data, SAMPLE debug data, and HIL input data from FXI(16, 2) to FXR(16, 2, S).</p>	pwang

continues on next page

Table D.1 – continued from previous page

Revision	Description	Author
1.0.2/ August 2021	<p>In the Hardware Programming Interfaces chapter,        - Updated the Finite State Machine of Baseband Core figure.        - Added the Memory Refresh section.        - In the Interrupts section, corrected the descriptions of IRQ bit order.</p> <p>In FMCW Waveform Generator chapter, fixed bank number errors in the Examples section.</p> <p>In the Constant False Alarm Rate (CFAR) Detectors chapter, updated descriptions of parameters for CA-CFAR, OS-CFAR, SOGO-CFAR, and NR-CFAR.</p> <p>In the Auto Gain Control (AGC) chapter, updated descriptions of monitoring-related AGC registers.</p> <p>In the Interference Avoidance and Mitigation chapter, changed the recommended setting of CFG_SAM_SPK_THRES_DB<sub>L</sub> for interference mitigation in the example.</p> <p>In the Data Collection and Hardware-in-the-Loop (HIL) chapter,        - In the FFT Output Data Size section and SAMPLE Output Data Size section, added notes about the difference between DMP_MID and DMP_FNL.        - In the DoA Debug Data Size section, added a description of power spectrum.</p> <p>In the Memory Mapping table, modified the shared RAM size from 2M to 1M.</p> <p>Updated descriptions in Baseband Register Table.</p> <p>Other refinement.</p>	pwang



## **IMPORTANT NOTICE AND DISCLAIMER**

Information in this document is provided solely to assist users to use Calterah's products and services.

Calterah Semiconductor Technology (Shanghai) Co., Ltd. ("Calterah") reserves the right to change, modify, amend this document as it sees fit. The information in this document is subject to change without notice.

Every effort has been made in the preparation of this document to ensure accuracy and completeness of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied, including but not limited to, the implied warranties of merchantability and fitness for a particular purpose or non-infringement of third party intellectual property rights.

Calterah owns the Intellectual Property Rights to the contents in this document and the Intellectual Property Right to the underlying technologies for Calterah's provision of service. Intellectual Property Right means all patents, copyrights, layout-designs, utility models, know-hows, trade secrets, trademarks, service marks, trade dress or other intellectual property rights to Calterah's products and services. Intellectual Property Rights shall include intellectual property rights in and of any variations and changes of company name, chip names, images of any forms of products and services, service marks, trademarks and products and services.

This document contains no express or implied authorization, license, or grant of right to use any trademarks, service marks, trade names, domain names, website names and any other prominent brand features of Calterah, including but not limited to, "calterah", "Calterah", "jiatelan", "Jiatelan" and "[calterah.com] (<https://www.calterah.com/>)" (collectively, "Marks"). Without Calterah's prior written consent, you may not display or use the Marks or apply for trademark registration or domain name registration of the Marks, alone or in combination in any way whatsoever, and may not express or imply to others that you have the right to display, use or otherwise dispose of the Marks. You shall indemnify and hold Calterah and its affiliates, officers, employees, agents and consultants harmless against any and all claims, costs, damages, and expenses, and reasonable attorney fees arising from your non-compliance with the terms and provisions of this document.



## BIBLIOGRAPHY

- [Berrick1973] Donald Barrick “FM/CW Radar Signals and Digital Processing” Jul. 1973.
- [Cai2011] T. Cai and L. Wang “Orthogonal Matching Pursuit for Sparse Signal Recovery With Noise” *IEEE trans. on information theory*, Jun., 2011
- [calterah2020] Calterah Semiconductor “Calterah Alps Reference Manual” 2020.
- [Rohling1983] Hermann Rohling “Radar CFAR Thresholding in Clutter and Multiple Target Situations” *IEEE Transactions on Aerospace and Electronic Systems* 1983

## **Bibliography**

---



# INDEX

## A

AGC, 185  
anti velocity ambiguity with chirp delay, 42  
Anti-VELAMB CD, 42, 232  
Anti-VELAMB with chirp delay, 232  
auto gain control (*AGC*), 43  
azimuth angle, 158

## B

bandwidth, 12  
basic FMCW frames, 46  
BPM, 151  
BPM demodulation (*DBPM*)  
    BPM demodulation coefficients, 163  
butter-fly engine, 100

## C

CA-CFAR, 112  
CFAR, 107  
CFL, 20  
CFX, 20  
Chebyshev Type II Filter, 90  
chirp, 12  
chirp shifting mode (*CS*), 41  
complex number, 20  
complex number with common exponent, 20  
Constant False Alarm, 107

## D

decimation filter, 90  
digital signal processor, 11  
Direction of Arrival (*DoA*), 14  
*DoA*, 127  
Doppler effect, 13  
down-sampling, 90  
DSP, 11

## E

elevation angle, 158

## F

FFT, 13, 97

FFT engine, 100  
field of view, 13  
fixed-point integer, 17  
fixed-point real number, 18  
FLR, 19  
FMCW, 12  
FMCW chirp, 89  
FMCW frame, 89  
FOV, 13  
frame interleaving, 43, 217  
frequency hopping, 201  
frequency hopping mode (*FH*), 40  
FXI, 17  
FXR, 18

## H

Hadamard Matrix, 154

## I

interference, 201

## M

MCU, 11  
memory mapping, 32  
MIMO, 151  
MIMO Combination, 108

## N

NR-CFAR, 112

## O

OS-CFAR, 112

## P

Peak Detector, 112  
phase scramble mode (*PS*), 41  
phase scrambling, 201  
pseudo-floating real number, 19

## R

radar baseband processing unit, 11  
radar microcontroller unit, 11

## Index

---

RBPU, 11  
rotate mode, 217  
RX, 13

### S

second-order section, 90  
side information, 158  
simple complex number, 20  
single mode, 217  
SISO Combination, 107  
SOGO-CFAR, 112  
steering vectors, 127

### T

TDM, 151  
TX, 12

### V

virtual array, 151  
virtual array mode (*VAM*), 39

### X

XOR Chain, 202

