
LLRF System For ESS – Software Module Technical Documentation

Author	Affiliation	Reviewer	Approver
Urša Rojec	Cosylab	Klemen Strniša	
Alexander Söderqvist	Cosylab		

DOCUMENT REVISION HISTORY

Version	Reason for revision	Date
1.0	Initial version	2014-03-07
1.1	<ul style="list-style-type: none"> First review comments Added asynReason list and NDS Class descriptions 	2014-06-12
1.2	<ul style="list-style-type: none"> Adding new functionality according to firmware updates, Added demo application description Removed obsolete appendices Added sis8300noAO.db, sis8300Reg.db and sis8300llrfReg.db to description Fixed GUI instructions – added new screenshots 	2014-12-22
	<ul style="list-style-type: none"> Second review comments 	2015-01-21
1.3	<ul style="list-style-type: none"> FF Table Speed Settings Modulator Ripple Filter Settings 	2015-01-23
1.4	<ul style="list-style-type: none"> Added EPICS Status tab screenshot, updated others 	2015-02-26
2.0	<p>Major changes due to</p> <ul style="list-style-type: none"> Transition from MA to IQ control in firmware Changes in the generic sis8300 epics module 	2015-02-27
2.1	<ul style="list-style-type: none"> Added Signal Monitoring Added New OPI screenshots Added Demo Screenshots Added RTM Settings Added Interlock Added Special Operating Modes 	2015-04-15
2.2	<ul style="list-style-type: none"> General Update of exported NDS parameters Added Demo Application (partially – GUI) 	2015-04-20
2.3	<ul style="list-style-type: none"> Updated the epics module to the latest changes in the code Added Control Table Generation OPI Added Timing OPI Added Calibration Procedure OPI 	2015-05-26

	<ul style="list-style-type: none"> • Added Special Operating Modes OPI • Added demo ioc startup settings • Added demo archiver explanation • Added timestamping information • Added hardware setup section with figures • Added rtm description settings • Added information section – where to find • Fixed the errors in current development system information (AI6 is hijacked also) • Updated the Main and overview screenshot and description • Added python, numpy and pyepics installation instructions • Added archiver installation and startup information • Removed cavity and reference DC offset 	
2.4	<ul style="list-style-type: none"> • Updated Scripts section • Added typical Operation Section 	2015-06-01
	<ul style="list-style-type: none"> • Updated the Main screen 	2015-06-03
2.5	<ul style="list-style-type: none"> • Added RTM connection Instructions 	2015-06-11
2.6	<ul style="list-style-type: none"> • Removed the user manual – it is now available on the wiki 	2015-10-06
2.7	<ul style="list-style-type: none"> • Update after major code refactoring 	2016-01-15
2.8	<ul style="list-style-type: none"> • Fixed broken references and figures. • Add notch filter • Added writing of control tables PV • New Channel Data Ready PV 	2016-12-08

TABLE OF CONTENTS

1	About This Document	6
2	Theory of Operation	7
2.1	Overview of Hardware and Software Components	7
2.2	Hardware Operation	9
2.3	Software Operation.....	10
2.3.1	Control Tables	11
2.3.2	Control System State Machine	12
3	Architecture	14
3.1	Kernel Module.....	14
3.1.1	Implementation.....	14
3.2	User-space Library	14
3.2.1	Implementation.....	14
3.2.2	Exported interface	14
3.2.3	Generic sis8300 interface and its altered functionality in LLRF context.....	16
3.3	EPICS Device Support - NDS.....	18
3.3.1	Responsibilities	18
3.3.2	Implementation.....	19
3.3.3	Driver Initialization Parameters	20
3.3.4	Exported interface	20
3.4	EPICS Database	42
3.4.1	Exported interface	42
3.5	Startup Snippets.....	67
3.6	Demo application	67
3.7	Software Version	67
3.8	Learning Feed Forward	67
4	References	69
5	List of Abbreviations.....	70
6	APPENDIX: Current Development System.....	71
7	APPENDIX: Control Table Generation	72

1 ABOUT THIS DOCUMENT

As this is a rather long description of the LLRF software functionality we understand that reading it all in one piece takes a lot of time. For a feel of the software workings we recommend that you at least take a look at http 1 user manual

Theory of Operation and EPICS Device Support - NDS (Specifically: 3.3.13.3.2, 3.3.3 and 3.3.4) chapters.

This document is a software reference manual. For the user manual see:

<https://ess-ics.atlassian.net/wiki/display/HAR/Low+Level+RF+System>

http 1 user manual

2 THEORY OF OPERATION

The LLRF system will be responsible for controlling the field in accelerating cavities throughout the entire accelerator, which includes RFQ, DTL, Spoke cavities and medium and high beta cavities. Each cavity will have a separate klystron, a topology which implies the use of a separate LLRF system for each cavity – klystron pair (from now on referred to as RF cell). The system will be responsible for maintaining the phase and amplitude stability of the field in that particular cavity, which will be achieved by monitoring the current state of the RF cell and providing a driving signal for the klystron.

At the core of the LLRF system will be a LLRF controller board that will use a combination of feedback and feedforward to compensate for field perturbations such as Lorentz force detuning, microphonics, bunch charge fluctuations, etc. Each of the feedback and feedforward will be responsible for a different type of perturbations; FF will try to compensate for repetitive (occurring on pulse to pulse bases), and FB for random errors.

The LLRF controllers will be implemented on the same hardware for all the RF cells along the accelerator.

In addition to field control, the LLRF system will also be included in cavity resonance control or frequency tuning which will be done in two steps – coarse frequency tuning with stepper motors and fine tuning with piezo motors. None of the mentioned systems exists yet.

2.1 Overview of Hardware and Software Components

General overview of software and hardware components of the LLRF system is depicted on Figure 1.

Hardware – the LLRF controller board – is represented by block 3 and will provide generic digitizer interface (3.1) alongside custom, LLRF specific firmware (3.2). The software part will be responsible for integration of the board into the ICS and will cover blocks 4, 5, 6, 10 and 11.

The LLRF controller board will be implemented on the same hardware for all the RF cells along the accelerator. The controller board is realized on a Struck SIS8300L digitizer board [1]. This as an AMC compliant to the MTCA.4 standard, which serves as the digital processing part of the LLRF. All the digital processing and control logic is realized on the on-board FPGA, by extending the generic Struck digitizer functionality. The analogue front end is realized on the RTM and can be different. The choice of the RTM will depend on more factors, the more obvious one of them being the RF frequency. Currently there are three types of RTMs being used during development SIS8900 [2], DWC8VM1 [3] and DS8VM1 [no manual yet]

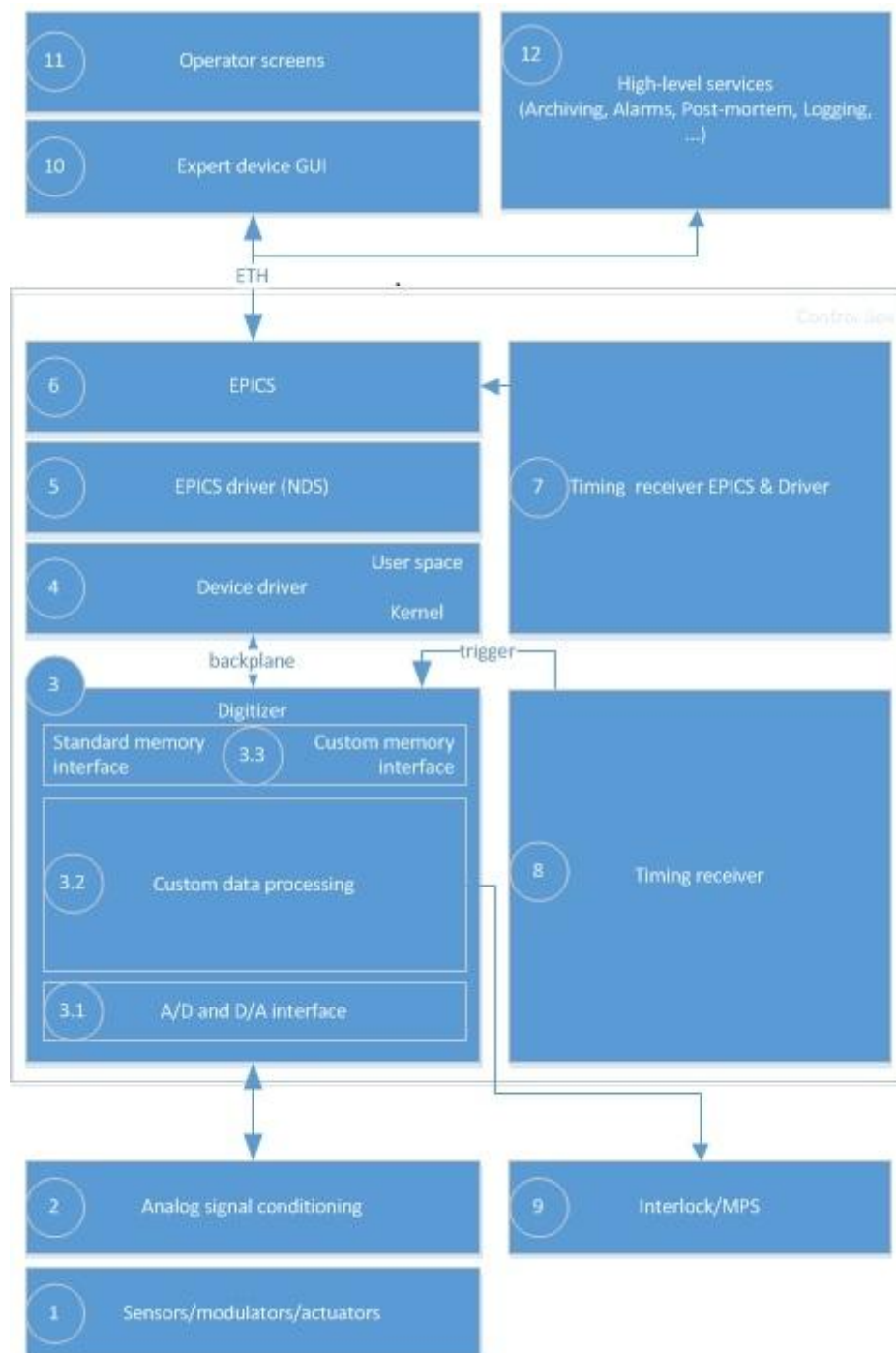


Figure 1 LLRF software and hardware components and their interactions to other parts of ICS (timing system, MPS). The software part of LLRF system has to cover blocks 4, 5, 6, 10 and 11. Block 3 represents the LLRF controller board.

2.2 Hardware Operation

In order to define the software architecture, some knowledge of hardware operation is needed. Figure 2 presents a LLRF control loop of one RF cell.

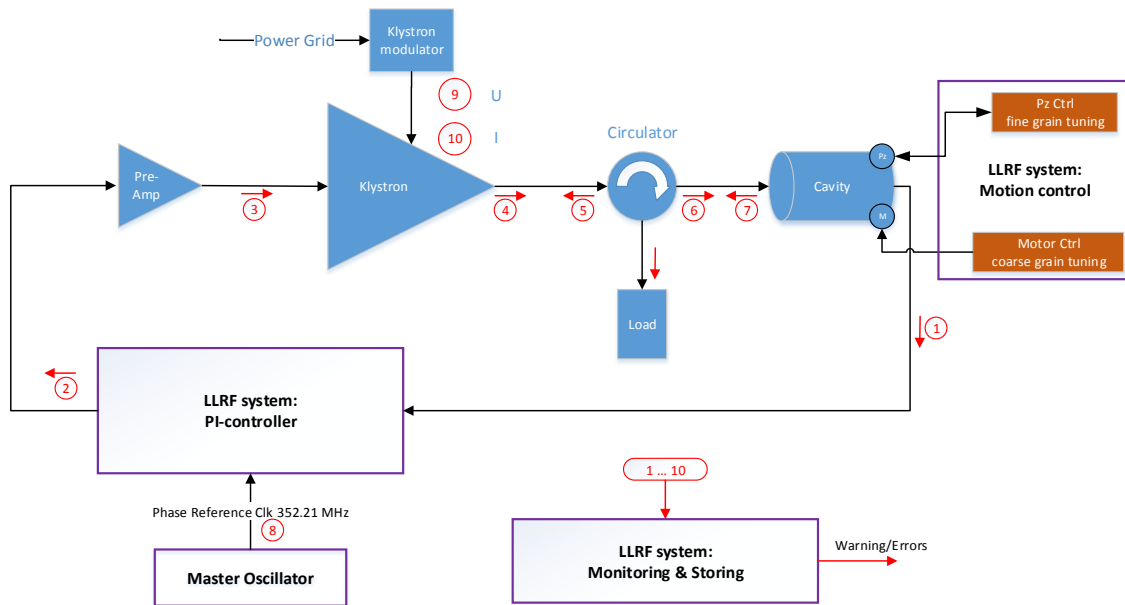


Figure 2 represents LLRF control loop an input signals for the LLRF controller board (labelled as LLRF system). Figure was taken from [1].

The PI controller is realized on the FPGA of the SIS8300L [1] digitizer board, and the 10 input signals arrive to the board over an RTM, connected to the board. The monitoring and storing part on Figure 2 represents the software – the scope of this document – while the Motion control part is not yet realized. The blue components are out of scope of LLRF as is the Master oscillator, which provides the RF reference.

The LLRF controller board takes 10 analogue inputs (Figure 2, Table 1):

AI Channel Number	Signal
1	Cavity probe
2	LLRF controller output (read back)
3	Pre-amplifier output
4	Klystron output
5	Circulator reflected signal
6	Cavity drive signal
7	Cavity reflected signal
8	Master oscillator

9	Klystron modulator U
10	Klystron modulator I

Table 1 List of LLRF controller board AI signals. **At this point in time, the development version does not yet include all the signals. See APPENDIX: Current Development System.**

that represent the current state of the RF cell. The signals serve as input for the LLRF control loop and get processed by an FPGA located on the Struck SIS8300L AMC (LLRF controller board). Result of processing are two analogue signals (phase and amplitude) used to drive the klystron.

In addition to 10 AI channels, the FPGA also provides two virtual channels, corresponding to PI error for magnitude and angle controller. The term virtual channel is used for the channels that for all software purposes behave like analogue input channels but rather than belonging to a direct physical input on the RTM, they are a result of some processing done by the controller. From the software point of view they are just waveforms stored in controller memory, which makes the interface to them undistinguishable from a physical data channel.

Main logic of the LLRF system is implemented on the LLRF controller board that processes the input in several functional blocks:

1. PI regulator,
2. Feed Forward correction (FF),
3. Klystron linearization block,
4. High Voltage Feed forward (HV FF),
5. Amplitude Limiter

Table 2: FPGA Processing Blocks. **At this point in time, the development version does not include all the blocks. See APPENDIX: Current Development System.**

Each of the blocks requires separate configuration which can be specific to an RF cell. The configuration is thus not provided by hardware, but needs to be set trough software by user. Having configurable blocks is just one of the features that allow for the use of same LLRF controller boards throughout the accelerator as mentioned in 1.

2.3 Software Operation

The software part of the LLRF control system is responsible for integrating the LLRF controller board into the ICS. It needs to provide a configuration for each of the HW functional blocks, readback of HW status and run the Learning Feed Forward algorithm (LFF). The algorithm is not a part of CS integration and is thus out of scope of this document. Software blocks, along with systems they connect to, are represented on Figure 1.

The larger part of SW responsibility is thus providing a communication between the user and the HW. HW status needs to be continuously updated (read from the LLRF controller) and provided to the user. This includes providing the user with access to all the AI signals listed in Table 1 and PI error as well

as read back of current configuration, represented in Table 2. In the other direction, from user to HW, the data is sent on demand rather than continuously.

2.3.1 Control Tables

The LLRF controller will require several control tables during operation, they are listed in Table 3:

1. Set Point table (SP)
2. Feed Forward table (FF)
3. Feed Forward Correction Table (FF_CORR), the result of LFF (see 3.4.1.10)
4. Klystron linearization table
5. High Voltage Feed forward table (HV FF)

Table 3 List of LLRF controller control tables, see also Table 2.

Control tables 4 and 5 can be directly mapped to FPGA functional blocks 3 and 4 (Table 3), while control tables 2 and 3 together constitute the configuration of FF functional block 2. Control table 1 represents the desired phase/amplitude (I/Q) of the field during ramp up phase.

There is a big conceptual difference between the tables 1 - 3 and tables 4 - 5. The last two tables compensate for non-ideal behaviour of physical LLRF components (nonlinear behaviour of the klystron and ripple and droop from the modulator output), while tables 1 – 3 concern themselves with the RF field. The modulator and klystron get calibrated during the commissioning phase, so the tables 4 and 5 remain constant during operation. This however, is not the case with tables 1 – 3. The three tables are responsible for RF field control and can change on pulse-to-pulse basis.

The method for generation of control tables is not yet defined, some comments on this can be found in the APPENDIX: Control Table Generation.

2.3.2 Control System State Machine

2.3.2.1 Hardware State Machine

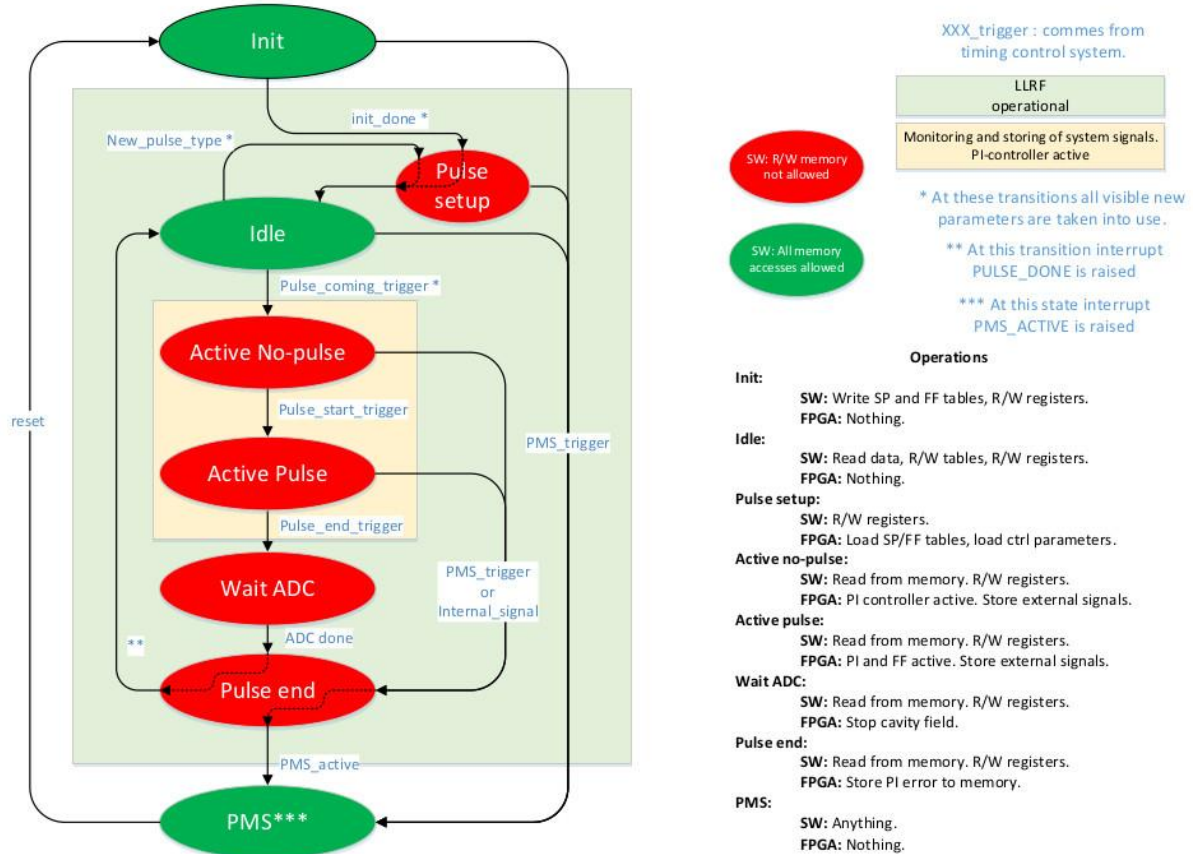


Figure 3 Hardware state machine

The firmware has its own state machine (depicted on Figure 3) that is tightly linked to three cavity states – prepare for beam, beam and no beam. These states are defined with 3 timing events plus one PMS event, which is why the LLRF board will need 4 trigger inputs, each for one timing event:

Event	Event Source	Software Interrupt
PULSE_COMING	Timing System	none
PULSE_START	Timing System	none
PULSE_END	Timing System	yes (after controller finishes and goes to IDLE)
PMS	PMS??	yes

Table 4 Timing events relevant for the operation of the LLRF system.

PMS event gets emitted only in case of machine error, but the other three events are emitted for every

pulse. They will always have to be present during normal operation. The PULSE_COMMING event will tell the LLRF when to start ramping up (playing the SP table), PULSE_START will tell the LLRF to hold the field (play the FF table) and PULSE_END will tell it to turn the field off (or ramp down). Events PULSE_COMMING and PULSE_END define the ACTIVE state, during which the controller is busy with the pulse. During this phase, the control loop is running and signals are being sampled and stored into board memory (storage is optional and set by NSAMPLES parameter, sampling and processing of the signal is not – see 0).

2.3.2.2 Software State Machine

The software state machine is realized on the EPICS level and is a trimmed version of the hardware state machine, meaning that it has less states. It groups all the states where the LLRF is operational (green area on Figure 4) into a single state called ON, which basically indicates that the controller is running. The INIT and PMS states are mapped to corresponding hardware states. In addition to these three, the software state machine also defines two additional states – ERROR and OFF. The detailed description of the states and their transitions can be found in 3.3.4.1.

Software will only access the board during the IDLE phase (restriction will be enforced at EPICS level), where it has full read and write access to registers and memory. To inform the SW when IDLE state begins, the controller will emit a PULSE_DONE interrupt when transition from PULSE_END to IDLE state occurs.

Upon receiving the PULSE_DONE interrupt, the SW will know that the controller is done with the pulse. It should then fetch the ADC (Table 1) and internal signal samples (PI error) from the controller memory and make the data from the past pulse (=sampled during ACTIVE phase) available to the user. After this, the SW will write the new control tables to controller memory and perform any parameter setup that needs to be done before the next pulse arrives. When finished with setup, the SW will arm the board, so that it will start waiting for the next PULSE_COMMING trigger.

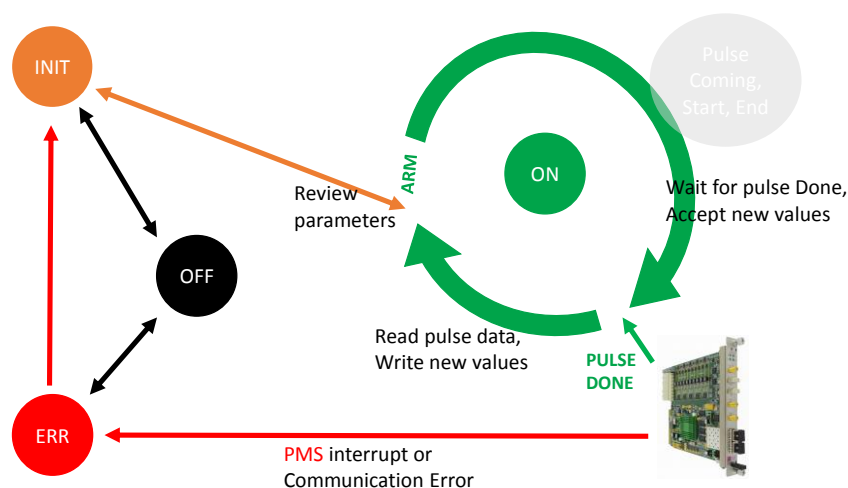


Figure 4: Software state machine.

3 ARCHITECTURE

Integration of the LLRF controller board into the ICS will be done in several blocks, as depicted in Figure 1. At the top level there is an Expert Screen (blocks 10 and 11 on Figure 1) that connects to the EPICS database (block 6 on Figure 1). The two blocks provide the user with a functionally sensible/hardware independent overview of the LLRF system, and enable access to the LLRF controller board from GUI or another CA client. Integration of card into EPICS is done through device support with the help of NDS framework. NDS (block 5 on Figure 1) provides communication between EPICS database and user-space API (block 4 on Figure 18) and is responsible for board configuration and tracking of the controller state. The two lowest lying blocks, kernel module and user-space library, are the only two layers that are aware of actual hardware specific implementation (such as register map).

3.1 Kernel Module

Kernel module represents the lowest lying SW layer and has direct access to HW registers. Its responsibility is to hold a list of all attached boards, to provide a register map, through which the HW registers are accessed from SW and handle DMA transfers to and from the board.

Since the layer provides raw access to HW registers, intimate knowledge of the LLRF controller register map is required to access the layer directly. To hide this, a user-space library is provided, which exports LLRF controller functionality in terms of descriptive function calls. In normal operation applications than never accesses the kernel module directly, but use the user-space library instead.

3.1.1 Implementation

A kernel module that handles DMA transfers, mapping of board registers and access to the board through standard *dev* interface was already developed as part of support for generic Struck SIS8300 firmware [2]. The existing module covers all the functionality and can be reused as-is.

3.2 User-space Library

The goal of user-space library is to hide the HW specific implementation by exporting an API that covers all the functionality provided by the controller board. The LLRF addition to the generic sis8300 user space library is stateless, meaning that it does not store any data but consists solely of function calls and structure definitions. None of the parameters, settings or data tables that are already stored in board's memory or registers are duplicated here.

The user space API provides communication between the kernel module and top level applications. In the case of LLRF controller this "application" will be the NDS, but there are no actual restraints to using the user-space API from other (not EPICS related) applications.

3.2.1 Implementation

Since the functionality of generic Struck firmware is already supported by [4], the LLRF specific firmware support will be added as a separate library that will depend on the generic one. An application that will want to use the LLRF specific functionality will thus have to include both libraries.

3.2.2 Exported interface

Exported interface is an API that covers the functionality of the LLRF controller [4]. Generic Struck

firmware for the SIS8300L digitizer is not included in support but provided by a separate library.

3.2.2.1 Conversion to and from device data format

For non-integer parameters, the board uses fixed point representation. The user space library provides a function to convert to/from double to these fixed number representations. This is hidden from the library user, since parameters are set through a series of exported functions. The fixed point is specified in the form of

- Signed(integer bits, fractions bits) or
- Unsigned(integer bits, fractional bits)

for every parameter separately (see [4]).

What is not hidden from the library user are PI error and FF and SP tables. All of them are stored in the memory as 32 bit wide samples. One sample contains information about angle and magnitude and a function is provided to either split a raw sample to angle and magnitude or to join the angle and magnitude into a 32 bit wide raw sample.

3.2.2.2 Memory map

Custom logic requires a special memory map. This includes setting the addresses for storing PI Error, SP and FF tables. User space library provides a function that sets this addresses based on the number of pulse types and maximum size for each of the SP and FF tables and PI error. The maximum allowed sizes are obtained from board registers.

Since the generic sis8300 library expects the ADC data to be stored at the beginning of the memory, this function also makes sure that the memory reserved for the custom logic is reserved at the end. This allows the reuse of generic functions for reading and memory setup for the ADC sampling.

This function should always be called when the board is powered on, or when a software reset is executed, because this resets all custom (LLRF specific) settings.

3.2.2.3 Cavity Signal

For the cavity signal, a read-only value is provided by the controller, which specifies the number of samples taken during active phase.

3.2.2.4 PI Error

Custom LLRF firmware provides an additional internal signal, which does not correspond to any of the physical AI channels, but represents the PI error. The difference in interface to this channel with respect to generic ADC channels is that there is no nsamples setting for PI error. Value for nsamples is provided as a read-only parameter after each pulse.

The library also provides a function to read the raw (in fixed point format, where PI error and magnitude waveforms are interleaved, see 3.2.2.1) PI Error values from controller memory.

3.2.2.5 Control Tables

A part of board memory is reserved for storing control tables (SP and FF), one for every pulse type. The interface to Control Tables is much the same as for a normal I/O channel, where pulse type corresponds

to channel number. The user-space library provides functions to write or read the raw table for the current pulse type or set the number of samples in the control tables belonging to current pulse type.

When writing a table, the library only checks if pulse type is out of range. Other than this, it does not perform any checks on data validity but simply copies a block specified by nsamples to board memory. It is up to the method calling the function to make sure that the content of tables is correct (also see 3.2.2.1).

3.2.2.6 Signal Monitoring

The controller allows for setup of min and max limit for 8 channels (Also see APPENDIX: Current Development System). ADC Channels 0 and 1 represent the cavity and reference input respectively, and do not have signal monitoring functionality. Signal monitoring setup allows the selection of signal type (either AC or DC) and a threshold value. User can then define on what conditions an alarm should be raised and what should the action on alarm be.

NOTE: One of possible actions of signal monitoring is to trigger interlock action is set to trigger interlock, then first harlink output will go high.

3.2.2.7 Trigger Setup

As already explained in 2.22.2, the LLRF doesn't behave as a generic DAQ card but needs 4 specific triggers to function (Table 4). Each of this triggers must be connected to a separate trigger line, which is why firmware offers 3 different trigger setups. Each of the setups define which trigger line represents what event, where the trigger line for PMS is common to all the setups.

3.2.2.8 Interlock

Controller provides 4 different types of interlock conditions on harlink inputs 0 – 3. In addition to generic options to enable external trigger on rising or falling edge, a high and low level condition is also implemented in the custom logic.

3.2.2.9 Special Operating Modes

The LLRF controller board can function in several operating modes [4]. Each of this modes must be set up and can also be operated in CW mode. Both are provided for in the library, CW mode can be managed with software triggers, and is explained in on the OPI (see http 1 user manual)

3.2.3 Generic sis8300 interface and its altered functionality in LLRF context

Some of the generic firmware functionality is altered when using the board with custom LLRF firmware. This normally affects some of the functions provided in the generic sis8300 user space library. The affected functions are listed in Table 5.

Generic Function	LLRF context
Arm the device	After each PULSE_DONE interrupt, the board has to be rearmed.
Disarm the device	Has no effect when done through software.

Pretrigger	On generic Struck FW there is an option to set pretrigger – samples to acquire before trigger. This functionality is no longer be available with LLRF custom FW. This setting will be ignored.
ADC nsamples	<p>On generic Struck FW there is an option to specify the number of samples that have to be acquired during acquisition.</p> <p>The LLRF FW is designed so that this setting only effects the amount of data written to RAM that can be readout by the user. The whole LLRF FPGA processing chain is ignorant of the setting, ADCs run constantly and the PI controller always gets input. (If this was not the case, the correct setting of NSAMPLES would be crucial, since PI needs to obtain current state of the LLRF system in order to work properly).</p>
Enable acquisition for ADC channel (ADC channel mask)	<p>All the ADC channels with connected signals have to be enabled while controller is running (see also APPENDIX: Current Development System). Disabling a channel would cause the controller to receive only zeroes for that input and thus improper operation.</p> <p>! IMPORTANT: If The channel 0 that requires cavity input is not enabled, the control loop will not start. This channel should always be enabled.</p>
AO Channels	The output of the board running custom LLRF FW is set by the FPGA and is used to drive the klystron. The applications should not use the write AO functions from generic FW.
DAC Setup	The board uses DAC output to drive the klystron. The generic DAC_CONTROL_REGISTER should not be touched directly. This setup should be done through the provided user-space library function.
ADC Setup	ADC tap delay needs to be configured when the board is started. This is done through a provided user space library function.
DAQ Done interrupt	<p>This interrupt is no longer in use and has been replaced by PULSE_DONE interrupt. User should not depend on this interrupt.</p> <p>UPDATE: although the interrupt can still be found in the Struck documentation, it is no longer connected.</p>
Trigger setup	<p>All the generic Trigger setup has no meaning with LLRF specific FW. The controller offers a custom register with 2 available trigger setups.</p> <p>Trigger settings in registers: LVDS_IO_CONTROL_REG and SAMPLE_CONTROL_REG are ignored</p>
Harlink Input	Harlink inputs, controlled in HARLINK_IN_OUT_CONTROL_REG are used to setup the interlock condition.

Software Interrupt	Custom logic offers two software interrupts, which are connected to the user interrupt line provided by generic struck FW. The two interrupts are PMS and PULSE_DONE. The reason for the interrupt can be read out from a custom register (GOP, see [4]).
--------------------	---

Table 5: Generic FW functions and their meaning in LLRF context

3.3 EPICS Device Support - NDS

EPICS device support module is responsible for integrating the card into EPICS and providing communication between the user-space API and EPICS database. It is realised with the help of NDS Framework [5]. Since NDS framework is focused on DAQ cards, bare NDS functionality had to be extended to support additional control options required by LLRF controller board.

3.3.1 Responsibilities

3.3.1.1 Pulse Type

The accelerator will have several possible pulse types/beam modes (not defined yet). Each of these pulse types could be different (length of the pulse, power, etc.) which is why each of them will require a separate SP and FF table. Distinction between different pulse types is one of the points where the SP and FF tables separate themselves from other control tables in Table 3.

It will be the responsibility of NDS layer to make sure that the pulse type is set up (meaning that the SP and FF table for the PT are loaded into memory), before allowing the user to select the PT.

3.3.1.2 Controller setup

The device support should provide access to all the settings that are needed by the controller, which can roughly be separated into 8 groups:

1. Non IQ sampling Setup
2. Vector Modulator setup
3. Modulator Ripple Filter setup
4. PI Controller Setup
5. Control Table setup
6. Data Acquisition Setup
7. Signal Monitoring Setup
8. Interlock and Trigger Setup

It will also perform some basic sanity checks on validity of those parameters.

3.3.1.3 PI Error RMS calculation and statistics display

Option to track the cumulative average for PI I and PI Q error since last time a controller setting was changed is also provided. Tracking also provides an option to ignore samples at the end of the pulse, or reset the RMS calculation on request.

3.3.1.4 Control Tables (FF and SP table)

Since the user space library writes the given SP or FF table to board memory without question, it will be in NDS responsibility to make sure that the data written is of correct size (as specified in TABLE_SIZE register) and format. It is not, however, responsible for the content of the tables.

During PULSE_ACTIVE state, the controller will then play out the tables. If the table does not extend through the whole interval between PULSE_COMING and PULSE_START for SP table, or between PULSE_START and PULSE_END for FF table, the controller will hold the last value in the table until the interval is finished [4].

3.3.2 Implementation

Since the generic Struck firmware is already supported in NDS [5] with a set of C++ Classes [6], the LLRF specific functionality is added by extending these Classes and adding new ones where necessary. All the classes and their additions with respect to generic EPICS module are described in this section.

A Device in NDS is modelled with Device, Channel Group and Channel Classes. The Struck SIS8300L LLRF Device has four channel groups with the following channels:

- Analog Input Channel Group (AI CG)
 - 10 Analog Input Channels (AI CH)
- Control Table Channel Group – SP tables (CT CG)
 - One Control Table Channel for each pulse type (CT CH)
 - One Control Table Channel for Special Operating Modes (CT SO)
- Control Table Channel Group – FF tables (CT CG)
 - One Control Table channel for each pulse type (CT CH)
 - One Control Table Channel for Special Operating Modes (CT SO)
- Controller Channel Group Class (CTRL CG)
 - 2 PI Channels (PI CH)
 - PI I Channel (PI I CH)
 - PI Q Channel (PI Q CH)
 - IQ Channel (IQ CH)
 - VM Channel (VM CH)
 - 1 Modulator Ripple Filter Channel (MR CH)
 - 4 Interlock Channels (ILOCK CH), one for every HARLINK input
- Signal Monitor Channel Group (SIGMON CG)
 - 10 Channels, each corresponding to an AI channel (SIGMON CH)

The physical AI channels map directly to AI Input channels, all other channels are virtual. The Control Table Channel group has two instances, one taking care of SP and the other of FF tables. Each Control table channel number maps directly to pulse type. The Controller Channel Group joins together all the parameters that are required to set up the custom part of the LLRF, except for Signal monitoring which is moved to its own CG. IQ, VM, MR and ILOCK CHs channels only hold parameter values, while both PI Channels act as data input channels that provide readout of the PI error for the previous pulse and setup of PI controller parameters.

The Device class is responsible for following the controller status and is in control of all Channel Group transitions. When the Device transitions to ON it starts waiting for PULSE_DONE interrupt and sends all Channel Groups into PROCESSING state. When the interrupt is received, the Device sends all the

Channel Groups into DISABLED state. When a Channel Group or Channel within the group leaves the processing state, it fetches data belonging to the pulse that just passed, and when it enters the disabled state it writes the new values to the controller.

During PROCESSING state CGs and CHs are accepting new values for controller parameters. The values are then taken into account with the next pulse, e.g. after the device is armed the next time. Each parameter or setting has a corresponding readback value, which gets updated when the parameter is actually written to the hardware. The readback thus provides information on the exact time the value was written to the controller.

If the setting is written to a shadow register (see [4] for the shadow register list), the controller takes the new value into account after an explicit call from software to update parameters. A call for update parameters happens before every arm of the board (if the parameters changed) and has its own corresponding readback which provides the exact time this was written to hardware. This allows one to track what parameters were used for each specific pulse.

3.3.3 Driver Initialization Parameters

In addition to standard parameters required by the `ndsCreateDevice` iocsh function [5], the driver requires two LLRF specific initialization parameters:

Parameter	Meaning
FILE	Specifies the Linux device node corresponding to the selected Struck SIS8300L board.
NUM_PULSE_TYPES	Number of pulse types that the device has to support. Tells the driver how many CT CHs to create in each CT CG.

Table 6: Driver initialization parameters

3.3.4 Exported interface

The interface exported by the NDS layer is a set of *asynReasons*, belonging to a Channel or a Channel Group. This chapter gives an overview of C++ Classes that are included in the EPICS module.

3.3.4.1 LLRF Device (sis8300llrfDevice Class)

The `sis8300llrfDevice` Class derives from `sis8300Device` Class [6] to provide LLRF specific functionality. The Class is responsible for card registration and CG management. It is also in control of the software state machine by implementing the NDS Device states defined in Table 8 and transitions between them (Table 9).

The lifecycle of the device starts with its creation at IOC initialization. After IOC initialization, the device is in OFF state and the card not yet registered with the user-space library. The Device automatically transitions to INIT state if its Enabled property is set, or waits for INIT message from the user. The condition for successful transition is that the `NUM_PULSE_TYPES` ≥ 0 , that the selected card (the device node via the FILE initialization parameter) is successfully opened and that the information about the device serial number and firmware version is read from the card successfully. Upon a

successful transition all the CGs are passed the device context (they in turn pass it on to their CHs) so that they are able to interact with the card. If the card registration fails, the Device goes into ERROR state.

When the Device enters INIT state, it first initializes the card. This includes:

- Setup the memory map
- Setup DAC
- Setup the clock source

It then calls initialize on all CGs (they in turn call initialize on all CHs), so that initial configuration can be read from device registers, and starts waiting for ON request from the user. At this point the user can configure the controller by specifying DAQ options, pulse type, various PI controller parameters, SP and FF tables.

When the device receives an ON request, it first checks if the selected pulse type has been setup correctly (SP and FF tables are set). If transition is successful the Device requests that all CGs write their data to the controller, sends an INIT DONE flag to the board and arms the controller. After this it sends CGs to PROCESSING state (They in turn send their CHs to PROCESSING state) and starts the pulse setup task which waits for PULSE_DONE or PMS interrupt from the board.

The pulse setup task is responsible for monitoring the controller state and controlling when CGs will go to PROCESSING or DISABLED. Unless an error occurs, or it is interrupted by the user the task will keep on repeating the following:

1. Wait for a software interrupt from the board
2. Receive interrupt,
 - a. If it was PMS go to ERROR state and stop the task
 - b. If it was PULSE_DONE go to 3
3. Send all CGs to Disabled state
 - a. When they leave PROCESSING, they will read the past pulse data from the controller
 - b. When they enter DISABLED state they will write new user settings to the controller and do callbacks for any settings that have changed – callbacks will update the setting readback values. The readback values will thus always reflect the current hardware settings
4. Check with CGs if any of the parameters have changed and determine the update reason for the board
5. Check if new pulse type was selected, if yes, check if the selected type is set up. If yes, write the new pulse id to the card
6. Send all CGs to PROCESSING state
7. Clear latched interrupts for PI overflow and VM magnitude limiter
8. Arm the board
9. Go to 1

Information that is included in the past pulse data are values that are expected to change on pulse-to-pulse basis. They are listed in Table 7.

Data	NDS Class
------	-----------

PI Error waveforms (2x)	PI CH
Calculated PI Error RMS (2x)	PI CH
PI Overflow Status (2x)	PI CH
ADC waveform data (10x)	AI CH
Number of samples acquired for PI err during ramp-up and active phase	CTRL CG
Total number of samples acquired during ramp-up plus active phase for cavity signal	CTRL CG
Total number of samples acquired during ramp-up plus active phase for PI error	CTRL CG
Vector Modulator magnitude limiter status	VM CH
ILOCK Status (4x)	ILOCK CH
Signal Monitor ILOCK, PMS and ALARM status (8x)	SIGMON CH
Maximum or minimum amplitude on a specific channel and current amplitude on that same channel	SIGMON CH

Table 7: Parameters that change on pulse to pulse basis and are read out after every PULSE_DONE interrupt from the device.

The sis8300llrf Device Class implements the following NDS Device states:

State	Description
OFF	The device file is not opened, and the controller cannot be accessed. In this state, the board can be replaced, hot-plugged or flashed with new firmware.
INIT	The device file is opened. All the groups have device context. The controller is IDLE. In this state it is possible to change clock settings.
ON	The controller is active and the control loop is running. Pulse setup task is running.
RESET	This is a transition state where a SW reset of custom logic is executed. After the reset the device can be put into INIT or OFF state.

ERROR	A PMS interrupt was received, or there was a problem in communication with the device.
-------	--

Table 8: Device NDS states

And transitions between them:

Source State	Destination State	Description
OFF	INIT	Try to open the device and read firmware version and serial number. Pass the device context to all the CGs and CHs and initialize the CGs and CHs.
OFF	ERROR	This transition occurs if the device cannot be opened or if the device is in PMS state when it is turned on.
INIT	ON	Check if the selected pulse type has FF and SP tables set. If yes, indicate INIT DONE to the device and arm it. Start the pulse setup task, that will wait for PMS or PULSE_DONE interrupt (see Table 4). This mode is also used for device setup.
ON	ERROR	The transition occurs when a PMS interrupt is received or if there is a problem in communication with the device. Stop waiting for PULSE_DONE and PMS interrupt.
Any except OFF	RESET	Issue a software reset of custom logic. Send all the LLRF Channel groups to RESET.
RESET	INIT	The device waits in RESET state (for clarity) and has to be manually moved out of it.
Any state except ON	OFF	When device transitions to OFF state file descriptor is released.

Table 9: Device state transitions

The Device Class implements the following parameters:

Asyn Reason	Asyn Interface	Description
-------------	----------------	-------------

State	asynInt32	See [5]
Command	asynOctetWrite	Supported messages are “ON”, “RESET”, “OFF” and “INIT”
Enabled	asynInt32	See [5]
Model	asynOctetRead	See [5]
Serial	asynOctetRead	See [5]
HardwareRevision	asynOctetRead	See [5]
FirmwareRevision	asynOctetRead	See [5]
SoftwareRevision	asynOctetRead	See [5]
OperatingMode	asynInt32	Used to select the operating mode of the controller.
ForceTrigger	asynInt32	To be used for special operating modes and during setup to force a specific FSM state or manual parameter update.
PulseType	asynInt32	Current Pulse Type. Max allowed value is defined at iocInit.
PulseDoneCount	asynInt32	Number of received pulse since the last INIT to ON transition
PulseMissed	asynInt32	Binary. Goes high if pulse count since last received user interrupt is bigger than one.
PMSAct	asynInt32	State of the PMS. Goes high if PMS interrupt was received from the board.
UpdateReason	asynInt32	Called whenever a request to the board is made to: <ul style="list-style-type: none"> • Init done = 0x1, • Take into account new parameters = 0x2 • New pulse type/update all = 0x4, • Take into account new FF table for the current pulse type 0x8 • Take into account new SP table for the current pulse type 0x10

		<ul style="list-style-type: none"> Do a software reset
Arm	asynInt32	Binary. Indicates when the device was armed from software.
PulseDone	asynInt32	Binary. Indicates when a PULSE_DONE interrupt was received from the device.
Status	asynInt32	Used mostly for tracking the controller state during development, has states ARMED, PULSE_DONE, CLEAR, PMS
SetupActive	asynInt32	Binary. Used to put the controller into setup mode.
SignalActive	asynInt32	Binary. Used to determine whether the controller is currently in active state (outputting a signal). Used in CW mode only.

Table 10: Device NDS properties

3.3.4.2 LLRF Base Channel Group (sis8300LlrfChannelGroup Class)

This Channel Group Class is a base Class for all LLRF specific channel groups. It implements or overrides the functionality of an NDS Channel Group Class. The class registers state transition handlers that correspond to LLRF controller states and provides functions for tracking parameter changes. The Channel Group is responsible for reporting if any changes were made on any of its channels. Tracking parameter changes is important to determine the update reason from the pulse setup task. The Class provides 4 virtual functions (see also [6]) that should be overridden by deriving classes:

- commitParameters*: write new parameter values associated with this CG to the controller.
- readParameters*: read all the current parameter values from hardware
- markAllParametersChanged*: Mark all the parameters this CG is responsible as changed. Call *markAllParametersChanged* on all channels. This will force a rewrite of all the parameters when a next call to *commitParameters* occurs.
- initialize*: Used for any initialization that requires access to the hardware and can thus not be done in IOC INIT phase. The default will also call initialize on all CHs. Default will call *markAllParametersChanged* and *commitParameters*.

The following state handlers are registered within the group:

State Handler	Description
ENTER PROCESSING	Set <i>updateReason</i> to 0

ENTER DISABLED	<p>The following actions are performed in the order they are listed:</p> <ol style="list-style-type: none"> 1. Check if CG is in IOC_INITIALIZATION state and return if it is. 2. Check if CG came from RESETTING state, send all CHs to DISABLED. 3. Call <i>commitParameters</i>.
ENTER RESET	<p>There are two state handles taking care of this transition. The following actions are performed in the order they are listed:</p> <ol style="list-style-type: none"> 1. Send all channels to RESETTING STATE. 2. Call <i>readParameters</i> to get the new values from hardware (after reset was executed). 3. Call <i>markAllParametersChanged</i> to force a rewrite of the values to hardware when returning to INIT state.

Table 11: LLRF Channel Group State handlers

The LLRF Channel Group Class implements the following configuration parameters, specified by NDS:

Asyn Reason	Asyn Interface	Description
State	asynInt32	See [5]
Enable	asynInt32	Overridden. CG cannot be disabled.
Command	asynOctetWrite, asynOctetRead	“START” and “STOP” messages will return an error, because state transitions are controlled by the Device Class, based on software interrupts, not the user.
ChannelDataReady	asynInt32	Signals when all data in channel have been updated.

Table 12: LLRF Channel Group NDS properties

3.3.4.3 LLRF Base Channel (sis8300llrfChannel Class)

This is a LLRF specific NDS ADIOChannel Class [5], from which all LLRF specific channels are derived. It provides commonly used functions and registers state handlers, relevant in LLRF operation. The core functions of this class are much the same as for LLRF CG Class and should be overridden by deriving Classes where necessary:

- *commitParameters*: If the CG is not in PROCESSING, than write new values for all the parameters that have changed to hardware and update the CG’s *updateReason* accordingly. Derived classes should override this function when necessary to write the parameter values corresponding to the specific channel

- *readParameters*: Read current parameter values from hardware
- *markAllParametersChanged*: Mark all parameters in the CH as changed. This will cause them to be recommitted to hardware.
- *initialize*: The function is intended for any type of initialization that requires access to hardware and can thus not be done before device enters the INIT state. Default function just calls *markAllParametersChanged* and *commitParameters*.

Table 13 gives a detailed description of state transitions, which are all hooked on the PULSE_DONE interrupt:

State handler	Description
ENTER DISABLED	Call <i>commitParameters</i> unless the CH is in IOC_INITIALIZATION state.
ENTER RESET	Call <i>readParameters</i> and <i>markAllParametersChanged</i> .

Table 13: LLRF Channel State handlers

The LLRF Channel Class implements the following configuration parameters specified by NDS:

Asyn Reason	Asyn Interface	Description
State	asynInt32	See [5]
Enabled	asynInt32	This property is read only. All channels used for LLRF specific data and settings are always enabled
Command	asynOctetWrite, asynOctetRead	“START” and “STOP” messages are overridden in this class, because state transitions are controlled by the Device Class and based on software interrupts, not the user.

Table 14: LLRF Channel NDS properties

3.3.4.4 LLRF Control Table Channel Group (sis8300llrfControlTableChannelGroup Class)

The LLRF Control Table Channel Group Class derives from LLRF Base Channel Group (sis8300llrfChannelGroup Class). A channel in this groups acts as a normal I/O channel. The channels are grouped into Control Table Channel Group Class based on the table type, which can be FF or SP. When the Control Table Channel Group initializes, it gets the maximum number of samples supported by the FW from device registers. At the time of CG creation, each group registers as many CHs as there are defined pulse types (specified by NUM_PULSE_TYPES parameter, Table 6).

In addition to inheriting NDS properties from Table 12, the class also implements the following additional properties:

Asyn Reason	Asyn Interface	Description
SamplesCount	asynInt32	Read only. Gives maximum allowed number of elements in a control table. Value is read from device registers when the device is turned on and is currently (consult [4] for up to date values) 0x01000 for SP tables 0x10000 for FF tables
MaxNelm	asynInt32	Maximum allowed elements in the FF or SP table. The value is obtained directly from hardware at transition to INIT state and does not change during operation.
FFTableSpeed	asynInt32	Feed forward table speed represents the number of clock cycles before next FF value is added to the PI input. In the interval [1,15] or every time a new PI sample is available The setting is only available for FF tables, using it for SP tables will return an error.

Table 15: Control Table Channel Group NDS properties

3.3.4.5 LLRF Control Table Channel (sis8300llrfControlTableChannel Class)

The Control Table Channel Class derives from LLRF Base Channel (sis8300llrfChannel Class). Each instance of Control Table Channel represents a pulse type corresponding to that channel number. The control table channel has two associated tables, I and Q.

Before the table is sent to hardware memory, both I and Q table are joined into a single table which is what actually gets written to the hardware. If I and Q table are not of the same size, the shorter table is filled up by the value of last element to get the same length for both tables. This is a viable solution, since the controller holds the last value until the end of pulse phase anyway [4].

The number of elements in the array is not directly settable. It gets set when the tables are joined and is the same as the number of elements in the larger table. The value of SamplesCount is used by the CG, to determine the size of the currently used Control Table and send it to the controller.

In addition to inheriting NDS properties from Table 14, the Class also defines the following new ones:

Asyn Reason	Asyn Interface	Description
-------------	----------------	-------------

SamplesCount	asynInt32	Read-only. Number of samples equals number of elements in the larger of AngleTable and MagnitudeTable.
ITable	asynFloat32Array	I part of the Control Table,
QTable	asynFloat32Array	Q part of the Control Table
RawTable	asynInt32Array	Raw table that contains 32 bit samples, containing both I and Q part. Basically the I and Q tables converted to a Signed(1,15) fixed point representation and interleaved, where table I is at offset 0.
FFTableMode	asynInt32	FF Table mode, can be hold last or circular. Circular is to be used with special operating modes (see [4]). The reason can only be used with FF table types.
WriteTable	asynInt32	Trigger reason to write specified tables to hardware.

Table 16: Control Table Channel NDS properties

3.3.4.6 LLRF Special Operation Control Table Channel (sis8300llrfControlTableChannelSpecOp Class)

This Class derives from LLRF Control Table Channel (sis8300llrfControlTableChannel Class) and provides settings for special operation modes (see [4]). It (ab)uses the I and Q table buffers from parent to be used as Magnitude and Angle tables in case of Magnitude or angle Controlled Signal Generator. In SP and FF CG, there is always one extra channel reserved at the end, which is used for special operating modes. In addition to properties defined in Table 16, the class defines the following ones:

Asyn Reason	Asyn Interface	Description
SamplesCount	asynInt32	Read-only. Number of samples equals number of elements in the larger of AngleTable and MagnitudeTable.
MagTable	asynFloat32Array	Magnitude part of the Control Table
AngleTable	asynFloat32Array	Angle part of the Control Table

Table 17: Control Table Special Operation Class NDS Properties

When using the controller in special operating modes, the mode can require either Magnitude and Angle or I and Q table. In the channel class itself (and also on the device) there are only two buffers, which

can contain either MA or IQ pair. The only difference between using the *MagTable* and *AngleTable* reasons from Table 17 and using the *ITable* and *QTable* reasons from Table 16 is the conversion of the double values to the hardware fixed point representation. Every time a new mode is used, both tables should be written down to avoid mixing up the two representations.

3.3.4.7 LLRF Controller Channel Group (sis8300llrfControllerChannelGroup Class)

The Controller Channel Group Class derives from LLRF Base Channel Group (sis8300llrfChannelGroup Class. It groups together all the channels that are responsible for monitoring and setup of the controller state.

In addition to inheriting NDS properties from Table 12, it also defines the following new ones:

Asyn Reason	Asyn Interface	Description
SamplesCntPIRampUp	asynInt32	Read-only. Number of PI errors sampled during ramp up phase (between PULSE-COMING and PULSE_START triggers)
SamplesCntPIActive	asynInt32	Read-only. Number of PI errors acquired during active phase (between PULSE_START and PULSE_END triggers)
SamplesCntPITotal	asynInt32	Read-only. Number of PI errors acquired during ramp up plus active phase (between PULSE_COMING and PULSE_END trigger).
SamplesCntADCTotal	asynInt32	Read-only. Number of ADC samples acquired per AI channel during ramp up and active phase (between PULSE_COMING and PULSE_END
OutputType	asynInt32	This is used to select either PI or FF driven output.
TriggerType	asynInt32	Selects which three backplane trigger lines to use for triggering, <ul style="list-style-type: none"> • MLVDS lines 0,1, 2 = 0 • MLVDS lines 4,5,6 = 1

Table 18: Controller Channel Group NDS properties

3.3.4.8 LLRF Non-IQ Sampling Channel (sis8300llrfIQSamplingChannel Class)

The sis8300llrfIQSamplingChannel Class extends the basic sis8300llrfChannel Class and defines the following asynReasons that represent IQ sampling settings:

Asyn Reason	Asyn Interface	Description
IQCavInpDelay	asynInt32	Cavity input delay. If enabled, sets number of clock cycles to delay cavity input as: Delay = value + 3, i.e. minimum delay is 3 CC. Used to align Cavity and Reference input at phase compensation. Limits are: [0,63]
IQCavInpDelayEn	asynInt32	Binary, used to enable or disable the Cavity input delay. Enable: 1, Disable: 0
IQAngleOffset	asynFloat64	IQ sampling angle offset. Used to compensate for different physical delays between cavity and reference signal. Used to adjust cavity input signal so that it is in phase with reference when a SP with 0 angle is used. Limits are: [- π , π]
IQAngleOffsetEn	asynInt32	Enable or disable the IQ angle Offset addition. Enable: 1, Disable: 0
NearIqParamM	asynInt32	Near IQ parameter M
NearIqParamN	asynInt32	Near IQ parameter N

Table 19: IQ Channel NDS properties

3.3.4.9 LLRF Vector Modulator Channel (sis8300llrfVMChannel Class)

The sis8300llrfVMChannel Class extends the basic sis8300llrfChannel Class and defines the following asynReasons that represent Vector Modulator Settings:

Asyn Reason	Asyn Interface	Description
MagnitudeLimitVal	asynFloat64	Set Magnitude limit value. Limits are: [-2 ¹⁵ , 2 ¹⁵ -2 ⁻¹⁶] → [0.0, 0.999984741211]
MagnitudeLimitEnable	asynInt32	Enable magnitude limiter. Limit value is MagnitudeLimitVal

		Enable = 1, Disable = 0
MagnitudeLimitStatus	asynInt32	VM Magnitude limiter status, Read-Only 1 = Active, 0 = Not active
InvertOutputI	asynInt32	Invert I output to compensate for Struck DAC inversion. Enable = 1, Disable = 0
InvertOutputQ	asynInt32	Invert Q output to compensate for Struc DAC inversion. Enable = 1, Disable = 0
SwapIqEn	asynInt32	Swap I and Q = 1, Do nothing = 0
PreDistEn	asynInt32	Pre-distort input to VM. Enable = 1, disable = 0
PreDistRC00	asynFloat64	Pre-distortion matrix, value RC00. Limits are: $[-2^1, 2^1 - 2^{-12}] \rightarrow [-2.0, 1.99975585938]$
PreDistRC01	asynFloat64	Pre-distortion matrix, value RC01. Limits are: $[-2^1, 2^1 - 2^{-12}] \rightarrow [-2.0, 1.99975585938]$
PreDistRC10	asynFloat64	Pre-distortion matrix, value RC10. Limits are: $[-2^1, 2^1 - 2^{-12}] \rightarrow [-2.0, 1.99975585938]$
PreDistRC11	asynFloat64	Pre-distortion matrix, value RC11. Limits are: $[-2^1, 2^1 - 2^{-12}] \rightarrow [-2.0, 1.99975585938]$
PreDistDCOI	asynFloat64	Pre-distortion DC offset for I part. Limits are $[-2^0, 2^0 - 2^{-15}] \rightarrow [-1.0, 1.99975585938]$
PreDistDCOQ	asynFloat64	Pre-distortion DC offset for Q part. Limits are $[-2^0, 2^0 - 2^{-15}] \rightarrow [-1.0, 1.99975585938]$

Table 20: VM Channel NDS properties

In addition to state transitions listed Table 13, this class defines the following state transitions:

State handler	Description
LEAVE PROCESSING	Read the Magnitude limit status

Table 21: VM Channel State Transition Handlers

3.3.4.10 LLRF Interlock Channel (sis8300llrfILOCKChannel Class)

The sis8300llrfILOCKChannel Class extends the basic sis8300llrfChannel Class and defines the following asynReasons that represent Interlock Channel Settings:

Asyn Reason	Asyn Interface	Description
getValueInt32	asynInt32	Harlink input status High = 1, Low = 0
ILOCKCond	asynInt32	Set Interlock Condition: <ul style="list-style-type: none"> • DISABLED = 0, • RISING EDGE = 1, • FALLING EDGE = 2, • HIGH LEVEL = 3, • LOW LEVEL = 4

Table 22: ILOCK Channel NDS Properties

In addition to state transitions listed Table 13, this class defines the following state transitions:

State handler	Description
LEAVE PROCESSING	Read the Harlink input Status

Table 23: ILOCK Channel State Transitions

3.3.4.11 LLRF PI Channel (sis8300llrfPChannel Class)

The LLRF PI channel Class derives from LLRF Base Channel (sis8300llrfChannel Class. In addition to inheriting NDS properties from Table 14, it also defines additional properties that represent settings or data for the I and Q PI controller.

Apart from reading and writing to hardware, this class also calculates RMS of the PI error waveform obtained after every pulse and does a cumulative average of the value. The average is calculated from last X pulses and gets reset whenever controller settings change. A maximum value of the RMS during these X pulses is also stored.

Asyn Reason	Asyn Interface	Description
PIGainK	asynFloat64	Set K gain for PI controller. Limits are: [-2 ⁸ , 2 ⁸ -2 ⁻²⁴] → [-128.0, 127.9999999404]
PIGainTsDivTi	asynFloat64	Set Ts/Ti gain for PI controller. Limits are: [-2 ⁸ , 2 ⁸ -2 ⁻²⁴] → [-128.0, 127.9999999404]
PISaturationMax	asynFloat64	Set Max Saturation for PI Controller. Limits are: [-2 ¹⁵ , 2 ¹⁵ -2 ⁻¹⁶] → [-32768.0, 32767.999984741211]
PISaturationMin	asynFloat64	Set Min Saturation for PI Controller. Limits are: [-2 ¹⁵ , 2 ¹⁵ -2 ⁻¹⁶] → [-32768.0, 32767.999984741211]
PIFixedFFVal	asynFloat64	Set fixed point FF value. Limits are: [-1, 1-2 ⁻¹⁵] → [-1.0, 0.999969482422]
PIFixedFFEnable	asynInt32	Use PIFixedFFVal instead of FF table Use fixed = 1, Use table = 0
PIFixedSPVal	asynFloat64	Set fixed point SP value. Limits are: [-1, 1-2 ⁻¹⁵] → [-1.0, 0.999969482422]
PIFixedSPEnable	asynInt32	Use PIFixedSPVal instead of SP table Use fixed = 1, Use table = 0
PIOverflowStatus	asynInt32	Overflow occurred = 1, No Overflow = 0
BufferFloat32	asynFloat32ArrayIn	Contains the PI error waveform from the last pulse.
RMSCurrent		RMS value of the PI error during ACTIVE phase (between PULSE_START and PULSE_END timing triggers, see Table 4), calculated from the data

		available through BufferFloat32
RMSSMNMIgnore	asynInt32	Number of samples to ignore at the end of every pulse when calculating the RMS
RMSAverage	asynFlot64	Cumulative average of RMS values for the last X pulses, where X can be obtained from RMSPulseCnt. The average is reset manually, or when any of the parameters on the device change (_UpdateReason != 0, see 3.3.4.1).
RMSMax	asynFloat64	Maximum RMS value in the last X pulses, where X can be obtained from RMSPulseCnt. The average is reset manually, or when any of the parameters on the device change (_UpdateReason != 0, see 3.3.4.1).
RMSPulseCnt	asynInt32	Number of pulses taken into account in the RMS average calculation.
RMSReset	asynInt32	Binary, used to manually reset the RMSAverage and RMSMax values and start fresh with the next pulse. RMSPulseCount will start again from 1.

Table 24: PI Channel NDS properties

In addition to state transitions listed Table 13, this class defines the following state transitions:

State handler	Description
LEAVE PROCESSING	<ul style="list-style-type: none"> • Read the PI Error waveform from the hardware • Calculate the RMS during the active phase • Calculate the new RMS cumulative average • Check if the new RMS is bigger than current RMS max value and store it if it is • Check the PI overflow status

Table 25: PI Channel State Transitions

3.3.4.12 LLRF Modulator Ripple Filter Channel (sis8300llrfModRippleFiltChannel Class)

The LLRF Modulator Ripple Filter Channel Class derives from LLRF Base Channel (sis8300llrfChannel Class. In addition to inheriting NDS properties from Table 14, it also defines additional properties that are specific to Modulator ripple filter settings:

Asyn Reason	Asyn Interface	Description
-------------	----------------	-------------

ModRippleFilConstS	asynFloat64	Modulator ripple filter constant S: [$-2^0, 2^0 - 2^{-31}$] \rightarrow [-1.0, 0.999969482422]
ModRippleFilConstC	asynFloat64	Modulator ripple filter constant C: [$-2^0, 2^0 - 2^{-31}$] \rightarrow [-1.0, 0.999969482422]
ModRippleFilConstA	asynFloat64	Modulator ripple filter constant A: [$0, 2^0 - 2^{-16}$] \rightarrow [-1.0, 0.999984741211]
ModRippleFilStartEvt	asynInt32	Modulator ripple filter start event defines the start of modulator ripple filter active period. Values can be: <ul style="list-style-type: none"> • PULSE_COMMING = 0, • PULSE_START = 1
ModRippleFilStopEvt	asynInt32	Modulator ripple filter stop event defines the end of modulator ripple filter active period. Values can be: <ul style="list-style-type: none"> • PULSE_START = 1, • PULSE_END = 2
ModRippleFilQEn	asynInt32	Binary, enable modulator ripple filter for Q part. Values can be: Enable = 1, Disable = 0
ModRippleFilIEn	asynInt32	Binary, enable modulator ripple filter for I part. Values can be: Enable = 1, Disable = 0

Table 26: Modulator Ripple Filter Channel NDS Properties

3.3.4.13 LLRF Notch Filter Channel (sis8300llrfModRippleFiltChannel Class)

The LLRF Notch Filter Channel Class derives from LLRF Base Channel (sis8300llrfChannel Class). In addition to inheriting NDS properties from Table 14, it also defines additional properties that are specific to Notch filter settings:

Asyn Reason	Asyn Interface	Description
NotchFilConstAReal	asynFloat64	Notch filter constant A real part: $[-2^0, 2^0 - 2^{-31}] \rightarrow [-1.0, 0.999969482422]$
NotchFilConstAImag	asynFloat64	Notch filter constant A imaginary part: $[-2^0, 2^0 - 2^{-31}] \rightarrow [-1.0, 0.999969482422]$
NotchFilConstBReal	asynFloat64	Notch filter constant B real part: $[-2^0, 2^0 - 2^{-31}] \rightarrow [-1.0, 0.999969482422]$
NotchFilConstBImag	asynFloat64	Notch filter constant A imaginary part: $[-2^0, 2^0 - 2^{-31}] \rightarrow [-1.0, 0.999969482422]$
NotchFilEn	asynInt32	Binary, enable notch filter. Values can be: Enable = 1, Disable = 0

Table 27: Modulator Ripple Filter Channel NDS Properties

3.3.4.14 LLRF Signal Monitor Channel (sis8300llrfSignalMonitorChannel Class)

The LLRF Signal Monitor Channel Class derives from LLRF Base Channel (sis8300llrfChannel Class). In addition to inheriting NDS properties from Table 14, it also defines additional properties that are specific to Signal Monitor settings:

Asyn Reason	Asyn Interface	Description
MagTreshold	asynFloat64	Magnitude threshold determines when an alarm is raised on this channel. It is used together with MonitorAlarmCnd. Limits are: $[0, 2^0 - 2^{-15}] \rightarrow [0.0, 0.999984741211]$
MonitorAlarmCnd	asynInt32	Alarm condition. Alarm is raised when the ADC signal goes: Over Treshold=0, Below Treshold=1 Where the threshold is pecified with MagTreshold.
MonitorStartEvtnt	asynInt32	This event defines the start of monitor active

		<p>period, it can be:</p> <ul style="list-style-type: none"> • PULSE_COMMING=0, • PULSE_START=1, • PULSE_END=2, • NEVER=3 <p>And has to be before MonitorStopEvnt, which defines the end of monitor active period.</p>
MonitorStopEvnt	asynInt32	<p>This event defines the end of signal monitor active period, it can be:</p> <ul style="list-style-type: none"> • PULSE_START=1, • PULSE_END=2, • PULSE DONE=3 <p>And has to be after MonitorStartEvnt which defines the start of monitor active period.</p>
MonitorPMSEn	asynInt32	<p>Trigger PMS if Alarm is raised</p> <p>Disabled=0, Enabled=1</p>
MonitorILOCKEn	asynInt32	<p>Trigger ILOCK if Alarm is raised</p> <p>Disabled=0, Enabled=1</p> <p>(see also APPENDIX: Current Development System)</p>
SignalTypeDC	asynInt32	<p>Set signal type, it can be:</p> <p>AC=0, DC=1</p>
MagCurrent	asynFloat64	<p>Current magnitude value on the corresponding ADC channel</p>
MagMinMax	asynFloat64	<p>Minimum or maximum magnitude value during the last monitor active period (defined with MonitorStartEvnt and MonitorStopEvnt). If MonitorAlarmCnd is set to trigger below threshold, this will return the mainimum magnitude, if it is set to trigger above threshold, it will return the maximum magnitude.</p>
SigmonAlarm	asynInt32	<p>Binary, shows the status of alarm on this signal monitor CH. Values are:</p>

		<p>Alarm active = 1, Not active = 0</p> <p>The alarm will be raised if the signal goes below or over magnitude threshold (depending on the choice of MonitorAlarmCnd) during the signal monitor active period (defined with MonitorStartEvnt and MonitorStopEvnt).</p>
SigmonPMS	asynInt32	<p>Binary, shows the status of PMS for this signal monitor CH. Values are:</p> <p>1 = PMS active, 0 = not active</p> <p>The PMS is raised if alarm is raised and if PMS triggering is enabled for the CH (with MonitorPMSEn).</p>
SigmonILOCK	asynInt32	<p>Binary, shows the status of interlock on this signal monitor CH. Values are:</p> <p>Interlock active = 1, not active = 0</p> <p>Interlock becomes active when alarm is raised and if interlock is enabled for the CH (with MonitorILOCKEn).</p>

Table 28: NDS Signal Monitor Channel Properties

In addition to state transitions listed Table 13, this class defines the following state transitions:

State handler	Description
LEAVE PROCESSING	<ul style="list-style-type: none"> • Read alarm status for the channel • Read PMS status for the channel • Read interlock status for the channel • Read maximum/minimum amplitude for the last pulse • Read current magnitude value

Table 29: Singal Monitor Channel State Transitions

3.3.4.15 LLRF Analog Input Channel Group (sis8300llrfAIChannelGroup Class)

The Analog Input Channel Group Class derives from generic sis8300 AI CG Class [6]. It overrides *asynReasons* that are not supported in the LLRF specific implementation. In this derived Class, the responsibility of the AI CG for triggering the acquisition is removed, since this is in the domain of the Device Class. It does not add any new state transitions and overrides two from the parent class (see [6]), their actions are defined in Table 30:

State handler	Description
ENTER PROCESSING	Override parent to do nothing
LEAVE PROCESSING	Override parent to do nothing
ENTER DISABLED	Keep parent's handler, that calls <i>commitParameters</i>

Table 30: NDS LLRF AI CG state transitions

This Class implements the following parameters:

Asyn Reason	Asyn Interface	Description
State	asynInt32	See [5]
SamplesCount	asynInt32	Number of samples to acquire. This only affects the number of ADC samples that will get stored into memory.
ClockSource	asynInt32	Overrides parent to prevent the changing the clock settings when the loop is running (Device is in ON state).
ClockFrequency	asynInt32	Overrides parent to prevent the changing the clock settings when the loop is running (Device is in ON state).
ClockDivider	asynInt32	Overrides parent to prevent the changing the clock settings when the loop is running (Device is in ON state).

Table 31: AI Channel Group NDS properties. Clock setting are meant to be used during development and cannot be changed while the controller is running = while device is in ON state.

Parameters not listed in Table 31 are unsupported or overridden. They are:

Asyn Reason	Asyn Interface	Reason for override
Command	asynOctetWrite	“START” and “STOP” messages are overridden in this class, because state transitions are controlled by the Device Class, based on software interrupts, not

		the user.
TriggerRepeat	asynInt32	Is used by the parent class for automatic rearm. In LLRF implementation, Device Class is responsible for arming the board.
TriggerDelay		Not supported
TriggerCondition	asynInt32	Not Supported in the same way. Generic Struck Trigger setup has no meaning.
Enable	asynInt32	CG cannot be disabled

Table 32: AI Channel Group overridden NDS properties

3.3.4.16 LLRF Analog Input Channel (sis8300llrfAIChannel Class)

The Analog Input Channel Class derives from generic sis8300 AI CH Class [6] for usage with channels AI0 (Cavity input) and AI1 (Reference input). It overrides *asynReasons* not supported by the LLRF specific implementation. It does not add any new state transitions and overrides two from the parent class (see [6]), their actions are defined in Table 30:

State handler	Description
ENTER PROCESSING	Keep parent
LEAVE PROCESSING	Extend parent to add signal magnitude and angle read
ENTER DISABLED	Keep parent's handler, that calls commitParameters

This Class implements the following parameters:

Asyn Reason	Asyn Interface	Description
State	asynInt32	See [5]
Enable	asynFloat64	Overridden, so that disabling of AI0 (cavity input) and AI1 (reference input) is not allowed.
SignalAngle	asynFloat64	Current signal Angle, should always be read together with SignalMagnitude after a new MA point is

		available (see NewMAPoint)
SignalMagnitude	asynFloat64	Current Signal Magnitude, should always be read together with Angle after a new MA point is available (see NewMAPoint)
SignalI	asynFloat64	Current Signal I value. Is calculated together with Q value when a new MA point is received from the device. It should always be read together with Q when a new MA point is available (see NewMAPoint)
SignalQ	asynFloat64	Current Signal Q value. Is calculated together with Q value when a new MA point is received from the device. It should always be read together with I when a new MA point is available (see NewMAPoint)
NewMAPoint	asynInt32	<p>Writing to this will force read of a MA point from the device. The record will get processed when a new MA and corresponding IQ point is available.</p> <p>When using the MA and IQ values, one should only tread out the pairs when this is processed, because the data is correlated.</p>

Table 33: AI Channel NDS properties

3.4 EPICS Database

EPICS database will be responsible for communication with the user. Records will be provided for configuration of all the LLRF board functional blocks and HW status update.

3.4.1 Exported interface

The interface exported by this block is a set of *EPICS process variables* that can be accessed through the CA. The templates are separated into several groups and have the following prefixes:

- *sis8300llrf-Main* prefix includes all the templates required for normal operation of the device,
- *sis8300llrf-RMSStatistics* includes extra records for resetting RMS statistics from the database,
- *sis8300llrf-Register* includes a list of LLRF-specific registers and allows one to read/write raw values from/to them,
- *sis8300llrf-Setup* includes records required for the setup procedure,
- *sis3800llrf-SpecOp* includes all the records needed to use the device in special operating modes

3.4.1.1 sis8300llrf-Main-Device.template

This template adds functionality to the generic sis8300Device.template [6]. In order to successfully load the template, the generic one must be loaded first. The added functionality is the following:

Name	Type	Description
\$(PREFIX)	mbbi	Adds RESETTING to the list of generic sis8300Device states
\$(PREFIX):PT \$(PREFIX):PT-RBV	longout, longin	Pulse Type.
\$(PREFIX):PMS	bi	PMS status, 1 if active, 0 if not
\$(PREFIX):ARM	bi	Used to track when the device was armed from software.
\$(PREFIX):PULSE_DONE	bi	Used to track when PULSE_DONE interrupts are received from the device.
\$(PREFIX):UPDATE_REASON	bi	Tracks calls to update parameters, that can: <ul style="list-style-type: none"> • Make shadow registers visible to the controller • Force the controller to load new SP/FF tables • Inform the controller of a new pulse type • Init done
\$(PREFIX):PULSEDONECNT	longin	Number of received PULSE_DONE interrupts since last transition from INIT to ON
\$(PREFIX):PULSEMISSED	bi	Pulse missed indicator. It will go high if the number of pulses we read out from the device between two arms != 1.
\$(PREFIX):STATUS	mbbi	Tracks controller status. Can be NONE, PMS, ARMED, PULSE_DONE and is mostly used for development purposes.
\$(PREFIX):RTM	mbbo	Adds the PINI option and default setting to the parent's record.

Table 34: sis8300llrfDevice.template records

The following macros must be defined when loading the template:

Macro	Description
PREFIX	Name prefix
ASYN_PORT	Asyn Port Name
PULSE_TYPE	Default Pulse Type
RTM	RTM type to select by default, can be: <ul style="list-style-type: none"> • SIS8900 = 0, • DWC8VM1 = 1, • DS8VM1 = 2, • NONE = 3

Table 35: sis8300llrfDevice.template macros

3.4.1.2 sis8300llrf-Main-ControlTable-CG.template

This template defines the database with records used to control and monitor the CT CG parameters

Name	Type	Description
\$(PREFIX):\$(CTRL_TABLE_TYPE)-STAT	mbbi	State of the channel group, see [5].
\$(PREFIX):\$(CTRL_TABLE_TYPE)-MAXNSAMPLES	longin	Maximum number of elements in a control table. Read only – information is obtained directly from the device.

Table 36: sis8300llrfControlTableChannelGroup.template records

The following macros must be defined to successfully load the sis8300llrfControlTableChannelGroup.template:

Macro	Description
PREFIX	Name prefix
ASYN_PORT	Asyn Port Name
CTRL_TABLE_TYPE	Either SP or FF

ASYN_ADDR	3 for SP, 4 for FF
-----------	--------------------

Table 37: sis8300llrfControlTableChannelGroup.template macros

3.4.1.3 sis8300llrf-Main-FFTable-CG.template

This template adds two FF specific records to the 3.4.1.2 template.

Name	Type	Description
\$(PREFIX):\$(CTRL_TABLE_TYPE)-TABLESPEED	mbbo,	Speed of the FF table, see 3.3.4.4
\$(PREFIX):\$(CTRL_TABLE_TYPE)-TABLESPEED-RBV	mbbi	

Table 38: sis8300llrfFFTableChannelGroup.template records

The following macros must be defined to successfully load the sis8300llrfFFTableChannelGroup.template:

Macro	Description
PREFIX	Name prefix
ASYN_PORT	Asyn Port Name
CTRL_TABLE_TYPE	FF
ASYN_ADDR	4

Table 39: sis8300llrfFFTableChannelGroup.template

3.4.1.4 sis8300llrf-Main-ControlTable-CH.template

This template defines the database with records used to control and monitor the CT CH parameters

Name	Type	Description
\$(PREFIX):\$(CTRL_TABLE_TYPE)-\$(CHAN_NAME=PT\$PULSE_TYPE))-STAT	mbbi	State of the channel group, see [5].

\$(PREFIX):\$(CTRL_TABLE_TYPE)-\$(CHAN_NAME=PT\$PULSE_TYPE))-I \$(PREFIX):\$(CTRL_TABLE_TYPE)-\$(CHAN_NAME=PT\$PULSE_TYPE))-I-GET	waveform	I table, the –GET record has to be manually processed and will read the table from hardware, convert it from IQ sample to a float I sample.
\$(PREFIX):\$(CTRL_TABLE_TYPE)-\$(CHAN_NAME=PT\$PULSE_TYPE))-Q \$(PREFIX):\$(CTRL_TABLE_TYPE)-\$(CHAN_NAME=PT\$PULSE_TYPE))-Q-GET	waveform	Q table, the –GET record has to be manually processed and will read the table from hardware, convert it from IQ sample to a float Q sample
\$(PREFIX):\$(CTRL_TABLE_TYPE)-\$(CHAN_NAME=PT\$(PULSE_TYPE))-WRTBL	bo	Write Table, process this record to write specified I and Q tables to hardware.
\$(PREFIX):\$(CTRL_TABLE_TYPE)-\$(CHAN_NAME=PT\$PULSE_TYPE))-SMNM-RBV	longin	Number of elements in the table that is actually written to hardware.
\$(PREFIX):\$(CTRL_TABLE_TYPE)-\$(CHAN_NAME=PT\$PULSE_TYPE))-RAWTABLE-GET	waveform	Raw table that is currently in the device memory (containing IQ samples). Record must be manually processed and will fetch the data from the device memory every time it is.

Table 40: sis8300llrfControlTableChannel.template records

The following macros must be defined to successfully load the sis8300llrfControlTableChannel.template

Macro	Description
PREFIX	Name prefix
ASYN_PORT	Asyn Port Name
CTRL_TABLE_TYPE	Either SP of FF
CTRL_TABLE_CG_NAME	Either sp or ff
PULSE_TYPE	The pulse this channel belongs to

CTRL_TABLE_MAX_NSAMPLES	Maximum number of elements in a control table
-------------------------	---

Table 41: sis8300llrfControlTableChannel.template macros

3.4.1.5 sis8300llrf-Main-Controller-CG.template

This template defines the database with records used to control and monitor the CTRL CG parameters. In addition to standard definitions, this template also provides control for PI error RMS statistics.

Name	Type	Description
\$(PREFIX):LLRFCTRL-STAT	mbbi	State of the channel group, see [5].
\$(PREFIX):TRGSETUP \$(PREFIX):TRGSETUP-RBV	mbbo, mbbi	Trigger setup, can be <ul style="list-style-type: none"> • MLVDS-012 = 0 • MLVDS-456 = 1
\$(PREFIX):PIERR-SMNM-TOTAL	longin	Total number of PI err samples, acquired during RAMP UP + ACTIVE phase.
\$(PREFIX):PIERR-SMNM-RAMPUP	longin	Number of PI error samples acquired during RAMP UP phase.
\$(PREFIX):PIERR-SMNM-ACTIVE	longin	Number of PI error samples acquired during ACTIVE phase.
\$(PREFIX):ADC-SMNM-TOTAL	longin	Number of ADC samples acquired during RAMP UP + ACTIVE phase.
\$(PREFIX):OUTPUT-DRIVESEL \$(PREFIX):OUTPUT-DRIVESEL-RBV	bo, bi	Select the source that will drive the output: <ul style="list-style-type: none"> • PI Driven (normal operation) = 0 • FF Driven = 1
\$(PREFIX):CHDATARDY	bi	Processes when channel data is ready, Its forward link can for example be used to trigger arbitrary functionality that sets new parameters on the device. If the chain of processing (database link from this record to the parameter record) is unbroken, the new

		parameter(s) are written to hardware before the board is armed again.
--	--	---

Table 42: sis8300llrfControllerChannelGroup.template records

The following macros must be defined to successfully load the sis8300llrfControllerChannelGroup.template

Macro	Description
PREFIX	Name prefix
ASYN_PORT	Asyn Port Name
PI_ERR_MAX_NSAMPLES	Maximum number of PI error samples – used as DRVH limit
PI_ERR_SNM	Default number of PI error samples
TRG_VAL	Default trigger setup (optional, default value is 0)
OUTPUT_DRIVE	Select the default source that will drive the output (optional, default value is 0)

Table 43: sis8300llrfControllerChannelGroup.template macros

3.4.1.6 sis8300llrf-Main-IQSmpl-CH.template

This template defines the database with records used to control and monitor the IQ CH parameters.

Name	Type	Description
\$(PREFIX):IQSMPL-STAT	mbbi	Channel status, see [5]
\$(PREFIX): IQSMPL-NEARIQM \$(PREFIX) IQSMPL-NEARIQM -RBV	ao, ai	Near IQ parameter M
\$(PREFIX): IQSMPL-NEARIQN \$(PREFIX): IQSMPL-NEARIQN -RBV	ao, ai	Near IQ parameter N
\$(PREFIX):IQSMPL-CAVINDELAYVAL \$(PREFIX):IQSMPL-CAVINDELAYVAL-	longout ,	Cavity input delay

RBV	longin	
\$(PREFIX):IQSMPL-CAVINDELAYEN \$(PREFIX):IQSMPL-CAVINDELAYEN-RBV	bo, bi	Cavity input delay enable
\$(PREFIX):IQSMPL-ANGOFFSETVAL \$(PREFIX):IQSMPL-ANGOFFSETVAL-RBV	ao, ai	IQ angle offset
\$(PREFIX):IQSMPL-ANGOFFSETEN \$(PREFIX):IQSMPL-ANGOFFSETEN-RBV	bo, bi	IQ angle offset enable

Table 44: sis8300llrfIQSamplingChannel.template records

The following macros must be defined to successfully load the sis8300llrfIOIQChannel.template

Macro	Description
PREFIX	Name prefix
ASYN_PORT	Asyn Port Name
ASYN_ADDR	2
IQ_ANG_DRVH, IQ_ANG_DRVL	High and low limit for the value of IQ angle
IQ_CAV_INP_DELAY_DRVH, IQ_CAV_INP_DELAY_DRVL	High and low limit for the value of cavity input delay

Table 45: sis8300llrfIQSamplingChannel.template macros

3.4.1.7 sis8300llrf-Main-VMCtrl-CH.template

This template defines the database with records used to control and monitor the VM CH parameters.

Name	Type	Description
\$(PREFIX):VM-STAT	mbbi	Channel Status, see [5]

\$(PREFIX):VM-MAGLIMEN \$(PREFIX): VM-MAGLIMEN-RBV	bo, bi	Enable/disable magnitude limiter
\$(PREFIX): VM-MAGLIMVAL \$(PREFIX): VM- MAGLIMVAL-RBV	ao, ai	Magnitude limiter value
\$(PREFIX): VM-MAGLIMSTAT	bi	Magnitude limit status, None=0, Active=1
\$(PREFIX): VM-INVIEN \$(PREFIX): VM-INVIEN-RBV	bo, bi	Enable inverse I output
\$(PREFIX): VM-INVQEN \$(PREFIX): VM-INVQEN-RBV	bo, bi	Enable inverse Q output
\$(PREFIX): VM-SWAPIQEN \$(PREFIX) VM-SWAPIQEN-RBV	bo, bi	Swap IQ. No = 0, Yes = 1
\$(PREFIX):VM-PREDISTEN \$(PREFIX):VM-PREDISTEN-RBV	bo, bi	Enable pre-distortion of the input to VM
\$(PREFIX):VM-PREDIST-RC00 \$(PREFIX):VM-PREDIST-RC00-RBV	ao, ai	VM pre-distortion matrix value for RC00
\$(PREFIX):VM-PREDIST-RC01 \$(PREFIX):VM-PREDIST-RC01-RBV	ao, ai	VM pre-distortion matrix value for RC01
\$(PREFIX):VM-PREDIST-RC10 \$(PREFIX):VM-PREDIST-RC10-RBV	ao, ai	VM pre-distortion matrix value for RC10
\$(PREFIX):VM-PREDIST-RC11 \$(PREFIX):VM-PREDIST-RC11-RBV	ao, ai	VM pre-distortion matrix value for RC11
\$(PREFIX):VM-PREDIST-DCOI	ao,	Pre-distortion DC offset for I component

\$(PREFIX):VM-PREDIST-DCOI-RBV	ai	
\$(PREFIX):VM-PREDIST-DCOQ	ao,	Pre-distortion DC offset for Q
\$(PREFIX):VM-PREDIST-DCOQ-RBV	ai	component

Table 46: sis8300llrfVMControlChannel.template records

The following macros must be defined to successfully load the sis8300llrfIOVMChannel.template

Macro	Description			
PREFIX	Name prefix			
ASYN_PORT	Asyn Port Name			
ASYN_ADDR	3			
MAGLIM_DRVH, MAGLIM_DRVL	Highest and lowest magnitude limit value accepted by hardware (see			
	PreDistRC10	asynFloat64	Pre-distortion matrix, value R	
			$[-2^1, 2^1 - 2^{-12}] \rightarrow [-2.0, 1.9997]$	
	PreDistRC11	asynFloat64	Pre-distortion matrix, value R	
			$[-2^1, 2^1 - 2^{-12}] \rightarrow [-2.0, 1.9997]$	
	PreDistDCOI	asynFloat64	Pre-distortion DC offset for I	
			$[-2^0, 2^0 - 2^{-15}] \rightarrow [-1.0, 1.9997]$	
	PreDistDCOQ	asynFloat64	Pre-distortion DC offset for Q	
			$[-2^0, 2^0 - 2^{-15}] \rightarrow [-1.0, 1.9997]$	
Table 20)				
PREDISTRRC_DRVH, PREDISTRRC_DRVL	Highest and lowest value for pre-distortion matrix element accepted by hardware (see			
	PreDistRC10	asynFloat64	Pre-distortion matrix, value R	

			$[-2^1, 2^1 - 2^{-12}] \rightarrow [-2.0, 1.9997]$
	PreDistRC11	asynFloat64	Pre-distortion matrix, value R $[-2^1, 2^1 - 2^{-12}] \rightarrow [-2.0, 1.9997]$
	PreDistDCOI	asynFloat64	Pre-distortion DC offset for I $[-2^0, 2^0 - 2^{-15}] \rightarrow [-1.0, 1.9997]$
	PreDistDCOQ	asynFloat64	Pre-distortion DC offset for Q $[-2^0, 2^0 - 2^{-15}] \rightarrow [-1.0, 1.9997]$
Table 20)			
PREDISTDC_DRVH, PREDISTDC_DRVL	Highest and lowest value for DC offset accepted by hardware (see		
	PreDistRC10	asynFloat64	Pre-distortion matrix, value R $[-2^1, 2^1 - 2^{-12}] \rightarrow [-2.0, 1.9997]$
	PreDistRC11	asynFloat64	Pre-distortion matrix, value R $[-2^1, 2^1 - 2^{-12}] \rightarrow [-2.0, 1.9997]$
	PreDistDCOI	asynFloat64	Pre-distortion DC offset for I $[-2^0, 2^0 - 2^{-15}] \rightarrow [-1.0, 1.9997]$
	PreDistDCOQ	asynFloat64	Pre-distortion DC offset for Q $[-2^0, 2^0 - 2^{-15}] \rightarrow [-1.0, 1.9997]$
Table 20)			
PREDISTORT_VM_OUT_EN	Set to 1 to enable pre-distortion by default		
INVERT_Q	Set to 1 to enable inversion by default		
INVERT_I	Set to 1 to enable the inversion by default		

Table 47: sis8300llrfVMControlChannel.template macros

3.4.1.8 sis8300llrf-Main-ILOCK-CH.template

This template defines the database with records used to control and monitor the Interlock CH parameters.

Name	Type	Description
\$(PREFIX):\$(ILOCK_CH)-STAT	mbbi	Channel Status, see [5]
\$(PREFIX):\$(ILOCK_CH)-HARINP	bi	Current HARlink input status, Low=0, High=1
\$(PREFIX):\$(ILOCK_CH)-CONDITION \$(PREFIX):\$(ILOCK_CH)-CONDITION-RBV	mbbo, mbbi	Interlock condition select, can be: <ul style="list-style-type: none"> • DISABLED = 0 • RISING_EDGE = 1 • FALLING_EDGE = 2 • HIGH_LEVEL = 3 • LOW_LEVEL = 4

Table 48: sis8300llrfILOCKChannel.template records

The following macros must be defined to successfully load the sis8300llrfPIChannel.template

Macro	Description
PREFIX	Name prefix
ASYN_PORT	Asyn Port Name
ILOCK_CH	ILOCK0, ILOCK1, ILOCK2, ILOCK3
ASYN_ADDR	<ul style="list-style-type: none"> • ILOCK0 = 5, • ILOCK1 = 6, • ILOCK2 = 7, • ILOCK3 = 8 (see ILOCK_CH macro)

Table 49: sis8300llrfILOCKChannel.template macros

3.4.1.9 sis8300llrf-Main-PI-CH.template

This template defines the database with records used to control and monitor the PI CH parameters. In addition it defines some records required for PI err RMS statistics.

Name	Type	Description
\$(PREFIX):\$(PI_TYPE)-STAT	mbbi	Channel Status, see [5]
\$(PREFIX):\$(PI_TYPE)-OVRFLW	bi	Overflow status, None=0, overflow=1
\$(PREFIX):\$(PI_TYPE)-FIXEDSPVAL \$(PREFIX):\$(PI_TYPE)-FIXEDSPVAL-RBV	ao, ai	Fixed SP value
\$(PREFIX):\$(PI_TYPE)-FIXEDSPEN \$(PREFIX):\$(PI_TYPE)-FIXEDSPEN-RBV	bo, bi	Enable/disable fixed SP
\$(PREFIX):\$(PI_TYPE)-FIXEDFFVAL \$(PREFIX):\$(PI_TYPE)-FIXEDFFVAL-RBV	ao, ai	Fixed FF value
\$(PREFIX):\$(PI_TYPE)-FIXEDFFEN \$(PREFIX):\$(PI_TYPE)-FIXEDFFEN-RBV	bo, bi	Enable/disable fixed FF
\$(PREFIX):\$(PI_TYPE)-GAINK \$(PREFIX):\$(PI_TYPE)-GAINK-RBV	ao, ai	PI gain K value
\$(PREFIX):\$(PI_TYPE)-GAINTSDIVTI \$(PREFIX):\$(PI_TYPE)-GAINTSDIVTI-RBV	ao, ai	PI gain Ts/Ti value
\$(PREFIX):\$(PI_TYPE)-SATMAX \$(PREFIX):\$(PI_TYPE)-SATMAX-RBV	ao, ai	Maximum saturation value
\$(PREFIX):\$(PI_TYPE)-SATMIN \$(PREFIX):\$(PI_TYPE)-SATMIN-RBV	ao, ai	Minimum saturation value
\$(PREFIX):\$(PI_TYPE)-ERR	waveform	PI error
\$(PREFIX):\$(PI_TYPE)-ERR-SNM-RBV	longin	Number of PI errors read
\$(PREFIX):\$(PI_TYPE)-RMS	ai	Current PI error RMS value

\$(PREFIX):\$(PI_TYPE)-RMS-AVERAGE	ai	Cumulative average PI error RMS value in the last RMS-PULSECNT pulses
\$(PREFIX):\$(PI_TYPE)-RMS-MAX	ai	Maximum PI error RMS value in the last RMS-PULSECNT pulses
\$(PREFIX):\$(PI_TYPE)-RMS-PULSECNT	longin	Number of pulses over which RMS statistics was tracked
\$(PREFIX):\$(PI_TYPE):RMS-SNMIGNORE \$(PREFIX):\$(PI_TYPE):RMS-SNMIGNORE-RBV	longout, longin	Number of samples to ignore at the end of the pulse when calculating the RMS
\$(PREFIX):\$(PI_TYPE)-RMS-RESET \$(PREFIX):\$(PI_TYPE)-RMS-RESET-RBV	bo, bi	Reset RMS statistics

Table 50: sis8300llrfPIChannel.template

The following macros must be defined to successfully load the sis8300llrfPIChannel.template

Macro	Description
PREFIX	Name prefix
ASYN_PORT	Asyn Port Name
PI_TYPE	PI-I or PI-Q
ASYN_ADDR	0 for PI-I, 1 for PI-Q
PI_ERR_MAX_NSAMPLES	Maximum number of PI error samples
FIXEDSP_DRVH, FIXEDSP_DRVL	Highest and lowest value for fixed SP value accepted by hardware (see Table 24)
FIXEDFF_DRVH, FIXEDFF_DRVL	Highest and lowest value for fixed FF value accepted by hardware (see Table 24)
GAINK_DRVH,	Highest and lowest value for K Gain value accepted by

GAINK_DRVL	hardware (see Table 24)
GAINTSDIVTI_DRVH, GAINTSDIVTI_DRVL	Highest and lowest value for Ts/Ti Gain value accepted by hardware (see Table 24)
GAIN_PREC	Number of decimal points for Ts/Ti Gain value accepted by hardware (see Table 24)
SATMAX_DRVH, SATMAX_DRVL	Highest and lowest value for Maximum saturation value accepted by hardware (see Table 24)
SATMIN_DRVH, SATMIN_DRVL	Highest and lowest value for Minimum saturation value accepted by hardware (see Table 24)

Table 51: sis8300llrfPIChannel.template macros

3.4.1.10 sis8300llrf-Main-ModRippleFilt-CH.template

This template defines the database with records used to control and monitor the Modulator ripple filter parameters.

Name	Type	Description
\$(PREFIX): MODRIPPFIL-STAT	mbbi	Channel Status, see [5]
\$(PREFIX): MODRIPPFIL-CONSTS \$(PREFIX): MODRIPPFIL-CONSTS-RBV	ao, ai	Modulator Ripple Filter Constant S
\$(PREFIX): MODRIPPFIL-CONSTC \$(PREFIX): MODRIPPFIL-CONSTC-RBV	ao, ai	Modulator Ripple Filter Constant C
\$(PREFIX): MODRIPPFIL-CONSTA \$(PREFIX): MODRIPPFIL-CONSTA-RBV	ao, ai	Modulator Ripple Filter Constant A
\$(PREFIX): MODRIPPFIL-STARTEVNT \$(PREFIX):MODRIPPFIL-STARTEVNT-RBV	mbbo, mbbi	Modulator Ripple Filter Start Event
\$(PREFIX): MODRIPPFIL-STOPEVNT \$(PREFIX): MODRIPPFIL-STOPEVNT-RBV	mbbo, mbbi	Modulator Ripple Filter Stop Event

\$(PREFIX): MODRIPPFIL-QEN	bo,	Enable Modulator ripple filter for Q part
\$(PREFIX): MODRIPPFIL-QEN-RBV	bi	
\$(PREFIX): MODRIPPFIL-IEN	bo,	Enable Modulator ripple filter for Q part
\$(PREFIX): MODRIPPFIL-IEN-RBV	bi	

Table 52: sis8300llrfModRippleFiltChannel.template records

The following macros must be defined in order to successfully load the template

Macro	Description
PREFIX	Name prefix
ASYN_PORT	Asyn Port Name
ASYN_ADDR	4
CONSTS_DRVH, CONSTS_DRVL	Highest and lowest value for modulator ripple filter constant S value accepted by hardware (see Table 26)
CONSTC_DRVH, CONSTC_DRVL	Highest and lowest value for modulator ripple filter constant C value accepted by hardware (see Table 26)
CONSTA_DRVH, CONSTA_DRVL	Highest and lowest value for modulator ripple filter constant A value accepted by hardware (see Table 26)

Table 53: sis8300llrfModRippleFiltChannel.template macros

3.4.1.11 sis8300llrf-Main-NotchFilt-CH.template

This template defines the database with records used to control and monitor the Modulator ripple filter parameters.

Name	Type	Description
\$(PREFIX): NOTCHFIL-CONSTAREAL	ao,	Notch Filter Constant A real part
\$(PREFIX): NOTCHFIL-CONSTAREAL-RBV	ai	
\$(PREFIX): NOTCHFIL-CONSTAIMAG	ao,	Notch Filter Constant A imaginary

\$(PREFIX): NOTCHFIL-CONSTAIMAG-RBV	ai	part
\$(PREFIX): NOTCHFIL-CONSTBREAL	ao,	Notch Filter Constant B real part
\$(PREFIX): NOTCHFIL-CONSTBREAL-RBV	ai	
\$(PREFIX): NOTCHFIL-CONSTBIMAG	ao,	Notch Filter Constant B real imaginary part
\$(PREFIX): NOTCHFIL-CONSTBIMAG-RBV	ai	
\$(PREFIX): NOTCHFIL-IEN	bo,	Enable Notch filter
\$(PREFIX): NOTCHFIL-IEN-RBV	bi	

Table 54: sis8300llrfNotchFiltChannel.template records

The following macros must be defined in order to successfully load the template

Macro	Description
PREFIX	Name prefix
ASYN_PORT	Asyn Port Name
ASYN_ADDR	4
CONSTS_DRVH, CONSTS_DRVL	Highest and lowest value for modulator ripple filter constant S value accepted by hardware (see Table 26)
CONSTC_DRVH, CONSTC_DRVL	Highest and lowest value for modulator ripple filter constant C value accepted by hardware (see Table 26)
CONSTA_DRVH, CONSTA_DRVL	Highest and lowest value for modulator ripple filter constant A value accepted by hardware (see Table 26)

Table 55: sis8300llrfModRippleFiltChannel.template macros

3.4.1.12 sis8300llrf-Main-SigMon-CG.template

This template defines records used to control the SIGMON CG

Name	Type	Description
------	------	-------------

\$(PREFIX):SMON-STAT	mbbi	Channel Group status, see [5]
----------------------	------	-------------------------------

Table 56: sis8300llrfSigmonChannelGroup.template records

The following macros must be defined in order to successfully load the template

Macro	Description
PREFIX	Device Prefix
ASYN_PORT	Asyn Port Name
ASYN_ADDR	Value should be 5

3.4.1.13 sis8300llrf-Main-SigMon-CH.template

This template defines the database with records used to control and monitor the Signal Monitor parameters.

Name	Type	Description
\$(PREFIX):\$(CHANNEL_ID)-SMON-STAT	mbbi	Channel Status, see [5]
\$(PREFIX):\$(CHANNEL_ID)-SMON-ALARMSTAT	bi	Signal monitor alarm status
\$(PREFIX):\$(CHANNEL_ID)-SMON-PMSSTAT	bi	Signal monitor PMS status
\$(PREFIX):\$(CHANNEL_ID)-SMON-ILCKSTAT	bi	Signal monitor Interlock status
\$(PREFIX):\$(CHANNEL_ID)-SMON-MAGCURR	ai	Current magnitude for the corresponding ADC channel
\$(PREFIX):\$(CHANNEL_ID)-SMON-MAGMINMAX	ai	Maximum or minimum magnitude for the corresponding ADC channel during the last signal monitor active period.
\$(PREFIX):\$(CHANNEL_ID)-SMON-MAGTRESHVAL	ao,	Signal monitor magnitude threshold value

\$(PREFIX):\$(CHANNEL_ID)-SMON-MAGTRESHVAL-RBV	ai	
\$(PREFIX):\$(CHANNEL_ID)-SMON-STARTEVNT \$(PREFIX):\$(CHANNEL_ID)-SMON-STARTEVNT-RBV	mbbi, mbbo	Event defining the start of signal monitor active period
\$(PREFIX):\$(CHANNEL_ID)-SMON-STOPEVNT \$(PREFIX):\$(CHANNEL_ID)-SMON-STOPEVNT-RBV	mbbo, mbbi	Event defining the end of signal monitor active period
\$(PREFIX):\$(CHANNEL_ID)-SMON-ALARMCND \$(PREFIX):\$(CHANNEL_ID)-SMON-ALARMCND-RBV	bo, bi	Alarm condition
\$(PREFIX):\$(CHANNEL_ID)-SMON-PMSEN \$(PREFIX):\$(CHANNEL_ID)-SMON-PMSEN-RBV	bo, bi	Enable/Disable PMS if alarm is raised
\$(PREFIX):\$(CHANNEL_ID)-SMON-ILOCKEN \$(PREFIX):\$(CHANNEL_ID)-SMON-ILOCKEN-RBV	bo, bi	Enable/Disable Interlock if alarm is raised
\$(PREFIX):\$(CHANNEL_ID)-SMON-ACDCSEL \$(PREFIX):\$(CHANNEL_ID)-SMON-ACDCSEL-RBV	bo, bi	Signal type select – AC or DC

Table 57: sis8300llrfSignalMonitorChannel.template records

The following macros must be defined in order to successfully load the template

Macro	Description
PREFIX	Name prefix

ASYN_PORT	Asyn Port Name
CHANNEL_ID	AI2, AI3, AI4, AI5, AI6, AI7, AI8, AI9
ASYN_ADDR	Can be 2, 3, 4, 5, 6, 7, 8, 9 and corresponds to the ADC channel number (see macro CHANNEL_ID)
MAGTRESH_DRVH, MAGTRESH_DRVL	Highest and lowest value for modulator ripple filter constant S value accepted by hardware (see Table 28Table 26)

Table 58: SignalMonitorChannel.template macros

3.4.1.14 sis8300llrf-Main-AI-CG.template

The template for AI Channel overrides some settings from sis8300AICChannelGroup.template [6]. The list of overridden settings can be found in Table 32. In order to successfully load the template, the mentioned generic AI CG template must be loaded first. The following macros must be defined when loading the template:

Macro	Description
PREFIX	Name prefix
CHANNEL_ID	Unique ID (usually AI, has to be the same as when loading the sis8300AICChannelGroup.template)
AI_NELM	Max number of ADC samples per one channel

Table 59: sis8300llrfAICChannelGroup.template macros

3.4.1.15 sis8300llrf-Main-AI-CH.template

The template for AI channel overrides some settings from sis8300AICChannel.template [6] and adds functionality specific to AI0 (cavity input) and AI1 (reference input). In order to successfully load the template, the mentioned generic AI CH template must be loaded first.

Name	Type	Description
\$(PREFIX):\$(CHANNEL_ID)-ENBL	bo	Should always be set to 1 when using the Struck SIS8300L in LLRF context (see Table 5)
\$(PREFIX):\$(CHANNEL_ID)-IN	ai	Overrides the single sample read option for AI channel

\$(PREFIX):\$(CHANNEL_ID)-ANG	ai	Signal Angle, see Table 33
\$(PREFIX):\$(CHANNEL_ID)-MAG	ai	Signal Magnitude, see Table 33
\$(PREFIX):\$(CHANNEL_ID)-I	ai	Signal I, see Table 33
\$(PREFIX):\$(CHANNEL_ID)-Q	ai	Signal Q, see Table 33
\$(PREFIX):\$(CHANNEL_ID)-GETNEWMAPOINT	bo	Read new MA point from the device and calculate the corresponding IQ values, see Table 33
\$(PREFIX):\$(CHANNEL_ID)-NEWMAPOINT	bi	New MA and calculated IQ point is available for readout, see Table 33

Table 60: sis8300llrfAIChannel.template records

The following macros must be defined when loading the template:

Macro	Description
PREFIX	Name prefix
ASYN_PORT	Asyn Port Name
ASYN_ADDR	Channel Number (0-9), corresponds to ADC channel number
CHANNEL_ID	Unique ID (usually AI0 to AI9, the same as when loading the sis8300AIChannel.template)
ENABLE	1 for enabled, 0 for disabled

Table 61: sis8300llrfAIChannel.template macros

3.4.1.16 sis8300llrf-Main-CalcFixedPointMagAng.template

This channel adds extra records, used to calculate Magnitude and Angle corresponding to the fixed SP and fixed FF point settings from the 33LLRF PI Channel (sis8300llrfPIChannel Class). This is mostly used during setup procedure (see

The following macros must be defined when loading the template:

Macro	Description
PREFIX	Name prefix
PI_ONE, PI_TWO	Has to correspond to PI channel names defined in Table 51. Normally PI-I and PI-Q

Table 65: sis8300llrf-RMS-statistics-reset.template macros

sis8300llrf-Setup.template).

Name	Type	Description
\$(PREFIX):PI-FIXED\$(FIXED_POINT_TYPE)MAG	calc	This is where the MA point is calculated from the IQ point. The VAL field of the record holds the magnitude value
\$(PREFIX):PI-FIXED\$(FIXED_POINT_TYPE)ANG	ai	Angle value

Table 62: sis8300llrf-Main-CalcFixedPointMagAng.template records

The following macros must be defined when loading the template:

Macro	Description
PREFIX	Name prefix
FIXED_POINT_TYPE	FF or SP

Table 63: sis8300llrf-Main-CalcFixedPointMagAng.template macros

3.4.1.17 sis8300llrf-RMS-statistics-reset.template

This template adds just one record, which allows for a “simultaneous” reset of both I and Q PI error RMS statistics (see LLRF PI Channel (sis8300llrfPIChannel Class)).

Name	Type	Description
------	------	-------------

\$(PREFIX):PI-RMS-RESET	fanout	Processing this record will cause a reset of both PI-I and PI-Q RMS average and Max value
-------------------------	--------	---

Table 64: sis8300llrf-RMS-statistics-reset.template records

The following macros must be defined when loading the template:

Macro	Description
PREFIX	Name prefix
PI_ONE, PI_TWO	Has to correspond to PI channel names defined in Table 51. Normally PI-I and PI-Q

Table 65: sis8300llrf-RMS-statistics-reset.template macros

3.4.1.18 sis8300llrf-Setup.template

This template defines all the records that are used during the initial setup of the controller.

Name	Type	Description
\$(PREFIX):SETUP-ACT \$(PREFIX):SETUP-ACT-RBV	bo, bi	Used to set or indicate that the setup is active. One should not start the setup procedure by writing to this record. The SETUP-START record is used for this.
\$(PREFIX):SIGNALACT	bi	Indicates if the signal is currently active. Only to be used when operating in Continuous Wave (CW) mode.
\$(PREFIX):SETUP-START	bo	Write 1 to this record to start the setup and 0 to stop/abort the setup.

Table 66: sis8300llrf-Setup.template records

The following macros must be defined when loading the template:

Macro	Description
PREFIX	Name prefix

ASYN_PORT	Asyn Port Name
-----------	----------------

Table 67: sis8300llrf-Setup.template macros

3.4.1.19 sis8300llrf-SpecOp-Device.template

This template defines all the records that are used when device is operating in special operating modes.

Name	Type	Description
\$(PREFIX):FORCETRIGG \$(PREFIX):FORCETRIGG-RBV	mbbo, mbbi	Used for sending a specific trigger to the device.
\$(PREFIX):OPMODE \$(PREFIX):OPMODE-RBV	mbbo, mbbi	Used for selecting a specific operating mode
\$(PREFIX):SIGNALACT	bi	Indicates if the signal is currently active. Only to be used when operating in Continuous Wave (CW) mode.

Table 68: sis8300llrf-SpecOp-Device.template records

The template must always be loaded after the 42sis8300llrf-Main-Device.template. The following macros must be defined when loading the template:

Macro	Description
PREFIX	Name prefix
ASYN_PORT	Asyn Port Name

Table 69: sis8300llrf-SpecOp-Device.template macros

3.4.1.20 sis8300llrf-SpecOp-ControlTable-CH.template

This template defines records for settings defines in LLRF Special Operation Control Table Channel (sis8300llrfControlTableChannelSpecOp Class).

Name	Type	Description
\$(PREFIX):\$(CTRL_TABLE_TYPE)-SM-	bo,	FF table mode

FFTABLEMODE \$(PREFIX):\$(CTRL_TABLE_TYPE)-SM-FFTABLEMODE-RBV	bi	
(PREFIX):\$(CTRL_TABLE_TYPE)-SM-ANG	waveform	Set the Angle part of the Control table
\$(PREFIX):\$(CTRL_TABLE_TYPE)-SM-ANG-GET	waveform	Processing this record will read the Angle part of the control table from device memory. It has to be processed manually. I/O interrupts are disabled.
\$(PREFIX):\$(CTRL_TABLE_TYPE)-SM-ANG-SMNM-RBV	longin	Current number of elements in the Control table that is actually written to memory (see 3.3.4.5)
\$(PREFIX):\$(CTRL_TABLE_TYPE)-SM-MAG	waveform	Set the magnitude part of the control table
\$(PREFIX):\$(CTRL_TABLE_TYPE)-SM-MAG-GET	waveform	Processing this record will read the magnitude part of the control table from device memory. It has to be processed manually. I/O interrupts are disabled.
\$(PREFIX):\$(CTRL_TABLE_TYPE)-SM-MAG-SMNM-RBV	longin	Current number of elements in the Control table that is actually written to memory (see 3.3.4.5)

Table 70: sis8300llrf-SpecOp-ControlTable-CH.template records

The template must always be loaded after sis8300llrf-Main-ControlTable-CH.template. The following macros need to be defined in order to successfully load the template:

Macro	Description
PREFIX	Name prefix
ASYN_PORT	Asyn Port Name
CTRL_TABLE_TYPE	FF or SP, see Table 41
CTRL_TABLE_CG_NAME	Name of the corresponding channel group, sp or ff, see Table 41

NUM_PULSE_TYPES	Number of pulse types, has to be the same as defined in Table 35: sis8300llrfDevice.template macros
-----------------	---

Table 71: is8300llrf-SpecOp-ControlTable-CH.template macros

3.5 Startup Snippets

Startup snippets loading the appropriate records are defined in the module. All the startup snippets are explained on the wiki page ()

3.6 Demo application

3.7 Software Version

The Struck SIS8300L LLRF user-space library up to version 1.2 was developed using:

- kmod-sis8300 version 1.4

The Struck SIS8300L LLRF epics module version 1.2 was developed using:

- EPICS Base 3.14.12.3
- AsynDriver 4.21
- NDS 2.3.1
- epics-sis8300 module

If you are using a different version of any part of the software consult the release notes for possible changes.

3.8 Learning Feed Forward

Learning Feed Forward Algorithm (LFF) will try to compensate for repetitive errors, such as Lorentz force detuning, by correcting the FF control table (Table 3). The development of the algorithm is out of scope of this document and will not be developed by ICS.

The interface of the algorithm with the LLRF software module will be on EPICS database level and will depend on the output of the algorithm. We propose two options:

1. The output of the algorithm is a FF table that replaces the previous FF table
2. The output of the algorithm is a FF correction table which needs to be added to the existing FF.

In the first case, the FF angle and FF magnitude tables are already available in the database. In the second case, the impact of the algorithm can be included as:

$$FF = FF_MAIN + FF_CORR.$$

This option requires additional development. In both cases, the output of the algorithm can be written to the corresponding waveform record through Channel Access by using any of the CA client interfaces

listed here: <http://www.aps.anl.gov/epics/extensions/> under "CA Client Interfaces to other tools and languages". List of supported languages and tools is extensive and we believe that it offers enough variety, so there is no plan to add support for any other language/tool.

In either case there is another decision that needs to be made: will the output of the algorithm already provide angle and magnitude table joined into one, or will this be left to software (see section 3.2.2.1).

4 REFERENCES

- [1] Struck, *SIS8300-L uTCA FOR PHYSICS Digitizer, Version: SIS8300L-M-2008-1-V100*, 2014.
- [2] Struck, *SIS8900 uTCA FOR PHYSICS RTM, Version: SIS8900-M-1-1-V104*, 2013.
- [3] Desy and Struck, *DWC8VM1 8 Channel Downconverter One Channel Vectormodulator RTM, Version: DWC8VM1-M-1-1-V101*, 2014.
- [4] F. Kristensen, *LLRF Control System For ESS - Specification, version 2.6*, Lund: LTH, 2015.
- [5] “NDS Software Developer Manual”.
- [6] K. Strnisa, *EPICS sis8300 Module Technical Documentation (rpm: codac-core-4.1-epics-sis8300-doc)*, Cosylab, 2014.
- [7] S. Peggs, *Technical Design Report*, Lund: European Spallation Source, 2013.
- [8] N. Claesson, *Data On Demand (DOD) Module Technical Documentation (rpm: codac-core-4.1-epics-dod-doc)*, Cosylab, 2014.
- [9] R. Stefanic, *Timing Reciever Module Technical Documentation (rpm: codac-core-4.1-epics-tr-doc)*, Cosylab, 2013.

5 LIST OF ABBREVIATIONS

Abbreviation	Definition
CS	Control System
ICS	Integrated CS
SW	Software
HW	Hardware
EPICS	Experimental Physics and Industrial Control System
LLRF	Low Level RF
AMC	Advanced Mezzanine Card
RTM	Rear Transition Module
FF	Feed Forward
LFF	Learning FF
SP	Set Point
HV	High Voltage
PI	Proportional Integral
CW	Continuous Wave
OPI	Operator Interface
NDS	Nominal Device Support
CSS	Control System Studio
BOY	Best OPI, Yet
PV	EPICS Process Variable
MTCA/ μ TCA/uTCA	MicroTCA
MTCA.4	uTCA For Physics

6 APPENDIX: CURRENT DEVELOPMENT SYSTEM

The development of firmware and software components for the LLRF system is being developed in parallel. On top of that, since there is a lot of functionality that the system will have to provide in the end, they are being developed and added to the system one after another. Not everything listed in the document is already implemented (either at the firmware or software level) or even defined properly. Here is a list of things to be aware of:

Function	Section reference	Description
Signals connected to the AI channels	2.2	at this point the board only takes two inputs, the cavity probe on the first AI channel (AI1) and the reference input on the second (AI2). Channels 3 – 5 can have any input and have signal monitors, channels 6 – 9 are hijacked by FPGA and contain intermediate processing results
FPGA processing blocks and Control Tables	2.2 2.3.1	Right now, only PI regulator and FF correction blocks are realized on the FPGA. Blocks 3-5 are not yet properly defined and are not included in the FPGA [4]. This of course affects the CTs that are available through software
Signal Monitoring	3.2.2.6	Signal monitoring will eventually be available on all channels except cavity and reference input. At this point, Channels 6 -9 are hijacked by the firmware so there are no signal monitors available (the functionality is there though). If interlock on alarm is enabled, the Harlink input 0 will go HIGH when alarm is triggered on a channel.

7 APPENDIX: CONTROL TABLE GENERATION

So far there have been no requests for generating the FF and SP tables from parameters, but based on the example from DESY we can assume that the tables will be generated in SW from some user-defined physical parameters, such as:

- RF field gradient,
- Phase,
- Smoothing type,
- Sampling frequency of the table,
- Filling and flat-top duration...

Generating a table can be viewed as another mode of operation of the controller. This adds up to three modes of operation:

1. User – define (SP or FF) table,
2. User – defined (SP or FF) fixed point,
3. Generate the (FF or SP) table from parameters