C Language API to the SNS LLRF Board XXX dangerously rough draft XXX

Larry Doolittle, LBNL

March, 2004

Introduction

Interface hardware connected to a host computer is traditionally documented by defining its registers' addresses and functions. The SNS Low Level RF (LLRF) system includes features that are complex and require tricky software support. Fortunately, the same Verilog code that is synthesized for the on-board FPGA can also be simulated. The tricky layer of software is therefore written in concert with the FPGA Verilog code, and their interaction is tested in simulation.

That pre-tested software implements a Hardware Abstraction Layer (HAL) Application Programming Interface (API) to which higher level control system code (e.g., EPICS) connects. This code is designed to be robust, standards-compliant, and portable C. It has been successfully used on Linux and vxWorks, on three hardware architectures (Pentium, ARM, and PowerPC).

As long as this HAL interface is kept stable, the boundary between hardware and software can be adjusted as needed, without requiring changes to the control system code. Re-builds are needed when the implementation changes, so there is a match between the FPGA code and the driver code. Revision codes are built into the structure so that discrepancies can be flagged at startup.

This documentation is part of a package (`llc-suite`) that includes the LLRF Verilog code, the HAL implementation, and tests that can verify its functionality (in simulation) without special hardware. This API is followed for all three generations of SNS LLRF hardware: MEBT, Interim, and FCM.

Setup

```
#include "llrf.h"
unsigned int *llrf_full_setup(int la, unsigned short **fbasep);
int llrf_teardown(unsigned short *fbase);
extern char *llrf_setup_errors[];
```

The include file has prototypes and associated definitions for all API features. Only features of this file that are documented here should be used; that file also includes local definitions for the LLRF API implementation.

The `llrf_full_setup()` routine should be called before any other activity takes place, and `llrf_-teardown()` called at the end of activity. With real hardware, `llrf_teardown()` is expected to be a no-op.

The input parameter `la` is the logical address of the FCM. It is ignored on the MEBT and Interim systems.

The `unsigned short *fbase` that gets created in the call to `llrf_full_setup` is the base address of the FPGA, and needs to be passed to all other hardware access functions. Please do not deference this pointer yourself! (XXX should I therefore change it to `void *`?)

The return code of `llrf_full_setup` is 0 on success. Non-zero return values should abort any attempt to work with the FPGA hardware. Return values are an index into `llrf_setup_errors[]`, which provides a human-friendly description of the error, or `"OK"` in case of success. These strings are printable ASCII characters only, and do not include any line terminator character.

Register Access

```
unsigned int get_fpga_word(unsigned short *fbase, unsigned int address);
void set_fpga_word(unsigned short *fbase, unsigned int address,
  unsigned int value);
```

Some features of the board are directly accessible through traditional register I/O. The `fbase` parameter is a board- and system- dependent identifier (base address) the FPGA of interest. `address` is an integer register number derived from one of the symbolic names documented below.

All registers are 16 bits wide. Registers marked (†) can be read and written, and hold their last written value. "Bumpless reboot" can be accomplished by updating the software based on the values read back.

Register† LLRF_DDS_FREQ

The offset frequency for the on-board Direct Digital Synthesizer (DDS). This is a signed 16-bit number that adjusts the frequency of cavity operation, relative to the Master Oscillator, in the range of ± 625 kHz. One bit corresponds to 19.073486328125 Hz exactly (assuming the PLL is locked to a perfect standard 50 MHz).

Register LLRF_STATUS

Read-only bit map of module status.

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
-	-	-	-	-	-	-	-	-	-	LRC2	LRC1	PLL	IL	RF_ON	KCM

LRC2,LRC1: FCM-specific readout of the configuration switches that define its position. 0=Left, 1=Center, 2=Right, 3=Invalid. Non-FCM systems read 0.

PLL: Instantaneous readout of the same signal described in the PLL bit of the LLRF_ERRORS register.

IL: Interlock status. 0=Ready, 1=Fault. Not available on the MEBT.

RF: Instantaneous readout of the RF_ON trigger/gate signal. Not generally used because its value depends on when it's read. Could be useful diagnosing cable faults.

KCM: Used internally to handshake changes to multiplier coefficients.

Register† LLRF_CONFIG

Read/Write access to a bit map of assorted configuration control.

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
RD	TC	DD	CW	PE	IS	FS	-	-	-	SR	IR	RN	-	PSA	TE

TE (Trace Select) Enable waveform storage to the history buffer

PSA (Phase Sense Adjust) 0 - sample 2.5 MHz on the rising edge of the 40 MHz clock, 1 - sample 2.5 MHz on the falling edge of the 40 MHz clock

RN (Run) Let the timing state machine move beyond state 0 (Idle)

IR (Ignore RF Off) Continue running the state machine with full time widths even when the RF_ON signal is removed

SR (Self Retrigger) Jump from Wait to Idle at the programmed time interval, even without a rising edge of RF_ON. In combination with IR, permits bench testing without an external trigger source.

FS (Feedback Select) Enable feedback terms to the RF output signal

IS (Integrate Select) Allow integrator to accumulate signal for the RF output. Required both for feedforward to apply, and for the integral term feedback to be used

PE (RF Permit)

CW (Continuous Wave) Operate the feedback loop at all times, independent of the timing state machine. This sets CW RF operation of the cavity.

DD (DDS Passthrough) Simply pass the DDS waveform though to the output in a CW state. Used when analyzing the output signal with a spectrum analyzer.

TC (Totalizer Channel) 0 - Channel D (phase reference), 1 - Channel C (cavity). Presumably the value 1 will be used with the pulse reference system. The value 0 is more generally useful for lab work.

RD (Raw DDS) Drive the output with the DDS waveform, but only during the pulse period. This gives long-pulse open-loop cavity operation with controllable frequency, which is not possible with the Feedforward table.

Register LLRF_ERRORS

Bit map of possible error conditions. This register self-clears when read; writing to it sets the handshake flag (the data written is ignored). A handshake error is defined as the occurrence of an RF pulse between the time this register is written and read.

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
-	-	-	-	-	-	PLL	HS	SE	SM	TS	CM	AOV	BOV	COV	DOV

PLL (pll_unlocked): As reported by the ADF4001 chip. Always 0 on the MEBT. Note that a sudden cutoff of the 50 MHz IF reference signal can leave the ADF4001 stalled, and not reporting an error on this signal. When checking for PLL errors, it is therefore important to also look at the VCXO cntl voltage from the slow ADC subsystem. That voltage will go to (near) zero when ADF4001 is in the stalled state.

SE (sync_error): The 2.5 MHz phase reference signal shows incoherency with the 40 MHz clock. Either the hardware is defective, or the `phase_sense_adjust` configuration bit needs to be toggled. Only applies to the MEBT systems, the others always read 0.

SM (sync_missing): The 2.5 MHz phase reference signal is not present, so the absolute phase of the measured and generated signals is not known. Only applies to the MEBT systems, the others always read 0.

TS (trigger_skipped): A rising edge of the RF on signal was detected when the timing system was not primed to accept it. Can be caused by a pulse repetition rate faster than compatible with the timing setup, or by ringing on the falling edge of the trigger signal.

CM (clk40_missing): The on board DSP clock is not present. Can be caused by incorrect cable hookups or signal levels on the MEBT. Some host computers (nanoEngines) on the Interim systems can trigger this event with spikes on the host clock signal that double-trigger the FPGA. Always 0 on the FCM.

AOV, BOV, COV, DOV: The input RF ADCs reported an overflow during the acquisition time window. Usually caused by input signals that are too strong. Can also be caused by too short a warmup time. Always 0 on the FCM.

Timing

The timing engine cycles through the following states:

Number	Name	duration
1	Idle	(`*(LLRF_TIMERS+0)` + 2) · 100ns
3	ADC warmup	(`*(LLRF_TIMERS+2)` + 1) · 100ns
4	Feedforward only	(`*(LLRF_TIMERS+3)` + 1) · 100ns
5	Feedback	(`*(LLRF_TIMERS+4)` + 1) · 100ns
6	Acquire decay curve	(`*(LLRF_TIMERS+5)` + 1) · 100ns
7	Inhibit retrigger	(`*(LLRF_TIMERS+1)` + 1) · 400ns
0	Wait for trigger	(`*(LLRF_TIMERS+7)` + 1) · 400ns

State 0 normally only ends with the rising edge of the RF_ON trigger/gate signal. When the SR bit of the LLRF_CONFIG word is set, it will also end at the specified time. Unless the IR bit of the LLRF_CONFIG word is set, states 4 and 5 will end in (in a single 100 ns cycle) if RF_ON is low.

The timer registers can be read as well as written (to support bumpless reboot). Reliable reads, however, require extra attention. The same timer must be read two times in a row, and the first answer discarded. Sorry.

XXX Need to add a picture.

Totalizer

Register† LLRF_TOTAL_DEL

Sets the delay from the beginning of RF state machine state 3 (ADC warmup) to the acquisition of the phase reference signal. The time delay is $(16n + 1) \cdot 0.1\mu$s. The phase reference signal is averaged for a fixed 25.6 μs. See also the TC bit of LLRF_CONFIG.

```
void get_phase_zero(unsigned short *fbase, double *mag, double *phase)
```

`phase` is in degrees, `mag` is in bits, based on 32768 full scale.

Stopwatch

```
void arm_stopwatch(unsigned short *fbase);
unsigned int get_stopwatch_time(unsigned short *fbase, unsigned int *swtime);
```

The stopwatch provides a measurement of the time between RF pulses, with a resolution of 200 ns. Once armed, the next pulse width will be measured. The result (`*swtime`) is the number of 200 ns ticks. If the closing event has not yet been recorded when `get_stopwatch_time` is called, it will return 0.

History Buffers

```
void import_trace(unsigned short *fbase, unsigned int addr,
  unsigned int average, int cnt_at_pulse_end, short dr[], short di[])
```

`addr` should be one of LLRF_TRACED, LLRF_TRACE1, LLRF_TRACE2, or LLRF_TRACE3 (XXX need another version of this routine to cover the "I already made a copy of the data via DMA" case). `average` is the value for which that buffer is configured. `cnt_at_pulse_end` is the value read from LLRF_UCAPE or LLRF_DCAPE. The routine stores Real and Imaginary components of the waveform to the `dr` and `di` arrays, respectively.

```
  unsigned short hist_config(unsigned int average);
```

Returns the proper value to store to LLRF_UHIST_CFG or LLRF_DHIST_CFG.

```
  void time_fill(float t[], int cnt_at_pulse_end, int average);
```

Curve Fitting

```
  int analyze_decay3(unsigned int npt, unsigned int shift, double dt,

   short red[], short imd[], double *df, double *bw);
```

Perhaps not terribly tightly bound to the hardware, this routine does the exponential curve fitting associated with measurement of cavity detune frequency. Return codes:

0 - success
1 - `npt` < 3
2 - arithmetic overflow
3 - the matrix S is singular

The computed values `*df` (delta frequency) and `*bw` (bandwidth) have units inverted from the sample interval time provided in `dt`. See `host/decay3.c` for more information.

Low Speed Serial Devices

All low speed serial devices are controlled from a single peripheral controller in the FPGA, and the underlying operations are not atomic. These routines are therefore not compatible with multi-threaded use (although their use can be freely mixed with other threads working with other subsystems on the board).

```
void sport_init(unsigned short *fbase);
```

Called once at system initialization to put the subsystem into a known good state.

```
void set_dac(unsigned short *fbase, unsigned int channel, unsigned int value);
```

The channel assignment is given in the following table for each of the systems. ‘Mod drive’ controls the RF Output level, where available. The `value` is in the range of 0 to 4095.

	MEBT	Interim	FCM
0	?	Mod drive	unused
1	?	Loopback	Mod drive
2	?	AO1	unused
3	?	AO2	
4	?		
5	?		
6	?		
7	?		
8	?		
9	?		
10	?		
11	?		
12	?		
13	?		
14	?		
15	?		

```
int get_adc(unsigned short *fbase, unsigned int channel);
```

The channel assignment is given in the following table for each of the systems.

	MEBT	Interim	FCM
0	Fwd power	VCXO cntl	VCXO cntl
1	Rfl power	LO level	1.5V-Imon
2	Cav power	AI1	1.5V
3	Fwd power trip	AI2	3.3V-D
4	Rfl power trip	Loopback	12V * 0.25
5	Cav power trip	NC	Mod bias
6	Interlock stat	15V * 0.24	RFO level
7	Phase error	Mod bias	LO level
8	Sys Phase		
9	Sys Amp		
10	Amp1 Amp		
11	Cav Phase		
12	Amp1 Phase		
13	Amp2 Phase		
14	Amp2 Amp		
15	Fwd Signal Phase		

```
void write_adf(unsigned short *fbase, unsigned long num);
```

Write a 24-bit word to the ADF4001 Phase Lock Loop (PLL) chip. Each such write takes approximately 7 μs to execute. The nominal values to load are 0x000093, 0x000010, 0x000501 for the Interim system, and 0x000093, 0x000020, 0x000501 for the FCM system. The MEBT systems do not have such a chip; in that context this routine does nothing. Consult the ADF4001 data sheet for additional information.

1-Wire^{TM} Devices

Need to reorganize DS2401/Interim routine to support DS1822 structure, and include `get_temperature()`!

```
struct dallas_interface {
  unsigned char id[NUM_ONEWIRE_CHIPS][8];
  double temp[NUM_ONEWIRE_CHIPS];
};
```

Temperatures are always given in degrees Celsius, or `INVALID_TEMPERATURE` when data is not available. The structure also includes members that are not listed above, which are used to store state between calls.

```
int dallas_ff_cycle(struct dallas_interface *me, int quiet);
```

This routine should be called periodically (e.g., 60 Hz) to update the temperatures in the data structure. Each temperature will be updated at approximately 0.75 second intervals. If `quiet` is less than 2, status messages may be printed.

This call runs "quickly" with a maximum execution time of about 10 μs. This routine is a "no-op" on MEBT systems. On Interim systems, it hooks into the thermometer implemented with the low speed serial peripheral controller, and therefore is not thread-safe with API calls for any of the other low speed devices. That routine is also slower than the real 1-Wire "fire and forget" peripheral, taking 70 μs to run.

```
struct dallas_interface *new_dallas_interface(unsigned short *fbase);
```

Assigns memory for a `struct dallas_interface`, and initializes it for use. On the Interim system, this routine also fills in the serial number of the first (and only) 1-Wire chip, and therefore takes a "long" time to run: about 9 ms. This routine does nothing on MEBT systems.