UberClock Register Map and Serial Interface

Contents

1 Overview

This document describes the UberClock control interface exposed via LiteX CSRs and the corresponding serial console commands implemented in `uberclock.c`. The goal is to provide a structured description of:

- register groups and their roles,
- serial commands and valid ranges,
- write/read helper conventions (including commit behavior),
- capture-to-DDR (UberDDR3/S2MM) control and UDP streaming.

1.1 Build-time presence of CSRs

Many features are guarded by compile-time macros (e.g. `#ifdef CSR_MAIN_...`). If a CSR is not present in the bitstream, the command will print a message like `CSR not present` or `Not built with UberClock CSRs`.

2 Register Map

This section groups similar registers together. For each group:

2.1 Input NCO control

Register(s)	Width / range	Role
`phase_inc_nco`	24-bit ($0..2^{24}-1$)	Phase increment for the input CORDIC/NCO. Controls the generated reference tone frequency.
`nco_mag`	signed 12-bit packed in low bits (-2048..2047)	Magnitude control for the input NCO. Written as `(v & 0x0fff)`; interpret as 12-bit signed.

2.2 Downconversion NCO controls (channels 1–5 and reference)

Register group	Width / range	Role
`phase_inc_down_1..5`	24-bit used ($0..2^{24}-1$)	Per-channel phase increment for the downconversion mixers/NCOs (channels 1–5).
`phase_inc_down_ref`	24-bit used ($0..2^{24}-1$)	Reference phase increment (if present) for downconversion alignment or shared reference.

2.3 CPU-generated tone controls (channels 1–5)

Register group	Width / range	Role
`phase_inc_cpu1..5`	24-bit ($0..2^{24}-1$)	Per-channel CPU-side NCO phase increment (channels 1–5).
`mag_cpu1..5`	signed 12-bit packed (-2048..2047)	Per-channel CPU-side NCO magnitude. Written as `(v & 0x0fff)`.

2.4 Main routing and output selection

Register(s)	Width / range	Role
`input_select`	small enum (0..3)	Selects the main signal source (e.g., ADC/NCO/-SUM/reserved).
`upsampler_input_mux`	small enum (0..2)	Selects what drives the upsampler input path (Gain/CPU/CPU NCO).
`output_select_ch1`	4-bit (0..15)	Selects DAC1 source.
`output_select_ch2`	4-bit (0..15)	Selects DAC2 source.

2.4.1 Legend: `input_select`

The mapping follows the RTL definition:

Value	Name	Signal
0	ADC	`filter_in`
1	Input NCO	`nco_cos`
2	Output of the 5 ch. acc.	`sum[13:2]`
3	Reserved	Same as value 2

Notes:

- `sum[13:2]` performs width reduction (14 bit → 12 bit) while preserving sign.
- Value `3` is currently not explicitly distinguished in RTL and aliases to the default path.

2.4.2 Legend: `output_select_ch1`, `output_select_ch2`

These registers independently select the signal routed to DAC channel 1 and channel 2. Both channels share identical selection semantics.

Value	Name	Signal
0	Downsampled value after gain ch1	`upsampled_gain_y1[15:2]`
1	Downsampled value after gain ch2	`upsampled_gain_y2[15:2]`
2	Downsampled value after gain ch3	`upsampled_gain_y3[15:2]`
3	Downsampled value after gain ch4	`upsampled_gain_y4[15:2]`
4	Downsampled value after gain ch5	`upsampled_gain_y5[15:2]`
5	Upconverted TX channel 1 output	`tx_channel_output1[15:2]`
6	Upconverted TX channel 2 output	`tx_channel_output2[15:2]`
7	Upconverted TX channel 3 output	`tx_channel_output3[15:2]`
8	Upconverted TX channel 4 output	`tx_channel_output4[15:2]`
9	Upconverted TX channel 5 output	`tx_channel_output5[15:2]`
10	NCO output	`nco_cos « 2`
11	ADC ch 0	`filter_in « 2`
12	ADC ch 1	`filter_in_1 « 2`
13–15	Sum (default)	`sum[?]`

Notes:

- All DAC inputs are reduced to 14 bit width before output.
- Left shifts (`«2`) compensate for internal scaling differences.
- Undefined values (13–15) fall back to the summed signal.

2.5 Gain and final scaling

Register group	Width / range	Role
gain1..5	32-bit signed/fixed-point	Per-channel gain coefficients applied in the datapath. Encoding is design-defined (e.g., Q-format).
final_shift	typically small integer (0..7 used)	Final right-shift / scaling stage after accumulation/mixing (design-defined).

Defaults:

- gain1..5 ← 0x40000000 (a gain of 1)
- final_shift ← 2

2.6 Debug selection

Register(s)	Width / range	Role
highspeed_dbg_select	(0..3)	Selects which high-speed debug signal is exposed.

Defaults:

- high-speed dbg ← 0

2.6.1 Legend: highspeed_dbg_select

Value	Name	Signal
0	ADC CH1	filter_in
1	ADC CH2	filter_in_1
2	5 channel sum output	sum[13:2]
3	NCO output	nco_cos

Notes:

- Intended for probing high-rate internal signals close to the core datapath.
- The summed signal is width-reduced to 12 bits with sign preservation.
- Values above 3 default to the input NCO path.

2.7 Upsampler direct input registers

Register(s)	Width / range	Role
upsampler_input_x	signed 16-bit packed in low bits	Directly writes X component into the upsampler path.
upsampler_input_y	signed 16-bit packed in low bits	Directly writes Y component into the upsampler path.

Notes:

- Currently only for one channel. To be extended in hardware.

2.8 ATAN2 Output registers

Register(s)	Width / range	Role
`phase`	signed 25-bit packed in low bits	Reads phase output of the CORDIC atan2.
`magnitude`	signed 16-bit packed in low bits	Reads magnitude output of the CORDIC atan2.

Notes:

- Currently only for one channel (ch1). To be extended in hardware/software.

2.9 Capture control (design/ramp to DDR + small capture buffer)

Register(s)	Width / range	Role
`cap_enable`	1-bit	Select capture source written to DDR via S2MM: 0=ramp→DDR, 1=design capture→DDR.
`cap_beats`	32-bit (>=1)	Number of 256-bit beats captured by the gateware (used by S2MM).

2.10 UberDDR3 / S2MM engine registers

Register group	Width / range	Role
`ubddr3_dma_addr0/addr1`	64-bit address split	DDR destination address for S2MM writes.
`ubddr3_dma_inc`	1-bit	Enable incrementing destination address.
`ubddr3_dma_size`	enum	Transfer size code (bus/32/16/8) depending on integration.
`ubddr3_ramp_len`	beats	Number of beats to write.
`ubddr3_dma_req`	pulse	Start DMA request.
`ubddr3_dma_busy`	read	Busy flag (poll until 0).
`ubddr3_dma_err`	read	Sticky error indicator (if present).
`ubddr3_calib_done`	read	DDR calibration status (if present).

3 CSR read and write helper functions

During the build process, all CSR access helper functions are automatically generated by the LiteX build system. The generated helpers are located in the build directory and collected in the header file `csr.h`.For each CSR register, a pair of strongly-typed C helper functions is provided: one for writing and one for reading the register value. The helpers follow a uniform naming convention.

Listing 1: Generated CSR access helpers

```
/* Write helper */
main_<REGISTER_NAME>_write(uint32_t value);

/* Read helper */
uint32_t main_<REGISTER_NAME>_read(void);
```

Notes:

- `<REGISTER_NAME>` corresponds to the CSR name as defined in the gateware.
- All write helpers accept a single 32-bit argument; narrower registers are packed into the lower bits as required.
- Read helpers take no arguments and return the current register value.

4 Serial Console Commands

This section documents the user-facing CLI commands and how they map to registers.

4.1 UberClock control commands

Command	Arguments	Description
`phase_nco`	`<0..524287>`	Write `phase_inc_nco`.
`nco_mag`	`<-2048..2047>`	Write `nco_mag` (12-bit signed packed).
`phase_down_1..5`	`<0..524287>`	Write `phase_inc_down_1..5`.
`phase_down_ref`	`<0..524287>`	Write `phase_inc_down_ref`.
`phase_cpu1..5`	`<0..524287>`	Write `phase_inc_cpu1..5`.
`mag_cpu1..5`	`<-2048..2047>`	Write `mag_cpu1..5` (12-bit signed packed).
`input_select`	`<0..3>`	Write `input_select`.
`upsampler_input_mux`	`<0..2>`	Write `upsampler_input_mux`.
`output_select_ch1/ch2`	`<0..15>`	Write `output_select_ch1/ch2`.
`gain1..gain5`	`<int32>`	Write `gain1..5`.
`final_shift`	`<0..7>`	Write `final_shift`.
`lowspeed_dbg_select`	`<0..4>`	Write `lowspeed_dbg_select`.
`highspeed_dbg_select`	`<0..3>`	Write `highspeed_dbg_select`.
`upsampler_x/upsampler_y`	`<-32768..32767>`	Write `upsampler_input_x/y` (16-bit signed packed).
`cap_arm`	none	Pulse `cap_arm`.
`cap_done`	none	Read `cap_done`.
`cap_rd`	`<idx>`	Write `cap_idx`, then read `cap_data`.
`cap_enable`	`<0|1>`	Write `cap_enable` and commit.
`cap_beats`	`<N>`	Write `cap_beats` and commit.

4.2 UberDDR3 / S2MM commands

Command	Arguments	Description
`ub_info`	none	Print DDR calibration state and CSR base.
`ub_mode`	none	Print current `cap_enable`.
`ub_setmode`	`<0|1>`	Set `cap_enable` and commit.

Command	Arguments	Description
ub_start	<addr> [beats] [size]	Start S2MM using current capture mode.
ub_ramp	<addr> [beats] [size]	Force ramp mode, then start S2MM.
ub_cap	<addr> [beats] [size]	Force capture mode, then start S2MM.
ub_wait	none	Poll DMA busy until done; flush caches; report error.
ub_hexdump	<addr> <bytes>	Hexdump DDR region (direct memory read).
ub_send	<addr> <bytes> <dst_ip> <dst_port>	Stream DDR region to PC via UDP using UBD3 header framing.

5 Read/Write Workflow Recipes

5.1 Capture-to-DDR recipe

Typical sequence:

1. Select mode: `cap_enable 0` for ramp or `cap_enable 1` for capture.
2. Set length: `cap_beats <N>`.
3. Start DMA: `ub_start <addr> <N>`.
4. Wait: `ub_wait`.
5. Optional verify: `ub_hexdump <addr> <bytes>`.
6. Stream to PC: `ub_send <addr> <bytes> <ip> <port>`.

6 Appendix: Notes and TODO