ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-WE1BC007

THE LCLS-II PRECISION TIMING CONTROL SYSTEM*

T. Johnson †, M. Browne, C. Pino
SLAC National Accelerator Laboratory, Menlo Park, USA

Abstract

The LCLS-II precision timing system is responsible for the synchronization of experiment optical lasers with the LCLS-II XFEL. The system uses both RF and optical references for synchronization. In contrast to previous systems used at LCLS the optical lasers are shared resources, and must be managed during operations.

The timing system consists of three primary functionalities: RF reference distribution, optical reference distribution, and a phase-locked loop (PLL). This PLL may use either the RF or the optical reference as a feedback source. The RF allows for phase comparisons over a relatively wide range, albeit with limited resolution, while the optical reference enables very fine phase comparison (down to attoseconds), but with limited operational range.

These systems must be managed using high levels of automation. Much of this automation is done via high-level applications developed in EPICS. The beamline users are presented with relatively simple interfaces that streamline operation and abstract much of the system complexity away. The system provides both PyDM GUIs as well as python interfaces to enable time delay scanning in the LCLS-II DAQ.

BACKGROUND

What is "Timing"?

LCLS is a pulsed machine, capable of operating at a variety of fixed frequencies, depending on machine state and experiment need. The X-ray pulses emitted by LCLS are on the order of 100's to 10's of femtoseconds (fs) in pulse duration. The optical lasers used to perform experiments at LCLS often have pulse durations on the same order, leading to extremely tight synchronization requirements for optical pump-probe experiments [1]. The precision timing system provides a level of synchronization beyond that which can be achieved with the event timing system. This system is capable of not only synchronizing the fs pulses, but also controlling relative arrival time of the pump laser with respect to the X-rays, enabling optical pump-probe experiments that investigate transient states of matter [1].

Changes for LCLS-II

The LCLS-I laser locker system has been described previously [2]. This system was constructed using a combination of commercial and SLAC-designed hardware, and uses a combination of EPICS [3] and Python [4] code to create a high level application for facility users. This system has been used quite successfully since its inception, with every hutch at LCLS utilizing these systems for Ti:Sapphire laser experiments. However, to meet the precision timing needs of LCLS-II, a new system had to be developed in order to accommodate changes in both the timing requirements as well as changes in the physical layout and deployment of the laser systems. The major timing changes from LCLS-I to LCLS-II are outlined below.

- RF reference. The LCLS-II precision timing system uses the Phase Reference Line (PRL) RF reference used by the LCLS-II accelerator for synchronization of accelerator hardware.
- Laser technology. The LCLS-II laser systems are based on OPCPA technology, rather than the Ti:Sapphire lasers used in LCLS-I, allowing for higher laser repetition rate.
- Laser delivery. Rather than installing a laser system in every hutch for every interaction point, shared laser systems are used in LCLS-II. An evacuated, remotely controlled laser beam transport system is used to deliver beam from any of 4 possible laser bays in the NEH to any experiment interaction point.
- Optical locking. The LCLS-II precision timing system offers two modes of phase locking: RF locking, and optical locking.

SYSTEM FUNCTIONALITY

The LCLS-I precision timing system had a high level of automation, providing automated system operation and a single "target time" control point, controlling the arrival time of the optical laser pulses with respect to the X-ray pulses. The changes from LCLS-I to LCLS-II described above have necessitated commensurate changes in the LCLS-II precision timing system. The LCLS-II precision timing system implements many of the features present in LCLS-I, as well as others that are completely new for LCLS-II. The major features of the LCLS-II precision timing system are listed below.

- RF reference distribution.
- Frequency locking.
- RF phase locking.
- RF bucket jump detection and correction.
- Target time control and read-back.
- Automated lock transition.

* WORK SUPPORTED BY U.S. D.O.E. CONTRACT DE-AC02-76SF00515
† tjohnson@slac.stanford.edu

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-WE1BC007

- System calibration.
- Optical reference distribution.
- Optical phase locking.
- Multi-user arbitration.

CORE SUBSYSTEMS

RF Reference Distribution

The RF distribution system is responsible for the transport and distribution of the RF reference. An overview of this system is shown in Fig. 1. The LCLS-II system utilizes RF-over-Fiber (RFOF) technology, rather than transmission over stabilized coaxial cable. Diurnal temperature changes can exceed 10 °C per hour, and more than 20 °C peak to peak. Such temperature variations contribute to observed drift at the point of use, approximately 40 fs/m/K for the SMF-28 telecommunications grade single mode fiber used in this system. Such changes must compensate for these changes while not contributing significant additive jitter to the signal.

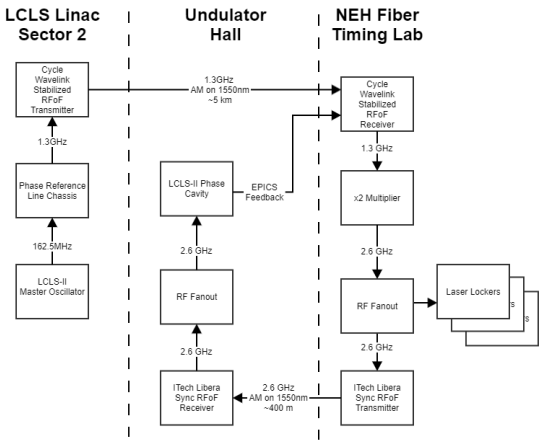

Figure 1: Overview of LCLS-II RF Distribution.

Two commercial RFOF systems were purchased for this purpose: the Wavelink system [5] from Cycle GmbH, and the Libera Sync [6] system from Instrumentation Technologies. The Wavelink system is responsible for transport of the PRL signal from its origin in Sector 2 of the LINAC to the NEH Fiber Timing Lab (FTL). This is the most demanding of the two fiber runs, extending over 5 km, much of which through buildings without climate control. The Libera Sync system is used to transport the RF reference from the FTL to the Undulator hall, where the RF reference is used by the LCLS-II phase cavity. This run is shorter, approximately 400 m, and is entirely within climate controlled buildings.

The LCLS-II phase cavity is used to measure the arrival time of the LCLS-II electron bunch with respect to the stabilized RF reference. A feedback system is used to keep the RF reference phase stable with respect to the beam arrival time. This drift corrected signal is then distributed to various points of use, such as the LCLS-II laser locker systems.

RF Phase Locking

RF Phase locking system used for LCLS-II is analogous to that used for LCLS-I, though the hardware and software has changed significantly. A block diagram of the RF locking scheme used is shown in Fig. 2. The particular example in Fig. 2 is that used for the Pulsed Fiber Timing System (PFTS), described in the following section.

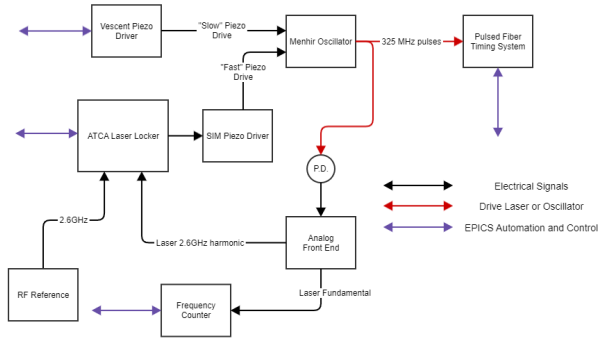

Figure 2: Overview of RF Locking.

The laser oscillator provides two piezo stages for adjusting the oscillator frequency. These are used by the laser locker to tune the frequency of the oscillator as well as adjust the phase. Two different piezo drivers are used: a commercial piezo driver from Vescent [7], and a low noise, high speed piezo driver developed in-house at SLAC. This fast piezo driver is the same design as used in the LCLS-I laser locker. The Vescent piezo driver is used to adjust the coarse control piezo motor, while the SLAC piezo driver is used for the fine control piezo motor.

The RF locking process begins with frequency locking. An EPICS EPID [8] is used to create a PID loop on the oscillator frequency as measured by a frequency counter. The output of the EPID record is applied to the coarse piezo. This feedback loop is capable of adjusting the frequency of the oscillator to within approximately 0.1 Hz of the setpoint.

Once the oscillator frequency has been adjusted to within 1-2 Hz of the setpoint (”frequency locked”), RF lock can be enabled. RF locking is performed using a custom phase locked loop implemented in firmware. The firmware was originally developed for the UED instrument at SLAC [9], this system has been modified and extended for use for LCLS-II. The LCLS-II laser locker utilizes hardware and EPICS drivers originally developed for the LCLS-II LINAC (the ”Common Platform”). The Common Platform is based on ATCA technology. The laser locker utilizes two boards originally designed for the LCLS-II low-level RF (LLRF) system: the LLRF down-mixer, and LLRF up-converter.

A photodiode is used to measure the output of the oscillator, creating an electrical signal that is then conditioned by an analog front end. The 2.6 GHz harmonic of the oscillator signal is compared to the 2.6 GHz RF reference provided by the RF distribution system within the RF down-mixer board. The phase difference between the two signals is calculated,

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-WE1BC007

and used to apply a correction to the fast oscillator piezo motor.

In addition to phase locking, the LCLS-II laser locker provides the ability to apply an overall phase shift to the oscillator with respect to the RF reference. A phase accumulator within the firmware keeps track of the total applied phase shift, enabling the system to reliably phase shift over many RF cycles.

Optical Phase Locking

In addition to RF locking, the LCLS-II precision timing system provides the facility for locking the drive laser to an optical reference. The optical locking system is based on the PULSE timing distribution system from Cycle GmbH [10]. This system splits an optical reference laser (the optical master oscillator, or "OMO") signal and utilizes feedback systems to stabilize the output optical signal. By RF locking the OMO to the LCLS-II RF reference (see Fig. 2), an extremely low jitter, low drift optical reference is created. This reference may then be distributed to points of use in the NEH laser hall and experiment hutches.

This stabilized optical reference can be used to make a more direct phase measurement than what is obtained when RF locking. Commercial two-color balanced optical cross-correlators (TCBOCs), available from Cycle GmbH [11], effectively perform an optical phase comparison, generating an error signal based on the level of temporal overlap of the two optical inputs. These TCBOC measurements are extremely sensitive, up to 5 mV/fs.

At LCLS, the PULSE timing distribution system has been purchased and installed with 7 available stabilized timing links. Three links are required for any one experiment; this configuration allows for two hutches to operate simultaneously, with one additional link to be used as a spare and for offline development. A fiber optic matrix switch is used as a multiplexer, enabling these 7 links to service 40 different timing endpoints across the NEH. The combination of the PULSE timing distribution system and fiber optic matrix switch is colloquially known at LCLS as the Pulsed Fiber Timing System (PFTS).

An overview of the system designed for LCLS-II is shown in Fig. 3. For a typical experiment, it is expected that up to 3 TCBOCs will be used to optically lock the system: directly after the oscillator, near the interaction point, and near the X-ray arrival time monitor (ATM). The signal of the oscillator TCBOC is digitized by a baseband ADC in the ATCA laser locker down-mixer. The oscillator frequency (65 MHz) exceeds the bandwidth of the balanced optical detector in the TCBOC, resulting in a DC error signal. For experiments, the output laser repetition rate will not exceed the LCLS-II X-ray beam rate (1 MHz), and thus the TCBOC pulse can be resolved. In this case, the TCBOC output must be digitized as a waveform and integrated to calculate the phase difference between the two input pulses. This is accomplished with SLAC-designed and -built digitizers, called "Wave8" digitizers. These waveform digitizers feature 8 channels, and utilize a custom protocol, PGP, for control and data exchange. Signal integration is performed in firmware, with the calculated value reported as a PV, and also integrated into the PGP data stream. The Wave8 digitizers are multiplexed, allowing one laser locker to utilize PGP data from any Wave8 in any hutch. This is accomplished via a PGP data switch, which allows for re-routing of input PGP data to the PGP input port of the laser locker.

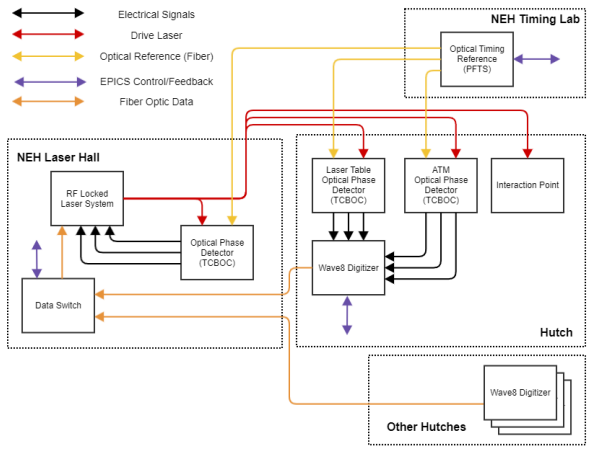

Figure 3: Overview of Optical Locking Scheme.

SOFTWARE

EPICS Automation

As mentioned previously, the LCLS-I laser locker uses a combination of EPICS automation and Python to create a high level application (HLA) for the user. This scheme has advantages, and has been used successfully for nearly 10 years, but is not without drawbacks. Notably:

- High coupling: The LCLS-I laser locker IOC and Python script are highly coupled. One does not function without the other.
- Low cohesion: The Python script used in the LCLS-I laser locker is a super-loop, continually checking PVs and changing operating mode as needed. Almost all automation in the system is performed within this loop.
- Reliability: EPICS is a battle-tested control system framework. Python is a mature programming language, but brings its own dependencies and is an additional layer on top of the EPICS control system stack.
- Simplicity: EPICS records can, in some ways, be simpler than highly OO Python classes. They are atomic units of functionality that may be linked together to achieve a particular behavior, much like an electronic circuit. Most of what the LCLS-I laser locker HLA is doing is: read a few PVs, make a calculation, write to a few PVs. This is very much in EPIC's wheelhouse.

Based on the points above, the decision was made to try to automate the LCLS-I laser locker, and timing system in

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-WE1BC007

general, using pure EPICS level automation. The majority of the functionality required is handled readily by EPICS base or some external records (e.g. the EPID record). However, in order to meet the requirements of the system, two missing functionalities were identified:

- Mutual exclusion: EPICS base does not provide a simple way to "lock" a resource from others at the record level. The LCLS-II precision timing system is a system of shared systems, and requires a simple way of signaling that a resource, e.g. a timing link or laser locker, is currently in use by another user.
- Simple automation routines: One of the reasons Python was used in the LCLS-I laser locker is that it is very easy to perform simple automation. For/while loops with branching logic, error handling, and so on are not easy to accomplish with EPICS base.

To meet these needs, two custom record types were developed at SLAC: the arbiter record, and the stepSequence record. These will be described in further detail in the following sections.

Arbiter Record

To help provide a mutual exclusion facility, the arbiter record was developed. The main fields of this record can be seen in Table 1. This record provides a simple and consistent interface for resource arbitration. This record does not truly lock a resource; it is a signaling mechanism to be used by other records, hence the name "arbiter". Up to 30 different "owners" of a resource may request to be the current owner of a resource via REQx fields. The arbiter record provides string fields for defining human readable names for each of these potential owners.

When a request is made, the arbiter record checks if the VAL field is > -1 (un-owned). If not, the VAL field is set to the integer REQx line that was set, the OWNER field is set to OWNx, and the REQx field is reset. If the resource is owned, then the request is ignored, but the REQx input is left high. If the resource is later released by the owner, then the previous request will be accepted if the REQx field is not modified by the requester. Request lines are checked in order, from REQ0 to REQ29. The STATE field is used to fairly check requests. This field keeps track of the previous owner. Once the current owner releases the resource, the record searches for new requests starting from this point, giving all request lines approximately the same priority.

stepSequence Record

To aid in the development of simple EPICS-level automation, the stepSequence record was developed. The main fields of this record can be seen in Table 2. This record was developed with the aim in mind of creating a simple and consistent state machine interface for interacting with EPICS automation. In any automation routine, there are three main concerns: starting the routine, stopping the routine, and checking the routine's status. This interface is provided by the stepSequence record's primary operator interface fields: REQ, ABRT, and STEPNAME.

The REQ and ABRT fields may be written to start or stop the stepSequence, respectively. The STEPNAME provides a user readable string to indicate the state of the stepSequence. The stepSequence record provides PREx and STEPNAMEx fields that are concatenated to build a complete description of the current state.

The stepSequence state machine can control up to 10 individual sub-steps. More complicated routines can be achieved by chaining together multiple stepSequences, creating arbitrarily long sequences. This is done by linking the ABRT, REQ and STATE fields of the sub-sequence to the ABRTx, REQx and STATEx fields of the parent sequence.

To aid in the use of the stepSequence record, an EPICS database preprocessor was developed. The preprocessor is integrated into the build system. During the build process, any database files suffixed with .dbs (rather than .db) are processed and converted into .db files for the resulting build. This preprocessor has been designed to recognize macros that represent many common programming structures (FOR, WHILE, IF/ELSE, etc). These macros can be used within .dbs files to generate the appropriate combination of stepSequence records and EPICS base records that accomplish the desired behavior.

An example from a .dbs used in the PFTS manager IOC is shown below. The "sequence" macro indicates that this should be decomposed into a stepSequence. The SET_INT and WAIT macros are used for writing long values and waiting for a number of seconds, respectively.

```
record(longout, $(P):$(H):R$(N):LAST_REQ) {
    field(PINI, "YES")
    field(VAL,  "0")
}

record(longout, $(P):$(H):R$(N):SRC_REQ) {
    field(VAL,  "0")
    field(FLNK, "$(P):$(H):R$(N):SRC_OWNREQ_ASUB")
    field(PINI, "YES")
    info(autosaveFields, "VAL")
}

sequence($(P):$(H):R$(N):REQ_SRC) {
SET_INT("Start Operation", $(P):$(H):R$(N):LAST_REQ, 0)
SET_INT("Src ownership", $(P):$(H):R$(N):SRC_REQ, 1)
WAIT("Waiting for src ownership",
     $(P):$(H):R$(N):SRC_OWNED, 5)
}
```

The above stepSequence macros are automatically decomposed into the records shown below by the database preprocessor. As can be seen in the resulting output, use of the stepSequence and stepSequence preprocessor allows for an IOC developer to easily generate EPICS automation code that can be difficult and tedious to produce by hand.

```
record(longout, $(P):$(H):R$(N):LAST_REQ) {
    field(PINI, "YES")
    field(VAL,  "0")
}
```

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-WE1BC007

Table 1: stepSequence Record Fields

Field	Summary
REQ0 to REQ29	30 separate unsigned long request lines. Write a non-zero value to request the arbiter, and zero to release it, or cancel the request if it hasn't been granted yet. These can be used as recursive locks if you just increment/decrement the field.
VAL	A long value indicating the current owner, or -1 if it is not currently owned.
OWN0 to OWN29	30 string input links. These give names to the thirty input requests.
STATE	Field to keep track of the previous owner. Used internally to fairly assign ownership.
OWNER	A string value giving the name of the current owner. If the corresponding OWN0-OWN29 field is not defined, this will be "REQn" if the owner is n, or "None" if the arbiter is not owned.
CLEAR	Writing a non-zero value will clear all requests and release the arbiter. This field is cleared after handling the request.

Table 2: stepSequence Record Fields

Field	Summary
VAL	The current step number, 0-9.
STEPNAME	The current step description. If the stepSequence is done, this is "DONE". If running, this is a combination of the current step prefix (PRE0 ... PRE9) and the step name of the current substep (the value of STEPNAME0 ... STEPNAME9). If both of these are not empty, this name will be both of these separated by ": ". If we are aborted, this will be the last running state name prefixed by "ABORT: ".
WAIT	Are we currently waiting for a substep to complete?
DLY	After asserting a substep request, delay for this number of seconds before checking the substep state.
DLYING	Are we currently delaying before checking a substep state?
REQ	The request interface for the stepSequence. Write a 1 to start the steqSequence.
ABRT	The abort interface for the stepSequence. Write a 1 to abort the stepSequence.
CLR	Write a 1 to move from the Aborted state to the Done state.
STATE	The state interface for the stepSequence.
REQ0 to REQ9	Links to request signals for each of the sub-steps.
ABRT0 to ABRT9	Links to abort signals for each of he sub-steps.
STATE0 to STATE9	Links to states for each of the sub-steps.
STEPNAME0 to STEPNAME9	State name strings for each substep.
PRE0 to PRE9	State name prefix strings for each sub-step.

```
record(longout, $(P):$(H):R$(N):SRC_REQ) {
    field(VAL,  "0")
    field(FLNK, "$(P):$(H):R$(N):SRC_OWNREQ_ASUB")
    field(PINI, "YES")
    info(autosaveFields, "VAL")
}

record(stepSequence, $(P):$(H):R$(N):REQ_SRC) {
    field(PRE0, "Start Operation")
    field(REQ0, "$(P):$(H):R$(N):REQ_SRC:_V0.PROC")
    field(PRE1, "Src ownership")
    field(REQ1, "$(P):$(H):R$(N):REQ_SRC:_V1.PROC")
    field(PRE2, "Waiting for src ownership")
    field(REQ2, "$(P):$(H):R$(N):REQ_SRC:ST2.PROC")
    field(ABRT2, "$(P):$(H):R$(N):REQ_SRC:AB2.PROC")
    field(STATE2, "$(P):$(H):R$(N):REQ_SRC:_S2 CPP")
}

record(longout, $(P):$(H):R$(N):REQ_SRC:_V0) {
    field(VAL, "0")
    field(OMSL, "supervisory")
    field(OUT, "$(P):$(H):R$(N):LAST_REQ PP")
}

record(longout, $(P):$(H):R$(N):REQ_SRC:_V1) {
    field(VAL, "1")
    field(OMSL, "supervisory")
    field(OUT, "$(P):$(H):R$(N):SRC_REQ PP")
}
```

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-WE1BC007

```
record(seq, $(P):$(H):R$(N):REQ_SRC:ST2) {
    field(SELM, "All")
    field(DOL1, "1")
    field(LNK1, "$(P):$(H):R$(N):REQ_SRC:_S2 PP")
    field(DOL2, "$(P):$(H):R$(N):REQ_SRC:TM2 NPP")
    field(LNK2, "$(P):$(H):R$(N):REQ_SRC:CT2 PP")
}

record(longout, $(P):$(H):R$(N):REQ_SRC:AB2) {
    field(VAL, "2")
    field(OUT, "$(P):$(H):R$(N):REQ_SRC:_S2 PP")
}


record(longout, $(P):$(H):R$(N):REQ_SRC:_S2) {
    field(VAL, "0")
}

record(longout, $(P):$(H):R$(N):REQ_SRC:TM2) {
    field(VAL, "5")
}

record(longout, $(P):$(H):R$(N):REQ_SRC:CT2) {
    field(VAL, "0")
}
```

An example of an automation routine achieved with the stepSequence framework is the calibration of the LCLS-II laser locker manager. An example user interface is shown in Fig. 4. The user has a few simple controls, as well as start/stop buttons and the system state. This routine steps the ATCA laser locker phase shifter through a range of phase shift and captures the measured time offset between a fixed trigger and the laser pulse at each step. This creates a mapping between oscillator pulse arrival and phase shift. The phase is swept through several oscillator pulses, and the phase shifter is then adjusted to the central time between two pulses. This becomes the new phase shifter "zero" value. The stepSequence is used to perform this calibration routine, executing the for loop while performing error handling and data validation.

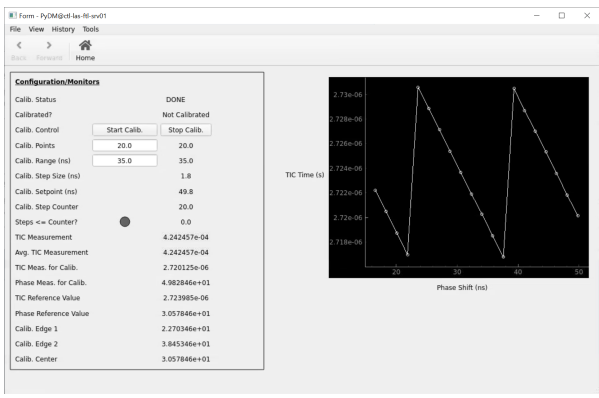

Figure 4: Example calibration using the stepSequence record.

The Laser Locker Manager and the PFTS Manager

Utilizing the tools described above, two high level applications have been developed for the LCLS-II precision timing system: the Laser Locker Manager (LLM), and the PFTS Manager (PFTSM). The LLM is analogous to the LCLS-I HLA; the PFTSM does not have an analog in LCLS-I. There is significant complexity to these systems; the LCLS-II precision timing system spans the entire facility. It is the responsibility of these HLAs to provide a clean user interface, and reliable operation while meeting the needs of LCLS-II.

The LLM is responsible for managing the synchronization and temporal overlap of the X-rays and optical drive laser. Short of RF and optical reference distribution, the LLM is responsible for all the functionality described in System Functionality.

An example PyDM [12] user interface is shown below in Fig. 5. This screen has been designed to look similar to the LCLS-I laser locker screen to aid in operator training. Relatively few user controls are required; a few configuration parameters are available and, most important, target time control. Expert screens for all features are accessible from this user panel, so that expert users may debug further if problems exist. One example is the Calibration expert screen shown in Fig. 4.

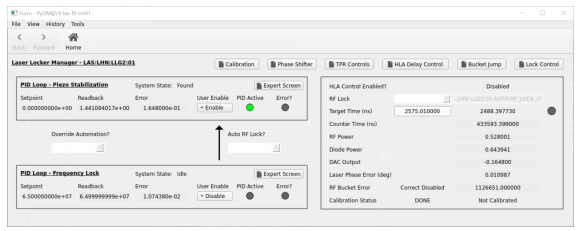

Figure 5: Example Laser Locker Manager User Interface.

An example user interface for the PFTSM is shown below in Fig. 6. This screen shows the use of the arbiter record quite clearly. The user has the ability to request between one and three PFTS links. These requests initiate a stepSequence which checks if any links are already assigned, requests them if not, and waits for ownership status to be returned. The state of each of these stepSequences ("DONE") is seen in the screen, as well as FLS (stabilized fiber links) ownership status. The current link owner, as captured by the individual link arbiter, is also displayed to help other notify other hutches of the current owner. Auto Overlap, another stepSequence responsible for locating the TCBOC signal in the relevant ADC window, is also executable by the user from this screen. As there is risk of system damage if the links are mishandled, the expert level controls are hidden from the user, and are available only to system operators with the requisite training.

CONCLUSION

The various sub-systems of the LCLS-II precision timing system has been developed and deployed. These systems are beginning RF locked commissioning, with optically locked commissioning to begin in Spring of 2024. Two new EPICS records have been developed to aid in the development of these systems: the arbiter record, and the stepSequence

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-WE1BC007

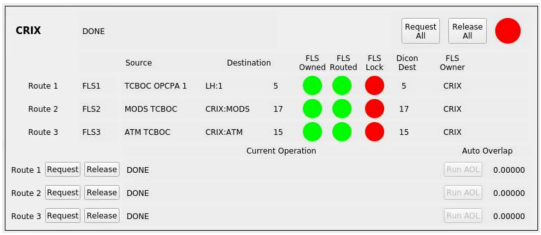

Figure 6: Example PFTS Manager User Interface.

record. These have been used to great effect in the two HLA's that support these systems.

ACKNOWLEDGEMENTS

Along with the listed authors on the paper, the authors would like to credit and acknowledge the following individuals for their past and present contributions to the LCLS-II precision timing system and related controls (in no particular order): Chengcheng Xu, Junyang Xiang, Stefan Droste, Lili Ma, Mike Skoufis, Marcio Paduan Donadio, Karl Gumerlock, Matt Weaver, Huanyu Song, and Joe Frisch.

REFERENCES

[1] J. M. Glownia *et al.*, "Time-resolved pump-probe experiments at the LCLS", *Opt. Express*, vol. 18, no. 17, pp. 17620–17630, 2010. doi:10.1364/OE.18.017620

[2] K. Gumerlock *et al.*, "A Low-Cost, High-Reliability Femtosecond Laser Timing System for LCLS", in *Proc. FEL'14*, Basel, Switzerland, Aug. 2014, paper THP080, pp. 917–921.

[3] EPICS, www.aps.anl.gov/epics/

[4] Python, https://www.python.org/

[5] Cycle GmbH Wavelink, https://www.cyclelasers.com/timing-distribution/wave/

[6] Instrumentation Technologies Libera Sync 3, https://www.i-tech.si/products/libera-sync-3/

[7] Vescent Slice-DHV, https://vescent.com/us/slice-dhv-dual-channel-high-voltage-amplifier.html

[8] The EPID record, https://millenia.cars.aps.anl.gov/software/epics/epidRecord.html

[9] L. Ma *et al.*, "SLAC UED LLRF System Upgrade", *arXiv:1910.02296 [physics.acc-ph]*, Oct. 2019. doi:10.48550/arXiv.1910.02296

[10] The PULSE Timing Distribution System, https://www.cyclelasers.com/timing-distribution/pulse/

[11] TCBOC – Two-Color Balanced Optical Cross Correlator, https://www.cyclelasers.com/synchronization-modules/tcboc/

[12] Python Display Manager (PyDM), https://github.com/slaclab/pydm