Data Acquisition Program for EIC DIRC Quartz Bar Quality Assurance Tests

Tyler Lemon, Mary Ann Antonioli, Peter Bonneau, Aaron Brown, Pablo Campero, Brian Eng, George Jacobs, Mindy Leffel,Marc McMullen, and Amrit Yegneswaran
Physics Division, Thomas Jefferson National Accelerator Facility, Newport News, VA 23606
December 19, 2023

The EIC DIRC quartz bar quality assurance (QA) tests require quantifying the current response of photodiodes induced by the test station laser after passing through the test station optics and quartz bar. To read the photodiode response, a custom transimpedance amplifier PCB was designed [1] and a program was developed that runs on an Arduino Uno R3 that reads from analog digital converters and transmits data via an RS232 serial interface to a local, 20-character-long, 4-row LCD screen. This note discusses the code developed for this program.

The EIC DIRC quartz bar QA program is a variant of C++ that includes specific libraries for running on an Arduino microcontroller. The annotated C++ code developed for the project is in Appendix A.

The QA program can be split into four sections—program initialization (code lines 1–74), device initialization (lines 75–103), function declaration (lines 104–118), and the DAQ loop (lines 119–320).

The initialization portion of the program declares and initializes global variables that will be used throughout the program.

The device initialization portion sets up and establishes communication to the analog digital converters and LCD screen used by the system. This section of code also includes some minor error handling that lets the user know if any analog digital converters or the LCD screen cannot be properly initialized.

The function declaration portion of the code sets up two custom functions—*resetFunc* provides software reset capabilities to the user, allowing the system to be reset via its serial interface (versus power cycling the Arduino or using the Arduino's on-board hardware reset button) and *writeDisplay* provides an interface for the code to write to all four lines of the LCD screen.

The DAQ loop portion is the section of code that runs indefinitely on the Arduino until the board is reset or powered down. This code can be split into two subsections—the DAQ portion (lines 119–182) and a serial command engine (lines 183–320).

The DAQ portion controls I^2C communication to the analog digital converters, reads raw counts from the analog digital converters, converts the raw data into voltages, and the voltages (data type float) into a string data type for writing to the LCD screen and serial interface.

The serial command engine portion allows the Arduino's RS232 serial connection to query or set parameters of the QA program. Table I contains the list of available commands, command syntax, and details of what each command does.

In summary, C++ code has been developed specifically for an Arduino Uno R3 board to acquire and process data for EIC DIRC quartz bar QA tests.

[1] M. McMullen, et al., *Data Acquisition Board for Testing the EIC DIRC Quartz Bars,* DSG Note 2023-54, 2023.

Command	Command syntax	Expected response
Read all voltages	r V	voltage of all channels in volts
Read single channel voltage	r V <ch>	voltage measured by channel in volts
Read sleep interval	r sleep	value of sleep interval
Read status	r status	status as an integer
Read single status bit	r status <bit>	value of bit entered
Read system name	r id	"EIC-DIRC_DAQ_System_0"
Read DAQ period	r daqPeriod	DAQ period value
Write sleep interval value	w sleep <val>	no response expected
Set single status bit	w status <bit> <val>	no response expected
Clear all status bits	w clearStatus	ACK: Status cleared
Perform software reset	w reset	ACK: Resetting…
Set DAQ period	w daqPeriod <val>	ACK: DAQ Period changed to <val>

TABLE I. List of available commands, command syntax, and command function.

APPENDIX A: EIC DIRC QA DAQ PROGRAM C++ CODE

```
/*
EIC - DIRC DAQ program

Tyler Lemon
Detector Support Group
November 2023

Program is written to run on an Arduino Uno R3 for reading from up to four ADC
inputs, displaying that information on a local LCD screen or printing to a
Serial interface.
Program also includes a serial command engine to allow reading/writing of
program parameters without having to re-program the Arduino
*/


#include <Adafruit_ADS1X15.h>                          //Import library for ADCs
#include "Wire.h"                                      //Import library for I2C comm.
#include "Adafruit_LiquidCrystal.h"                    //Import library for LCD screen

Adafruit_ADS1115 adc0;                                 //Set up alias for ADC0
Adafruit_ADS1115 adc1;                                 //Set up alias for ADC1
Adafruit_LiquidCrystal lcd(0);                         //Set up alias for LCD screen

                                                       //Note: for line 14 - 59, code
                                                       //      is declaring and
                                                       //      initializing any
                                                       //      variables needed later

unsigned long daqPeriod = 200; //ms                   //DAQ period

unsigned long tSleep = 120000; //ms                   //Sleep duration that LCD stays
                                                       //      on for
int wakePin = 2; //DIO pin                             //Pin that wake button will use
int wake = 0;                                          //Wake status value
unsigned long tOffset = 0;                             //Holding variable for storing
                                                       //      system time that wake
                                                       //      button was last pressed
int lastPrint = 0;                                     //Holding variable for storing
                                                       //    system time that LCD was
                                                       //      last refreshed at
unsigned int printPer = 500; //ms                      //Period that LCD is refreshed
int16_t res0;                                          //Raw ADC counts for Ch0
float v0;                                              //Voltage read at Ch0
String v0Str;                                          //String for Ch0 measurement
                                                       //      to print to Serial
String v0Msg;                                          //String for Ch0 measurement
                                                       //      to print to LCD
int16_t res1;                                          //Raw ADC counts for Ch1
float v1;                                              //Voltage read at Ch1
String v1Str;                                          //String for Ch1 measurement
                                                       //      to print to Serial
String v1Msg;                                          //String for Ch1 measurement
                                                       //      to print to LCD
int16_t res2;                                          //Raw ADC counts for Ch2
float v2;                                              //Voltage read at Ch2
String v2Str;                                          //String for Ch2 measurement
                                                       //      to print to Serial
String v2Msg;                                          //String for Ch2 measurement
```

```
                                                    //      to print to LCD
int16_t res3;                                       //Raw ADC counts for Ch3
float v3;                                           //Voltage read at Ch3
String v3Str;                                       //String for Ch3 measurement
                                                    //      to print to Seriak
String v3Msg;                                       //String for Ch3 measurement
                                                    //      to print to LCD
float multiplier = 0.1875F; //V\bit                 //Conversion factor for ADC
                                                    //      counts to volts
String cmd = "";                                    //Empty string variable
                                                    //      for storing
                                                    //      received commands
int strLen = 10;                                    //Length of strings to
                                                    //      print on LCD screen
int16_t stat = 0;                                   //16-bit status byte

void setup() {                                      //ENTER SETUP FUNCTION that
                                                    //      only runs once at start
  pinMode(wakePin, INPUT);                          //Set wakePin I/O as an input
  lcd.begin(20, 4);                                 //Initialize LCD screen
  lcd.setBacklight(HIGH);                           //Turn on LCD screen backlight
  lcd.setCursor(0,1);                               //Print start message on screen
  lcd.print("    EIC DIRC DAQ    ");                //      ...
  lcd.setCursor(0,2);                               //      ...
  lcd.print("  system  starting  ");                //      ...
  Serial.begin(9600);                               //Initialize serial connection
                                                    //      with 9600 baud rate
  Serial.println("\nEIC DIRC DAQ Initializing..."); //Print start message to serial

  if (!adc0.begin(0x48)) {                          //If ADC0 is not able to be
                                                    //      connected to:
    Serial.println("Failed to initialize ADC0.");   //Print error message to serial
    bitSet(stat, 0);                                //set ADC0 status bit to 1
  }

  if (!adc1.begin(0x49)) {                          //If ADC1 is not able to be
                                                    //      connected to:
    Serial.println("Failed to initialize ADC1.");   //Print error message to Serial
    bitSet(stat, 1);                                //Set ADC1 status bit to 1
  }

  delay(2000); // ms                                //Wait two seconds
  lcd.clear();                                      //Clear LCD screen
}                                                   //END OF SETUP FUNCTION

void(* resetFunc) (void) = 0;                       //Function for software reset of
                                                    //      program

void writeDisplay(String L0, String L1, \           //Function for printing data to
String L2, String L3){                              //      LCD screen
  lcd.setCursor(0,0);                               //      ...
  lcd.print(L0);                                    //      ...
  lcd.setCursor(0,1);                               //      ...
  lcd.print(L1);                                    //      ...
  lcd.setCursor(0,2);                               //      ...
  lcd.print(L2);                                    //      ...
  lcd.setCursor(0,3);                               //      ...
  lcd.print(L3);                                    //      ...
```

```
}                                                   //      ...

void loop() {                                       //ENTER LOOP THAT RUNS
                                                    //      INDEFINITELY UNTIL
                                                    //      IT IS STOPPED BY POWER
                                                    //      OFF OR RESET


  wake = digitalRead(wakePin);                      //Read wakePin digital input
  if (wake == 1){                                   //If wake = 1;
    tOffset = millis();                             //set tOffset to system time
    lcd.setBacklight(HIGH);                         //turn on LCD backlight
    wake = 0;                                       //Reset wake to zero
  }
  if (millis() - tOffset > tSleep){                 //If sleep period has elapsed
                                                    //      since the wake button
                                                    //      was pressed:
    bitSet(stat,2);                                 //Set sleep status bit
    lcd.clear();                                    //Clear LCD screen
    lcd.setBacklight(LOW);                          //Turn off LCD backlight
  }
  else{                                             //If sleep period has not
                                                    //      elapsed:
    bitClear(stat,2);                               //Clear sleep status bit
  }

  if (bitRead(stat,0) == 0) {                       //If ADC0 status bit = 0:
    res0 = adc0.readADC_Differential_0_1();         //Read ADC0 Ch0 (Ch0)
    v0 = res0 * multiplier / 1000;                  //Ch0 raw ADC counts to Volts
    v0Str = String(v0, strLen);                     //Convert Ch0 to string of
                                                    //      length 10
    v0Msg = "Ch0 = "+v0Str.substring(0,9)+" V";     //Assemble Ch0 message for LCD
    res1 = adc0.readADC_Differential_2_3();         //Read ADC0 ch1 (Ch1)
    v1 = res1 * multiplier / 1000;                  //Ch1 raw ADC counts to Volts
    v1Str = String(v1, strLen);                     //Convert Ch1 to string of
                                                    //      length 10
    v1Msg = "Ch1 = "+v1Str.substring(0,9)+" V";     //Assembly Ch1 message for LCD
  }
  else {                                            //If ADC0 status bit != 0:
    v0Msg = "Ch0 = ADC ERROR ";                     //Assemble error message for
    v1Msg = "Ch1 = ADC ERROR ";                     //      LCD screen
  }

  if (bitRead(stat,1) == 0) {                       //If ADC1 status bit = 0:
    res2 = adc1.readADC_Differential_0_1();         //Read ADC1 Ch0 (Ch2)
    v2 = res2 * multiplier / 1000;                  //Ch2 raw ADC counts to volts
    v2Str = String(v2, strLen);                     //Convert Ch2 to string of
                                                    //      lenth 10
    v2Msg = "Ch2 = "+v2Str.substring(0,9)+" V";     //Assembly Ch2 message for LCD
    res3 = adc1.readADC_Differential_2_3();         //Read ADC1 Ch1 (Ch3)
    v3 = res3 * multiplier / 1000;                  //Ch3 raw ADC counts to volts
    v3Str = String(v3,strLen);                      //Convert Ch3 to string of
                                                    //      length 10
    v3Msg = "Ch3 = "+v3Str.substring(0,9)+" V";     // Assemble Ch3 message for LCD
  }
  else {                                            //IF ADC1 status bit != 0:
    v2Msg = "Ch2 = ADC ERROR ";                     //Assemble error message for
    v3Msg = "Ch3 = ADC ERROR ";                     //      LCD screen
  }
```

```
175
176   if ((millis() - lastPrint) >= printPer){          //If time since last LCD refresh
177                                                     //      has been exceeded:
178     lastPrint = millis();                           //Set lastPrint to current
179                                                     //      system time
180     writeDisplay(v0Msg,v1Msg,v2Msg,v3Msg);          //Update LCD screen
181   }
182
183   if (Serial.available() > 0){                      //If there are bits available at
184                                                     //      the Serial interface:
185     cmd = Serial.readString();                      //Store serial data to cmd
186     String inp = cmd;                               //Initialize inp string, set
187                                                     //      equal to cmd
188     String inpStrings[5];                           //Initialize a 5-element array
189                                                     //      of strings
190     int inpCount = 0;                               //Initialize counter for number
191                                                     //      of arguments in cmd
192     while (inp.length() > 0){                       //While length of inp is > 0:
193       int index = inp.indexOf(" ");                 //Look for spaces in inp
194       if (index == -1){                             //If no spaces found:
195         inpStrings[inpCount++] = inp;               //store inp as first element of
196                                                     //      input array
197         break;                                      //Exit while loop to line 203
198       }
199       else {                                        //If spaces were found in inp:
200         inpStrings[inpCount++] = inp.substring(0,index);    //Split inp at spaces,
201                                                             //      store in array
202         inp = inp.substring(index+1);                       //Cut string at location
203                                                             //      of space, return
204                                                             //      to line 192
205       }
206     }
207
208     String vOut[4] = {v0Str,v1Str,v2Str,v3Str};     //Make an array of the voltages
209     String access = inpStrings[0];                  //first element of input array
210                                                     //      is access mode (r or w)
211     String cmdIn = inpStrings[1];                   //2nd = command to execute
212     String arg0 = inpStrings[2];                    //3rd = command argument 0
213     String arg1 = inpStrings[3];                    //4th = command argument 1
214     String arg2 = inpStrings[4];                    //5th = command argument 2
215
216     if (access == "r"){  //'r' = read               //If command access mode is 'r':
217       if (cmdIn == "V"){                            //  If command in  is "V"
218         if (arg0 != ""){                            //  and arg0 is not empty,
219           int ch = arg0.toInt();                    //  convert arg0 to int
220           if (ch < 4) {                             //  is arg0 <= num. of channels
221             Serial.println(vOut[ch]+"V");           //  print ch.'s data to serial
222           }
223           else {                                    // If arg0 > num. channels
224             Serial.println("ERROR3: Invalid channel"); //   print error message
225           }
226         }
227         else {                                      //If no args with V command
228           Serial.println(vOut[0]+"V;"+vOut[1]+\     //Print to serial all four
229           "V;"+vOut[2]+"V;"+vOut[3]+"V");           //      channels' voltages
230         }
231       }
232
```

```
233
234       else if (cmdIn == "sleep"){                    //If command is "sleep":
235         Serial.println(tSleep);                      //Print sleep timeout duration
236                                                      //      to Serial
237       }
238       else if (cmdIn == "status"){                   //If command is "status":
239         if (arg0 == "") {                            //  and arg0 is empty
240           Serial.println(stat);                      //  print status byte as int
241         }
242         else {                                       //  If arg0 is not empty,
243           int bIn = arg0.toInt();                    //  convert arg0 to int
244           if (bIn > 16) {                            //  Check whether arg0 < 16
245             Serial.println("ERROR4: Invalid status bit"); \\if not print error msg.
246           }
247           else {                                     // Otherwise print, specific
248             Serial.println(bitRead(stat,bIn));       // bit's value to serial
249           }
250         }
251       }
252       else if (cmdIn == "id"){                       //if cmdIn = "id"
253         Serial.println("EIC-DIRC_DAQ_System_0");     //print system name to serial
254       }
255       else if (cmdIn == "daqPeriod"){                //if cmdIn = "daqPeriod"
256         Serial.println(daqPeriod);                   //print error message to serial
257       }
258       else {
259         Serial.println("ERROR2: Command not recognized"); //Print error message if
260       }                                                    //        cmdIn is not
261     }                                                      //        accounted for
262
263     else if (access == "w"){  \\ "w" = write        //If access mode is "w":
264       if (cmdIn == "sleep") {                      //  if cmdIn is sleep
265         if (arg0 != "") {                          //  and arg0 is not empty,
266           tSleep = arg0.toInt();                   //  set tSleep to arg0
267         }
268         else {                                     //If no arg0,//print error msg.
269           Serial.println("ERROR5: Command requires\
270            additional input: w sleep <val>");
271         }
272       }
273       else if (cmdIn == "status"){                 //if cmdIn is "status"
274         if ((arg0 != "") and (arg1 != "")) {       //  and both arg0 and arg1
275           if ((0 <= arg0.toInt() <= 15) and \      //  are not empty and have
276           (0 <= arg1.toInt() <= 1)) {              //  appropriate values:
277             if (arg1.toInt() == 1) {               //if arg1 = 1
278               bitSet(stat,arg0.toInt());           //set bit defined by arg0 to 1
279             }
280             else {                                 //otherwise,
281               bitClear(stat,arg0.toInt());         //set arg0 bit to 0
282             }
283           }
284           else {                                   //If not enough arguments or
285             Serial.println("ERROR6: Command"+\     //      the args are not
286             "requires two inputs: w status"+\      //      within the expected
287             "<bit> <val>");                        //      range, print error msg.
288           }
289         }
290       }
```

```
291
292       else if (cmdIn == "clearStatus") {            //if cmdIn is "clearStatus"
293         stat = 0;                                   //clear status byte to zero
294         Serial.println("ACK: Status cleared");      //print acknowledge message
295       }
296       else if (cmdIn == "reset") {                  //if cmdIn = "reset":
297         Serial.println("ACK: Resetting...");        //print acknowledge message
298         resetFunc();                                //Call software reset func
299       }
300       else if (cmdIn == "daqPeriod") {              //if cmdIn = "daqPeriod":
301         if (arg0 != "") {                           //  and arg0 is not empty:
302           daqPeriod = arg0.toInt();                 //  set daqPeriod to arg0
303           Serial.println("ACK: DAQ Period"+\        //print acknowledge message
304           "changed to"+String(daqPeriod));          //      ...
305         }
306         else {                                      //if arg0 is not present:
307           Serial.println("ERROR7: Command"+\        //print error message
308           "requires additional input: w daqPeriod <val>");
309         }
310       }
311     }
312     else {                                                  //access != w or r
313       Serial.println("ERROR1: Access mode not recognized");     //print error msg.
314     }
315   }
316   delay(daqPeriod);                                 //wait for DAQ
317                                                     //      period to
318                                                     //      elapse
319 }                                                   //END OF MAIN LOOP
320                                                     //RETURN TO LINE 123
```