PRL-TN-2022-114
Technical Note

Liquid Crystal Variable Retarder (Lcvr) Driver For Studying Transient Nematic Effect

by

Shibu K. Mathew, Bireddy Ramya & Ankala Raja Bayanna
(Udaipur Solar Observatory)

भौतिक अनुसंधान प्रयोगशाला, अहमदाबाद
Physical Research Laboratory, Ahmedabad

Published by
The Dean's office, PRL.

LIQUID CRYSTAL VARIABLE RETARDER (LCVR) DRIVER FOR STUDYING TRANSIENT NEMATIC EFFECT

Shibu K. Mathew*, Bireddy Ramya & Ankala Raja Bayanna

Received 10 September 2021; revised 05 January 2022; accepted 07 February 2022; published 16 February 2022.

Abstract
Liquid Crystal Variable Retarders (LCVRs) are widely used for polarization measurements. At Udaipur Solar Observatory we use two LCVRs for the polarization measurements. The measured polarization is utilized to infer the magnetic field on the solar surface. The retardance change in the LCVR on the application of a voltage is made use for modulating the incoming polarization to measurable intensities (Stokes I, Q, U, and V parameters). In solar polarimetry, fast switching of the LCVR retardance (in the order of a few milliseconds) is required to measure polarization states in the same seeing conditions. The switching time required to obtain various polarization states is one of the limitations in using the LCVRs for solar applications. In this report, we describe a precise 'voltage driver circuit' for applying appropriate voltages to the LCVR. We also discuss the application of a high voltage pulse to invoke the Transient Nematic Effect (TNE) in the LCVRs which reduces the transition/switching times and makes it useful for fast polarimetry.

Keywords
Multi-Application Solar Telescope, Polarimetry, Narrow-band Imager

[1] *Udaipur Solar Observatory, Physical Research Laboratory, Udaipur, Rajasthan*
***Corresponding author**: shibu@prl.res.in

Contents

1. Introduction

Liquid Crystal Variable Retarders (LCVR) are used in solar polarimetry for measuring active region magnetic field. The polarimetry is done by measuring the Stokes I, Q, U, and V intensity images by sequentially switching the retardance of the crystals by applying a drive voltage. The rapid variations in the atmospheric condition (seeing) play a negative role in these measurements as it produces distortions in the images taken in sequence. Usually, the difference between the consecutive images taken in orthogonal polarizations is used for producing the Stokes images. As the measurements are sequential, the change in the seeing condition can introduce artifacts in these measurements. Also, the short-term evolution of the solar active regions within the image acquisition period can introduce spurious signals in the measurements. The above facts necessitate the rapid measurements of orthogonal polarization for solar polarimetry. The switching speed of the LCVRs plays a crucial role in solar polarimetry. With the availability of very fast CCD and sCMOS cameras, the main constraint in using the LVCRs for fast solar polarimetry is its response time. In Multi-Application Solar Telescope at Udaipur Solar Observatory (MAST-USO), we use two LCVRs for making the Stokes images for solar magnetic field measurements (Venkatakrishnan, et.al. [2017]; Mathew, et.al. [2017]; Tiwary, et.al. [2017]). The response time of the LCVRs can be minimized by applying a low duration high voltage pulse before the application of the retardance voltage, which in effect results in Transient Nematic Effect (TNE). In this technical report, we give emphasis to the driver circuit used for applying the retardance voltages to the LCVRs. The test results showing the dependence of these pulses on the response time are also discussed.

2. Liquid Crystal Variable Retarder voltage driver

2.1 LCVR construction

LCVR is made of liquid crystals, the birefringence of which can be altered by applying a voltage. The construction of a typical LCVR is shown in Figure 1, the liquid crystal is filled in the cavity made of two highly polished glass windows which are separated by spacers of a few microns. The Indium Tin Oxide (ITO) coating on the inside faces of the windows acts as the electrodes. The tipping of the liquid crystal molecules with applied voltage results in the variation of the birefringence and thus the retardance. We use LCVRs from Meadowlark Optics®.

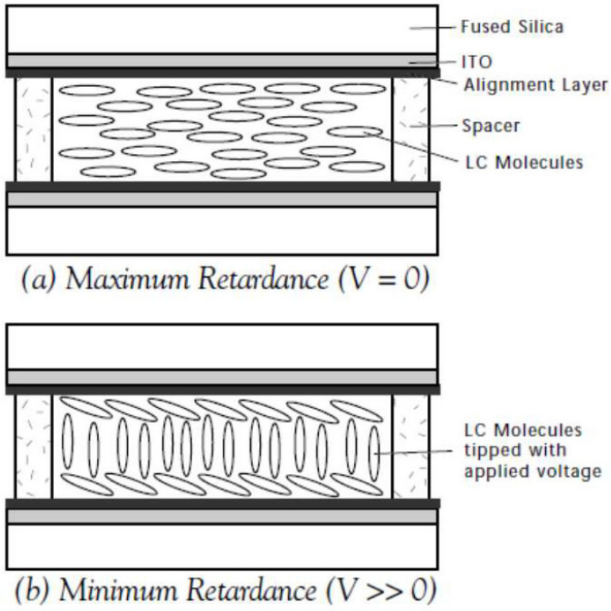

Figure 1. A vertical cut through an LCVR showing the alignment of liquid crystal molecules, with and without an applied voltage. ITO-coated glass plates are used as transparent electrodes for the application of voltage across the LCVR cell (Taken from Meadowlark Optics®, liquid crystal catalogue: `https://www.meadowlark.com/store/pdfs/liquidcrystals.pdf`)

.

2.2 Voltage driver for LCVR

LCVR requires a square ac voltage for changing the retardance. A dc voltage can damage the liquid crystal layer due to ionic build-up (Perlmutter & David Doroski [1996]). Figure 2 shows the waveform of a typical drive voltage used for the LCVR. For driving the LCVRs and testing them, a voltage driver circuit is developed in-house at USO. We follow a similar circuitry of the LCVR driver used in Global Oscillation Network Group (GONG)'s instrument for magnetic field measurements. Figure 3 and Figure 4 show the schematic of the LCVR driver and the circuitry, respectively.

As mentioned earlier, the LCVR requires an ac square wave, with a mean zero drive voltage. For producing the symmetric ac pulses, two precision op-amps, OP37, in inverting and non-inverting modes are used (LM741 can also be used by making appropriate changes in offset-null pin connections). The output of the amplifiers goes to two of the source inputs (S1 & S2) of an analog multiplexer (ADG508, from Analog Devices®), where the select pin A0 is driven by a square wave oscillator (LM555, working at 2 kHz) or a PIC Microcontroller square wave generator having adjustable frequencies, Figure 6 shows the circuit diagram. PORTB of PIC16F873A is used as an input port, the switches connected to pins B7, B6, B5, B4 are used to select the frequency of the clock output pulse. The microcontroller reads the input from PORTB, depending on the switch position it adjusts the frequency of the clock pulses at the output pin RA2. The program for deriving different frequencies from the microcontroller is listed in the Appendix. An external crystal oscillator of 12MHz is used for timing and clock operations. The square wave selects one of the source inputs alternatively which in effect provides a symmetric ac waveform at the output of the multiplexer with a mean zero voltage. From here the output is taken to another buffer op-amp and to a gain stage for further amplification and then to the LCVR. The input voltage to op-amps is jumper selectable

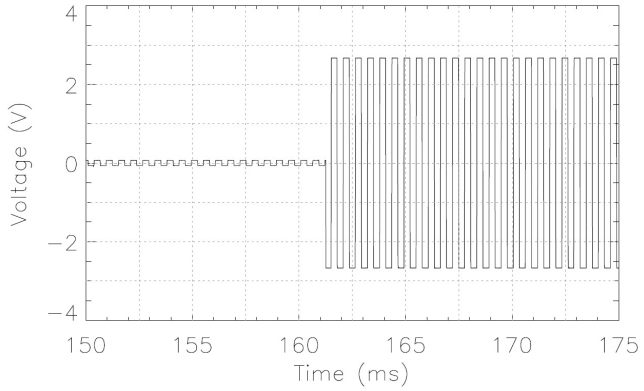

Figure 2. A typical waveform of the voltage applied to the LCVR derived from the designed voltage driver circuitry. A 2 kHz square wave is used for the LCVR, modulated by the amplitude of DC voltage which determines the retardance value.

and can be given from an external voltage source or from a digital to analog converter (DAC). A 16-bit DAC from Texas Instruments® (DAC714P) is used for the latter. The selected DAC can provide output voltages within ±10 V, with a ±1 LSB linearity. The 16-bits give an approximate resolution of 0.3 mV at the DAC output. DAC714 uses serial data input and the input pins can be driven by any compatible serial interface. Figure 5 shows the timing diagram forthe DAC serial data in (SDI)

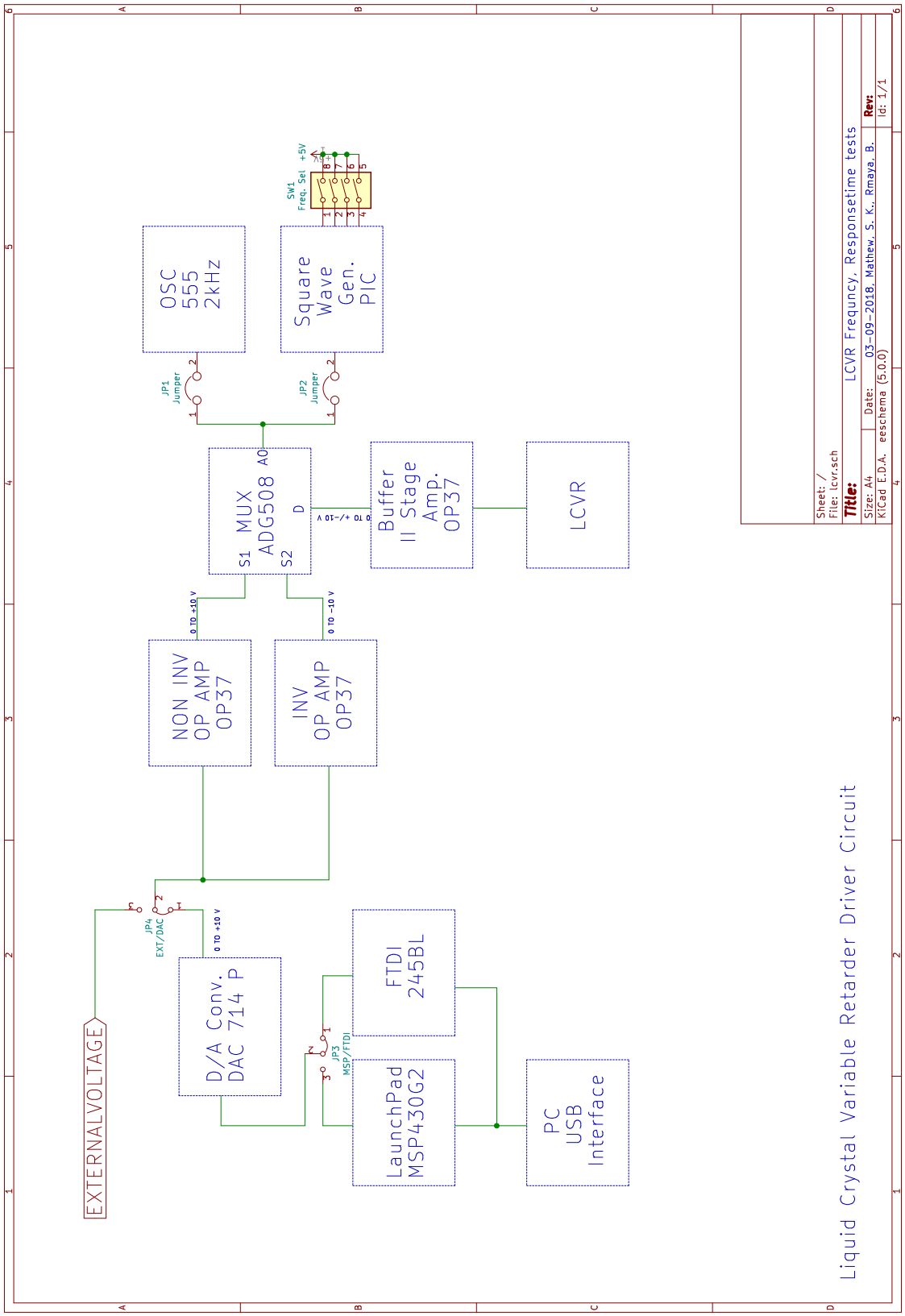

Figure 3. Schematic diagram for the LCVR driver circuit. For the tests we used both Lanuchpad and FTDI for driving the DAC.

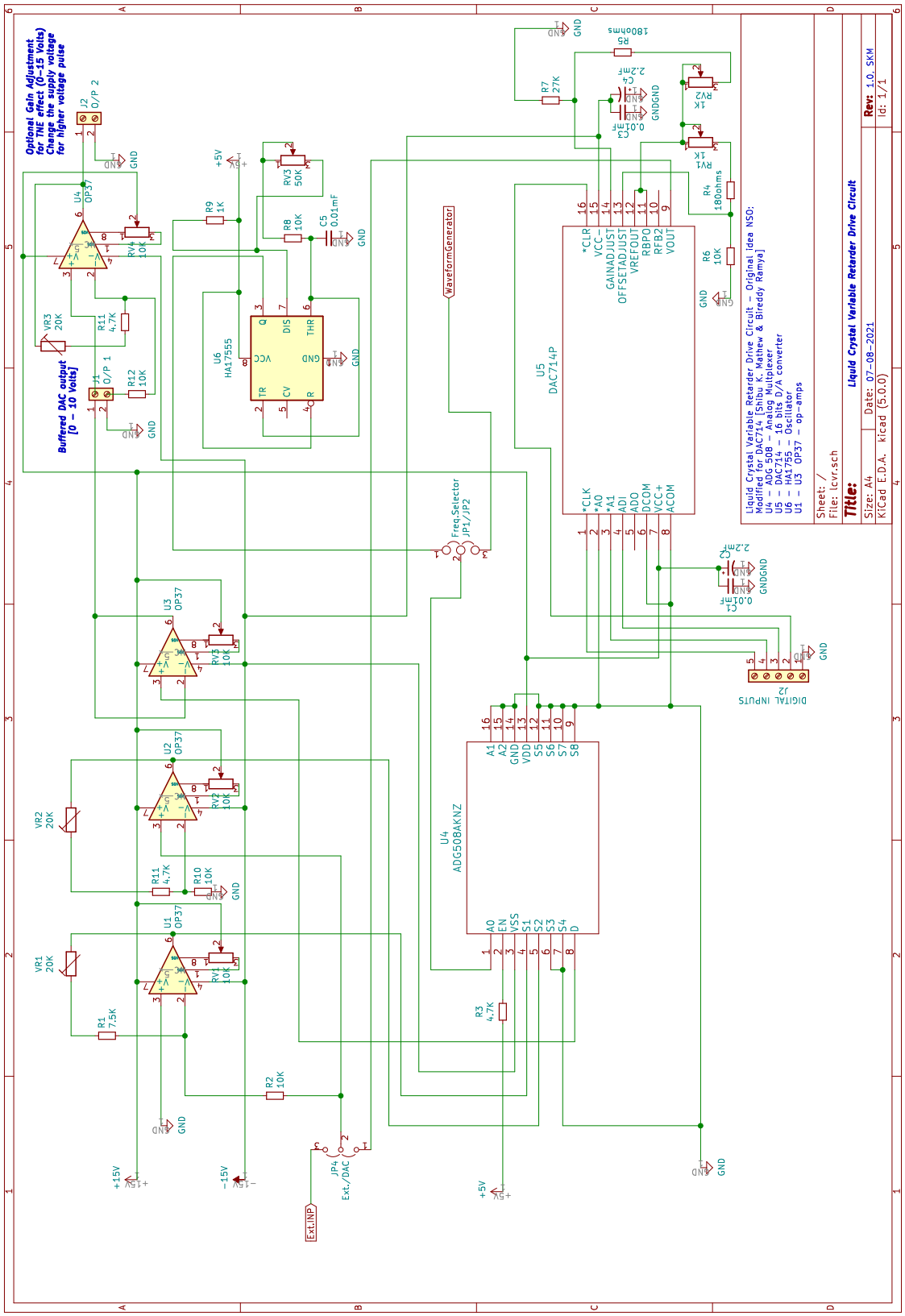

Figure 4. Circuit diagram for the LCVR driver electronics. The op-amp U4 is used for additional gain for producing the short duration high voltage TNE pulses.

along with clock (CLK) and Latch Data ($\overline{A1}$) . We follow this timing diagram shown in Figure 5, with the exception of ($\overline{A0}$) permanently tied to ground in our application. The DAC714 can be driven by a Micro-

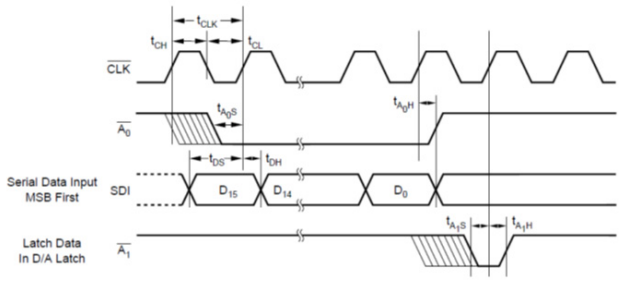

Figure 5. The timing diagram for the serial data in for DAC714. Taken from TI® DAC714 datasheet.

controller or can be directly interfaced to a computer through an FT245BL, USB to parallel FIFO chip from Future Technology Devices International Ltd. (FTDI®). For microcontroller (MSP) option we used a development board – Launchpad, from TI®. The interfacing diagrams for both options are given in Figure 7. While conducting the tests for the TNE effect, the DAC is driven by the Microcontroller to avoid any delay in transmitting the data between the computer and the FTDI chip and thus the uncertainty in the timings.

The Launchpad kit, which employs the MSP430G2 Microcontroller from TI®, is used as the first option for driving the DAC (An Arduino® Uno board can also be used for this purpose). The board provides several GPIO, out of which 4 pins are used for interfacing the DAC. Refer to Figure 7 and the programs in the Appendix for the pin connections (P1.0 → SDI, P1.3 → CLR, P1.4 → CLK, P1.5 → $\overline{A1}$). The onboard USB programmer along with Energia® IDE is used for programming and uploading the firmware. A simple C program or any other serial communication program (like PuTTY/HyperTerminal) can be used for communicating with the Launchpad. An example C program is given in the Appendix. The board could be programmed for any sequence of voltage pulses. The remaining GPIOs in the Launchpad can be used for synchronizing LCVR switching and the data acquisition with other instruments.

The second option used for driving the DAC is by an FTDI® USB to parallel chip. FT245BL is a high-speed FIFO converter, which can be connected to the PC USB port. Four parallel output pins from the FT245BL are used for interfacing with the DAC (D0 → SDI, D1 → CLK, D2 → $\overline{A1}$, D3 → CLR). A sample C program for driving the DAC with an FTDI chip is given in the Appendix. Since the program makes use of the functions from the 'D2XX' driver from FTDI, make sure that the appropriate driver, library, and include files are installed

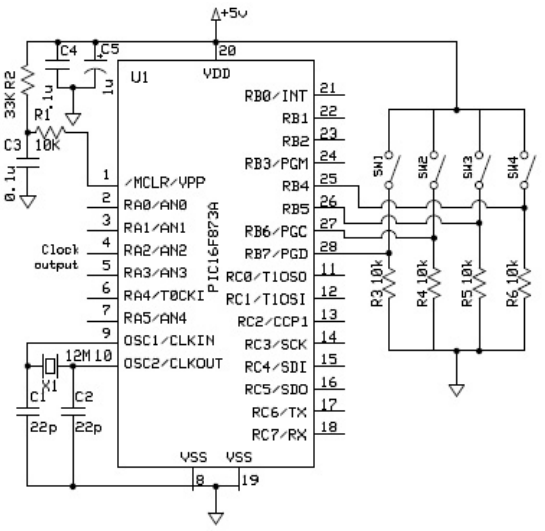

Figure 6. PIC16F873A variable frequency clock generator. Frequencies can be changed by selecting the DIP switches.

before using the program.

The DAC714 is designed to accept binary two's complement (BTC) with MSB first. A full-scale positive voltage is produced for the $7FFF_H$, full-scale negative voltage for 8000_H, and mid scales zero volts for 0000_H. In both the cases of MSP and FTDI, the input voltage is converted into BTC. In the case of MSP, the voltage is transmitted to the microcontroller through the USB and the float to BTC conversion is done in the controller. Each of the 16-bits combined with the clock and other pulses was applied to the GPIO ports in sequence to produce the required clocking of the data. In the case of FTDI, a similar procedure is followed, except the conversion of float to BTC is done in the computer. More details on the conversion and the data clocking are given in the example programs.

3. Experimental set-up to measure the LCVR retardance and response time

In this section, we report a few test results of the measurements carried out using the above circuitry. For measuring various critical parameters of the LCVR an experimental set-up as shown in Figure 8 is used. The LCVR is kept in between a set of linear polarizers; with its fast axis at 45° with respect to one of the linear polarizers. The other polarizer is mounted on a rotation stage (Newport Make), for alternating the polarization angle between 0° and 90°. An LED laser beam passes through the set-up and is imaged on a photo-detector (Centronic®, Series 5T). The photo-detector circuit amplifies the photocurrent to a measurable voltage which is then taken to one of the channels of a USB data acquisition (A/D converter, NI6320 with a capability of

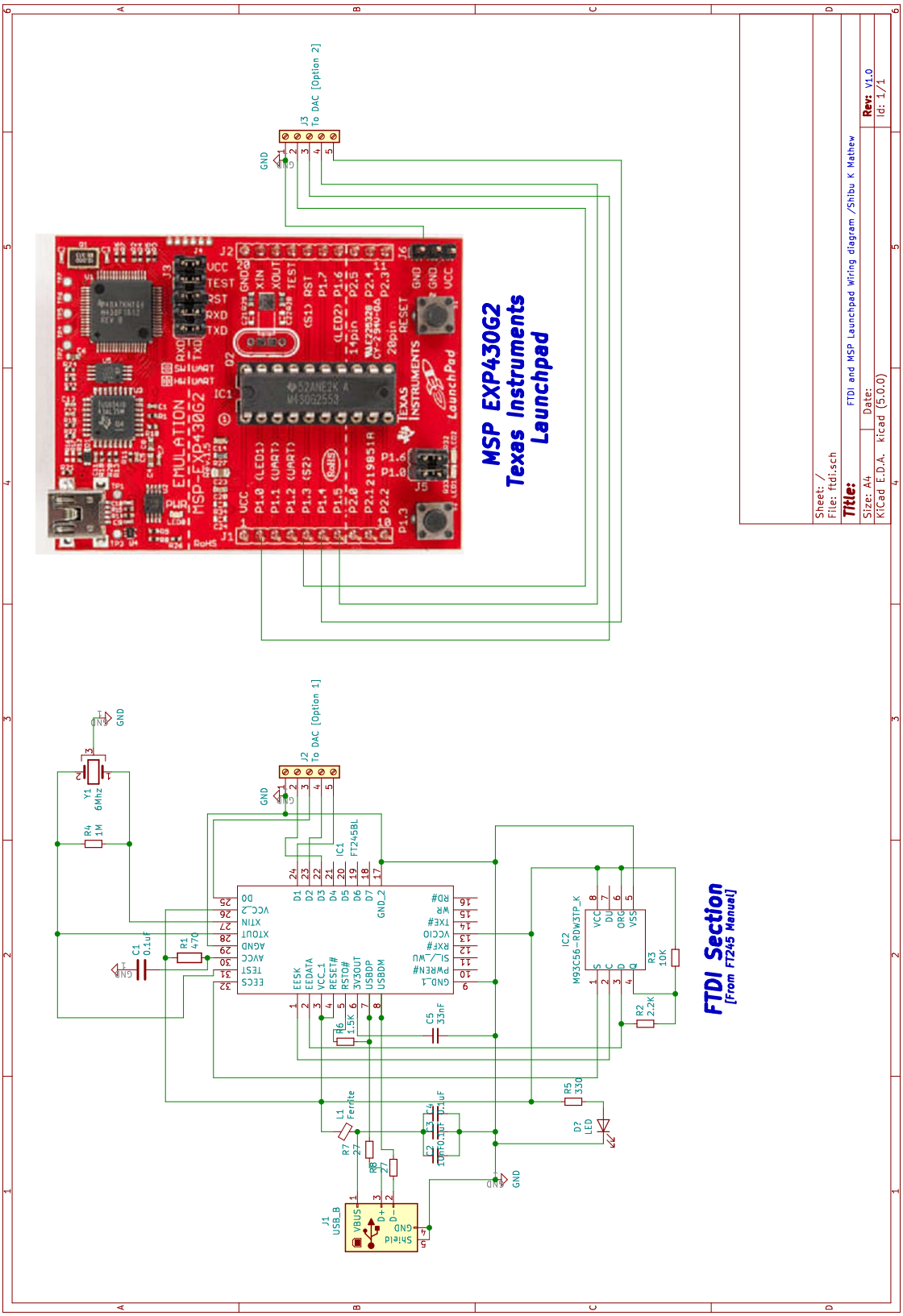

Figure 7. FTDI and TIMSP wiring diagram for driving the DAC. The pin connections are as per the programs listed in the Appendix.

250kS/s) module from National Instruments (NI®). In order to measure the voltages from the photo-detector and that applied to the LCVR simultaneously, another channel of the data acquisition system is used for sampling the DAC voltage. The LCVR voltage, rotation of the linear polarizer, and data acquisition can be controlled and synchronized with the computer.

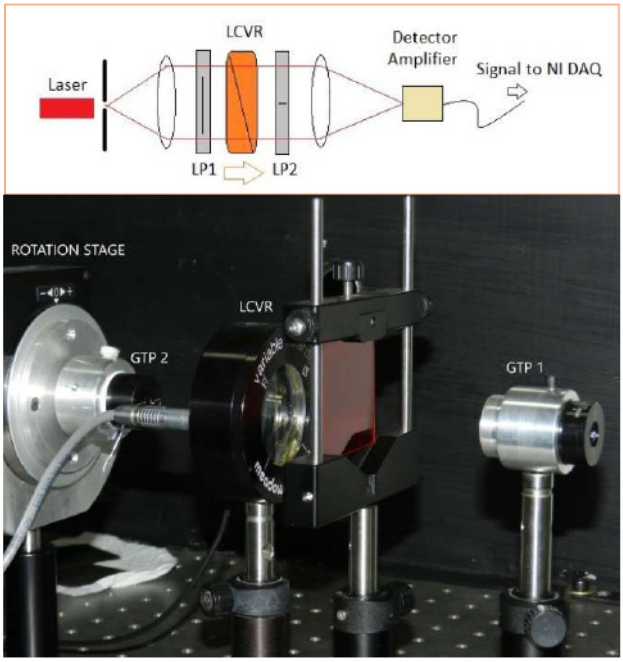

Figure 8. Experimental set-up used for the retardance measurements. GTP1 and GTP2 are Glan-Tylor prisms, the LCVR is kept between GTP1 and GTP2 with its fast-axis at to the GTP1 polarization axis.

3.1 Retardance measurement

For this, the DAC was operated with the FTDI and the computer. Two intensity measurements (i.e. the corresponding voltages from the photo-detector) were taken for each LCVR voltage by making the second linear polarizer (GTP2) parallel and perpendicular (0°, 90 ° → $I_{\|}$, $I_{\perp}$) with respect to the first one. The retardance in degrees can be calculated from these measurements by;

$$\delta_{\nu} = \frac{180}{\pi} \times \left[\frac{I_{\|} - I_{\perp}}{I_{\|} + I_{\perp}}\right] \quad (1)$$

After correcting the phase wrapping, we calculated the half-wave ($\lambda/2$) and quarter-wave ($\lambda/4$) voltages for this particular LCVR as 1.93V and 2.76V, respectively. Figure 9 shows the measured retardance.

3.2 LCVR response time and TNE effect

For conducting this experiment, we use the Launchpad to drive the DAC. The optical configuration similar to the earlier one is used; except that the GTP1 and

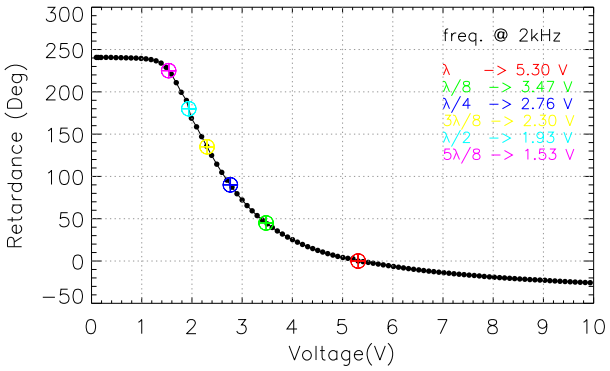

Figure 9. Measured retardance of an LCVR from Meadowlark optics®. A C program was used to automate the entire measurement along with the application of voltages.

the GTP2 were kept in a crossed position. The voltage in the photo-detector, which is a proxy to the intensity change, is recorded along with the applied voltage to the LCVR. Figure 10 shows the result from one such data set, where the LCVR is at $\lambda/4$ retardance voltage. We computed the response time as the time taken for the LCVR to reach a flat retardance from the moment when the voltage is changed. It took around 25 ms to reach a flat retardance. Figure 11 shows the TNE effect, where the application of a small high voltage pulse accelerates the retardance change. The Launchpad is programmed

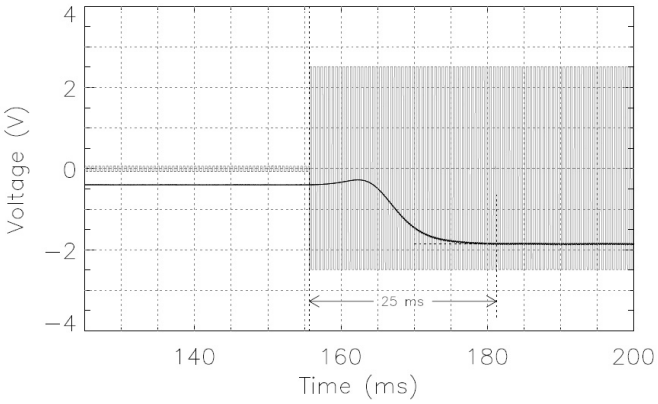

Figure 10. $\lambda/4$ retardance voltage applied to the LCVR. The pulses are voltage applied to the LCVR, whereas the solid line is the response recorded as the intensity change in the output beam. To reach a flat response, the LCVR took around 25 ms.

to drive the DAC with two voltage steps, a small 2 ms, 15V pulse and then with the voltage. It would be possible to use a slightly higher value for the voltage pulse (Meadowlark Optics uses an 18 Volts pulse), we restrict our voltage pulse to 15 volts because using a very high voltage pulse may destroy the molecular alignment per-

manently. The time period of 2 ms is opted to have at least 4 high voltage pulses get applied to the LCVR. The high voltage pulse reduces the response time due to the TNE effect, and the flat $\lambda/4$ retardance is achieved within 17ms. The optimization of the spike amplitude and spike width needs to be done for different retardance values, with which we can further reduce the response time. We have successfully used the fabricated circuit to conduct many more experiments with the LCVR. For example, the variable PIC frequency generator is used for conducting experiments on the frequency dependence of the drive voltage to the retardance changes.

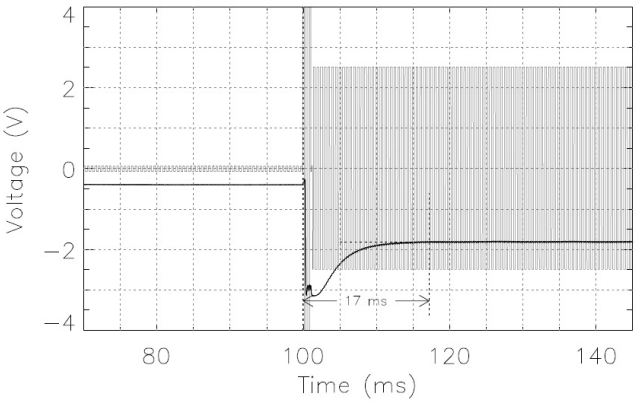

Figure 11. Same as Fig. 10, except a small voltage spike of ±15V for 2 ms applied before the $\lambda/4$ retardance voltage. The response time here was reduced to 17 ms.

4. Conclusion

Liquid Crystal Variable Retarders are widely used for polarimetry applications. In this technical note, we report the development of a voltage driver circuitry, precise voltage source using a 16-bit converter, and its interfacing with the computer are discussed. We also include some of the test results and source codes for driving the LCVRs. We conducted the test on the TNE effect and found that it is possible to reduce the response time of the LCVR by applying a high voltage spike before the application of the required retardance voltage.

Acknowledgment

We acknowledge the contribution of Late Mr. S. K. Gupta for taking part in the design of the circuit, making the schematic, PCB, and some part of the circuit fabrication. We also acknowledge the work of Mr. Divyanshu Tak from Nirma University, Ahmadabad, for helping with some part of the data acquisition work.

The editor Prof. D. Pallamraju, thanks the reviewers Himanshu Mazumdar and Y B Acharya for their assistance in evaluating this Technical Note.

References

Venkatakrishnan P., Mathew S. K., Srivastava N., Bayanna A. R., Kumar B., Ramya Bireddy, Jain N., Saradava M., The multi application solar telescope. *Current Science*, 113(4), 2017.

Mathew S. K., Bayanna A. R., Tiwary A. R., Ramya B., Venkatakrishnan P., First observations from the multi-application solar telescope (mast) narrow-band imager. *Solar Physics*, 292:106, 2017.

Tiwary A. R., Mathew S. K., Bayanna A. R., Venkatakrishnan P., Yadav R. Imaging spectropolarimeter for the multi-application solar telescope at udaipur solar observatory: Characterization of polarimeter and preliminary observations. *Solar Physics*, 292:49, 2017.

Perlmutter S. H., David Doroski, Degradation of liquid crystal device performance due to selective adsorption of ions. *Applied Physics Letters*, 69:1182, 1996.

Appendix

[Note: The program listed here are only for the regular operation of LCVR. Example programs for running the LCVR with a short duration pulse and other codes can be obtained from shibu@prl.res.in. The USB drivers required for operating the TI Launchpad MSP430G2 and FTDI245BL chip can be obtained from the respective company websites.]

A) Energia (Arduino like) sketch for driving DAC using TI Launchpad MSP430G2

```
// Driving the DAC714 TI 16-bit DAC with
// MSP EXP430G2 Launchpad Board
// Shibu K. Mathew
// Programs written for LCVR
// voltage input through PuTTY or any other programs,
// through VCP

#define SDI  01
#define CLK  16
#define AIO  32
#define CLR  08

void setup()
{
pinMode(2, OUTPUT);  // Equivalent number on P1OUT = // 01 -- DAC SDI [Pin no. 04]
pinMode(6, OUTPUT);  // Equivalent number on P1OUT = // 16 -- DAC CLK [pin no. 01]
pinMode(7, OUTPUT);  // Equivalent number on P1OUT = // 32 -- DAC AIO [pin no. 02]
pinMode(5, OUTPUT);  // Equivalent number on P1OUT = // 08 -- DAC CLR [pin no. 16]
pinMode(14,OUTPUT);
Serial.begin(9600);
}
void loop()
{
float inpV=0.0;
if (Serial.available() > 0)
{
inpV=Serial.parseFloat();
Serial.println(inpV, DEC);
int j,dt;
float AFA,dacvol,con,dig_in;
long DDI,MASK;
int i,DD=dig_in,BYTA,td,tdd;
con=32767.0/10.0;
AFA = 1.0;         // If additional amplification
// change the factor
dacvol=inpV/AFA;
dig_in=(float)(dacvol*con);
BYTA=0;
td =1;
tdd=2*td;
MASK=32768;
DDI = dig_in;
if (DDI < 0)                        // Two complement for -ve values
{
DDI=-1.0*DDI;
DDI=~(DDI);
```

```
DDI=DDI+1;
}
for (i=0;i<=15;i++)       // Bit- shifting for faster operations
{
if ((DDI & MASK) == 0)
{DD = 0;}
else
{DD = 1;}
MASK = MASK >> 1;

BYTA   = SDI*DD+CLK*1+CLR*1+AIO*1;
P1OUT  = BYTA;
delayMicroseconds(td);
BYTA   = SDI*DD+CLK*0+CLR*1+AIO*1;
P1OUT  = BYTA;
delayMicroseconds(tdd);
BYTA   = SDI*DD+CLK*1+CLR*1+AIO*1;
P1OUT  = BYTA;
delayMicroseconds(td);
}
delayMicroseconds(td);
BYTA   = SDI*DD+CLK*0+CLR*1+AIO*1;
P1OUT  = BYTA;
delayMicroseconds(td);
BYTA   = SDI*DD+CLK*0+CLR*1+AIO*0;
P1OUT  = BYTA;
delayMicroseconds(td);
BYTA   = SDI*DD+CLK*1+CLR*1+AIO*0;
P1OUT  = BYTA;
delayMicroseconds(td);
BYTA   = SDI*DD+CLK*1+CLR*1+AIO*1;
P1OUT  = BYTA;
delayMicroseconds(td);
digitalWrite(14,HIGH); // Lit the Green LED
delayMicroseconds(td);
digitalWrite(14,LOW);
}
}
```

B) Example C programs for running DAC through Launchpad connected to PC USB port

a) Main program

```
// Main Program calling functions
// used for communicating with Launchpad
// and DAC, DAC voltages provided as floating point
// Shibu K. Mathew, USO/PRL
// From command prompt usage 'LaunchPad voltage'

#include <iostream>
#include <cstddef>
#include <cstring>
#include <stdio.h>
#include <cstdlib>
#include <windows.h>
```

```
#include <stdlib.h>
#include <cstring>
#include <ctime>
#include <time.h>
#include <math.h>
#include <conio.h>
#include <dir.h>
#include "Func.h"
int main(int argc, char *argv[])
{
float inp;
sscanf(argv[1],"%f",&inp);
if (inp < 0.0)
{
printf("Restrict the voltage to positive\n");
printf("DAC will produce negative voltage\n");
printf("but not suitable or required for\n");
printf("LCVR switching \n");
exit(1);
}
HANDLE LaunchPad;

// Check the COM port to which the Launchpad is connected
// change the COM port name accordingly

char port_LaunchPad[]="\\\\.\\COM3" ;
LaunchPad=LaunchPad_GetHandle(port_LaunchPad);
LaunchPad_Write(LaunchPad,inp);
LaunchPad_CloseHandle(LaunchPad);
}
```

b) C functions

```
// TI Lanchpad communication
// Written by Shibu K. Mathew
// Version 1
// General include files

#include <iostream>
#include <cstddef>
#include <cstring>
#include <stdio.h>
#include <cstdlib>
#include <windows.h>
#include <stdlib.h>
#include <cstring>
#include <ctime>
#include <time.h>
#include <math.h>
#include <conio.h>
#include <dir.h>
#include <string>
#include "Func.h"
#define BUFF_SIZ 100
```

```
// Function 1. LaunchPad_GetHandle - SKM
HANDLE LaunchPad_GetHandle(const char * port)
{
HANDLE LaunchPad = CreateFile(port,GENERIC_READ|GENERIC_WRITE, 0,NULL,OPEN_EXISTING,0,NULL);
COMMTIMEOUTS cto = { 1, 100, 1000, 0, 0 };
DCB dcb;
SetCommTimeouts(LaunchPad,&cto);
memset(&dcb,0,sizeof(dcb));
dcb.DCBlength = sizeof(dcb);
dcb.BaudRate = 9600;
dcb.fBinary = 1;
dcb.Parity = NOPARITY;
dcb.StopBits = ONESTOPBIT;
dcb.ByteSize = 8;
SetCommState(LaunchPad,&dcb);
return LaunchPad;
}

//Function 2. LaunchPad_CloseHandle - SKM
void LaunchPad_CloseHandle(HANDLE LaunchPad)
{
CloseHandle(LaunchPad);
}

//Function 3. LaunchPad_Write - SKM
void  LaunchPad_Write(HANDLE LaunchPad, float Value)
{
float orV=Value;
if (Value < 0.0)
{
Value=-1.0*Value;
}
float rem;
int n0,n1,n2,n3,n4,n5,n6,n7,n8;
char inpLaunchPad[8];
DWORD write=0;

n0=int(Value/1000.0);
rem=Value-n0*1000.0;
n1=int(rem/100.0);
rem=rem-n1*100.0;
n2=int(rem/10.0);
rem=rem-n2*10.0;
n3=int(rem);
rem=rem-n3;
rem=int(rem*10000.0);
n4=int(rem/1000.0);
rem=rem-n4*1000.0;
n5=int(rem/100.0);
rem=rem-n5*100.0;
n6=int(rem/10.0);
rem=rem-n6*10.0;
n7=int(rem);

switch (n0)
```

```
{
 case 0: inpLaunchPad[0]='0';
 break;
 case 1: inpLaunchPad[0]='1';
 break;
 case 2: inpLaunchPad[0]='2';
 break;
 case 3: inpLaunchPad[0]='3';
 break;
 case 4: inpLaunchPad[0]='4';
 break;
 case 5: inpLaunchPad[0]='5';
 break;
 case 6: inpLaunchPad[0]='6';
 break;
 case 7: inpLaunchPad[0]='7';
 break;
 case 8: inpLaunchPad[0]='8';
 break;
 case 9: inpLaunchPad[0]='9';
 break;
 }
 switch (n1)
{
 case 0: inpLaunchPad[1]='0';
 break;
 case 1: inpLaunchPad[1]='1';
 break;
 case 2: inpLaunchPad[1]='2';
 break;
 case 3: inpLaunchPad[1]='3';
 break;
 case 4: inpLaunchPad[1]='4';
 break;
 case 5: inpLaunchPad[1]='5';
 break;
 case 6: inpLaunchPad[1]='6';
 break;
 case 7: inpLaunchPad[1]='7';
 break;
 case 8: inpLaunchPad[1]='8';
 break;
 case 9: inpLaunchPad[1]='9';
 break;
 }

 switch (n2)
{
 case 0: inpLaunchPad[2]='0';
 break;
 case 1: inpLaunchPad[2]='1';
 break;
 case 2: inpLaunchPad[2]='2';
 break;
 case 3: inpLaunchPad[2]='3';
```

```
break;
case 4: inpLaunchPad[2]='4';
break;
case 5: inpLaunchPad[2]='5';
break;
case 6: inpLaunchPad[2]='6';
break;
case 7: inpLaunchPad[2]='7';
break;
case 8: inpLaunchPad[2]='8';
break;
case 9: inpLaunchPad[2]='9';
break;
}
switch (n3)
{
case 0: inpLaunchPad[3]='0';
break;
case 1: inpLaunchPad[3]='1';
break;
case 2: inpLaunchPad[3]='2';
break;
case 3: inpLaunchPad[3]='3';
break;
case 4: inpLaunchPad[3]='4';
break;
case 5: inpLaunchPad[3]='5';
break;
case 6: inpLaunchPad[3]='6';
break;
case 7: inpLaunchPad[3]='7';
break;
case 8: inpLaunchPad[3]='8';
break;
case 9: inpLaunchPad[3]='9';
break;
}
switch (n4)
{
case 0: inpLaunchPad[4]='0';
break;
case 1: inpLaunchPad[4]='1';
break;
case 2: inpLaunchPad[4]='2';
break;
case 3: inpLaunchPad[4]='3';
break;
case 4: inpLaunchPad[4]='4';
break;
case 5: inpLaunchPad[4]='5';
break;
case 6: inpLaunchPad[4]='6';
break;
case 7: inpLaunchPad[4]='7';
break;
```

```
case 8: inpLaunchPad[4]='8';
break;
case 9: inpLaunchPad[4]='9';
break;
}
switch (n5)
{
case 0: inpLaunchPad[5]='0';
break;
case 1: inpLaunchPad[5]='1';
break;
case 2: inpLaunchPad[5]='2';
break;
case 3: inpLaunchPad[5]='3';
break;
case 4: inpLaunchPad[5]='4';
break;
case 5: inpLaunchPad[5]='5';
break;
case 6: inpLaunchPad[5]='6';
break;
case 7: inpLaunchPad[5]='7';
break;
case 8: inpLaunchPad[5]='8';
break;
case 9: inpLaunchPad[5]='9';
break;
}
switch (n6)
{
case 0: inpLaunchPad[6]='0';
break;
case 1: inpLaunchPad[6]='1';
break;
case 2: inpLaunchPad[6]='2';
break;
case 3: inpLaunchPad[6]='3';
break;
case 4: inpLaunchPad[6]='4';
break;
case 5: inpLaunchPad[6]='5';
break;
case 6: inpLaunchPad[6]='6';
break;
case 7: inpLaunchPad[6]='7';
break;
case 8: inpLaunchPad[6]='8';
break;
case 9: inpLaunchPad[6]='9';
break;
}
switch (n7)
{
case 0: inpLaunchPad[7]='0';
break;
```

```
case 1: inpLaunchPad[7]='1';
break;
case 2: inpLaunchPad[7]='2';
break;
case 3: inpLaunchPad[7]='3';
break;
case 4: inpLaunchPad[7]='4';
break;
case 5: inpLaunchPad[7]='5';
break;
case 6: inpLaunchPad[7]='6';
break;
case 7: inpLaunchPad[7]='7';
break;
case 8: inpLaunchPad[7]='8';
break;
case 9: inpLaunchPad[7]='9';
break;
}

if (orV >= 0.0)
{
char LaunchPad_Temp[]={inpLaunchPad[0],inpLaunchPad[1],inpLaunchPad[2],inpLaunchPad[3],
'.',inpLaunchPad[4],inpLaunchPad[5],inpLaunchPad[6],inpLaunchPad[7]};
WriteFile(LaunchPad,LaunchPad_Temp,sizeof(LaunchPad_Temp),&write,NULL);
}

if (orV <  0.0)
{
char LaunchPad_Temp[]={'-',inpLaunchPad[0],inpLaunchPad[1],inpLaunchPad[2],inpLaunchPad[3],
'.',inpLaunchPad[4],inpLaunchPad[5],inpLaunchPad[6],inpLaunchPad[7]};
WriteFile(LaunchPad,LaunchPad_Temp,sizeof(LaunchPad_Temp),&write,NULL);
}
}
```

c) delay function

```
#include <iostream>
#include <cstddef>
#include <cstring>
#include <stdio.h>
#include <cstdlib>
#include <windows.h>
#include <stdlib.h>
#include <cstring>
#include <ctime>
#include <time.h>
#include <math.h>
#include <conio.h>
#include <dir.h>

void delay(double seconds)
{
LARGE_INTEGER timestart, timeend,  systemfrequency;
double millisecondsleft;
QueryPerformanceFrequency(&systemfrequency);
```

```
QueryPerformanceCounter(&timestart);
QueryPerformanceCounter(&timeend);
__int64 timetaken = (timeend.QuadPart - timestart.QuadPart);
millisecondsleft = (timetaken)*1000. / systemfrequency.QuadPart;
while (millisecondsleft < (double)seconds*1e3)
{
QueryPerformanceCounter(&timeend);
__int64 timetaken = (timeend.QuadPart - timestart.QuadPart);
millisecondsleft = (timetaken)*1000. / systemfrequency.QuadPart;
}
}
```

d) Include file 'Func.h'

```
// Include file
// Functions definitions written for LCVR tests
// Shibu K. Mathew - version 1.1
void delay(double);
HANDLE LaunchPad_GetHandle(const char *);
void LaunchPad_Write(HANDLE,float);
void LaunchPad_CloseHandle(HANDLE);
```

b) Example C program for driving DAC using FTDI245BL chip

a) main program

```
// Driving the DAC714 through FTDI245BL
// Shibu K. Mathew, V1.0, USO/PRL

#include <iostream>
#include <cstddef>
#include <cstring>
#include <stdio.h>
#include <cstdlib>
#include <windows.h>
#include <stdlib.h>
#include <cstring>
#include <ctime>
#include <time.h>
#include <math.h>
#include <conio.h>
#include <dir.h>
#include "Func.h"
#include "ftd2xx.h"

int main(int argc, char *argv[])
{
FT_HANDLE FTDI=FTDI_Handle();
float inpV;
if (argc < 2)
{
printf("Usage FTDI_DAC Voltage\n");
exit(1);
}
if (argc >= 2)
```

```
{
sscanf(argv[1],"%f",&inpV);
printf("Given input Volatge = %f\n",inpV);
}
FTDI_ClockData(FTDI,inpV);
FTDI_CloseHandle(FTDI);
}
```

b) functions 'FTDI_driver.cpp'

```
// Functions written for FTDI 245
// For working with DAC741
// Shibu K Mathew
// V1.0, USO/PRL

#include <iostream>
#include <cstddef>
#include <cstring>
#include <stdio.h>
#include <cstdlib>
#include <windows.h>
#include <stdlib.h>
#include <cstring>
#include <ctime>
#include <time.h>
#include <math.h>
#include <conio.h>
#include <dir.h>
#include "Func.h"
#include "ftd2xx.h"
#define SDI0   1
#define A10     2
#define CLK0  4
#define CLR0  8

// Function 1. FTDI_Handle -SKM
FT_HANDLE FTDI_Handle(void)
{
char des[15] = "SKM-BISQUE";    // Name in the //EPROM programmed using 'FT_Prog' from FTDI
// Change it accordingly
UCHAR Mask = 255;
UCHAR Mode = 1;
FT_STATUS ftStatus;
FT_HANDLE ftHandle;
ftStatus = FT_OpenEx(des, FT_OPEN_BY_DESCRIPTION, &ftHandle);
ftStatus = FT_SetBaudRate(ftHandle, 460800);
ftStatus = FT_SetDataCharacteristics(ftHandle, FT_BITS_8, FT_STOP_BITS_1, FT_PARITY_NONE);
ftStatus = FT_SetBitMode(ftHandle, Mask, Mode);
return ftHandle;
}

// Function 2. FTDI_ClockData -SKM
void FTDI_ClockData(FT_HANDLE fthandle,float inp0)
{
FT_STATUS ftStatus;
int kk;
```

```
DWORD BytesWritten;
UCHAR BTW=0;
float amp_factor=1.0;
float dac_vol0=(inp0)/amp_factor;
float con=32767./10.0;
unsigned int mask=32768;
unsigned int ddi0=dac_vol0*con;
int DDCON0;
if (ddi0 < 0 )
{
ddi0=-1.0*ddi0; //  Two's compliment, for bitshifting take the absolute value
ddi0=~(ddi0);   //  Make a binary invert
ddi0=ddi0+1;    //  Add  one
}
UCHAR DATA_BUF[52];

int counter=0;
for (kk=0;kk<=15;kk++)
{
if ((ddi0 & mask) == 0)
{DDCON0=0;}
else
{DDCON0=1;}
mask=mask>>1; //shift the mask for next bit
DATA_BUF[counter]=SDI0*DDCON0+CLK0*1+CLR0*1+A10*1;
counter++;
DATA_BUF[counter]=SDI0*DDCON0+CLK0*0+CLR0*1+A10*1;
counter++;
DATA_BUF[counter]=SDI0*DDCON0+CLK0*1+CLR0*1+A10*1;
counter++;
}
DATA_BUF[counter]=SDI0*DDCON0+CLK0*0+CLR0*1+A10*1;
counter++;
DATA_BUF[counter]=SDI0*DDCON0+CLK0*0+CLR0*1+0*A10;
counter++;
DATA_BUF[counter]=SDI0*DDCON0+CLK0*1+CLR0*1+0*A10;
counter++;
DATA_BUF[counter]=SDI0*DDCON0+CLK0*1+CLR0*1+A10*1;

ftStatus = FT_Write(fthandle,&DATA_BUF,sizeof(DATA_BUF),&BytesWritten);
}
// Function 3. FTDI_CloseHandle - SKM
void FTDI_CloseHandle(FT_HANDLE ftHandle)
{
FT_Close(ftHandle);
}
c) include file 'Func.h'

// Include file
// Function definitions for all the functions
// Shibu K. Mathew - version 1.1

#include "ftd2xx.h"

void FTDI_ClockData(FT_HANDLE, float);
```

```
FT_HANDLE FTDI_Handle(void);
void FTDI_CloseHandle(FT_HANDLE);
```

c) PIC 16F873A variable frequency square wave generator program

```
// LCVR Control, Ramya B,24-08-2018, PIC16F873A
// Generate Clock at Different Frequencies
// ccp2=gen1=RC2,ccp1=gen2=RC1
// PIC16F873A

#define _XTAL_FREQ 12000000

#include<pic.h>
#include<math.h>
#include<htc.h>
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include "delay.h"

//__CONFIG(FOSC_HS & WDTE_OFF & PWRTE_ON & BOREN_OFF & LVP_OFF & CPD_OFF & WRT_OFF & CP_OFF);
#pragma config BOREN = OFF
#pragma config CPD = OFF
#pragma config WRT = OFF
#pragma config FOSC = HS
#pragma config WDTE = OFF
#pragma config CP = OFF
#pragma config LVP = OFF
#pragma config PWRTE = ON
//#include "delay.h"

void main()
{
TRISA=0x00;
TRISB=0xFF;
int old_PORTB=0xFF;
while(1)
{
if(PORTB != old_PORTB)
{
switch(PORTB)
{
case 0x00: {dvalue=0.5*1.0;//1KHZ,1ms
break;}
case 0x20: {dvalue=0.5*(1.0/1.5);//1.5KHZ
break;}
case 0x10: {dvalue=0.5*(1.0/2.0);//2.0KHZ
break;}
case 0x30: {dvalue=0.5*(1.0/2.5);//2.5KHZ
break;}
case 0x80: {dvalue=0.5*(1.0/3.0);//3.0KHZ
break;}
case 0xA0: {dvalue=0.5*(1.0/3.5);//3.5KHZ
break;}
case 0x90: {dvalue=0.5*(1.0/4.0);//4.0KHZ
```

```
break;}
case 0xB0: {dvalue=0.5*(1.0/4.5);//4.5KHZ
break;}
case 0x40: {dvalue=0.5*(1.0/5.0);//5.0KHZ
break;}
case 0x60: {dvalue=0.5*(1.0/5.5);//5.5KHZ
break;}
case 0x50: {dvalue=0.5*(1.0/6.0);//6.0KHZ
break;}
case 0x70: {dvalue=0.5*(1.0/6.5);//6.5KHZ
break;}
case 0xC0: {dvalue=0.5*(1.0/7.0);//7.0KHZ
break;}
case 0xE0: {dvalue=0.5*(1.0/7.5);//7.5KHZ
break;}
case 0xD0: {dvalue=0.5*(1.0/8.0);//8.0KHZ
break;}
case 0xF0: {dvalue=0.5*(1.0/8.5);//8.5KHZ
break;}
}// switch end
}//if end
PORTA=0xFF;
__delay_ms(dvalue);
PORTA=0x00;
__delay_ms(dvalue);
old_PORTB=PORTB;
}//while end
}// main end
```

PRL research encompasses the earth the sun immersed in the fields and radiations reaching from and to infinity, all that man's curiosity and intellect can reveal

पीआरएल के अनुसंधान क्षेत्र में समविष्ट हैं पृथ्वी एवं सूर्य जो निमीलित हैं चुंबकीय क्षेत्र एवं विकिरण में अनंत से अनंत तक जिन्हे प्रकट कर सकती है मानव की जिज्ञासा एवं विचारशक्ति