Kentech Instruments Ltd

OPERATIONS MANUAL

for

PLPS

Serial no. J02*****

24/02/03

Kentech Instruments Limited, Unit 9, Hall Farm Workshops, South Moreton, Oxfordshire OX11 9AG, U.K.
International Tel: +44 1235 510 748 International Fax: +44 1235 510 722
email info@kentech.co.uk Web : http://www.kentech.co.uk/

CONTENTS

(i) Disclaimer
(ii) EMC
1 Introduction
2 Specifications
3 Use
4 Software

(i) DISCLAIMER

There are high voltage power supplies (1.5kV) present in this instrument when the unit is operating. Do not remove any covers from the PLPS or expose any part of its circuitry. In the event of malfunction, the PLPS must be returned to Kentech Instruments Ltd or its appointed agent for repair.

Kentech Instruments Ltd accepts no responsibility for any electric shock or injury arising from use or misuse of this product. It is the responsibility of the user to exercise care and common sense with this highly performance equipment.

The accessible terminals of this instrument are protected from hazardous voltages by basic insulation and protective grounding via the IEC power input connector. It is essential that the ground terminal of this connector is earthed via the power lead to maintain this protection.

If cleaning is necessary this should be performed with a soft dry cloth or tissue only.

EMC warning

PLPS programmable laser pulse shaper

This equipment includes circuits intentionally designed to generate fast rise, high voltage signals and the EM emissions will be sensitive to the details of the load. If the PLPS is used with the pockels cell provided then EM emissions will fall within EN55011 Emissions Specification for Industrial, Scientific and Medical equipment however the use of other types of pockels cell or other loads may result on emissions which exceed these levels.

1 INTRODUCTION

The PLPS programmable laser pulse shaper is a user programmable high voltage digital to analogue converter driven by fast ram and controlled by an embedded microcontroller. The PLPS is supplied with a modulator which may be driven between maximum transmission and maximum extinction under software control.

The sample rate is nominally 150MHz and the electrical risetime at the pockels cell is nominally 20nsecs (i.e. there are three samples per unit risetime).

The PLPS is designed to compensate for gain depletion in long pulse laser amplification systems. The maximum electrical pulse duration is 10us, which corresponds to nominally 1500 voltage samples.

The maximum output voltage is >700V which corresponds to 100% gain modulation at 1.3um with the modulator provided.

The modulator uses a Y-cut lithium tantalate crystal in which the field is applied longitudinally. There is usually some intrinsic birefringence in such modulators and a DC bias is provided to minimise the transmission the zero programme setting.

2 SPECIFICATIONS

Optical modulator
Lithium tantalate amplitude modulator for 1-1.3um operation comprising matched pair of 4 x 4 x 17mm Y-cut crystals in temperature compensating arrangement to give half wave voltage of ~7--V (at 1.3um). Crystal capacitance ~23pF. Single layer AR coating. Single sided SMA connector.

Fixing centres:
2 x M3 tapped holes, 15mm spacing across optical axis.

Electrical drive

Rise/fall	~20ns
Load	<=30pF
Duration	<=10us
Sample rate	~150MHz
Number of samples	1500 (3000 bytes)
Trigger	TTL
Proportional monitor	~2V into 50ohms, divided down from output
Interface	RS232, 9600 baud
Power	100-240V ac, >30VA

Controls/indicators

AC power	Rocker
Trigger input	BNC
Modulator drive	SMA on short flying lead
Porportional monitor	BNC
PC bias	Screwdriver access
Power LED	Red
Triggered LED	Green
Load LED	Amber
Save/recall LED	Amber

3 USE

The PLPS will normally be mounted over the modulator. Tapped spacers are provided on the base of the electronics assembly to fit suitable supports. The modulator has two tapped holes on its base to allow the mounting of it to an adjustable optical mount.

It is assumed that the user s familiar with the use of optical modulators and will provide suitable optics (such as a polarizer) and will align these optics appropriately.

Attach the PLPS to the modulator with the short flying SMA lead on the base of the electronics assembly.

A dumb RS232 terminal or computer with an emulator programme will be required to programme the PLPS.

The 9 pin RS232 connector pins are:

2	RS232, out
3	RS232, in
5	Ground

The electrical waveform at the modulator may be viewed via the proportional monitor output. This signal is divided down from the signal at the modulator.

The optical modulator function is quite non-linear, not least because of the sin-squared behaviour of the optical modulator. The user must provide suitable programming data in order to produce the required optical waveform.

The modulator will in general have some inherent birefringence and a bias potential will be required to minimise optical transmission. There is an internal bias supply which may be adjusted via the pockels cell bias hole on the side of the PSU. A small electrical screwdriver will be required. This adjustment provided a DC bias between 0 and ~250V. Set this voltage to give minimum optical transmission.

The sample clock is a quick start oscillator based on a resonant transmission line so the sample frequency is NOT crystal referenced. If the sample timing is critical we suggest that a calibration parameter is included in any host PC control application to define the time per sample, which should be determined by measurement.

4 SOFTWARE

At power up the PLPS will look at the state of the start-up selector switch and will initialise with either a null waveform or WFM0 or WFM1. This process takes a few seconds.

The Save/recall LED indicates access to the internal NV memory. The Load LED indicates that the fast RAM array is being programmed.

When a trigger signal is received the triggered LED will be illuminated for a short while.

After initialisation the PLPS will issue a banner message.

The internal microprocessor runs a Forth operating system and all commands are Forth definitions. Commands must be upper case and are followed by a <CR>.

Correct interpretation and execution of a command is followed by an OK prompt. An error will cause an error message to be returned.

The local processor keeps the waveform in RAM and downloads it to the DAC RAM when instructed.

We suggest that the first thing to try is to set up a dumb terminal and try typing commands to the PLPS from the keyboard. Each <CR> will result in an OK prompt - this is a good way to check that the link is alive.

There follows a typical RS232 dialogue with the PLPS. The baud rate is 9600, 8 data bits, 1 stop bit, no parity.

The typed text is in red, the response is in black. Note that the PLPS echoes back all characters in the command line.

The text in blue is a description of the command above and is not part of the dialogue.

```
Power up - 3 second delay then:

Programmable Laser Pulse Shaper
Copyright Kentech Instruments Ltd 2003
Firmware V1.0

<CR>  ok
-EN_TRIG  <CR>  ok
Stop triggering
<CR>  ok
<CR>  ok
<CR>  ok
<CR>  ok
<CR>  ok
<CR>  ok
<CR> results in an 'ok' response i.e. the local processor is alive and well
+EN_TRIG  <CR>  ok
Restart triggering
0WFM <CR>  ok
Fill RAM image of waveform with zeros
0 2000 0 50 RAMP  <CR>  ok
Generate a ramp going from 0 to 2000, starting at sample 0 and ending at 50
0 0 50 100 RAMP  <CR>  ok
0 3000 101 140 RAMP  <CR>  ok
3000 2000 141 150 RAMP  <CR>  ok
2000 2000 151 200 RAMP  <CR>  ok
0 0 201 250 RAMP  <CR>  ok
1000 1000 251 270 RAMP  <CR>  ok
Generate a flat section, value 1000, starting at sample 251 and ending at 270
WFM->DAC  <CR>  ok
Send the waveform in RAM to the DAC RAM.
<CR>  ok
0WFM  <CR>  ok
0 3500 0 1500 RAMP  <CR>  ok
WFM->DAC  <CR>  ok
Generate a 10us long ramp
<CR>  ok
2000 0 500 PULSE  <CR>  ok
Generate a 2000 high pulse from sample 0 to 500 and immediately send it to the
DAC RAM.
<CR>  ok
<CR>  ok
100 TWEAK <CR>
```

Invoke the tweak function starting at sample 100

```
2 Sample # = 100 , Value = 2000
```

Typing ‘2’ increases the current sample, 1 will decrease it

```
2 Sample # = 100 , Value = 2100
2 Sample # = 100 , Value = 2200
2 Sample # = 100 , Value = 2300
2 Sample # = 100 , Value = 2400
4 Sample # = 101 , Value = 2000
```

Typing ‘4’ moves to the next sample, 3 to the previous one

```
4 Sample # = 102 , Value = 2000
4 Sample # = 103 , Value = 2000
4 Sample # = 104 , Value = 2000
1 Sample # = 104 , Value = 1900
1 Sample # = 104 , Value = 1800
1 Sample # = 104 , Value = 1700
1 Sample # = 104 , Value = 1600
<esc>  ok
```

Type ‘esc’ to leave the tweak function
The tweak function ignores all keys except 1,2,3,4 and <esc>. 1 and 2 edit the sample value, 3 and 4 move between samples. The edited sample is immediately sent to the DAC RAM so the output can be edited in real time. The <esc> key exits tweak and returns to the Forth operating system.

```
<CR>  ok
<CR>  ok
<CR>  ok
@WFM0  <CR>  ok
```

Fetch waveform from NVRAM 0 (similarly for @WFM1)

```
WFM->DAC  <CR>  ok
```

Send waveform to DAC RAM

```
!WFM0  <CR>  ok
```

Store current waveform in NVRAM 0 (similarly for !WFM1)

```
<CR>  ok
@WFM1  <CR>  ok
WFM->DAC  <CR>  ok
<CR>  ok
<CR>  ok
20 STEPSIZE !  <CR>  ok
```

Reduce the TWEAK step size from 100 (default) to 20

```
100 TWEAK <CR>
1 Sample # = 100 , Value = 200
1 Sample # = 100 , Value = 220
1 Sample # = 100 , Value = 240
<esc>  ok
```

```
NB the 1,2,3,4 and <esc> keys are not echoed when in tweak
1000 STEPSIZE !  <CR>  ok
100 TWEAK <CR>
1 Sample # = 100 , Value = 1420
1 Sample # = 100 , Value = 2420
1 Sample # = 100 , Value = 3420
1 Sample # = 100 , Value = 2420
<esc>  ok
```

In order to transfer a complete waveform record, binary waveform transfer functions are provided. These are:

TXWFM, RXWFM and **RXWFM->DAC**

If these functions are attempted from the keyboard the PLPS may appear to freeze as it is either waiting for 1500 samples (3kbytes) or is transmitting 3kbytes.

The format of a sample is 12 bit, 2 bytes, LSB first. In practice only 10 bits are used.

There are some test functions provided, the most useful of which being:

TEST2

After typing this command the PLPS will repeatedly return the state of various internal power supplies, including the PC bias setting. This process will continue until the next RS232 character is received.

Note that there is a time constant of a second or so on these voltages so, if this function is used while adjusting the PC bias voltage, wait long enough for the voltage to settle before noting a voltage setting.

TEST3 generates a series of small steps which may be useful when characterising the transfer funtion.

List of the useful commands.

All commands are in upper case and must be followed by <CR>. All commands are echoed. All arguments are integers and will be clipped. There must be at least one space between the argument (if required) and the command.

+EN_TRIG
Enable triggering
-EN_TRIG
Disable triggering
WFM->DAC
Send the waveform in RAM to the DAC RAM
0WFM
Zero the waveform in RAM
value SETALL
Set all 1500 samples of the waveform in RAM to ‘value’ and send it to the DAC RAM
address TWEAK
Enter the TWEAK function, starting at sample number ‘address’
value start end PULSE (V,S,E ---) WFM->DAC ;
Zero the waveform in RAM, produce a pulse of height ‘value’, running from sample ‘start’ to sample ‘end’ and send it to the DAC RAM
!WFM0
Store the current waveform in RAM in NVRAM 0 - NB this is not necessarily the same as the waveform in the DAC RAM
@WFM0
Fetch the waveform in NVRAM 0 into RAM
NB if this is not followed by ‘WFM->DAC’ it will not appear at the output.
!WFM1
Store the current waveform in RAM in NVRAM 1 - NB this is not necessarily the same as the waveform in the DAC RAM
@WFM1
Fetch the waveform in NVRAM 1 into RAM
NB if this is not followed by ‘WFM->DAC’ it will not appear at the output.
startval endval startadd endadd RAMP
Replaces samples ‘startadd’ to ‘endadd’ with a linear ramp from ‘startval’ to ‘endval’. NB ‘WFM->DAC’ is also required to send it to the output.
bias !BIAS
Internal adjustment
?HT
Internal measurement
?20V
Internal measurement

TXWFM

Transmit current waveform in RAM in binary. After the following characters are sent:

   ‘TXWFM <CR>’ the PLPS replies with:

   ‘TXWFM <CR> followed by 3000 binary characters, two bytes per sample, most significant byte first, then:

   ‘ ok <CR>’

RXWFM

Receive a binary waveform record into RAM. After the following characters are sent:

    ‘RXWFM <CR>’ the PLPS replies with:

    ‘RXWFM ‘

It then waits until it has received 3000 bytes then replies with

   ‘ ok <CR>’

RXWFM->DAC

As above except that the waveform is automatically sent to the DAC RAM after reception.

?SUMWFM

Print the checksum of the waveform in RAM, modulus 2^16

value address !VAL

Store ‘value’ in sample ‘address’ in RAM

NB if this is not followed by ‘WFM->DAC’ it will not appear at the output.

address @VAL

Fetch the value stored in sample ‘address’ onto the Forth stack - NB this is not necessarily the value in the DAC RAM. To print the value use the Forth print command ‘.’ :

address @VAL .

Print the value stored in sample ‘address’

This is the Forth source code we used in the host computer (a Mac) to test the data transfer. It was writen in MacForth from Creative Solutions.

The data is stored locally in ‘tdsbuffer’.
@VALUES gets the current waveform
!VALUES sends the local waveform in the host to the PLPS but does not load it into the DAC RAM
?SUMVALUES does a local checksum
N ?VALUES prints out N values from the local waveform buffer
RAMP generates a local ramp
EXAMPLE generates a local ramp, sends it to the PLPS and requests the PLPS to put it in the DAC RAM

```
anew graphmarker

decimal

30000 minimum.object

: S.KEY   ( -- char ) begin s.?terminal not while ?abort repeat s.key ;

create tdsbuffer 3000 allot

: @VALUES ( --) 0S.BUFFER
   " TXWFM" count DUP>R s.type s.cr
   R> 1+ 0 DO S.KEY DROP LOOP
   3000 0 do  s.key  tdsbuffer i+ c! loop
   5 0 do s.key drop loop ;

: !VALUES ( --) 0S.BUFFER
   " RXWFM" count DUP>R s.type s.cr
   R> 1+ 0 DO S.KEY DROP LOOP
   3000 0 do  tdsbuffer i+ c@ S.EMIT loop
   5 0 do s.key drop loop ;

: ?SUMVALUES
 ( --) 0 3000 0 DO tdsbuffer i+ C@ + LOOP 65535 AND . ;


: ?VALUES 0 DO
```

```
i 2* tdsbuffer + c@ 256 *
i 2* tdsbuffer + 1+ c@ + . CR LOOP ;


: RAMP
1500 0 DO i 20 * dup i 2* tdsbuffer + 1 + c!
256 / i 2* tdsbuffer + c! loop ;

: EXAMPLE
RAMP !VALUES
" WFM->DAC" count s.type s.cr
 ;
```

Below is a quick and dirty BASIC code to do the same sort of thing written in TrueBasic for the Mac. The red bit uploads the current waveform in PLPS ram. The following black bit assembles bytes into 12 bit samples and prints a few. The blue bit generates and sends a ramp using the RXWFM->DAC command which automatically puts the waveform in the DAC RAM the it then drops into a terminal emulator loop.
There is no error checking and it will hang if it gets out of phase during an upload.
If the PLPS hangs due to being sent too few characters after an RXWFM then sent it carriage returns until it replies with ‘ok’.

```
DIM values(3000)

LIBRARY "Comlib*"
LET baud = 9600
CALL Com_open (#1, 1, baud, "D8")      ! Open comm line at 9600 baud
PRINT " Baud rate = ", baud
PRINT " Type OPTION q to quit"
PRINT " NB - UPPER CASE FOR COMMANDS"

CALL Receive (s$)
LET s$ = ""


LET str$ = "TXWFM"
CALL Send (str$)
CALL Send (Chr$(13))
```

```
DO
   CALL Receive (s$)
   LET value$ = value$ & s$
   LET s$ = ""
   IF len(value$) >3005 then EXIT DO
LOOP

! print out the first few values
FOR i = 1 to 20
    PRINT ord(value$[i*2+5:i*2+5])*256 + ord(value$[i*2+6:i*2+6])
NEXT i

LET send$ = " "
FOR i = 1 to 1500
    LET thisvalue = i
    LET msb = int(thisvalue/256)
    LET lsb = thisvalue - msb*256
    LET send$ = send$ & Chr$(msb) & Chr$(lsb)
NEXT i

LET str$ = "RXWFM->DAC"
CALL Send (str$)
CALL Send (Chr$(13))
CALL Send (send$)


DO
   CALL Receive (s$)                ! get any input from network
   CALL Output (s$)                 ! routine to print it on screen
   IF key input then                ! get anything the user's typed
      GET KEY k
      SELECT CASE k
      CASE 207                      ! option-q -- end session
           STOP
      CASE 186                      ! option-b -- send break
           CALL Send_break
      CASE else                     ! else send to the network
           CALL Send (Chr$(k))
      END SELECT
   END IF
LOOP

SUB Output (s$)                     ! Handle CR & LF characters
```

```
    DO                            ! first strip all CRs
       LET i = Pos(s$,Chr$(13))   ! find first CR
       IF i = 0 then EXIT DO      ! none -- all done
       LET s$[i:i] = ""           ! remove the first
    LOOP

    DO                            ! now end line on line-feed
       LET i = Pos(s$,Chr$(10))   ! find next line-feed
       IF i = 0 then EXIT DO
       PRINT s$[1:i-1]            ! print each separate line
       LET s$ = s$[i+1:maxnum]    ! remove that line
    LOOP

    PRINT s$;                     ! print partial line

END SUB

END
```

Embedded source code (In Forth)

```
$C000 VDP !

DECIMAL

$FFDC CONSTANT SSR
: ?KEY SSR C@ $40 AND ;

0 VARIABLE P7DATA
0 VARIABLE ADATA

3000 VARIABLE NUMSAMPLES
0 VARIABLE TEMP

0 VARIABLE REG0
0 VARIABLE REG1
0 VARIABLE REG2
0 VARIABLE REG3
0 VARIABLE REG4
0 VARIABLE REG5
0 VARIABLE REG6
0 VARIABLE REG7
0 VARIABLE REG8
0 VARIABLE REG10

0 VARIABLE THIS
0 VARIABLE WFM 4096 ALLOT
0 VARIABLE STEPSIZE
0 VARIABLE SV
0 VARIABLE EV
0 VARIABLE SA
0 VARIABLE EA


: ADATA>A ( -- ) ADATA C@ $81E0 PC! ;
: !A ( b --) ADATA C!  ADATA>A ;
: @A ( -- b) ADATA C@ ;
: 0PORTAB ( -- ) 5 CFIGM !  5 $81F0 PC!  0 !A ;
: @B ( --N ) $81D0 PC@ ;
: 0P7 $FF $FF8C C! ;         ( make P7 outputs )
: !P7 DUP P7DATA ! $FF8E C! ;              ( write P7 )
: @P7 P7DATA @ ;
: +LOADLED
@P7 254 AND !P7 ;
: -LOADLED
@P7 1 OR !P7 ;

: +RECALLED
@P7 253 AND !P7 ;
: -RECALLED
@P7 2 OR !P7 ;

( TO DRIVE EPLD FROM PA )
: !EPLD ( n a -- )
          63 AND 128 + !A
          63 AND !A ;

: !REG0 ( N-- )
DUP REG0 ! 0 !EPLD ;
: !REG1 ( N-- )
DUP REG1 ! 1 !EPLD ;
: !REG2 ( N-- )
DUP REG2 ! 2 !EPLD ;
: !REG3 ( N-- )
DUP REG3 ! 3 !EPLD ;
: !REG4 ( N-- )
DUP REG4 ! 4 !EPLD ;
: !REG5 ( N-- )
DUP REG5 ! 5 !EPLD ;
: !REG6 ( N-- )
DUP REG6 ! 6 !EPLD ;
: !REG7 ( N-- )
DUP REG7 ! 7 !EPLD ;
: !REG8 ( N-- )
DUP REG8 ! 8 !EPLD ;
: INTCLK 0 9 !EPLD ;
: !REG10 ( N-- )
DUP REG8 ! 8 !EPLD ;

: +NCE0 ( -- )
REG0 @ 1 OR !REG0 ;
: -NCE0 ( -- )
REG0 @ 255 1 - AND !REG0 ;
: +NOE0 ( -- )
```

```
REG0 @ 2 OR !REG0 ;
: -NOE0 ( -- )
REG0 @ 255 2 - AND !REG0 ;
: +ALL_NW_LO ( -- )
REG0 @ 4 OR !REG0 ;
: -ALL_NW_LO ( -- )
REG0 @ 255 4 - AND !REG0 ;

: +NCE1 ( -- )
REG0 @ 8 OR !REG0 ;
: -NCE1 ( -- )
REG0 @ 255 8 - AND !REG0 ;
: +NOE1 ( -- )
REG0 @ 16 OR !REG0 ;
: -NOE1 ( -- )
REG0 @ 255 16 - AND !REG0 ;

: +EN_TRIG ( -- )
REG0 @ 32 OR !REG0 ;
: -EN_TRIG ( -- )
REG0 @ 255 32 - AND !REG0 ;

: +NCE2 ( -- )
REG1 @ 1 OR !REG1 ;
: -NCE2 ( -- )
REG1 @ 255 1 - AND !REG1 ;
: +NOE2 ( -- )
REG1 @ 2 OR !REG1 ;
: -NOE2 ( -- )
REG1 @ 255 2 - AND !REG1 ;

: +QSO ( -- )
REG1 @ 8 OR !REG1 ;
: -QSO ( -- )
REG1 @ 255 8 - AND !REG1 ;

: +FORCE_TRIG ( -- )
REG4 @ 1 OR !REG4 ;
: -FORCE_TRIG ( -- )
REG4 @ 255 1 - AND !REG4 ;

: +INTRESET ( -- )
REG1 @ 32 OR !REG1 ;
: -INTRESET ( -- )
```

```
REG1 @ 255 32 - AND !REG1 ;

: DATA->EPLD ( N N -- )
DUP 5 !EPLD
64 / 6 !EPLD
DUP 7 !EPLD
64 / 8 !EPLD ;

: EPLD->RAM
0 10 !EPLD
2 10 !EPLD
4 10 !EPLD ;

: PROGMODE ( -- )
-QSO +NOE0 +NOE1 +NOE2 -EN_TRIG
0 0 DATA->EPLD EPLD->RAM
-FORCE_TRIG INTCLK -INTRESET ;

: RUNMODE ( -- )
-FORCE_TRIG
+EN_TRIG
-NOE0
-NOE1
-NOE2
+INTRESET
-INTRESET
+QSO ;

: !ZEROS ( N-- )
0 0 DATA->EPLD EPLD->RAM
+ALL_NW_LO
0 DO INTCLK LOOP
-ALL_NW_LO ;

: WFM->DAC
+LOADLED
PROGMODE
+INTRESET
-INTRESET
1 !ZEROS
750 0 DO
WFM I 4 * + @ WFM I 4 * + 2 + @
DATA->EPLD EPLD->RAM INTCLK LOOP
```

```
400 !ZEROS
RUNMODE
-LOADLED ;

: 0WFM
2048 0 DO 0 WFM I 2* + ! LOOP ;

: +RAMP
1500 0 DO I 1800 1500 */ WFM I 2* + ! LOOP
WFM->DAC ;

: ++RAMP
2048 0 DO I 3 * WFM I 2* + ! LOOP
WFM->DAC ;

: -RAMP
1500 0 DO 1500 I - 3500 1500 */ WFM I 2* + ! LOOP
WFM->DAC ;

: +-RAMP
1024 0 DO
I  4 * WFM I 4 * + !
1023 I - 4 * WFM I 4 * + 2 + !
LOOP
WFM->DAC ;

: SETALL ( V-- )
1500 0 DO DUP WFM I 2* + ! LOOP DROP
WFM->DAC ;

: .VALS
." Sample # = " THIS @ . ." , Value = " WFM THIS @ 2* + @ . CR ;

: +VAL
WFM THIS @ 0 MAX 1499 MIN 2* + DUP
@ STEPSIZE @ + 4095 MIN SWAP ! .VALS ;
: -VAL
WFM THIS @ 0 MAX 1499 MIN 2* + DUP
@ STEPSIZE @ - 0 MAX SWAP ! .VALS ;

: +THIS
 1 THIS @ + 1499 MIN THIS ! .VALS ;
: -THIS
 -1 THIS @ + 0 MAX THIS ! .VALS ;
```

```
: CLOCKS ( N-- )
0 DO
INTCLK LOOP ;

: VAL->DAC
PROGMODE
+INTRESET
-INTRESET
THIS @ 2/ 1 + CLOCKS
THIS @ 2/ 4 * DUP WFM + @ SWAP 2+ WFM + @
DATA->EPLD EPLD->RAM
RUNMODE ;

: TWEAK ( -- )
DEPTH 0 = IF 0 THEN
0 MAX 1499 MIN
THIS !
CR .VALS
BEGIN
KEY
 CASE
          49 OF -THIS 0 ENDOF
          50 OF +THIS 0 ENDOF
          51 OF -VAL VAL->DAC 0 ENDOF
          52 OF +VAL VAL->DAC 0 ENDOF
          27 OF 1 ENDOF
          0
ENDCASE UNTIL
CR ." Reloading DACs" WFM->DAC CR ;

: PULSE ( V,S,E --- )
0WFM 1500 MIN 1 MAX DUP ROT
SWAP 1 - MIN 0 MAX
DO DUP WFM I 2* + ! LOOP DROP
WFM->DAC ;

( STUFF FOR 24C65 )
( SET TO ADDRESS 2 )
( WORD ADDRESS DOES NEED TO BE DIVISIBLE BY 2 )

: BEAD
   DUP 0 $2000 U/ NIP 2* SWAP $1FFF AND SWAP ;
```

```
: EEC! ( c addr -- )
   $4000 +
   BEAD >R
   DUP >< $A0 R> +
   STARTI2C
   4 0 DO S1BYTE R1BIT DROP
       LOOP STOPI2C 10 MS ;

: EEC@ ( addr -- c )
   $4000 +
   BEAD DUP >R
   $A0 + STARTI2C
   S1BYTE R1BIT DROP
   DUP >< S1BYTE R1BIT DROP
          S1BYTE R1BIT DROP
   $A1 R> + STARTI2C S1BYTE
   R1BIT DROP
   R1BYTE STOPI2C ;

: S8-> DUP 0< 128 * SWAP $7FFF AND 256 / + ; ( N -- N SHIFT R 8 BITS )

: EE! ( WORD ADDRESS -- )
                    DUP 1+ ROT DUP $FF AND ROT EEC!
                    S8-> SWAP EEC! ;

: EE@ ( ADDRESS -- WORD )
                    DUP EEC@ $100 * SWAP 1+ EEC@ + ;

: !WFM0 +RECALLED
1500 0 DO I 2* WFM + @ I 2* EE! LOOP -RECALLED ;

: @WFM0 +RECALLED
1500 0 DO I 2* EE@ WFM I 2* + ! LOOP -RECALLED ;

: !WFM1 +RECALLED
1500 0 DO I 2* WFM + @ I 2* 3000 + EE! LOOP -RECALLED ;

: @WFM1 +RECALLED
1500 0 DO I 2* 3000 + EE@ WFM I 2* + ! LOOP -RECALLED ;

: +RAMP
2048 0 DO I 2* WFM I 2* + ! LOOP
WFM->DAC ;
```

```
: RAMP ( SV EV SA EA -- )
0 MAX 1500 MIN DUP EA !
SWAP MIN SA !
EV ! SV !
EA @ 1+ SA @ DO
I SA @ - EV @ SV @ - EA @ SA @ - */ SV @ +
WFM I 2* + ! LOOP  ;

: !BIAS
0 MAX 250 MIN 0 D-A ;

: @SW ( --B )
@B 192 AND ;

: INIT
125 !BIAS
0PORTAB
0P7
0 !A
0 !REG0
0 !REG0
0 !REG1
0 !REG2
0 !REG3
0 !REG4
0 !REG5
0 !REG6
0 !REG7
0 !REG8
0 !REG10
100 STEPSIZE !
PROGMODE
1500 !ZEROS
-RECALLED
RUNMODE
+FORCE_TRIG -FORCE_TRIG
@SW 128 = IF @WFM1 THEN
@SW 64 = IF @WFM0 THEN
@SW 192 = IF 0WFM THEN
WFM->DAC
3000 NUMSAMPLES !
  ;
```

```
: TXbytes ( addr # --) OVER + SWAP DO I C@ EMIT LOOP ;
: RXbytes ( addr # --) OVER + SWAP DO (KEY) I C! LOOP ;
: TXWFM ( --) WFM NUMSAMPLES @ TXbytes ;
: RXWFM ( --) WFM NUMSAMPLES @ RXbytes ;
: RXWFM->DAC ( --) WFM NUMSAMPLES @ RXbytes WFM->DAC ;
: ?SUMWFM ( --) 0 NUMSAMPLES @ 0 DO WFM I + C@ + LOOP U. ;

: !VAL ( V,A -- )
0 MAX 1499 MIN 2* WFM + ! ;
: @VAL ( A -- V )
0 MAX 1499 MIN 2* WFM + @ ;

: !VAL->DAC ( V,A -- )
0 MAX 1499 MIN DUP THIS ! 2* WFM + !
VAL->DAC ;

: TEST0 ( -- )
( CYCLE DATA BUS FROM TDS )
BEGIN TEMP @ 1+ DUP TEMP ! !A ?KEY UNTIL ;

: TEST1 ( -- )
( MAKES RAM DATA TOGGLE )
0 10 !EPLD
BEGIN
63 5 !EPLD
0 6 !EPLD
0 7 !EPLD
63 8 !EPLD
100 MS
0 5 !EPLD
63 6 !EPLD
63 7 !EPLD
0 8 !EPLD
100 MS ?KEY UNTIL ;

: TEST2 ( -- )
BEGIN CR
0 A-D 21000 653 */ . ." MV " CR
1 A-D 512 - 197 60 */ . ." VOLTS BIAS " CR
2 A-D 1340 278 */ . ." VOLTS " CR
1000 MS ?KEY UNTIL ;
```

```
: TEST3 ( -- )
( STEPS )
0WFM
20 0 DO
I 100 * DUP I 1 - 100 * DUP 100 + RAMP LOOP
WFM->DAC ;

: ?20V 0 A-D 21000 653 */ . ." mV" CR ;
: ?BIAS 1 A-D  512 - 197 60 */ . ." VOLTS BIAS " CR ;
: ?HT 2 A-D 1340 278 */ . ." Volts" CR ;

: -FORTH  0   [ ' FORTH NFA ] LITERAL C! ; ( needed to get cold start )

: BANNER CR
." Programmable Laser Pulse Shaper" CR
." Copyright Kentech Instruments Ltd 2003" CR
." Firmware V1.0 " CR ;

: ABORT  ( initialisation of Forth system, falling into interpret loop
   SP!
   DECIMAL 0 OFFSET !
   [COMPILE] FORTH  DEFINITIONS  QUIT ;

: 0VARIABLES
   $C000 1- $FB80 $0A +ORIGIN !          ( dictionary full when it meets
ram
   $8800 DP !  $8800 FENCE !             ( set base of dictionary
   0 BLK !   0 IN !   0 OUT !            (
   0 SCR !   0 OFFSET !  0 STATE !       ( set a few user variables
   -1 DPL !   0 CSP !   0 R# !   0 HLD ! (
    ;  ( get context, current right

: GO 9600 BAUD 0VARIABLES BANNER INIT -FORTH ABORT ;


SET GO
```