



# Reading Spartan-3A Device DNA



A Reference Design for the Spartan-3A Starter Kit  
Includes an LCD display driver

**PicoBlaze™**

Ken Chapman  
Xilinx Ltd

Rev2 – 10<sup>th</sup> July 2007



# Limitations

**Limited Warranty and Disclaimer.** These designs are provided to you “as is”. Xilinx and its licensors make and you receive no warranties or conditions, express, implied, statutory or otherwise, and Xilinx specifically disclaims any implied warranties of merchantability, non-infringement, or fitness for a particular purpose. Xilinx does not warrant that the functions contained in these designs will meet your requirements, or that the operation of these designs will be uninterrupted or error free, or that defects in the Designs will be corrected. Furthermore, Xilinx does not warrant or make any representations regarding use or the results of the use of the designs in terms of correctness, accuracy, reliability, or otherwise.

**Limitation of Liability.** In no event will Xilinx or its licensors be liable for any loss of data, lost profits, cost or procurement of substitute goods or services, or for any special, incidental, consequential, or indirect damages arising from the use or operation of the designs or accompanying documentation, however caused and on any theory of liability. This limitation will apply even if Xilinx has been advised of the possibility of such damage. This limitation shall apply notwithstanding the failure of the essential purpose of any limited remedies herein.

This design module is **not** supported by general Xilinx Technical support as an official Xilinx Product. Please refer any issues initially to the provider of the module.

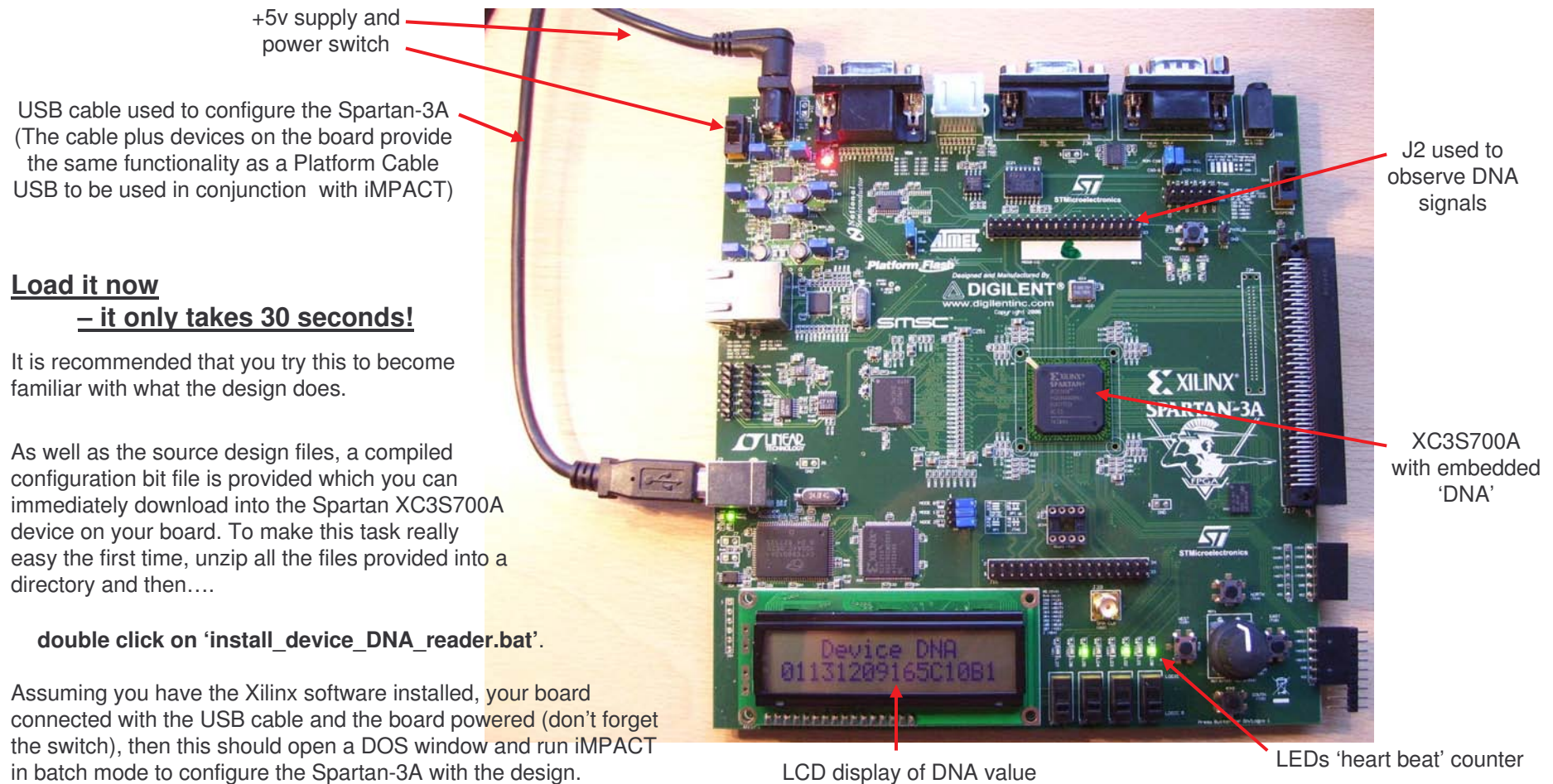
Any problems or items felt of value in the continued improvement of KCPSM3 or this reference design would be gratefully received by the author.

Ken Chapman  
Senior Staff Engineer – Spartan Applications Specialist  
email: chapman@xilinx.com

The author would also be pleased to hear from anyone using KCPSM3 with information about your application and how PicoBlaze has been useful.

# Design Overview

This design reads the unique device DNA value embedded in the XC3S700A Spartan-3A device and displays it on the LCD display. Device DNA can be used to provide a serial number for your own products, be useful in hardware and software design security techniques or be sued as part of product registration procedures. This document provides details of the reference design which employs a PicoBlaze processor to perform most of the tasks. As well as enabling the DNA feature of the device to be evaluated, it is hoped that the design will provide a useful reference for PicoBlaze users especially those interested in using LCD character modules with an 8-bit data interface as provided on the 3A starter kit (note that the 3E starter kit has the 4-bit interface).



## Load it now

**– it only takes 30 seconds!**

It is recommended that you try this to become familiar with what the design does.

As well as the source design files, a compiled configuration bit file is provided which you can immediately download into the Spartan XC3S700A device on your board. To make this task really easy the first time, unzip all the files provided into a directory and then....

**double click on 'install\_device\_DNA\_reader.bat'.**

Assuming you have the Xilinx software installed, your board connected with the USB cable and the board powered (don't forget the switch), then this should open a DOS window and run iMPACT in batch mode to configure the Spartan-3A with the design.

# Design Operation

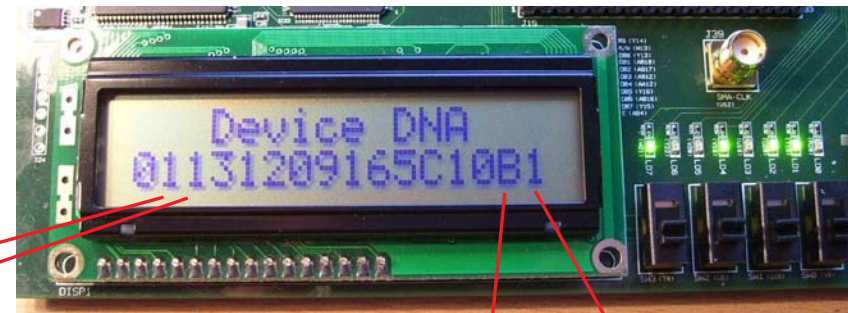
The design provided has a fixed functionality and it is left to you to make any additions or changes as you wish.



Following configuration the LCD should display the simple design title information shown here. This display will persist for 10 seconds. The 8 LEDs represent a simple binary counter implemented in PicoBlaze code and responding to interrupts generated at 1 second intervals. It is when the lower 4-bits of this counter contain the value '1010' (ten) that the display will change.

The device DNA is a 57-bit value but it is displayed as a 16 digit hexadecimal value of 64-bits with the most significant bits padded with zero. The DNA value shown on my board is 01131209165C10B1 hex.

PicoBlaze reads and displays the DNA value every 16 seconds (each time the lower 4 LEDs have the pattern '1010'). PicoBlaze actually reads the DNA value in less than 50µs but the LCD display is much slower making this update observable even though the value remains the same. The reading repeats to facilitate observation of signals with an oscilloscope if desired.

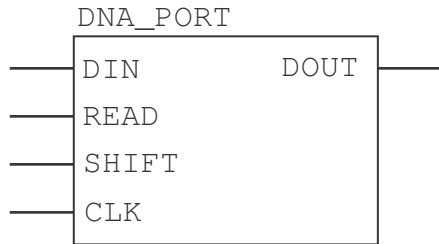


Device DNA



Of the is a 57-bits forming the device DNA value, 55 bits are a unique value and 2 bits are a fixed '10' pattern. The DNA is read serially, and convention states that the value is presented most significant bit first. It is the most significant bits that are the fixed '1' and '0' and may be used to confirm a correct read process if desired. The most significant 2 digits of the LCD display should always be '01' hex because they represent the fixed '1' of bit 56 padded with zeros to form a byte (00000001). The next most significant digit should always fall in the range 0 to 7 hex since bit 55 should always be the fixed '0'.

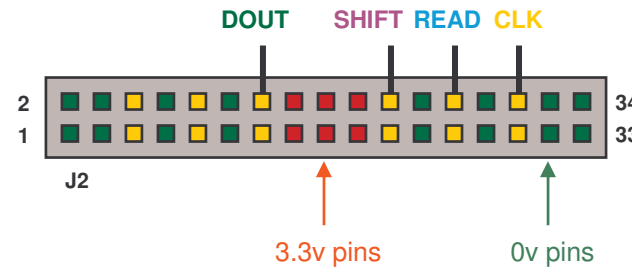
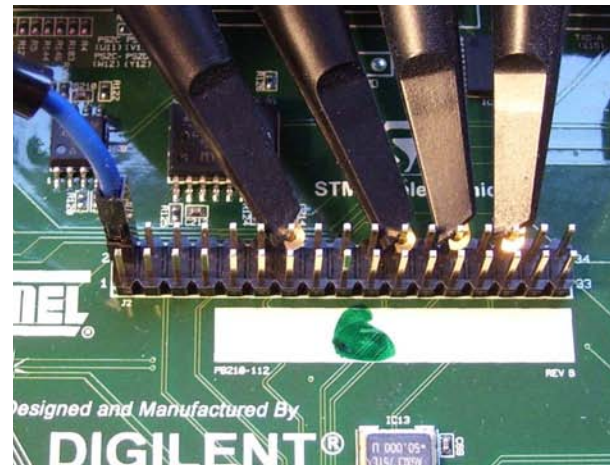
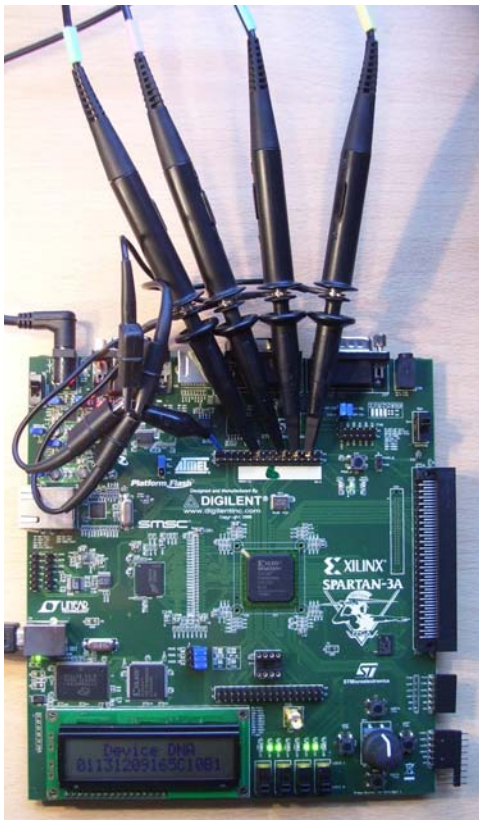
# DNA Port Signals



The DNA is read internally to the Spartan-3A device using the 'DNA\_PORT' primitive from the Xilinx library.

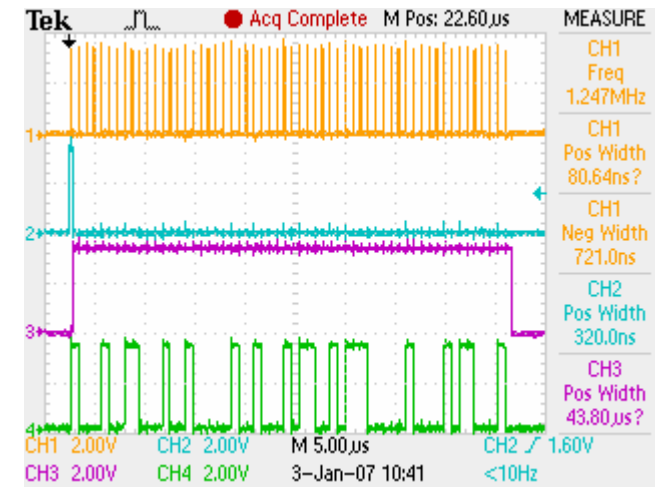
The port is essentially a 57-bit shift register with serial input DIN and serial output DOUT. The shift register advances with rising edges of a clock provided to the CLK input providing the SHIFT control is High.

In order to read the unique DNA value of the device the shift register must first be loaded by driving the READ input control High whilst a rising edge clock is applied. This will immediately present bit 56 (the fixed value '1') at DOUT.



To enable you to evaluate the DNA feature in more detail, the design also connects the CLK, READ, SHIFT and DOUT signals to convenient pins on header J2 as shown.

The oscilloscope screen shot below shows the complete read process being performed by PicoBlaze in approximately 44µs (more details later).

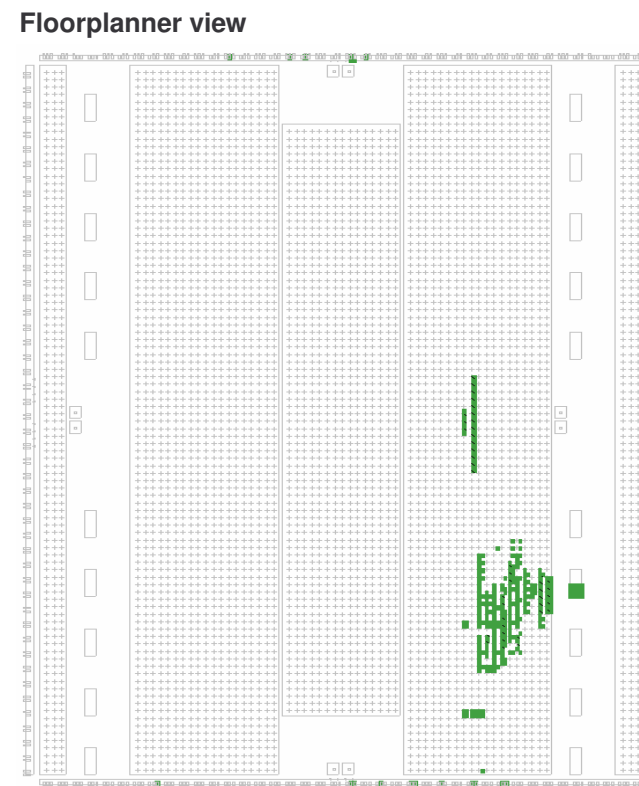
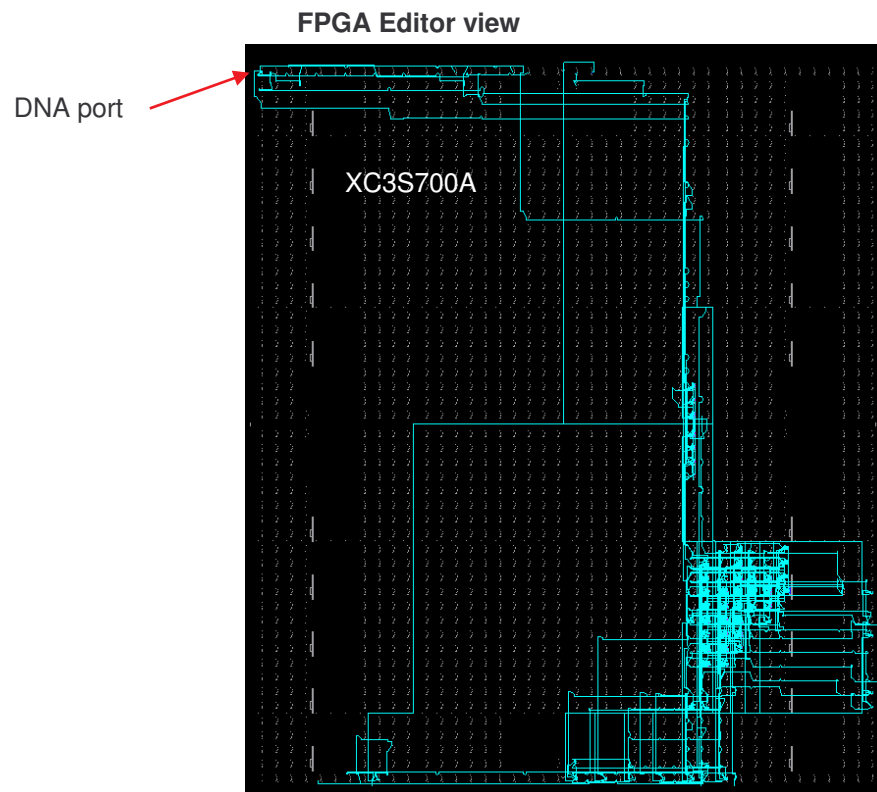


# Design Size

This reference design occupies less than 5% of the XC3S700A device. The majority of the logic resources are the 96 slices required to implement PicoBlaze. However, it should be recognised that only 26% of the program memory provided by the single Block RAM (RAMB16BWE) is currently used and therefore even this small amount of resources are capable of implementing much more.

## MAP report

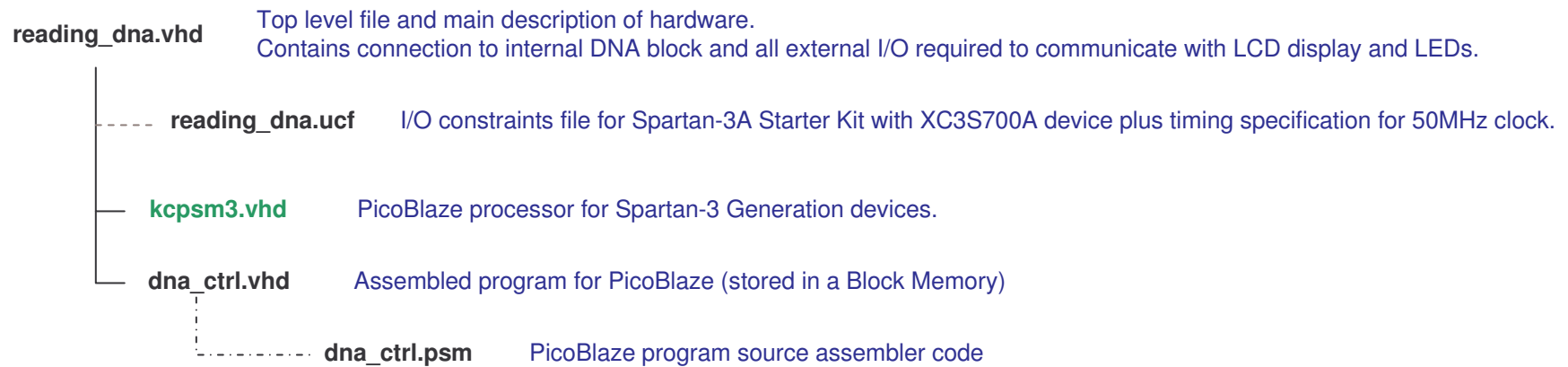
Number of occupied Slices:	122 out of	5,888	2%
Number of RAMB16BWEs:	1 out of	20	5%



# Design Files

For those interested in the actual design implementation, the following pages provide some details and an introduction to the source files provided. As well as these notes, the VHDL and PicoBlaze PSM files contain many comments and descriptions describing the functionality.

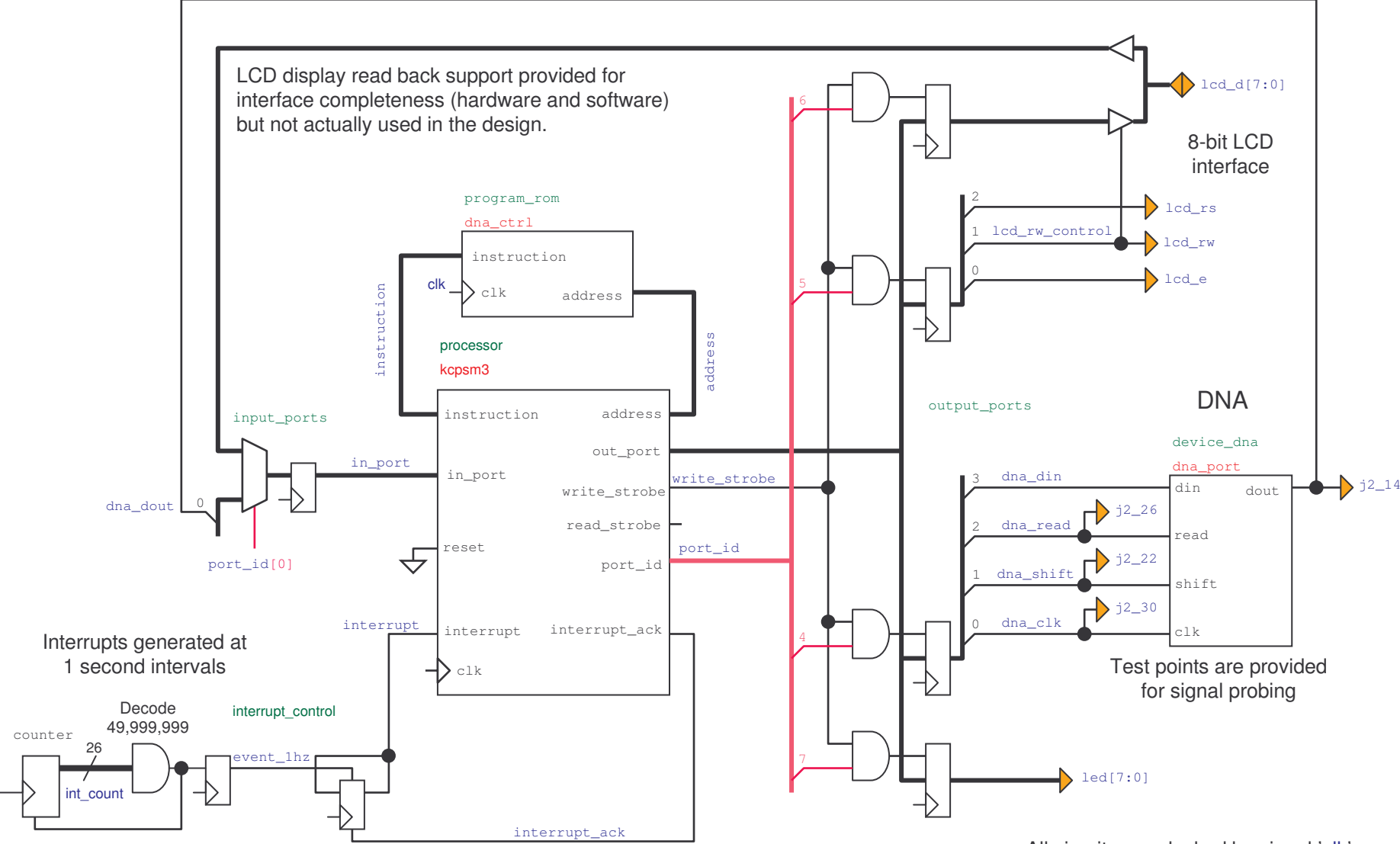
The source files provided for the reference design are.....



Note: The file shown in **green** is not included with the reference design as is provided with PicoBlaze download. Please visit the PicoBlaze Web site for your free copy of PicoBlaze, assembler and documentation.

[www.xilinx.com/picoblaze](http://www.xilinx.com/picoblaze)

# PicoBlaze Design



All circuits are clocked by signal 'clk' (50MHz) unless otherwise shown.



# How PicoBlaze Reads DNA

PicoBlaze simply generates the required CLK, READ and SHIFT control signals by writing to output port 10 hex and then reads DOUT using input port 00 hex. It is also possible for PicoBlaze to control the value applied to DIN but this connection is currently forced Low in the PicoBlaze software. The following extracts of assembly code can be compared with the oscilloscope screen shots. The timing of PicoBlaze code execution is highly predictable since all instructions take 2 clock cycles. The design uses the 50MHz oscillator provided on the starter kit and therefore every instruction executes in 40ns.

```
DNA_clk_pulse: XOR s0, DNA_clk
                OUTPUT s0, DNA_control_port
                XOR s0, DNA_clk
                OUTPUT s0, DNA_control_port
                RETURN
```

**CLK** is controlled by bit0 of register 's0'. This bit is toggled to become '1' which is output to the 'DNA\_control\_port'. It takes 2 instructions to toggle back to '0' and output again so **CLK** pulses are 80ns.

```
read_device_DNA: LOAD s0, DNA_read
                 OUTPUT s0, DNA_control_port
                 CALL DNA_clk_pulse
                 LOAD s0, DNA_shift
                 OUTPUT s0, DNA_control_port
```

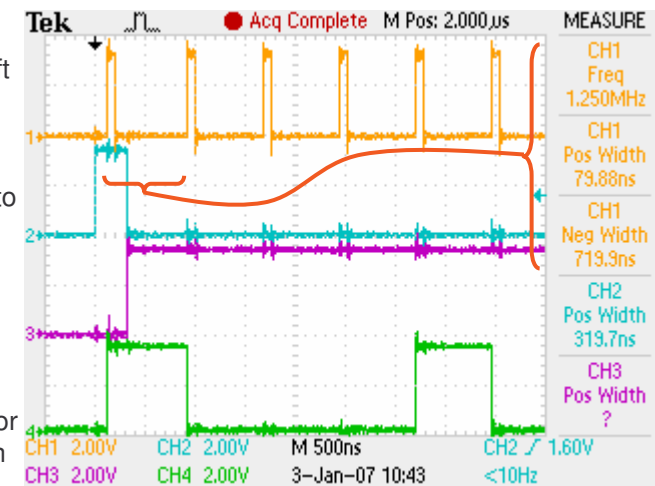
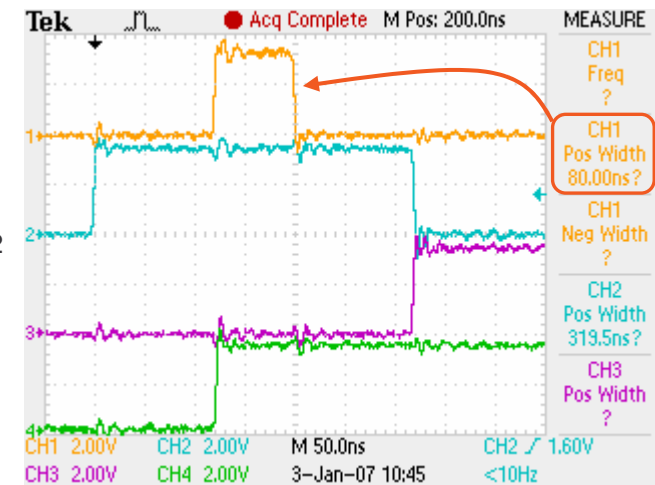
The **READ** signal is controlled by bit2 of 's0' and is driven High (for a total of 8 instructions taking 320ns) whilst CLK is pulsed. Notice how **DOUT** outputs the fixed '1' associated with bit56 of the DNA in response to the rising edge of CLK. **SHIFT** is controlled by bit1 of 's0' and is driven High as the READ is returned Low.

```
read_DNA_loop: LOAD s3, 00
               LOAD s2, 39
               INPUT s1, DNA_read_port
               TEST s1, DNA_dout
               SLA s3
               SLA s4
               SLA s5
               SLA s6
               SLA s7
               SLA s8
               SLA s9
               SLA sA
               CALL DNA_clk_pulse
               SUB s2, 01
               JUMP Z, store_DNA
               JUMP read_DNA_loop
```

The 57-bit DNA value is read into a 64-bit shift register formed of the register set....

[sA,s9,s8,s7,s6,s5,s4,s3] ←

During each iteration the value of the **DOUT** signal is read and tested to set the carry flag to the same value. The carry is then shifted into the LSB of the register set. Finally the CLK is pulsed to advance to the next bit of the DNA value. Each iteration requires a total of 19 instructions taking 760ns (note that the oscilloscope auto measurement figures are based on the timing between the CLK pulse for READ and the first CLK pulse of SHIFT which is 20 instructions taking 800ns).

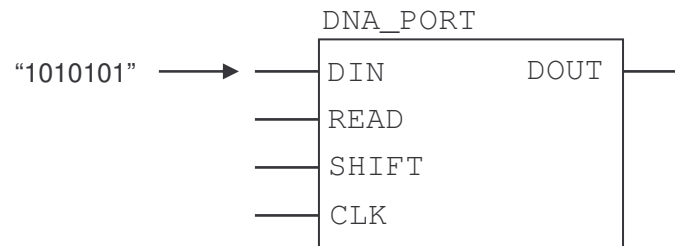


# DNA Design Exercises

Modify the PicoBlaze program (dna\_ctrl.psm) to extend the reading sequence from 57 to 64 bits.

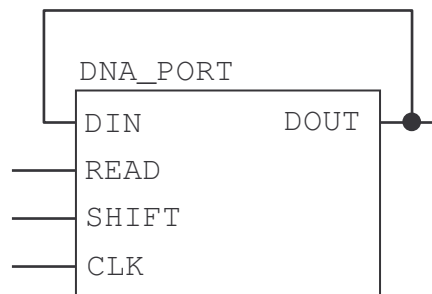
Apply the pattern “1010101” to the ‘DIN’ input whilst reading DOUT (it is what you write during the first 7 cycles of CLK that you will later observe).

Confirm that your new displayed value correctly represents the 57-bit DNA value followed by the data presented to DIN.



Retaining your new program which reads 64 bits from the DNA port, modify the hardware design such that DIN is used to recycle the DNA output value.

Confirm that your new displayed value represents the 57-bit DNA value followed by a repeat of the 7 most significant bits.



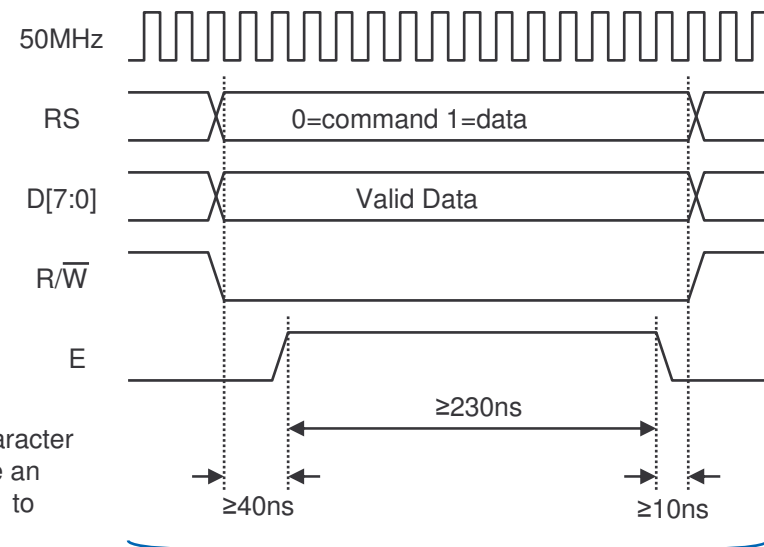
# LCD Display Timing

Once mastered, the LCD display is a practical way to display a variety of information using standard ASCII characters and even allows you to create some of your own. However, these displays are not fast and PicoBlaze uses software delay routines to slow down execution to a suitable rate.

## 8-bit Write Operation

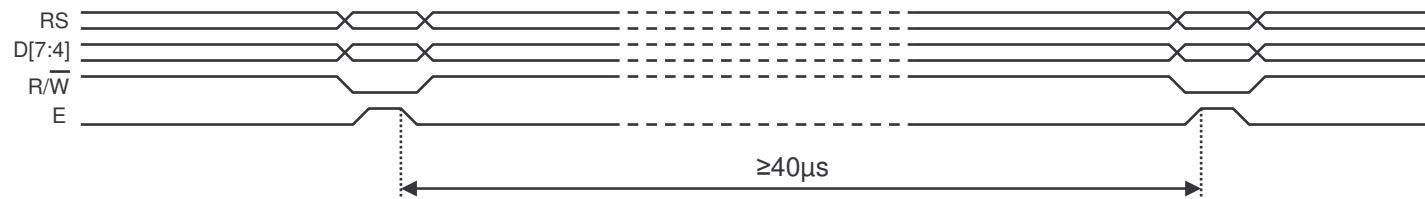
This timing diagram shows a single write operation being performed. The diagram is approximately to scale showing the minimum times allowed for setup, hold and enable pulse length relative to a 50MHz clock (20ns period). The data D[7:0], Register Select (RS) and write control (R/W) must be set up at least 40ns before the enable E goes High. Enable must be High for at least 230ns which is almost 12 clock cycles at 50MHz.

Hint – In a write only system, the R/W signal can be tied Low permanently.



## Display Communications

After initial display communication is established, all data transfers are 8-bit ASCII character codes, data bytes or 8-bit addresses. Following an 8-bit write operation, there must be an interval of at least 40µs before the next communication. This delay must be increased to 1.64ms following a clear display command.



Exercise – Implement a hardware based state machine which obeys the timing requirements for an 8-bit write of data including the 40µs delay. Modify the PicoBlaze interface and code to use your hardware circuit to write to the display. Under what circumstances would this approach be useful?

# PicoBlaze LCD Timing

In this design, PicoBlaze is used to implement the LCD communication 100% in software exploiting the fact that all instructions execute in two clock cycles under all conditions to implement accurate delays for signal and communication timing. This reference design uses the 50MHz oscillator provided on the starter kit so all instructions execute in 40ns.

```
CONSTANT delay_1us_constant, 0B
delay_1us: LOAD s0, delay_1us_constant
wait_1us: SUB s0, 01
          JUMP NZ, wait_1us
          RETURN
```

```
delay_40us: LOAD s1, 28
wait_40us: CALL delay_1us
          SUB s1, 01
          JUMP NZ, wait_40us
          RETURN
```

```
LCD_pulse_E: XOR s4, LCD_E
              OUTPUT s4, LCD_output_port
              CALL delay_1us
              XOR s4, LCD_E
              OUTPUT s4, LCD_output_port
              RETURN
```

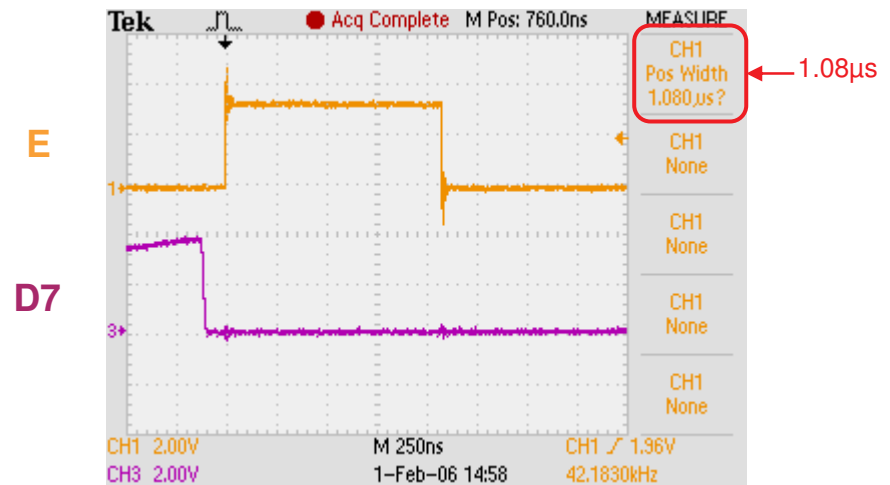
The enable 'E' pulse is formed by toggling the 'E' signal High on the output port, waiting for 1 $\mu$ s and then toggling 'E' Low again. This oscilloscope screen shot shows a single enable pulse observed at the LCD display pin. Data bit D7 is also shown (see code on next page for more details).

Exercise – 1 $\mu$ s is greater than the 230ns minimum requirement for the enable pulse width. Modify the code to generate an enable pulse which is 250ns.

The PicoBlaze program supplied implements a 1 $\mu$ s delay in software which it then uses as the base for all operations. This subroutine is invoked with a 'CALL delay\_1us' which then LOADs register s0 with 11 (0B hex). This in turn causes the 'SUB' and 'JUMP NZ' instructions to execute 11 times before 'RETURN' completes the routine. This means that a delay of exactly 1 $\mu$ s is formed by the 25 instructions each taking 40ns.

Creating other delays such as the 40 $\mu$ s required between 8-bit transfers is then a simple case of calling the 'delay\_1us' the appropriate number of times. In this case 40 (28 hex) times results in slightly more than 40 $\mu$ s due to the executions of instructions within the routine itself.

Exercise – Calculate the exact number of instructions, clock cycles and delay provided by the 'delay\_40us' subroutine. Prove your result either by simulation or preferably by running a test design on the Starter Kit and making measurements.



# PicoBlaze LCD Timing

```
LCD_write_data: OUTPUT s5, LCD_output_port ;data output
LOAD s4, 04 ;RS=1 Data, RW=0 Write, E=0
OUTPUT s4, LCD_control_port
CALL LCD_pulse_E
CALL delay_40us ;wait >40us
RETURN
```

The writing of 8-bit data is then achieved simply by outputting the desired data value (in this case provided in register 's5') to the D[7:0] data bits of the LCD display and setting the control signals to RW=0 and RS=1. Writing command instructions to the display is the same but requires that RS=0.

```
disp_DNA: LOAD s5, character_D
CALL LCD_write_data
LOAD s5, character_N
CALL LCD_write_data
LOAD s5, character_A
CALL LCD_write_data
RETURN
```

This code shows how the text string 'DNA' is written to the LCD display by repeatedly loading register 's5' with each ASCII code and calling the above routine. Note how the 'LCD\_write\_data' routine includes the 40µs delay required between each write operation.

The connections to the LCD display can be probed on the LCD module itself.

Exercise – Connect an oscilloscope and confirm that there is a delay of at least 40µs between each write ('E' pulse).

The LCD display is powered by a +5v supply. This means that if the display is read it could potentially drive the data signal lines above the maximum level (4.6v) allowed by the Spartan-3A pins which are powered by 3.3v. Therefore the display data pins have been connected via 390Ω series resistors to limit the current and voltage during read operations.



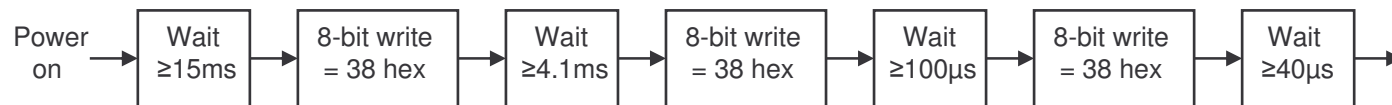
Hint – This design provides all the mechanisms (hardware and software) to enable the display to be read but does not actually need to use it. Most applications of character modules only require write operations but the interface to LCD graphic modules are identical and read operations are very useful when performing plotting algorithms.

# LCD Display Initialisation

Before the display can be used for the first time, there is an initialisation sequence which must be followed to allow communication to take place. These sequences are ideally suited to an 8-bit processor such as PicoBlaze. Besides the relative complexity of the sequence, the process is only executed once and then the processor is available to perform other tasks including the control on the display itself.

**Hint** - The PicoBlaze code provided includes a subroutine called 'LCD\_reset' which performs this initialisation sequence.

The first part of this sequence is to establish that the 8-wire data interface is being used. Note that some of these delays are in milliseconds.



The 8-wire interface is now established which means that all subsequent communication is formed of the 8-bit writes described previously. Note that these writes are all commands and therefore RS must be Low. The next part of the sequence is used to establish how the display should operate.



Function Set = 38 hex : This code describes the display type as being 2 lines, 5×7 dots per character and using 8-wire communication (again).

Entry Mode = 06 hex

0 0 0 0 0 1 1 0

- Do not shift display left or right when writing a character
- Increment cursor position when writing a character

Display Control = 0C hex

0 0 0 0 1 1 0 0

- Cursor blinking (character flashes) off
- Cursor (line under character) off
- Display is ON

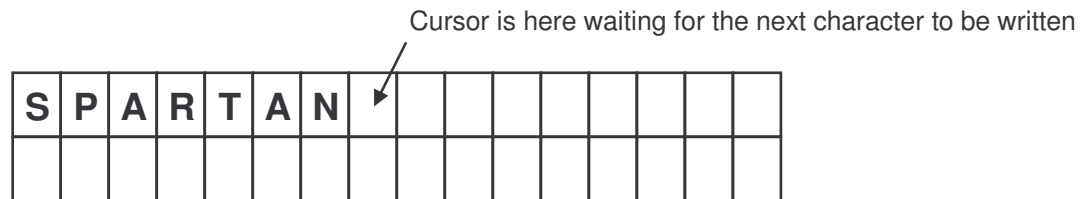
**Exercise** – Implement a hardware state machine which can perform the LCD initialisation sequence. Compare the size of your implementation with the 96 slices required to implement a PicoBlaze processor. Does the claim of the first paragraph hold true?

# LCD Display Control

The most common operation is simply to write ASCII characters to be displayed. These are considered to be data and hence RS will be High. With the display set up and reset as described previously, writing a series of characters will automatically result in their display on the top line of the display.

```
LOAD s5, character_S
CALL LCD_write_data
LOAD s5, character_P
CALL LCD_write_data
LOAD s5, character_A
CALL LCD_write_data
LOAD s5, character_R
CALL LCD_write_data
LOAD s5, character_T
CALL LCD_write_data
LOAD s5, character_A
CALL LCD_write_data
LOAD s5, character_N
CALL LCD_write_data
```

Following the execution of this code, the display will look like this.



If you continue to write characters, they will eventually fall off the end of the first line but they will not automatically appear on the second line. This is because the memory map of this display is not consecutive from line to line. The diagram below shows the memory address of each character location.

To set the cursor in the position required for subsequent characters to be displayed, the 8-bit memory address of that position must be written to the display as a command (RS=0). Note that this can not be confused with any of the other commands since the MSB is always '1' in these addresses.

The ability to move to any particular location and write a few characters is ideal for the display of various information as it becomes available such as time, date, measurements and short status messages.

The PicoBlaze code provided includes a subroutine called 'LCD\_cursor'. Load register 's5' with the desired position. The upper nibble (value 1 or 2) will define the line and the lower nibble the character position on the line (0 to F). E.g. s5 = 2B will position the cursor at the 12<sup>th</sup> position on line 2.

1	80	81	82	83	84	85	86	87	88	89	8A	8B	8C	8D	8E	8F
2	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	CA	CB	CC	CD	CE	CF
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F