Implementing and Testing Efficient Video Line Stores

(Includes 7 ready to use macros for popular line lengths)

Ken Chapman
Xilinx Ltd
28th June 2006

Rev.1

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 not-withstanding the failure of the essential purpose of any limited remedies herein.

These design modules are **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 these reference designs would be gratefully received by the author.

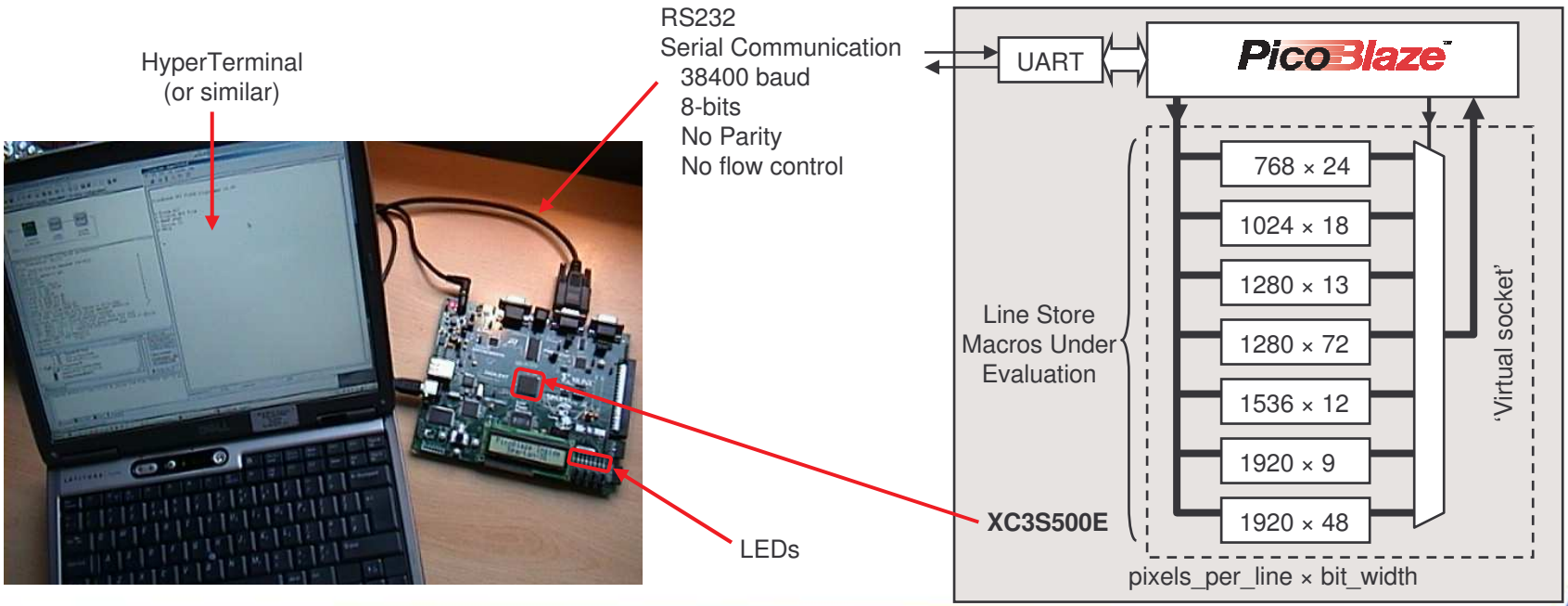
Ken Chapman
Senior Staff Engineer – Spartan Applications Specialist
email: chapman@xilinx.com

Introduction

This reference design for the Spartan-3E Starter Kit serves two quite different purposes. As such, you may be interested in one particular aspect or both.

Hardware Development and Testing of Macros - The design provides an example of how the Starter Kit can be used as a test bed for macros enabling real implementations to be evaluated during development. This technique can be applied to many parts of designs and helps reduce the burden of testing and debugging when putting a final system together. Such 'real simulation' can also be significantly faster than using a traditional software simulator since the logic is working at full clock rate. When testing a macro, it is not always necessary to have all the peripherals and connectors that the final system will have. The macro is effectively placed in a 'virtual socket' within the Spartan-3E device and some means provided to stimulate and monitor the macro. In this example PicoBlaze is used as a convenient way to control and monitor the macros under test with an RS232 (UART) link to the PC providing the human interface (HyperTerminal). So in fact the test design only uses 2 pins on the Spartan device and all others are 'virtual pins' (OK, I used the LEDs too).

Efficient Video Line Store Macros - In this case the macros under evaluation are a set of highly efficient video line stores implemented using Block Memory (BRAM). Line stores are often used when performing image processing algorithms. In recent years the resolution of images has been increasing resulting in more pixels per line and pixels of greater resolution (more bits to represent each colour). Unless these line stores are implemented efficiently it becomes very difficult to implement an adequate number of line stores on a Spartan-3E device. This reference design provides 7 ready to use line store macros all of which can be evaluated using this design. If your main interest in this reference design is purely to use one or more of these macros then you may wish to advance directly to page 16 (without passing GO and without collecting £200!).



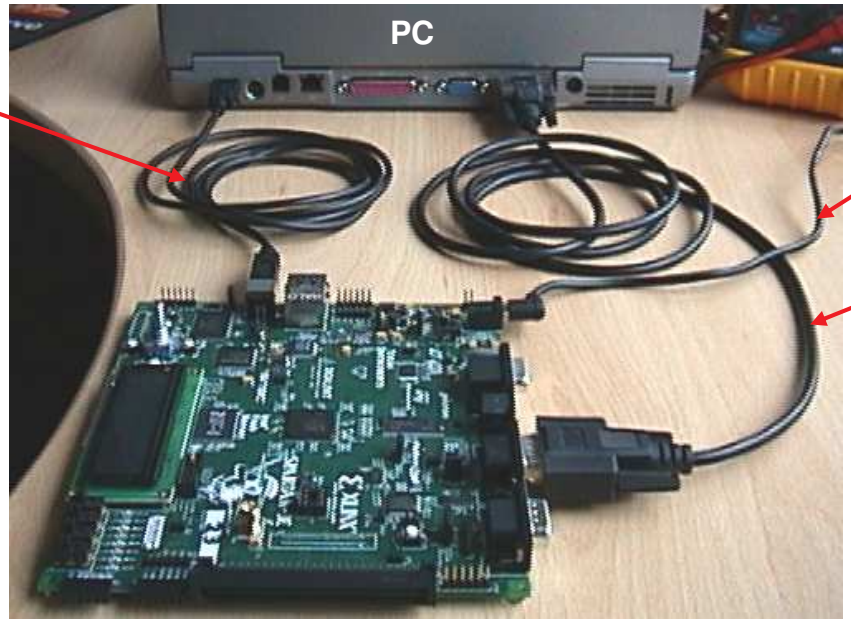
Using the Test Design

The design is provided as a configuration BIT file for immediate programming of the Spartan XC3S500E provided on the Spartan-3E Starter Kit. Source design files are also provided for those more interested in the intricacies of the design itself.

USB cable plus some devices on board essentially provide the same functionality as a Platform Cable USB and is used in conjunction with iMPACT.

Initially used to configure the Spartan-3E with the PicoBlaze based design (BIT file).

Can subsequently be used to update the PicoBlaze program stored in an internal Block Memory (BRAM) allowing rapid software changes and experiments (see JTAG_loader documentation provided with PicoBlaze) .



+5v supply
Don't forget to switch the board on too!
(SWP)

RS232 Serial Cable.
Used for operating the design and obtaining results.

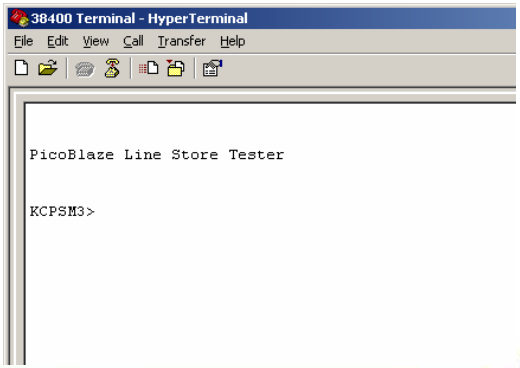
Cable connects J9 on the board to your PC serial port. For this you will need a male to female straight through cable (critically pin2-pin2, pin3-pin3 and pin5-pin5).

Quick Start - Configure the Spartan-3E with the design

To make this task really easy the first time, unzip all the files provided into a directory and then….

double click on 'install_line_store_tester.bat'.

Assuming you have the Xilinx software installed, your board connected with the USB cable and the board powered (don't forget the power switch), then this should open a DOS window and run iMPACT in batch mode to configure the Spartan-3E with the design (configuration BIT file). You should see the LED 'LD0' turn on and a message appear on your PC terminal window (see following pages for HyperTerminal set up).

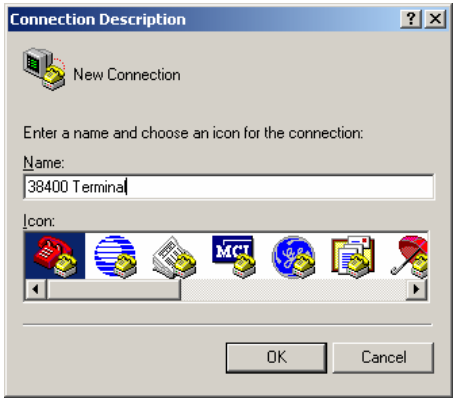


Serial Terminal Setup

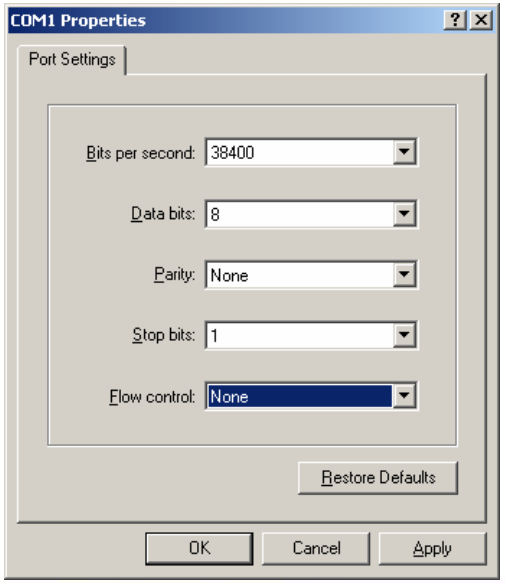
An RS232 serial link is used to communicate with the design. Any simple terminal program can be used, but HyperTerminal is adequate for the task and available on most PCs.

A new HyperTerminal session can be started and configured as shown in the following steps. These also indicate the communication settings and protocol required by an alternative terminal utility.

1) Begin a new session with a suitable name.
HyperTerminal can typically be located on your PC at
Programs -> Accessories -> Communications -> HyperTerminal.



2) Select the appropriate COM port (typically COM1 or COM2) from the list of options. Don't worry if you are not sure exactly which one is correct for your PC because you can change it later.



3) Set serial port settings.

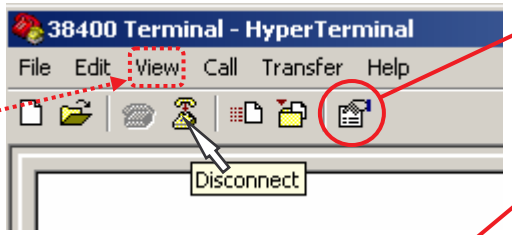
Bits per second : 38400
Data bits: 8
Parity: None
Stop bits: 1
Flow control: None

Go to next page to complete set up…

HyperTerminal Setup

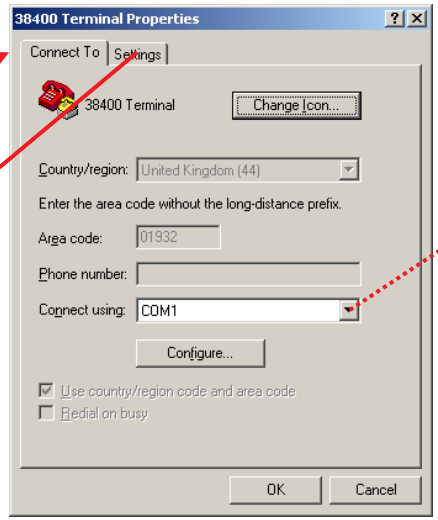
Although steps 1, 2 and 3 will actually create a Hyper terminal session, there are few other protocol settings which need to be set or verified for the PicoBlaze design to work as expected.

4 - Disconnect



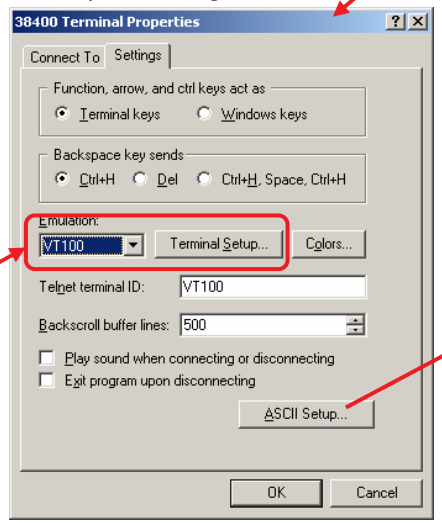
Optional step…..
Set Font to
Courier New,
Regular, 10

5 - Open the properties dialogue

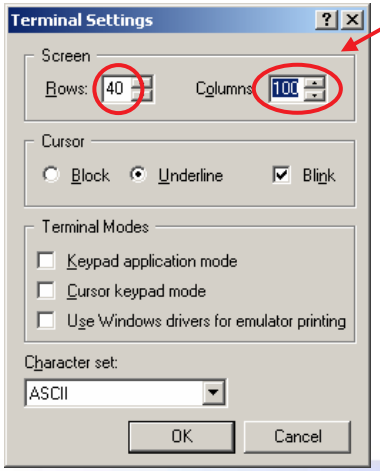


To select a different COM port and change settings (if not correct).

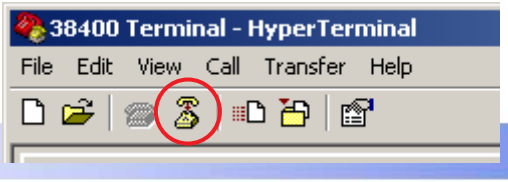
6 - Open Settings

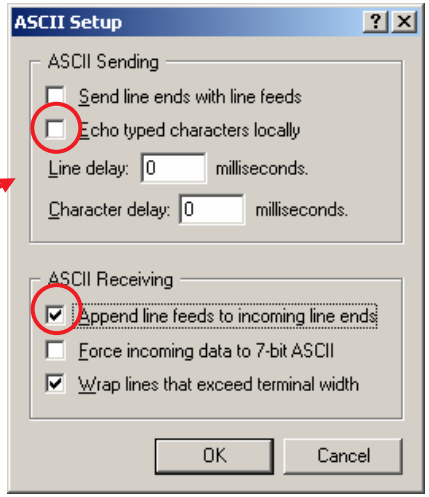


7 - Select VT100 and then click ‘Terminal Setup’
Set ‘Rows’ to 40 and ‘Columns’ to 100.
(You will probably need to stretch main screen later to fit this size).



9 - ‘OK’ the boxes to get back to main screen and then Connect.





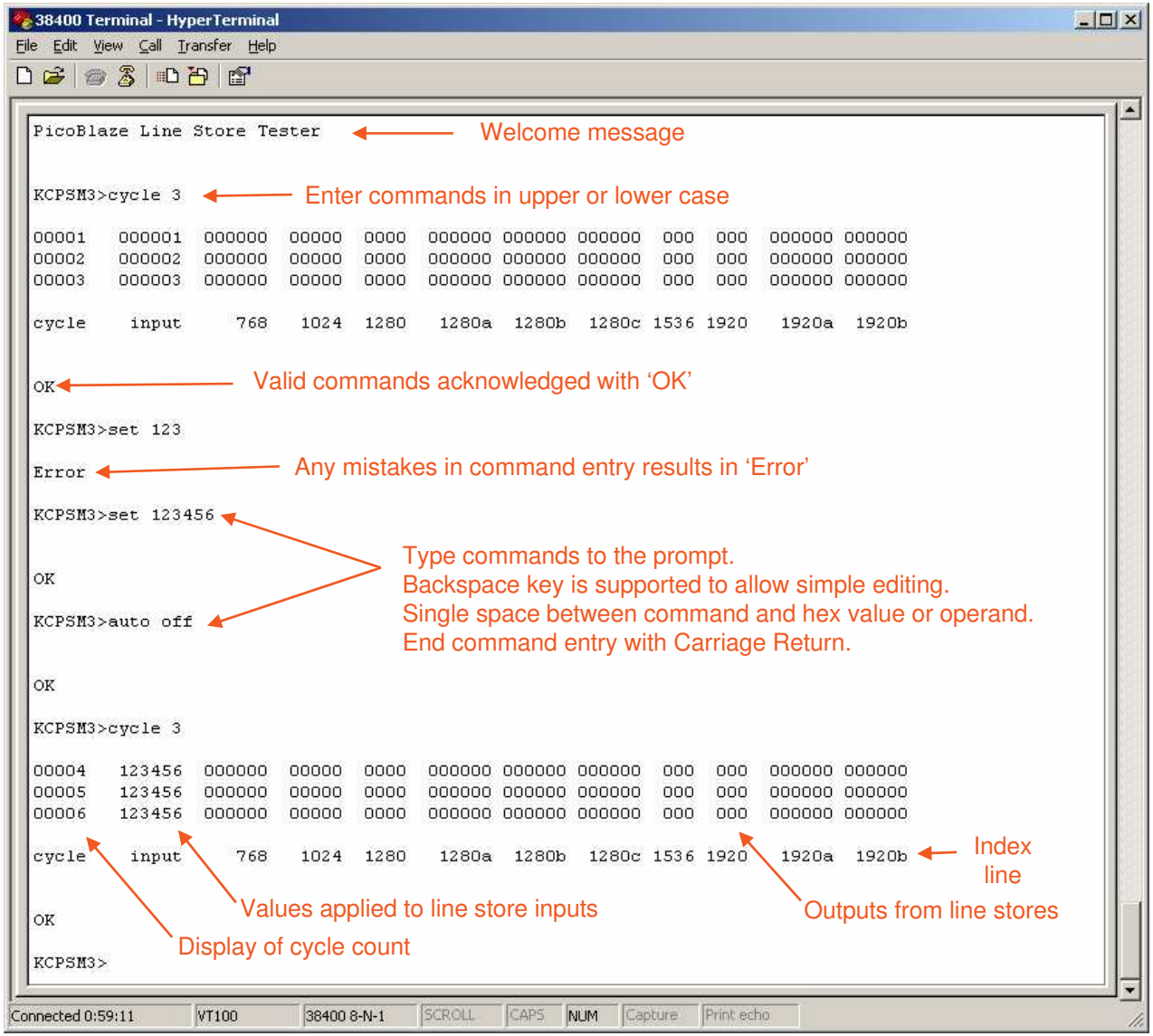
8 - Open ASCII Setup

Ensure boxes are filled in as shown.

The design will echo characters that you type so you do not need the ‘Echo typed characters locally’ option.

The design transmits carriage return characters (OD_{HEX}) to indicate end of line so you do need the ‘Append line feeds to incoming line ends’ option to be enabled.

Terminal Commands



All tests are performed using simple commands (see next page) entered at the terminal.

The value at the output of each line store is display with a simple 'index' line to help identify each output.

Terminal Commands

SET hhhhhh - Set the 24-bit value to be applied to the input of all line stores where 'hhhhhh' is a 6 digit hex value.
Line stores requiring less than 24-bits will are provided with the least significant bits of this value.
By default the design is initialised or 'RESET' with the value set to 000001 hex.

CYCLE n - Test the line stores for 'n' clock cycles where 'n' is a decimal value in the range 1 to 9999.
The input to the line stores during each test cycle will be depend on previous use of SET and AUTO commands.
The outputs from all line stores will be displayed depending on the previous use of FAST command.

AUTO ON / **AUTO OFF** – With AUTO mode turned on, the 24-bit value applied to the inputs of the line stores will automatically increment after each test cycle. By default the design is initialised or 'RESET' with AUTO turned ON and the status of AUTO is indicated by LED 'LD0' on the board.

FAST ON / **FAST OFF** – With FAST mode turned on, the display of results during a CYCLE command are suppressed except for the last cycle. In this design the speed of the test is limited by the communication rate of the RS232 interface. When FAST mode is enabled, the speed benefits of testing using real hardware become apparent. By default the design is initialised or 'RESET' with FAST turned OFF and the status of FAST is indicated by LED 'LD1' on the board.

RESET - Initialise the test design.
All line stores are purged of existing values by repeatedly writing the value 000000 hex.
Cycle counter is reset.
Input to line stores set to 000001 hex with AUTO mode ON.
FAST mode is turned off.

Example Test 1

```
KCPSM3>cycle 3

00001   000001  000000  00000  0000  000000 000000 000000  000  000  000000 000000
00002   000002  000000  00000  0000  000000 000000 000000  000  000  000000 000000
00003   000003  000000  00000  0000  000000 000000 000000  000  000  000000 000000

cycle    input     768   1024  1280   1280a  1280b  1280c 1536 1920   1920a  1920b


OK

KCPSM3>fast on


OK

KCPSM3>cycle 1915

01918   00077E  00047E  0037E  027E  00027E 000000 000000  17E  000  000000 000000

cycle    input     768   1024  1280   1280a  1280b  1280c 1536 1920   1920a  1920b


OK

KCPSM3>fast off


OK

KCPSM3>cycle 5

01919   00077F  00047F  0037F  027F  00027F 000000 000000  17F  000  000000 000000
01920   000780  000480  00380  0280  000280 000000 000000  180  000  000000 000000
01921   000781  000481  00381  0281  000281 000000 000000  181  001  000001 000000
01922   000782  000482  00382  0282  000282 000000 000000  182  002  000002 000000
01923   000783  000483  00383  0283  000283 000000 000000  183  003  000003 000000

cycle    input     768   1024  1280   1280a  1280b  1280c 1536 1920   1920a  1920b


OK
```

Hex value applied to line stores

Decimal Test cycle counter since start

'Index' tells you that column of results above are the outputs from a 1920 stage line store

Using the default initial settings the CYCLE command will generate input values (in hexadecimal) that match the test cycle count. All cycles are displayed and after the last line there is an 'index' to identify each line store output.

Although all line stores are being tested in parallel by this design, in this example my interest was to check the macros supporting 1920 stages of delay. To perform and display 1915 test cycles would take approximately 42 seconds so the fast mode is useful to make rapid progress; actually appears to be instantaneous ☺.

Hint – Use FAST ON to get close to the cycles of interest and then revert to FAST OFF to see the detail.

As the 1921st test cycle is reached, the output from the 1920 stage line store is showing the value 000001 hex which was input during the 1st test cycle.

Example Test 2

```
KCPSM3>set 123456

OK
KCPSM3>cycle 1
00001   123456  000000  00000  0000  000000 000000 000000  000  000  000000 000000
cycle    input     768   1024  1280   1280a  1280b  1280c 1536 1920   1920a  1920b

OK
KCPSM3>set 789abc

OK
KCPSM3>cycle 1
00002   789ABC  000000  00000  0000  000000 000000 000000  000  000  000000 000000
cycle    input     768   1024  1280   1280a  1280b  1280c 1536 1920   1920a  1920b

OK
KCPSM3>set aaaaaa

OK
KCPSM3>auto off

OK
KCPSM3>cycle 1285
```

In this example a specific pattern is being applied to the line store inputs.

First the value 123456 hex is set and then applied for one test cycle

Second the value 789ABC hex is set and then applied for one test cycle

Finally the value AAAAAA hex is set and by using the AUTO OFF command this value will be applied for all subsequent test cycles.

Again, although all line stores are being tested in parallel, in this case my interest was the line store macros providing1280 stages of delay.

Example Test 2 continued

```
.
.
.
01266   AAAAAA  AAAAAA  2AAAA   0000   000000 000000 000000   000   000   000000 000000
01267   AAAAAA  AAAAAA  2AAAA   0000   000000 000000 000000   000   000   000000 000000
01268   AAAAAA  AAAAAA  2AAAA   0000   000000 000000 000000   000   000   000000 000000
01269   AAAAAA  AAAAAA  2AAAA   0000   000000 000000 000000   000   000   000000 000000
01270   AAAAAA  AAAAAA  2AAAA   0000   000000 000000 000000   000   000   000000 000000
01271   AAAAAA  AAAAAA  2AAAA   0000   000000 000000 000000   000   000   000000 000000
01272   AAAAAA  AAAAAA  2AAAA   0000   000000 000000 000000   000   000   000000 000000
01273   AAAAAA  AAAAAA  2AAAA   0000   000000 000000 000000   000   000   000000 000000
01274   AAAAAA  AAAAAA  2AAAA   0000   000000 000000 000000   000   000   000000 000000
01275   AAAAAA  AAAAAA  2AAAA   0000   000000 000000 000000   000   000   000000 000000
01276   AAAAAA  AAAAAA  2AAAA   0000   000000 000000 000000   000   000   000000 000000
01277   AAAAAA  AAAAAA  2AAAA   0000   000000 000000 000000   000   000   000000 000000
01278   AAAAAA  AAAAAA  2AAAA   0000   000000 000000 000000   000   000   000000 000000
01279   AAAAAA  AAAAAA  2AAAA   0000   000000 000000 000000   000   000   000000 000000
01280   AAAAAA  AAAAAA  2AAAA   0000   000000 000000 000000   000   000   000000 000000
01281   AAAAAA  AAAAAA  2AAAA   1456   123456 000000 000000   000   000   000000 000000
01282   AAAAAA  AAAAAA  2AAAA   1ABC   789ABC 000000 000000   000   000   000000 000000
01283   AAAAAA  AAAAAA  2AAAA   0AAA   AAAAAA 000000 000000   000   000   000000 000000
01284   AAAAAA  AAAAAA  2AAAA   0AAA   AAAAAA 000000 000000   000   000   000000 000000
01285   AAAAAA  AAAAAA  2AAAA   0AAA   AAAAAA 000000 000000   000   000   000000 000000
01286   AAAAAA  AAAAAA  2AAAA   0AAA   AAAAAA 000000 000000   000   000   000000 000000
01287   AAAAAA  AAAAAA  2AAAA   0AAA   AAAAAA 000000 000000   000   000   000000 000000

cycle    input     768   1024   1280    1280a   1280b   1280c 1536 1920    1920a   1920b


OK

KCPSM3>
```

24-bit macro

Because the CYLCE command was issued with FAST OFF all results are displayed (but it took a minute to run!).

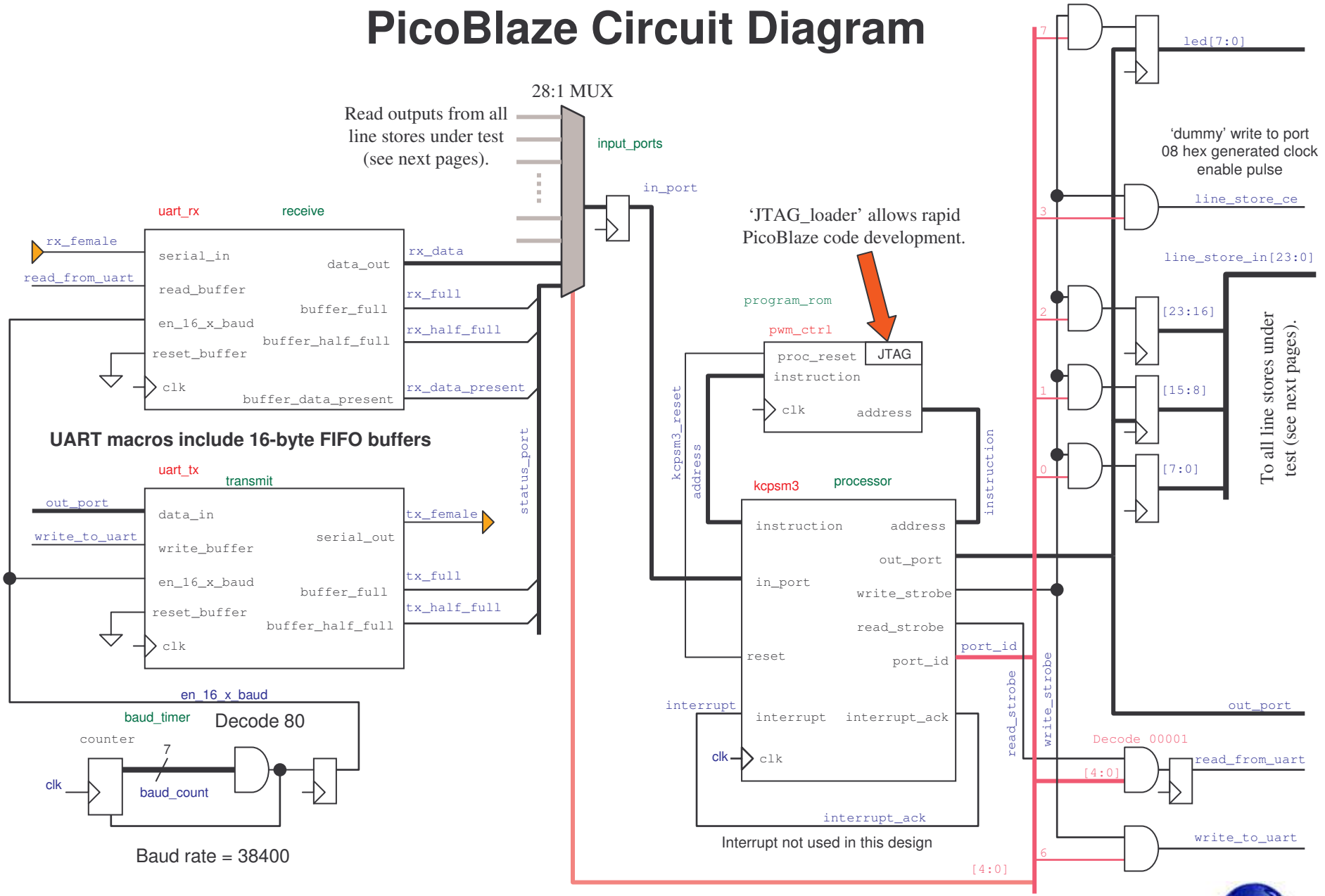
You can see that the shorter line stores have filled with the fixed AAAAAA hex value (some line stores are less than 24-bits resulting in what appears to be a different value at first glance) and those that are longer have yet to show any values other than their initial clear states.

As the test reaches cycle 1281 the special values applied during test cycles 1 and 2 appear at the outputs of the 1280 stage line stores.

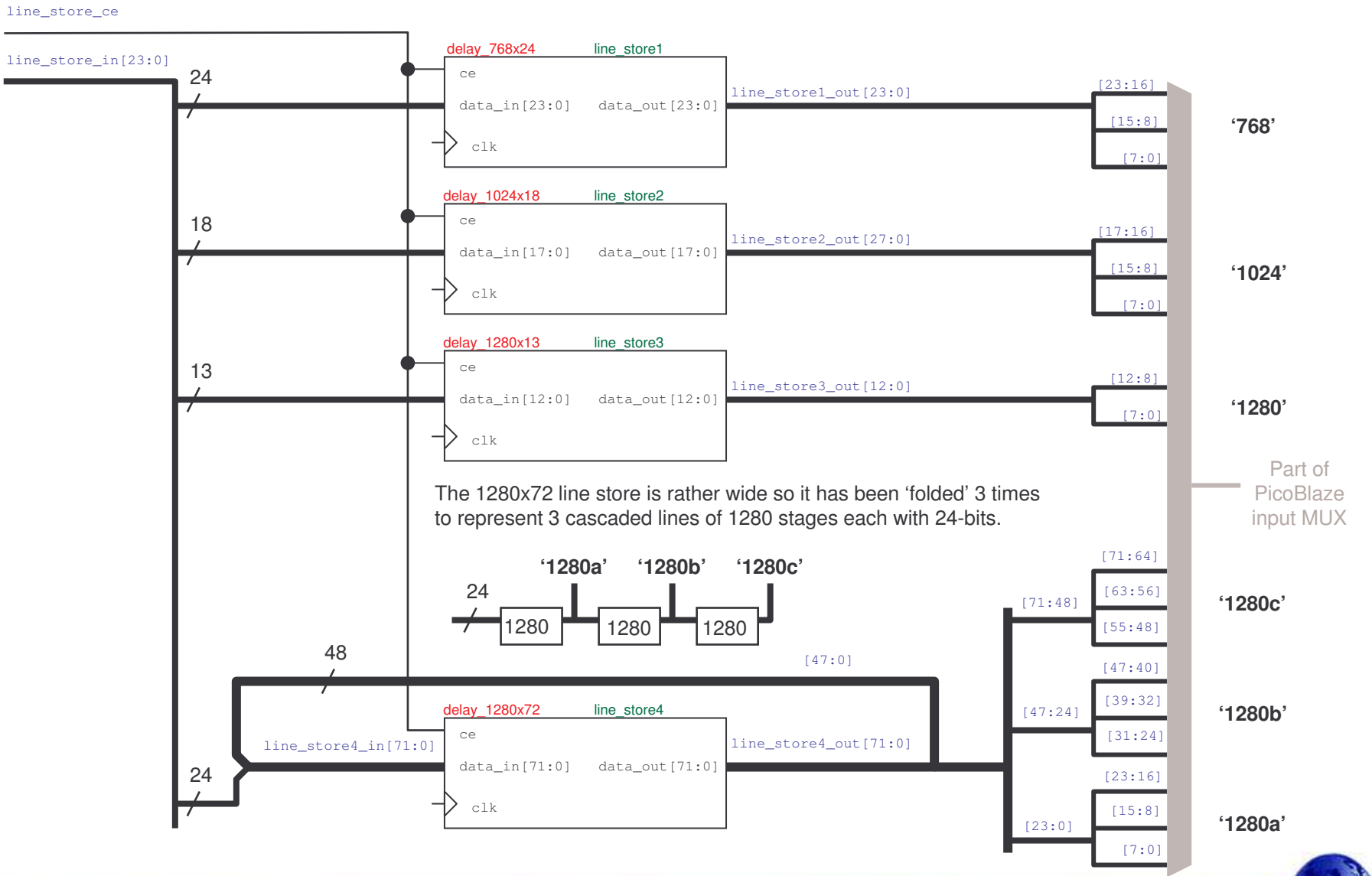
This macro supports 13-bit data so the original 24-bit value has been truncated to just the least significant bits

```
123456 hex = 0001 0010 0011 0100 0101 0110
789ABC hex = 0111 1000 1001 1010 1011 1100
```

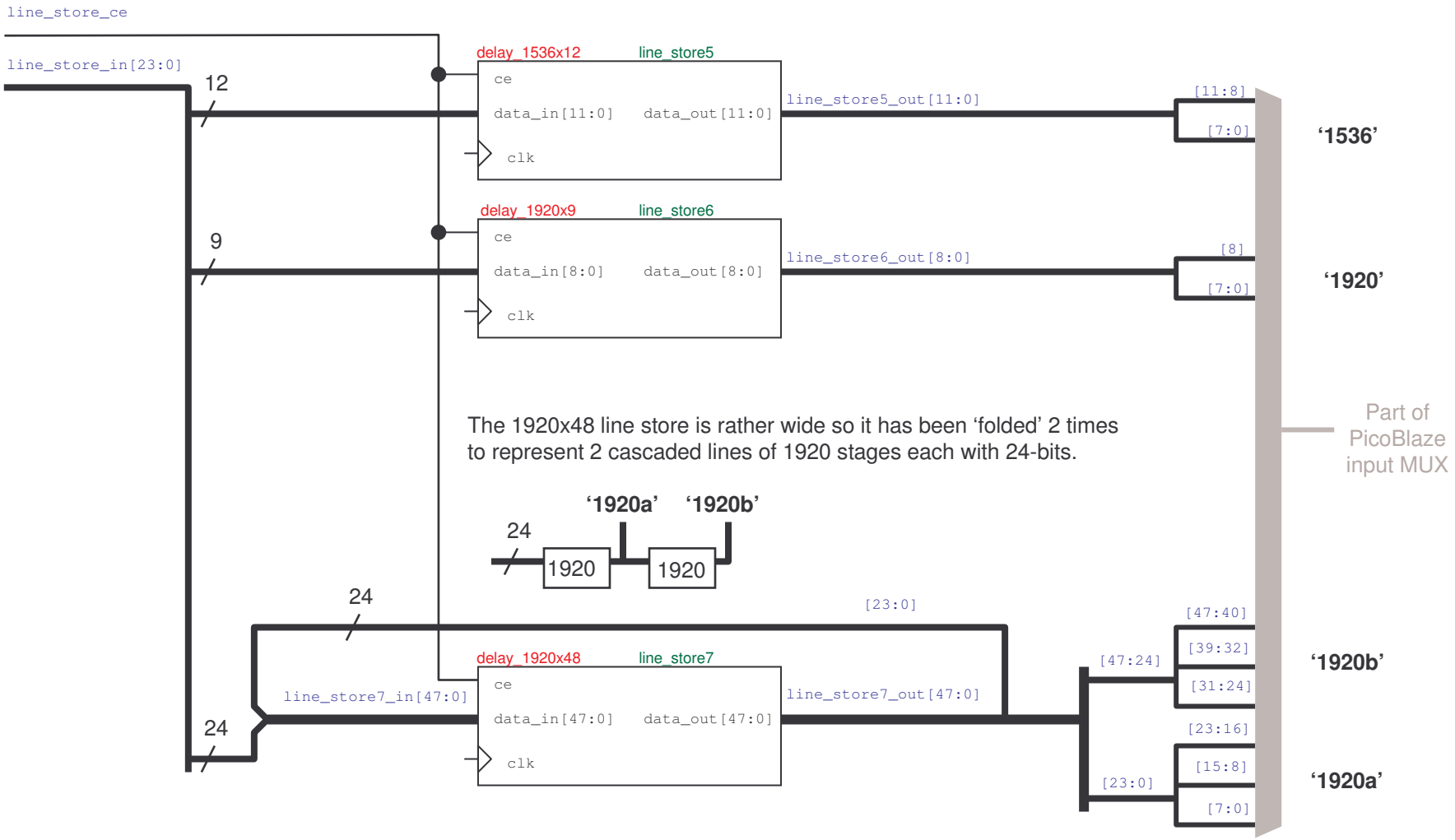
PicoBlaze Circuit Diagram



Line Stores Under Test - Circuit Diagram 1



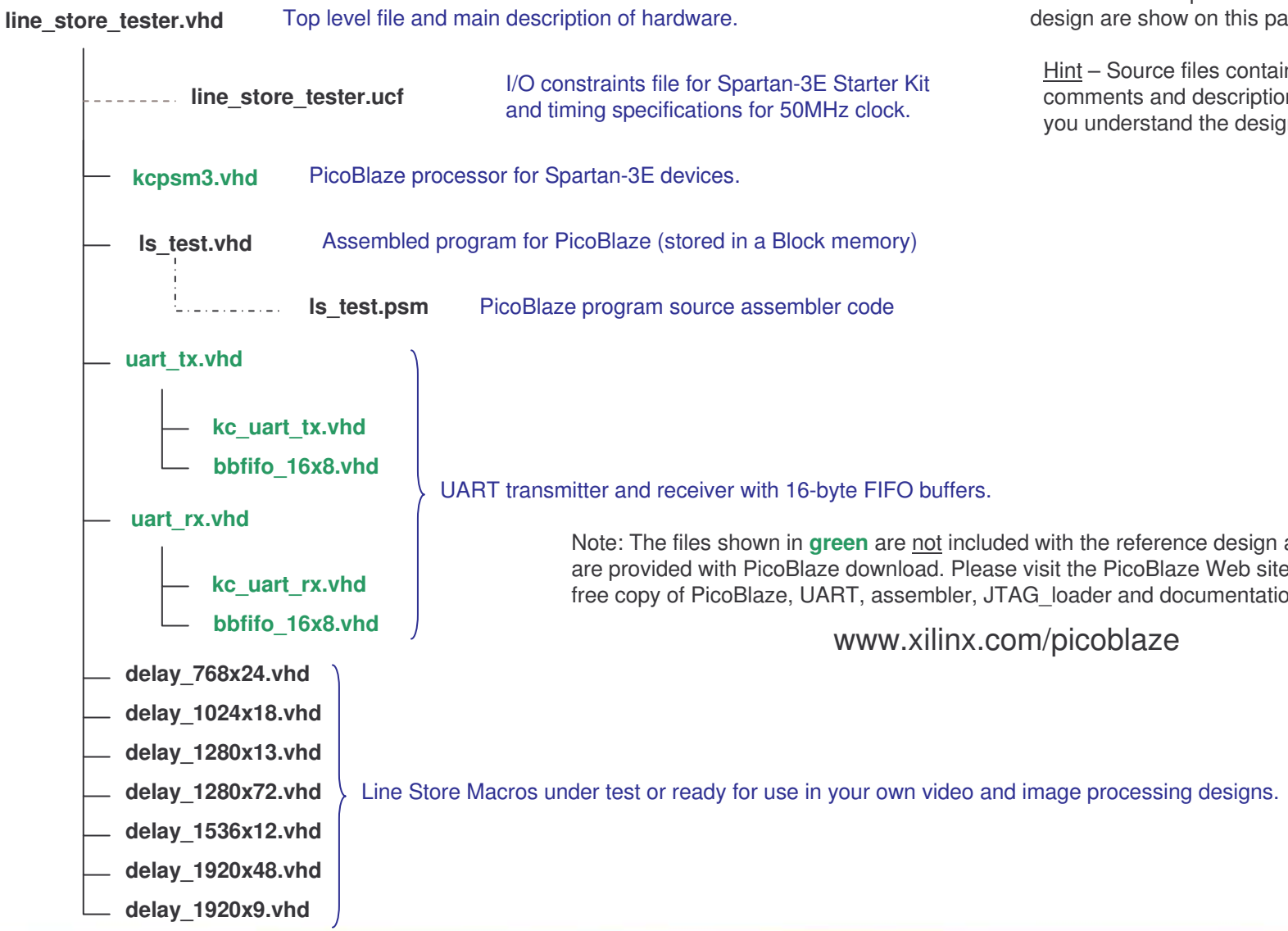
Line Stores Under Test - Circuit Diagram 2



Design Files

The source files provided for the reference design are show on this page.

Hint – Source files contain many comments and descriptions to help you understand the design further.

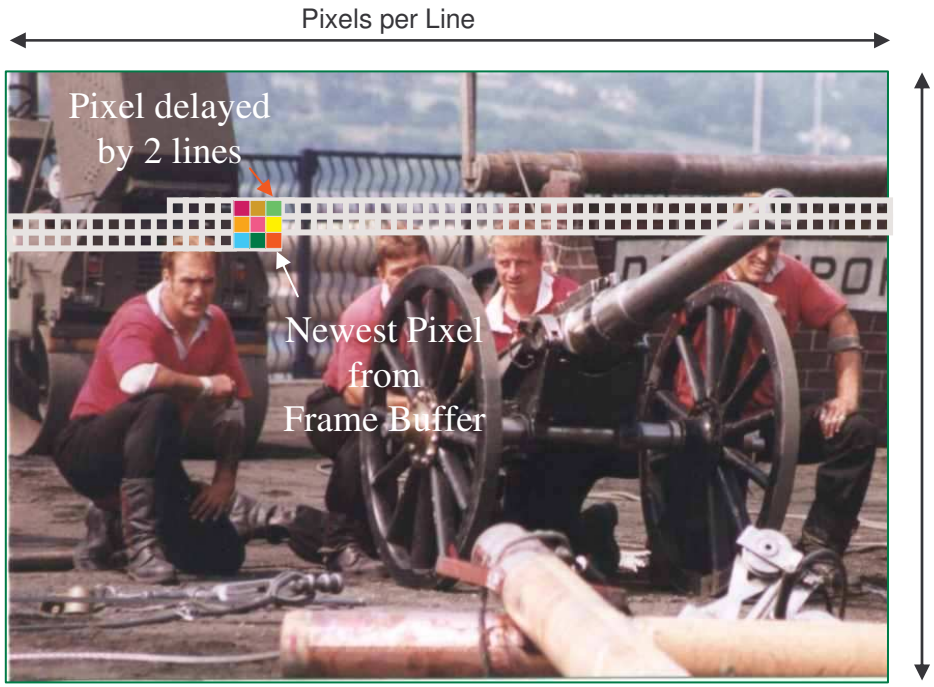


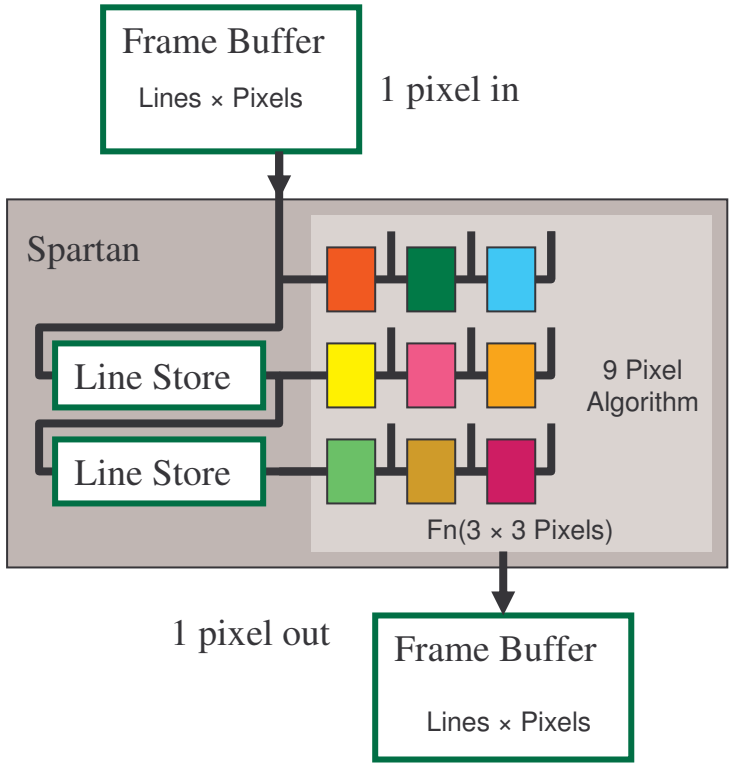
Note: The files shown in **green** are not included with the reference design as they are provided with PicoBlaze download. Please visit the PicoBlaze Web site for your free copy of PicoBlaze, UART, assembler, JTAG_loader and documentation.

www.xilinx.com/picoblaze

Line Stores

A video image is made up of many lines with each line being formed of many pixels. As you would expect, a Higher resolution display has a larger number of pixels per line and a larger number of lines per screen than a lower resolution display. A pixel is typically described as an intensity figure represented by an 8, 10 or 12-bit value. Of course colour displays will need to describe the intensity of red, green and blue elements of each pixel requiring anything up to (12×3) 36-bits to describe. Some systems may also include pixel correction value, infrared data etc.





In the majority of applications, it is impractical to store one or more whole images inside a Spartan devices and therefore some form of external storage is required. The bandwidth of this external memory then restricts how many pixels can be accessed at the pixel rate which can cause problems when implementing 2-D algorithms. As shown above, an image is formed by scanning from left to right and top to bottom. This only requires one pixel to be read from external memory at the pixel rate (although higher clock rates may be required to access multiple bytes for red, green and blue definition of each pixel). of higher quality. In 2-D processing, it is necessary to have access to all the pixels in a block. A simple shift register is all that is required to remember several pixels on the line currently being read but to access the pixel above or below initially implies another read form the external frame buffer which is probably not achievable in the time available (especially as the address would not be consecutive and suitable for burst reading). The solution are line stores implemented using on-chip memory which hold all the pixels for a complete line. These are in effect shift registers but only allowing access at their beginning and end. As a pixel in written into the line store, the output is presenting the corresponding pixel of the line directly above.

Typical Line Store Sizes

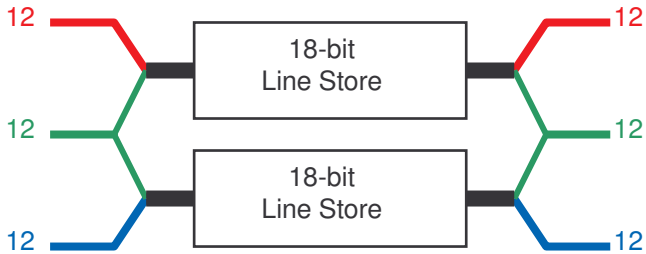
There are so many different sizes and formats of video displays that it is hard to keep up with them all. However, the following short table covers many of the more common sizes indicating the typical range of pixels per line. It is interesting to observe how the number of pixels typically relate to a multiples of 64, 128, 256 or 512 pixels even if their lengths are not powers of 2 in their own right.

The number of bits required to represent each pixel is really of less significance in terms of the implementation but is important to consider because of the demands it will place on a design overall. A dominant issue when implementing line stores efficiently in a Spartan device has more to do with human nature than to do with anything technical. We seem to have an overwhelming desire to keep things separated into their own 'little boxes' and functions. When it comes to implementing line stores, engineers often have this unnecessary desire to implement each line store in total isolation. They often fail to notice how red, green and blue pixel data can either be packed together to form a single wider data value or how a pixel value could be split into several pieces to make use of spare capacity in other line stores and then recombined later. So please bare this in mind that whilst using the macros supplied.

Line length	Length factors	Typical Bit Widths
640	5 × 128	8 BW, RGB
720		8 BW, RGB
768	3 × 256	8 BW, RGB
800	25 × 32	8 BW, RGB
1024	1 × 1024	8,10 BW,RGB
1152	9 × 128	8,10,12 RGB
1280	5 × 256	8,10,12 RGB
1366		10,12 RGB
1536	6 × 256 or 3 × 512	10,12 RGB
1920	15 × 128	10,12 RGB

12-bit red, green and blue amounts to 36-bits of data per pixel. There is no reason to keep the different colours in separate physical line stores as all will be delayed by the same amount.

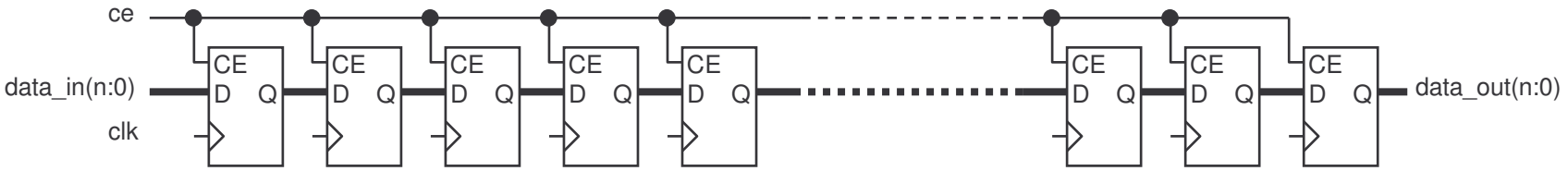
It would also be possible to split pixel data to fill otherwise unused capacity of a line store rather than use separate line stores for each colour.



Note that packing of pixels from different lines is also possible since all pixels on all lines advance at the same time (see page 19).

Line Stores Provided

This reference design package contains 7 pre-implemented line stores for the most common line lengths encountered. In each case the objective has been to provide the highest number of 'line-bits' per Block Memory (BRAM) and enable highly efficient designs to be implemented in Spartan devices. A 'line-bit' is a delay of 1-bit corresponding to the number of pixels on one line of a video display. The equivalent circuit for a line store macro is shown below.



Since the bit sizes of pixels are so variable, the focus was simply to provide the maximum number of line-bits in each case. Macros can then be connected in parallel or packed to achieve the total number of line-pixels required by your system. There is an example of such 'line-pixel packing' on the next page.

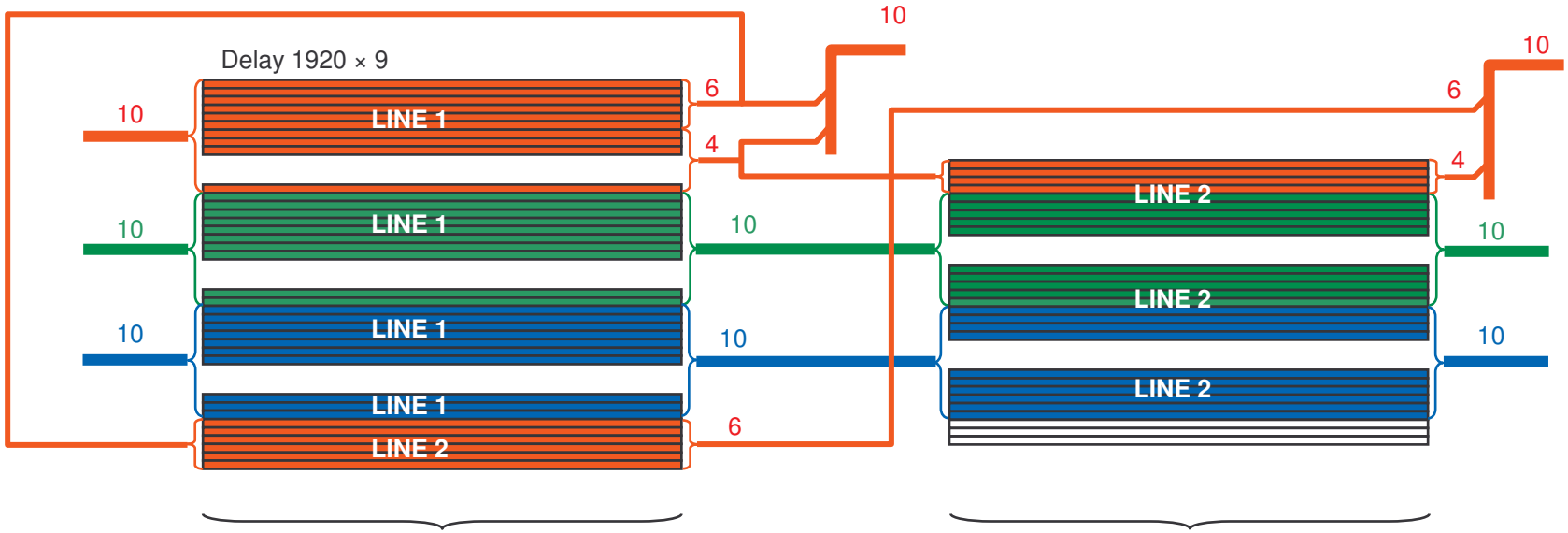
The following chart indicates the maximum number of line-bits that can be implemented in each device in the Spartan-3E family using the various macros provided. Divide the number in the table by the number of bits used to define your pixels (bits × colours) to calculate the maximum number of full line stores which can be implemented on a given device. For example, The XC500E is able to support a maximum of 288 line-bits of length 1280. If pixels are defined by 10-bits red, 10-bits green and 10-bits blue then we must divide the line-bits figure by 30 and this tells us that this device would support a maximum of 9 full RGB line stores.

Device	XC3S100E	XC3S250E	XC3S500E	XC3S1200E	XC3S1600E	
Number of BRAMs	4	12	20	28	36	**Macros supplied**
Maximum 768 Line-Bits	96	288	480	672	864	**delay_768x24 (1 BRAM)**
Maximum 1024 Line-Bits	72	216	368	504	648	**delay_1024x18 (1BRAM)**
Maximum 1280 Line-Bits	52	170	288	399	517	**delay_1280x13 (1 BRAM)** **delay_1280x72 (5 BRAMs)**
Maximum 1536 Line-Bits	48	144	240	336	432	**delay_1536x12 (1BRAM)**
Maximum 1920 Line-Bits	36	114	192	267	345	**delay_1920x9 (1 BRAM)** **delay_1920x48 (5 BRAMs)**

Example - Packing Line Stores

In this example we see how 2 line stores of 1920 pixels for a 3×3 pixel image processing algorithm can be implemented using 7 Block Memories (BRAMs). In this case each pixel is defined by 10-bits red, 10-bits green and 10-bits blue.

As described later, a single BRAM results provides a delay of 1920 stages but only for 9-bit data (**delay_1920x9.vhd**). However, since all colours of a pixel and all lines must advance at the same time (common pixel clock and clock/pixel enable), there is no need to keep colours or lines artificially separated allowing anything to be packed together in any order. This diagram shows one possible configuration using the 7 macros (BRAMs).



Line 1 is formed by packing the three 10-bit colour values into each BRAM rather than keeping each colour separate.

This means that the although 4 BRAMs have been used, the forth is still able to support another 6-bit bus which is then used to implement part of the second line.

Line 2 is formed using less BRAMs because it can steal 6 line-bits from the BRAMs primarily used to implement the first line. It in turn has 3 line-bits spare to help implement another line.

It is easy to see that not taking the opportunity to pack lines together would result in 8 BRAMs instead of the 7 shown. Failing to pack different colours together would yield very poor results. It may be nice to keep things separate when designing, but the costs can be significant.

Delay Length Sanity Check!

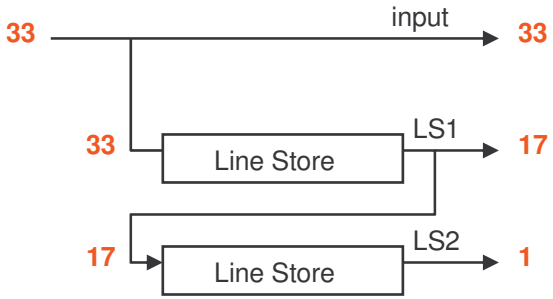
It is all too east to become confused about what is exactly the correct the length for a line store such that all the correct pixel data is available at the same time (typically the same clock cycle). It is therefore a good idea to consider a ridiculously small example in some detail. You may wish to skip over this description now, but don't be surprised if at some point in the future you need to come back to it once the actual implementation of lines stores is discussed.

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64
145	146	147	148	149	150	151	152	153	154	155	156	157	158	159	160

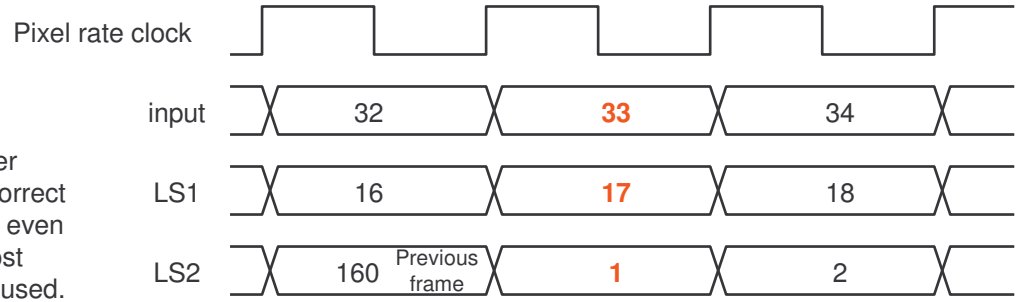
This example shows a very small display which is just 16 pixels wide and 10 pixels tall. We can number each pixel as shown.

The image is scanned left to right and top to bottom so the pixels will arrive in the order in which they are numbered here.

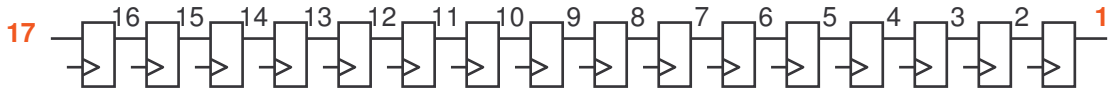
During fly-back (line and frame) the line stores must be disabled if the pixel clock is free running.



The object of line stores is to enable a vertical column of pixels to be observed simultaneously. So as highlighted, when pixel 33 is being received by the display at the start of the 3rd line, it should be possible to view pixel 17 and pixel 1. Likewise, as pixel 57 is received on the 4th line, then pixels 41 and 25 should be presented by the line stores. This is obviously delay of 16 pixels per line, but the key is to ensure that absolutely the correct data is visible as shown in the timing diagram below.



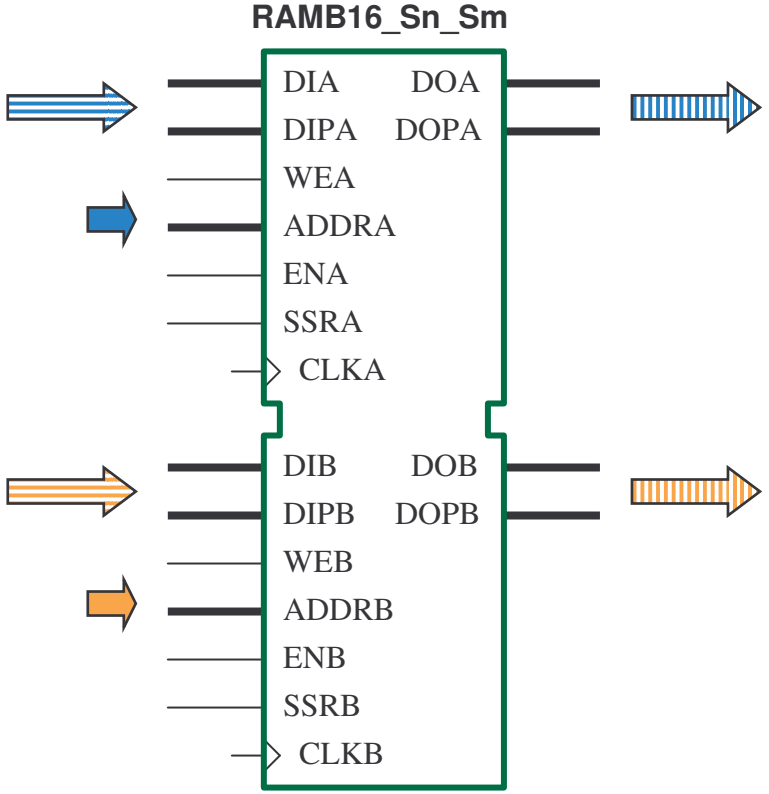
In theory, the line store can be implemented using a shift register equal to the length of the line. This automatically presents the correct pixel at the output as the new pixel is being applied. In practice, even the highly efficient SRL16E mode of the Spartan slices is not cost effective and techniques using Block Memory (BRAM) must be used.



READ FIRST Mode = Quad Port Memory

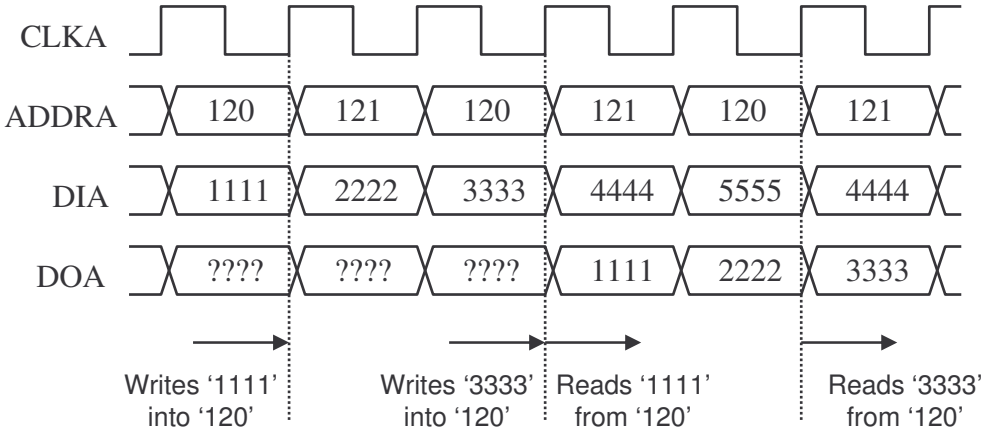
Each Block Memory (BRAM) is dual port but actually has 4 data ports; 2 data ports are always associated with data being written into the memory and the other 2 ports are always associated with reading data out of the memory. This is a significant difference to external memory devices in which a data port is typically bidirectional and consequently shared between writing and reading operations.

Since each Spartan BRAM is dual port, it is only natural that it should allow any 2 memory locations to be written simultaneously or any 2 memory locations to be read simultaneously. It is also natural that dual port memory should allow one port to be used to write information whilst the other port is being used to read information (e.g. when implementing a FIFO). However, in addition to these obvious modes, each BRAM has a special 'READ_FIRST' mode which enables both the write and read ports associated with each memory port to be used simultaneously. This means that each dual port BRAM can actually allow 2 values to be written and 2 values to be read every clock cycle. Although there are limitations to the possible address combinations, each BRAM can be considered a quad port memory in certain applications and video line stores are one such application that can exploit this pseudo quad port characteristic.



READ_FIRST mode allows the data stored at a given address to be read out on the DO/DOP lines at the same time that a new value applied to the DI/DIP lines is being stored at that same address.

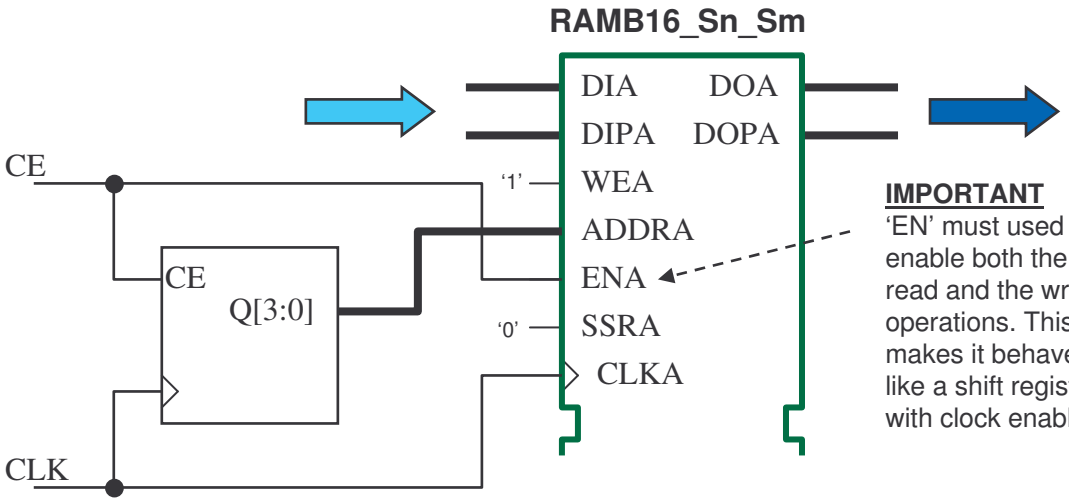
In the example below, the write enable (WEn) and enable (ENn) are always active (WEA=ENA=1) such that data is being written and read every clock cycle. The example emphasizes that the value '1111' previously stored at address '120' is read out at the same time that the new value '3333' is written.



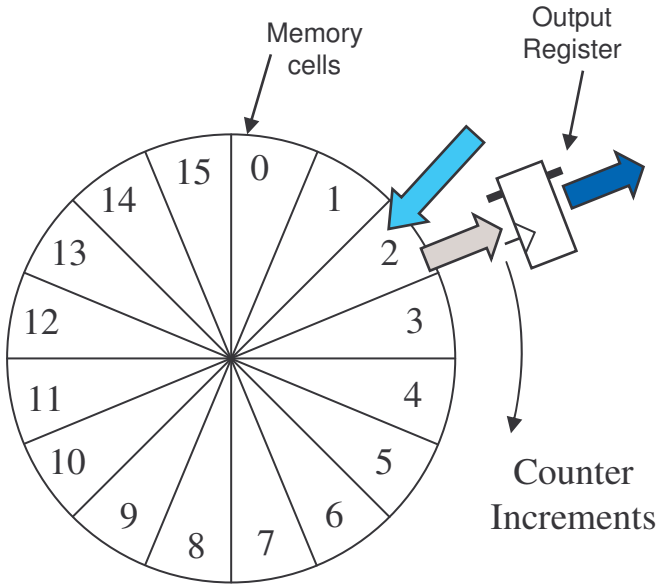
Cyclic Buffers For Delay Lines

The READ_FIRST mode allows each port of the BRAM to implement a fixed length delay using a cyclic buffer technique. A simple binary counter is all that is required to set the length of the delay. On each rising clock edge (when the write enable and enable are both High), three things will happen:-

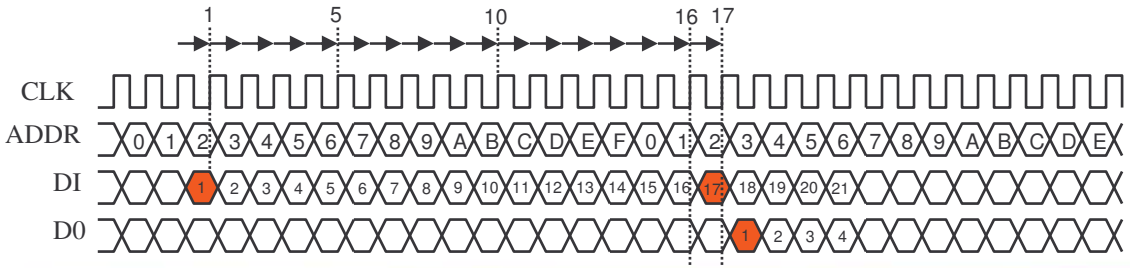
1) The value currently stored at the address defined by the counter will be transferred to the DO/DOP output lines where it will remain.
2) The new value provided at the DI/DIP input lines will be stored at the same address (over writing the previous value).
3) The counter will advance to the next location ready for the next clock cycle.



IMPORTANT
'EN' must used to enable both the read and the write operations. This makes it behave like a shift register with clock enable.



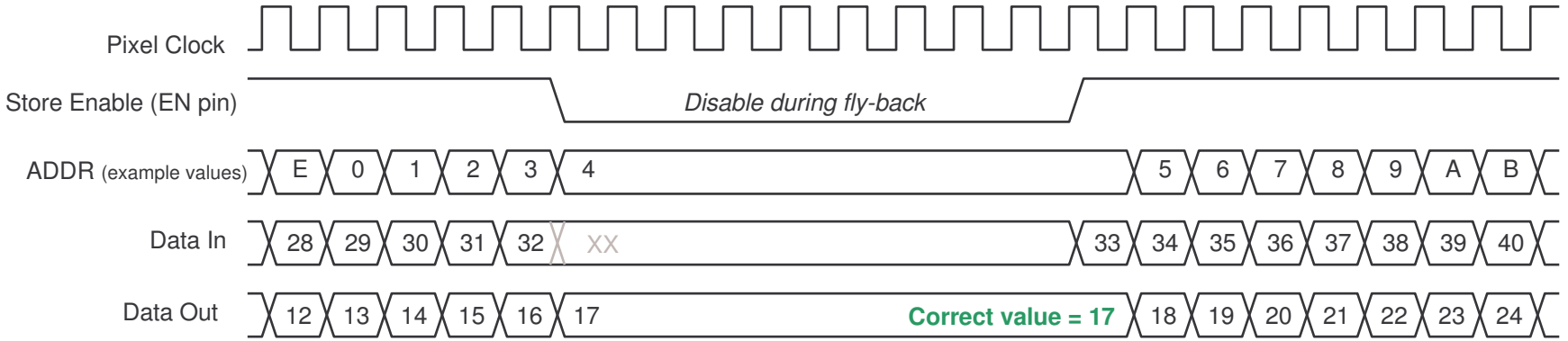
In this example a 4-bit counter has 16 states (0 to F hex) corresponding to 16 memory locations. This is where it is tempting to think that this structure is providing the functionality of a 16 pixel line store but closer inspection of the timing diagram below shows that this is not the case. Compare this diagram with the situation presented on page 20 where each pixel is numbered in ascending order. Do not become distracted by the actual address (ADDR) values as that is just the counter cycling round the 16 states and memory locations.



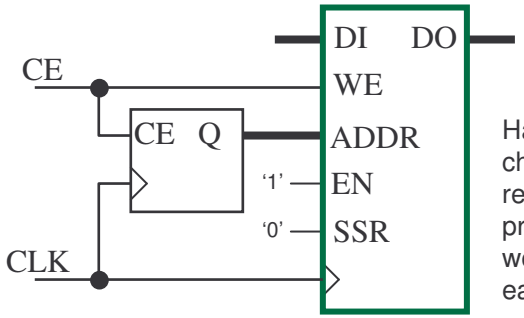
What we see is that the structure is actually the equivalent of a 17 stage shift register. This is because the READ_FIRST mode is the equivalent of having a synchronous register on the output of the memory which provides one more delay. Therefore **in order to have the equivalent of a 16 stage shift register, the cyclic counter must have only 15 states** (line length – 1). Important: Counting 0 to 14 (E hex) is 15 states (do not confuse terminal count with the number of counter states).

Why using ‘EN’ is Vital for Delay Lines

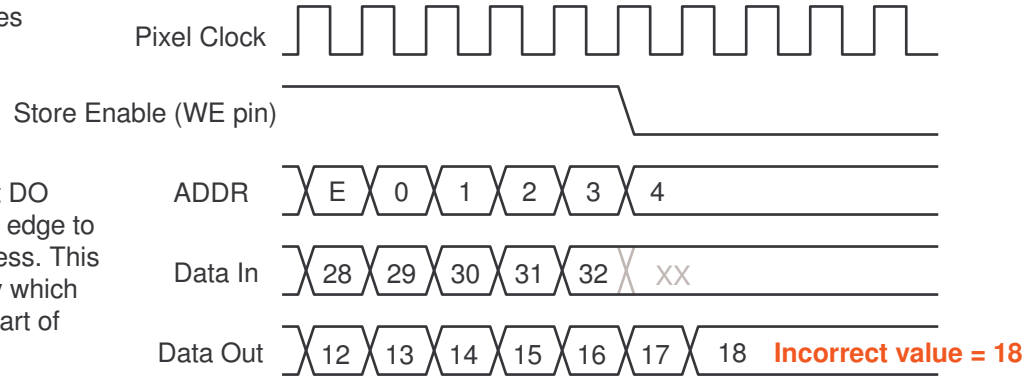
Each port of the BRAM has a write enable pin (WE) and an enable pin (EN). These are subtlety different and their correct use is vital when implementing delay lines. The ‘EN’ pin is a global enable for that port. If this signal is Low (‘0’) then the port is totally disabled. Nothing can be written and nothing can be read. The value at the DO and DOP output will remain static regardless of all other inputs. The ‘WE’ pin is the write enable. Providing the ‘EN’ is active High (‘1’) then the value presented on the DI and DIP pins will be stored at the location defined by the address applied to the ‘ADDR’ pins. When implementing a delay line suitable for a line store, it is vital that the ‘EN’ pin is used as the enable control and that the ‘WE’ is permanently active High. The reason for this only becomes clear when considering what happens when the delay is disabled as would be the case during use when display fly-back is being performed and the delay is providing a line store. Let’s return to the idea of a 16-pixel line store and look at what should happen during line fly-back…



Notice how the write of the last pixel of the second line (pixel 32) results in the output from the BRAM changing to the first pixel of the second line (pixel 17). This pixel is then is then available when the first pixel of the third line is presented (pixel 33). This is consistent with the operation of a shift register with clock enable or with the cyclic buffer implemented using BRAM provided the ‘EN’ pin is used. In contrast, look what happens if the ‘WE’ pin is used as the enable. Everything is working well whilst the enable is High, but as soon as the enable is deactivated the new address presented by the counter causes a further read.



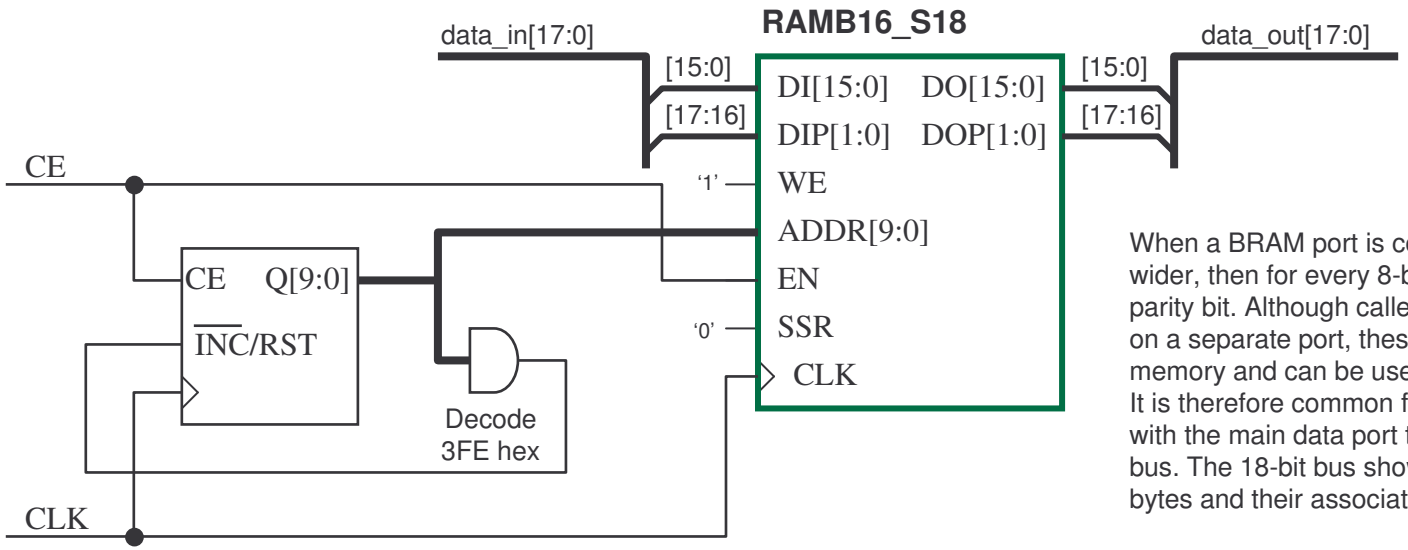
Having EN=‘1’ means that DO changes on the next clock edge to reflect the change of address. This presents pixel 18 too early which would give a fault at the start of each line.



1024 Pixel Line Store (18-bit)

delay_1024x18.vhd

The 1024 pixel line store is a direct implementation of a cyclic buffer and results in almost perfect system efficiency providing that you fully use the available bit width by packing pixels together to maximise the use of the 18-bits which each BRAM implements. In this case there is no reason to use the dual port capability of the BRAM and so the single port memory of aspect ratio 1024×18 with READ_FIRST mode is used.



When a BRAM port is configured to be 8-bits or wider, then for every 8-bits there is an additional parity bit. Although called parity bits and provided on a separate port, these bits are just ordinary memory and can be used to store more pixel data. It is therefore common for these bits to be merged with the main data port to provide a single wider bus. The 18-bit bus shown here is formed of two bytes and their associated two parity bits.

The 10-bit counter will automatically initialise to zero (no global reset required when using Spartan devices) and then increment for each clock edge that the enable is High. When the counter reaches 1022 (3FE hex) it will force the counter to roll back to zero on the next qualified clock edge. This gives the counter 1023 states rather than its natural 1024 states and means that one memory location (address 3FF hex) will never be used in this design.

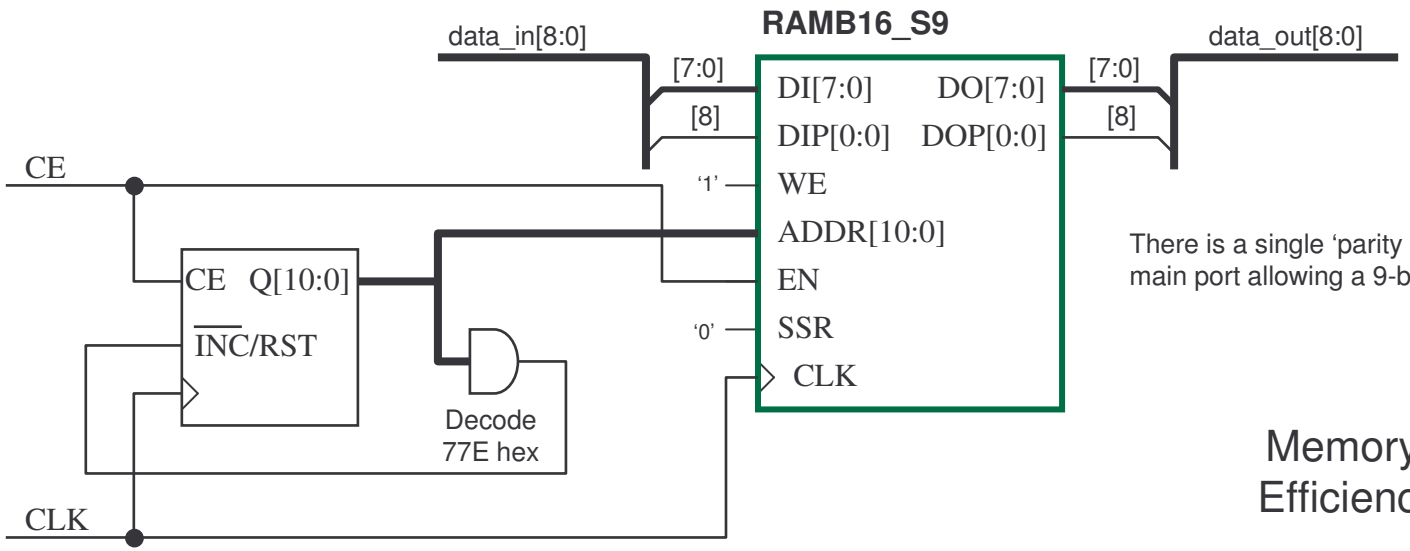
Memory Efficiency = 99.9%

The 10-bit address counter requires only 5 slices to implement plus a little more to decode the terminal count value. In most designs this number of slices is negligible and having a separate address counter for each BRAM line store will enable simple layout and design flow. If however slices are at a premium in your design, the counter can be shared between multiple BRAMs effectively forming a single line store of increased bit width. In this test design some address counters are routed to 5 BRAMs and yet they could overate close to 200MHz clock rate indicating that separate counters are not required purely to meet performance even at HDTV pixel rates.

1920 Pixel Line Store (9-bit)

delay_1920x9.vhd

The 1920 pixel line store can also be a very direct implementation using single port memory single port memory of aspect ration 2048×9 with READ_FIRST mode.



There is a single 'parity bit' to go with the 8-bit main port allowing a 9-bit bus to be supported.

Memory Efficiency = 93.7%

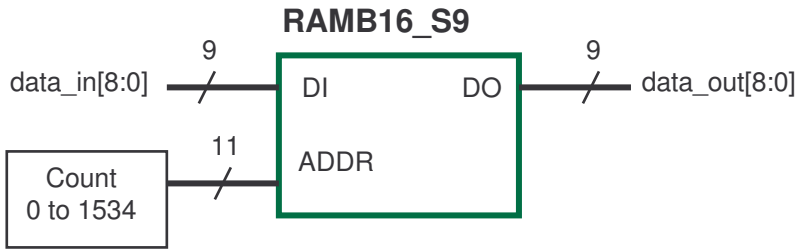
The 11-bit counter will automatically initialise to zero (no global reset required when using Spartan devices) and then increment for each clock edge that the enable is High. When the counter reaches 1918 (77E hex) it will force the counter to roll back to zero on the next qualified clock edge. This gives the counter 1919 states and means that 129 memory locations (address 77F to 7FF hex) are not used in this design.

The inefficiency of 6.3% represents the 1,161 bits of memory that are unused in this case. In theory, the maximum number of bits for a line length of 1920 that can be supported by a single BRAM is 9.605 (18432/(1920-1)) indicating that this implementation is providing the maximum number of complete line-bits that are possible which is probably acceptable in most cases. Later we will see that this wasted space can be recovered.

1536 Pixel Line Stores

The 1536 pixel line store presents an issue since 1536 is not even close to being a power of 2. In fact it is exactly mid way between 1024 and 2048 which can easily lead to a very inefficient implementation of a line store.

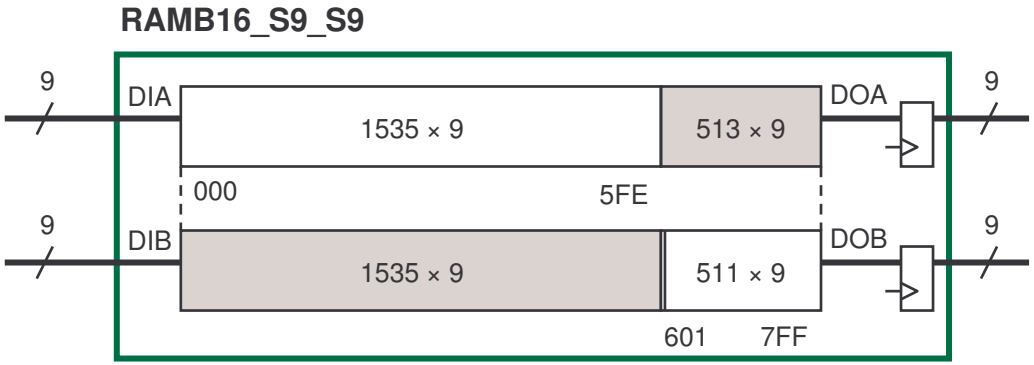
An obvious attempt is to employ the 2048×9 aspect ration of BRAM since this is the depth aspect greater than the 1536 delay required. The simplified diagram below indicates what this simple solution can offer.



1536 Line Store (9-bit)

Efficiency = 75% (unacceptable!)

The data width is only able to support 9 line-bits resulting in poor use of the memory. The 11-bit counter has 1535 states meaning that 513 memory locations are unused (5FF to 7FF hex). That is a total of 4,617 bits that are not used in this design. Clearly this is an unacceptable waste of memory since it would appear adequate to implement a further 3 line-bits. Fortunately there is a solution based on the observation that 1536 is a multiple of 256 even if it is not a power of 2 in its own right and that so far only used one of the two ports provided on the BRAM because we have used the READ_FIRST mode.



This diagram shows that if the ‘A’ port of a dual port memory is used to implement a 1536 line store of 9-bits it leaves 513 locations of 9-bits unused as was the case with the single port implementation discussed above. However, it now becomes clear that the ‘B’ port can be used to access that unused space. Providing the address range on the ‘B’ port is kept in the range 5FF to 7FF it will have no effect on the ‘A’ port operation and the dual port memory has effectively been divided into two single port memories albeit of different sizes.

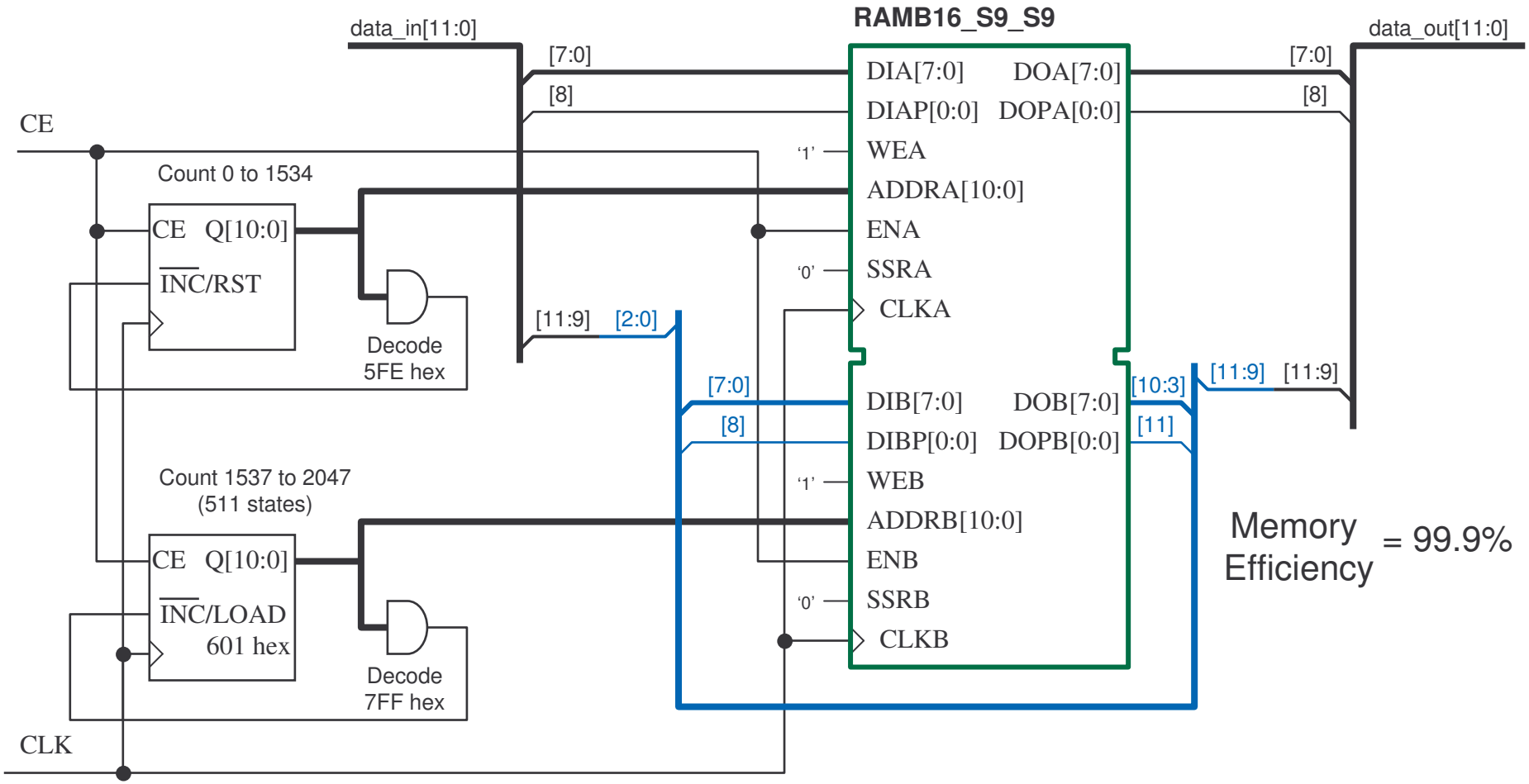
It is now possible for the ‘B’ port to implement a delay of 512stages of 9-bits. Cascading 3 delays of 512 stages then yields the desired 1536 stage delay. Since 9-bits conveniently divides by 3, the ‘B’ port is able to provide an additional 3 line-bits and utilise all but 2 locations (18 bits) of the memory.

1536 Line Store (12-bit) Efficiency = 99.9% (acceptable!)

1536 Pixel Line Store (12-bit)

delay_1536x12.vhd

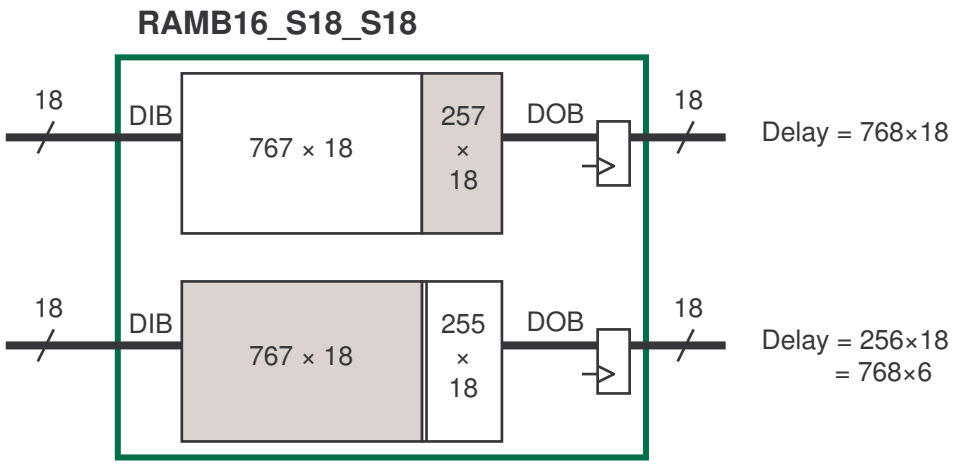
Dual port combined with READ_FIRST mode is enabling virtually all the memory to be used to provide a 1536 line store of 12-bits.



'B' port is used to form a 512 stage delay which is used 3 times by 3 bits. The diagram to the right shows the equivalence of the signals being used.

768 Pixel Line Stores

The 768 pixel line store can exploit the same dual port technique. This time using the fact that 768 is 3×256 to pack the remaining space via the second port.



768 Line Store (24-bit)

Memory Efficiency = 99.9%

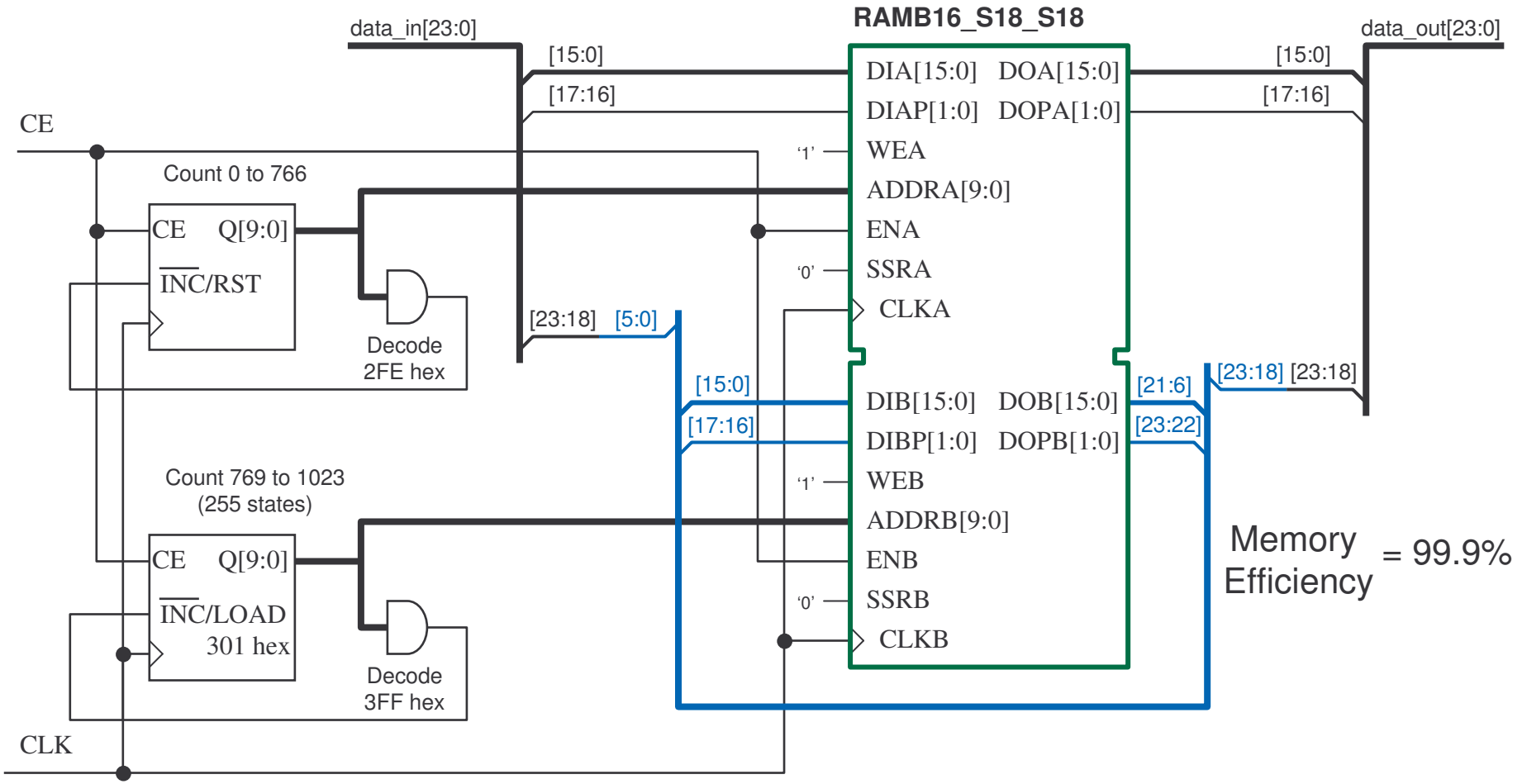
The BRAM is used in 1024×18 aspect ratio since this is deep enough to support a direct delay of 768 stages for 18-bits. This then leaves enough memory to implement 256 stage delays for another 18-bits. Since 768 is 3×256, then each additional bit just has to pass through the 'B' port delay 3 times. Once again we are faced with a convenient division of the bits which allow the 18-bit port to provide exactly 6 additional bits of line delay.

24-bits is an ideal system fit for 8-bit RGB pixel data.
One complete line store per BRAM.

768 Pixel Line Store (24-bit)

delay_768x24.vhd

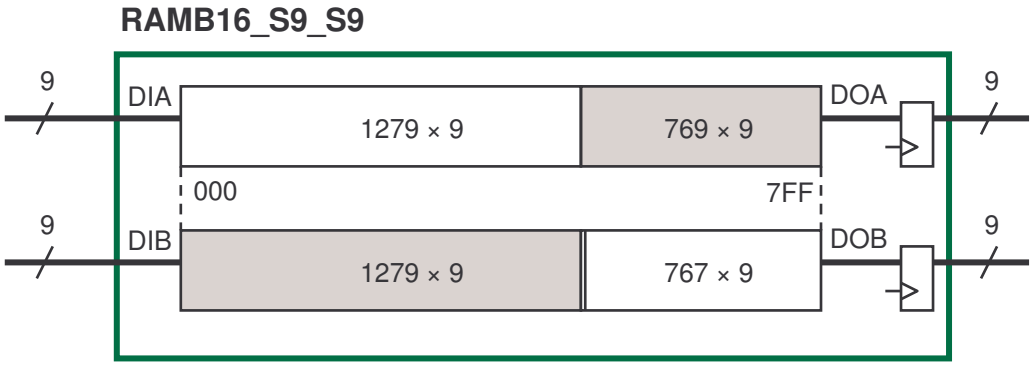
Dual port combined with READ_FIRST mode is enabling virtually all the memory to be used to provide a 768 line store of 24-bits.



'B' port is used to form a 256 stage delay which is used 3 times by 6 bits. The diagram to the right shows the equivalence of the signals being used.

1280 Pixel Line Stores

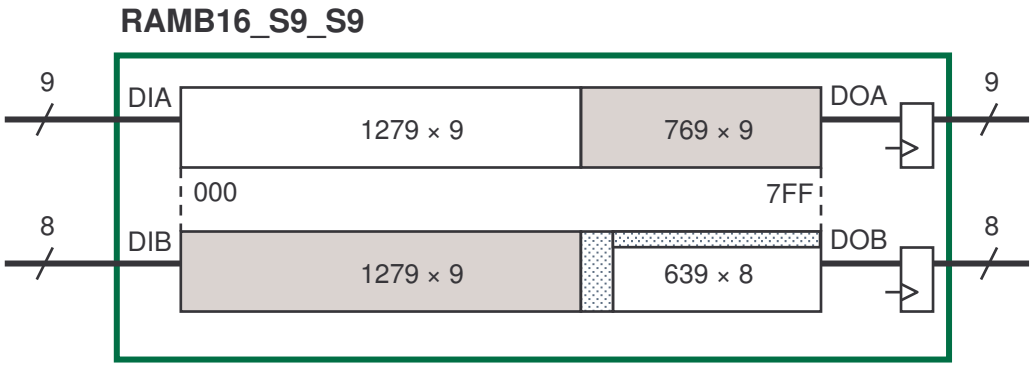
The 1280 pixel line store is even more of a challenge. To only use a BRAM to form 9-bits of delay of this length would mean a memory efficiency of only 62.5% which is something we just can not accept when using Spartan device in high volume applications. So once again the solution is to exploit the dual port of the memory to unlock that unused potential.



'A' port directly implements a delay of 1280 stages of 9 bits using 1279 memory locations and the synchronous READ_FIRST mode.

Although the 'B' port unlocks enough memory to form a delay of 768 stages of 9 bits, this length of delay is not immediately convenient for creating further delays of 1280 stages since it is 5 × 256.

For greatest memory efficiency, the 'B' port can be used to form delay of 640 stages which is half of the required 1280 stages. This then enables additional bits of data to be supported resulting in a total of 13 bits with a memory efficiency of 90.1%.



'A' port directly implements a delay of 1280 stages of 9 bits using 1279 memory locations and the synchronous READ_FIRST mode.

'B' port provides delay of 640 stages of 8 bits which can be used to form a delay of 1280 × 4.

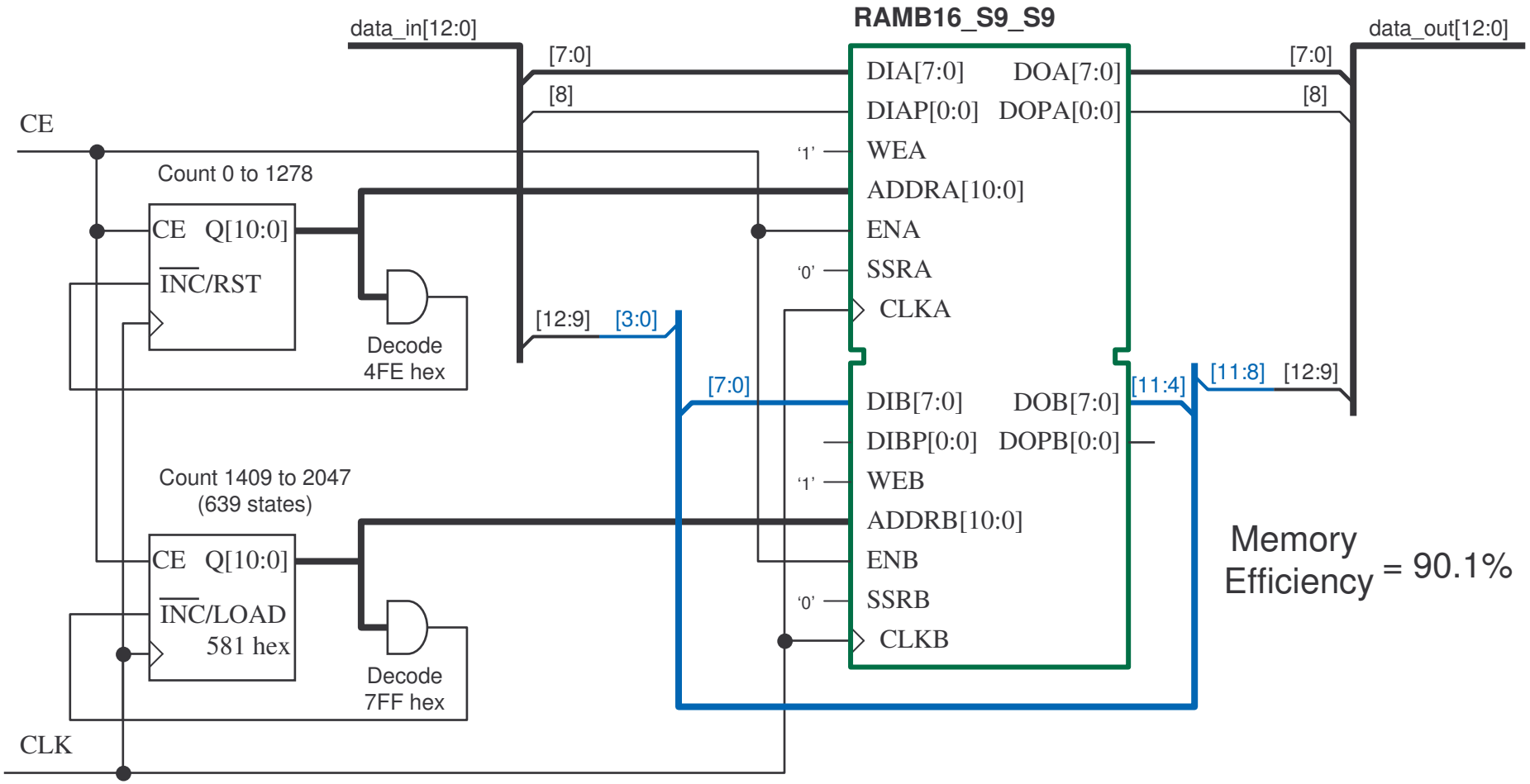
130 memory locations are completely unused and the 'parity bit' will be unused on the 'B' port.

Note: For every 2 instances of this structure, the 640 × 1 delay associated with the 'parity bit' of the 'B' port in each instance could be used to form an additional line. However we will soon see that multiple BRAMs working together can achieve a higher system efficiency.

1280 Pixel Line Store (13-bit)

delay_1280x13.vhd

Dual port combined with READ_FIRST mode is enabling 90% of the memory to be used to provide a 1280 line store of 13-bits.

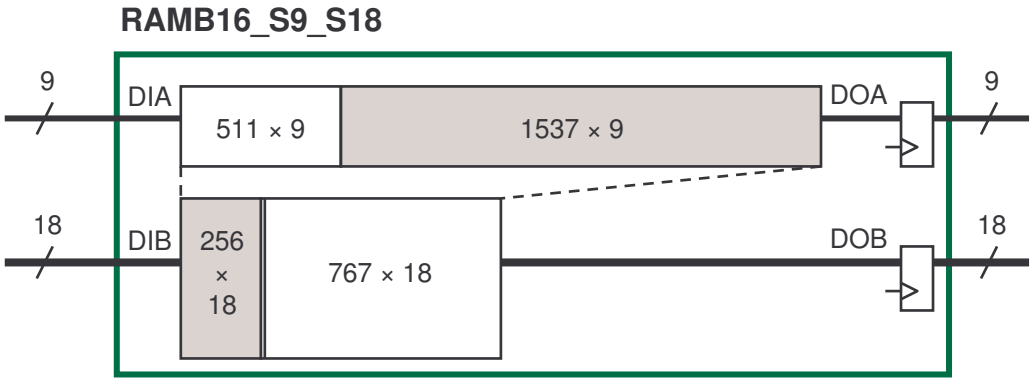


'B' port is used to form a 640 stage delay which is used 2 times by 4 bits. The diagram to the right shows the equivalence of the signals being used.

Improving 1280 Pixel Line Store Density

Even though use of the second port has enabled a 13-bit line store to be implemented in a single BRAM, the efficiency is still only 90.1%. The implementation is wasting 1809 bits of memory which is clearly enough to implement at least one more 1280x1 line delay. The solutions is to return to the observation that 1280 is equivalent to 5 × 256 and implement more line stores using several BRAMs arranged to support various multiples of 256 delay.

To further assist, we will exploit yet another feature of the BRAM; that being the ability for each port of a BRAM to be configured as a different aspect ratio. When using a mixed aspect ratio, the same memory is still accessed from each port but it is presented a different way on each port. In this case we will be using the RAMB16_S9_S18 primitive. The data width of the 'B' port is twice that of the 'A' port (18-bits verses 9-bits) but the 'B' port only has half the address range of the 'A' port (1024 location verses 2048). In other words, each 18-bit word located at a single address of the 'B' port appears as two 9-bit words located at two adjacent addresses of the 'A' port.



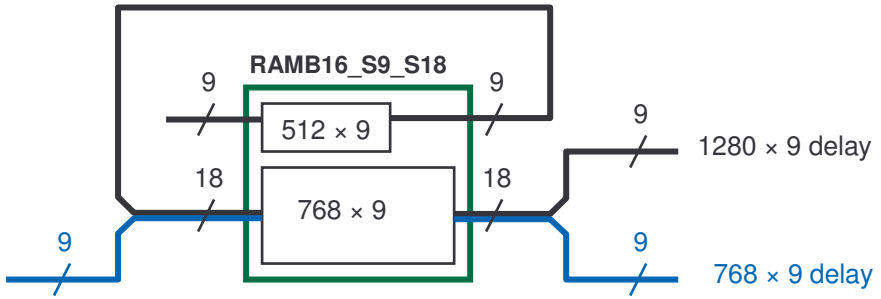
If the 'A' port is used to implement 9-bit delays of 512 stages, then there is still just over three quarters of the memory unused (1537×9). When this memory is viewed from the 'B' port it still appears that three quarters of the memory is unused, but it is now presented as 768×18 which is adequate to implement 18-bit delays of 768.

So this BRAM is 99.9% efficient when implementing….

512 × 9 delay where 512 = 2 × 256

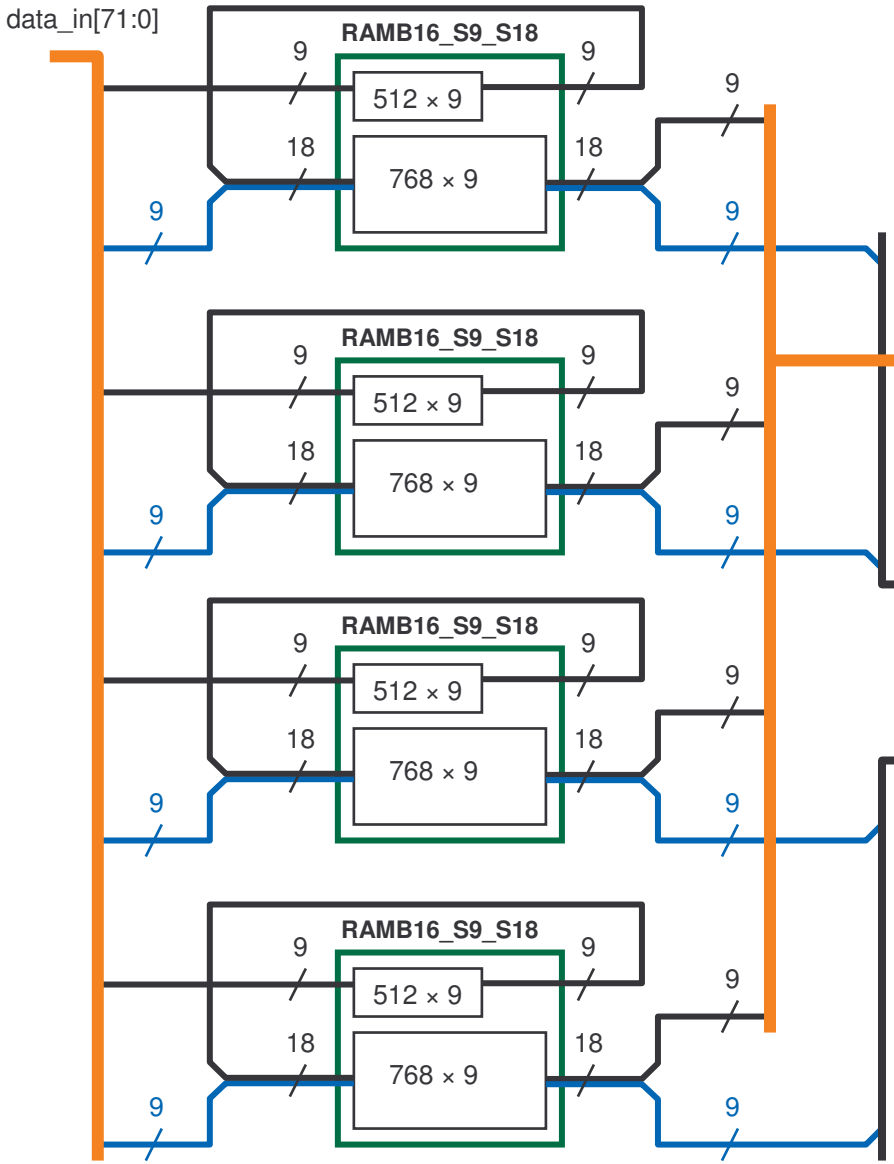
768 ×18 delay where 768 = 3 × 256

Initially this combination of delays does not appear to be very helpful. However, cascading the 512 stage delay with the 768 stage delay does result in the desired 1280 stage delay but only for 9-bits. On its own, this is only 62.4% efficient because the remaining 768 stage delay of 9-pixels is unused. All that is required to turn these unused delays into full 1280 stage delays are more 512 stage delays which are a very convenient power of two which can be formed in another BRAM.



1280 Pixel Line Store (72-bit)

delay_1280x72.vhd

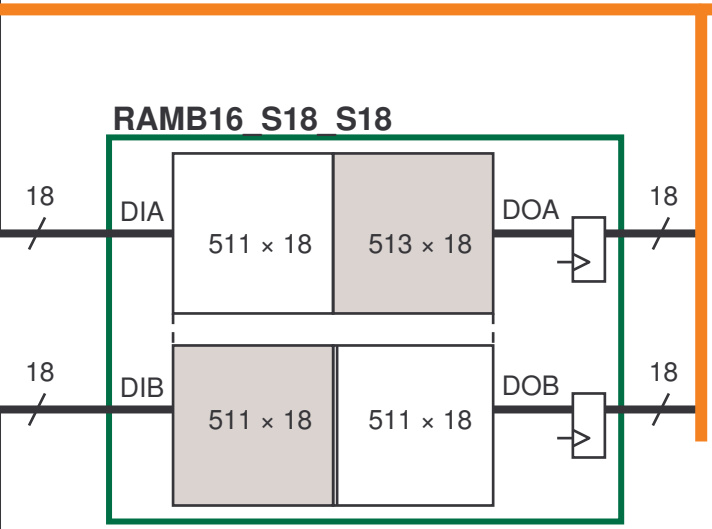


Using the mixed 512×9 and 768×18 combination of BRAM four times (the address counters can be shared) results in a desired 1280×36 delay but also provides a 768×36 delay which is not useful and would be wasteful.

Fortunately a single BRAM can then be configured to provide the 512×36 delays required to supplement all the 768 stage delays.

Efficiency = 99.8%

Average of 14.4 line-bits/BRAM

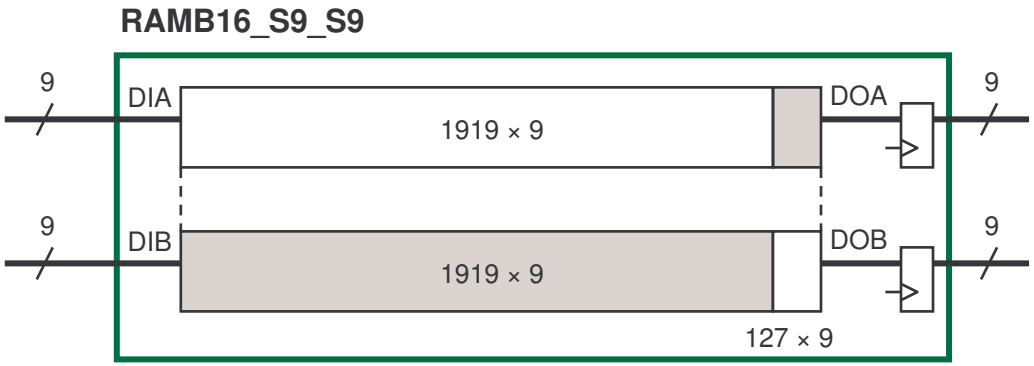


The same 9-bit address counter can be used for both ports since the delay implemented by each is the same. The actual address required by the BRAM is then 10-bits so the MSB should be forced Low on one port and High on the other to divide the memory into two halves. This counter can also be used to address the 'A' port in the first four BRAMs whilst forcing the remaining two MSBs to "00".

Hint – A single port BRAM with 36-bit aspect ratio is not used as this would prevent access to the dedicated multiplier located next to the BRAM.

1920 Pixel Line Store (48-bit)

delay_1920x48.vhd



Although the 1920 delay of 9 bit is already 93.7% efficient, it still leaves capacity to implement a 9-bit delay of 128 stages. This does not seem particularly useful until you remember that 1920 is equivalent to 15×128. Therefore combining this otherwise wasted delay from several BRAMs can provide enough to form a few more complete line-bits of 1920 stages.

5 BRAMs provides 48-bits and is the best fit

5 × ‘A’ ports each providing 9-bits of full 1920 stage delay = 45 lines

5 × ‘B’ ports each providing 9-bits of 128 stage delay = 45 delays of 128 stages
= 3 lines of 1920 stages

In gaining the 3 additional bits the data width of 48-bits also becomes a more convenient fit for 12-bit pixel data.

$$\text{Memory Efficiency} = 99.9\%$$