HUNT ENGINEERING
Chestnut Court, Burton Row,
Brent Knoll, Somerset, TA9 4BP, UK
Tel: (+44) (0)1278 760188,
Fax: (+44) (0)1278 760199,
Email: sales@hunteng.demon.co.uk
http://www.hunteng.co.uk
http://www.hunt-dsp.com

Implementing an FFT with the HERON-FPGA Family

Rev 1.1 T.Hollis 11-05-05

Introduction

The HERON-FPGA family is ideal for many of the building blocks of digital communications. Providing large easily-programmed gate arrays, often combined with interface elements like ADC or DACs, they can be used to implement many system components.

Many DSP systems rely on the transformation of discrete data between the time and frequency domains, using Discrete Fourier Transforms (DFT). The Fast Fourier Transform (FFT) algorithm is an efficient technique, both in terms of time and hardware requirement, to implement a Discrete Fourier Transform in an FPGA.

This note goes through the design of an FFT for implementation in a HUNT ENGINEERING HERONIO2V2 using the high speed A/D and D/A converters on the module. The design is written in VHDL using the 'Transient Analysis(Ex2)' example as its start point, this includes all the HUNT ENGINEERING Hardware Interface Layer to interface to the Heron Interface, the Hunt Serial Bus(HSB) and both A/D and D/A converters on the IO2V2.

The Example uses the Xilinx 'CoreGen' to generate the building blocks which will make up the FFT so allowing the example to demonstrate designing a FFT rather than writing VHDL code.

What is a FFT

One of the main tools in Digital Signal Processing is the Digital Fourier Transform, which allows discrete set of voltage versus time samples to be transformed into a discrete set of magnitude versus frequency and phase versus frequency samples. There is also an inverse transform that allows the frequency domain data to be transformed back into the time domain. The two domains provide complementary information about the same data.
The Fast Fourier Transform is an algorithm for computation of the Digital Fourier Transform that is efficient, both in terms of the hardware requirement and speed in which a transform can be accomplished. In this example the FFT is performed by a Xilinx CoreGen block that employs a Cooley-Tukey radix-4 decimation in frequency (DIF) FFT to compute the DFT of a 1024 point complex sequence. In general, this algorithm requires the calculation of columns or ranks of radix-4 butterflies, sometimes referred to as dragonflies. Each processing rank consists of N/4 dragonflies. For n = 1024 there are 5 dragonfly ranks, with each rank comprising 256 dragonflies.
The DFT is usually considered as a technique that allows meaningful inspection of a particular frequency component as relative to observing the time waveform. The transform can also have other advantages in the processing of data. For example if it is required to convolve two discrete time sequences, the equivalent operation in the frequency plane is multiplication. In some applications the transform of the data and then multiplication could be faster and require less hardware than computing the convolution in the time domain.
The Fourier Transform relationship exists not just between the time and frequency domains, for example in sonar systems where there is an array of hydrophone elements in space, there is a Fourier Transform relationship between the elements and the angular acoustic beams that can be formed.

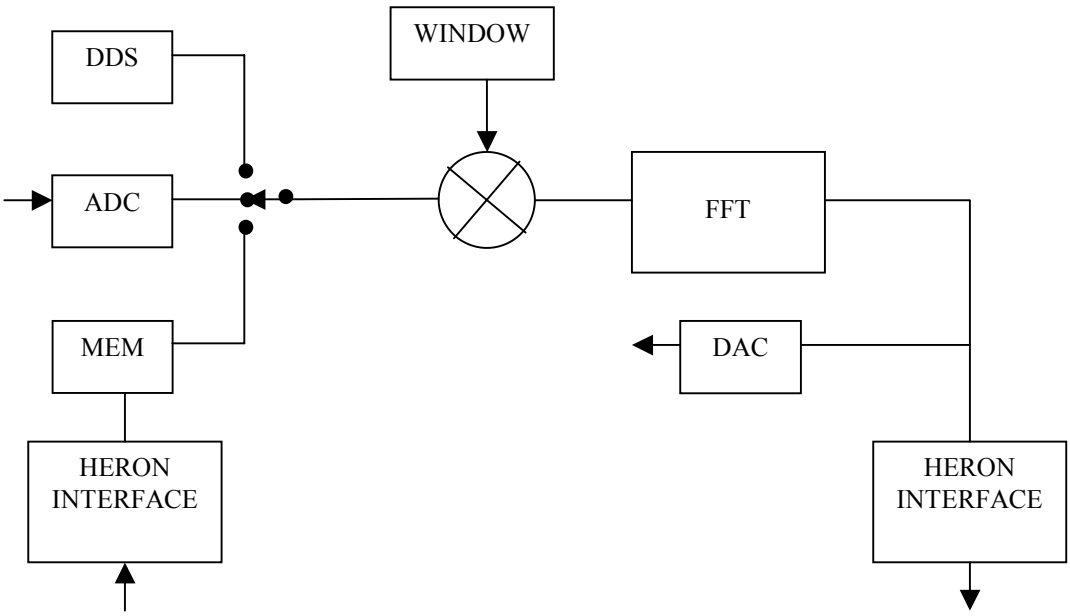


Figure 1 – FFT Example Block Diagram.

FFT Example

The block diagram in figure 1 shows the structure of the FFT example implemented in the FPGA on the HERONIO2V2. The FFT in the design is a 1024 point complex FFT/IFFT implemented using a Xilinx CoreGen block. The FFT is running with a 100MHz clock and is completing an FFT every 4096 clock cycles, or 40.96 micro Seconds.

To make the example flexible, and also to demonstrate some of the characteristics/pitfalls that can be encountered when using an FFT, the HUNT ENGINEERING Serial Bus (HSB) link has been used to control the configuration of the example using values written to registers in the FPGA. If the HSB is not available to you then the example can still be used as the registers have default values that connect the ADC-A output to the FFT block using a Kaiser-Bessel window. The output of the FFT is available at the DAC-A output. To synchronise the signal on the DAC output to the frames of data from the FFT there is a synchronisation pulse on the least significant pin (I/O0) of the digital I/O connector.

There are three possible signal sources for the FFT, the default source is ADC-A on the HERONIO2V2, the second source is a Direct Digital Synthesiser (DDS) which allows a full 16bit complex sine wave to be applied to the FFT. The frequency of the DDS output, which has a resolution of 1.5Hz, is determined by a value loaded over the HSB into registers in FPGA. The default value is 10.0MHz. The third source is a 1024x32 block of memory in the FPGA which can be written to over the Heron interface. This source is very flexible as any data can be written into the memory, this allows not only forward transforms but also if a suitable pattern for a frequency domain signal is loaded, an inverse transform into the time domain can be performed.

The DDS and the Memory sources allow full complex signals to be applied to the FFT, the ADC-A signal source sets the quadrature component of the input signal to zero.

The FFT block is a powerful tool but must be used with care if the output values are to represent what you want them to. The windowing of the input data to the FFT is an important example of this. In this example two data windows are available, a rectangular window where all the input data values have the same weighting and a Kaiser-Bessel window which shades out the values towards the ends of the data window. Which of these two window is selected is determined by a register value written over the HSB. The default value selects the Kaiser-Bessel window.

The FFT block is a 1024 point complex FFT/IFFT generated using the Xilinx Core Generator. For this example the Triple Memory Space (TMS) configuration has been used so that the FFT block is supplied with data 100% of the time and no potential computation cycles are wasted. There are two inputs/working memory banks on the input to the FFT block, which allows one block to be the FFT core's working memory while the other is loaded with a new block of data. On completion of the transform the results are available in a third block of memory, and the two input memory blocks swap rolls.

The complex output values from the FFT are linked to the Heron interface, and also indirectly to DAC-A on the HERONIO2V2. The output from the FFT block has both 'Phase' and 'Quadrature' components, the signal on DAC-A is the sum of the squares of these components.

The signal to the DAC-B output is the Phase component of the input signal to the FFT prior to the sample window being applied.

Using the Example FFT

The FFT project for the HERON-IO2V2 is supplied on the HUNT ENGINEERING CD, along with bit streams that can be loaded directly onto the HERON-IO2V2. The bit streams can be found under 'fpga\io2v2\fft_example' and the name of the bitstream reflects the FPGA part number. I.e. 2v1000fg456 is for the HERON-IO2V2 when fitted to a HEPC9 carrier board. If you are using the HEPC8 carrier board you will need to use the bit stream that ends with '_pc8', so for this example, the correct bit stream will be 2v1000fg465_pc8.

DSP example software can be found in the DSP directory and an example that can be run from the host in the host directory.

The FPGA project files can be found in the ISE directory, and the SRC directory for the VHDL source and window coefficient files. If you make changes to the project and re-build it you can change the functionality to be whatever you want.

Basic implementation

The simplest demonstration of the FFT example is to fit an IO2V2 to a mother board such as an HEPC8 or HEPC9 and downloading the applicable bitstream. The default configuration will give ADC-A as the FFT input, a Kaiser-Bessel window, and magnitude squared output of the FFT on DAC-A output.

Connect an RF signal generator's 50 Ohm output to the ADC-A input of the HERON-IO2V2, with a frequency of 10.0MHz and an output level of +5dBm, which is approximately 2Volts peak to peak into 200 Ohms. If the HERON-IO2V2 does not have a 200 Ohm input impedance then the signal level on the RF signal generator will need to be adjusted to give 2Volts peak to peak on the IO2V2 input. If the IO2V2 is DC coupled then the inputs will require a DC bias.

The signal from the signal generator is now digitised at a rate of 100MspSecond. The blocks of data to be transformed are multiplied by the Kaiser-Bessel window coefficients and loaded into one of the input/working memory banks of the FFT block. The complex output of the FFT block are squared and added together to give the magnitude squared of the frequency components. This is linked to the DAC-A output of the HERONIO2V2 and can be connected to an oscilloscope. To synchronise the blocks of output spectra from the DAC-A output with the oscilloscopes time base a synchronisation pulse is output on the 'Digital I/O' connector pin I/O0.

The DAC-B output from the HERONIO2V2 is linked to the Phase input signal to the FFT block prior to it being windowed.

The 1024 output values from the FFT will be clocked out of DAC-A at a rate of 100MspSecond, so will take (1024/100MHz) = 10.24micro Seconds. An oscilloscope set on a time base of 1microsecond/div and 10 division horizontal scale will display 10microSeconds. The maximum output level of the DAC's on the HERONIO2V2 is +/-1Volt, so as the output is always positive the maximum output signal level from the DAC of +1Volt. For a 10MHz, +5dBm signal the output level is about 350milliVolts.

The output level will depends on the window type and signal source. For this basic implementation the output level from FFT is on the low side for two reasons. Firstly because the signal from signal generator is real not complex the level is 6dB's down and further because a Kaiser-Bessel window is used the level is approximately a further 8dB's down. The output from the DAC is also the sum of squares. To compensate for this the default mode DAC output level has an extra gain of x8 implemented in the FPGA.

C6000 Example Based Software

HEART carrier like HEPC9

If you have a HEART based module carrier like HEPC9, you can run the DSP based example code with your HERON-IO2 and C6000 modules fitted to any slots. Then you need to configure the HEART connections between the modules using HeartConf, or alternatively use default routing jumpers to make the connection.

If you use HeartConf, then a network file that connects the two modules as follows

```
# For HUNT ENGINEERING's Device Driver API use:
# BD API        Board_type      Board_Id        Device_Id
#-------------------------------------------------------------------
# Using API
BD API HEP9A 0 0
#
# Nodes description
# ND  BD_nb  ND_NAME  ND_Type  CC-id HERON-ID  filename(s)
#-------------------------------------------------------------------
  c6   0      dspmodule  ROOT     (0)   00000001  mydspprog.out
  fpga 0      heronio2   normal         00000002  nofile

  ibc  0      ibc1        normal         0x06
  pcif 0      NodeC       normal         0x05

#-------------------------------------------------------------------
#       from:slot  fifo  to:slot   fifo   timeslots
#-------------------------------------------------------------------

heart      dspmodule  0      heronio2   0     1
heart      heronio2   0      dspmodule  0     1
```

could be used to make a connection from FIFO#0 of the DSP module to FIFO#0 of the HERON-IO2. Of course you need to modify the HERON-IDs to correctly show which slots your modules are fitted to.

If you prefer to use the default routing jumpers then you can simply fit the jumpers to 0, on both modules for both input and output FIFOs. In this case you are selecting to use timeslot 0 to connect between the FIFO#0 of each module to the other.

HEPC8 carrier

If you have an HEPC8, then you can also fit your HERON-IO2 and C6000 modules to any HERON slots. In this case the FIFO numbers that each module uses will depend on the module slots you have chosen. An example would be if the C6000 module is in slot1, and the HERON-IO2 is on slot3, then the DSP will use its FIFO#2, and the HERON-IO2 would use its FIFO#3.

DSP example : What it does

We supply a ‘C’ source file for a C6000 based HERON module. Before you can use it you need to make a new project that matches the C6000 module type that you have.

The program is called ‘fft_example.c’ and it is located in the ‘dsp’ sub-directory of this example. This example can be used both as a confidence check and a starting point for your development.

Once the DSP program is compiled and loaded onto the DSP it will be used to download values to the IO2V2 control registers via the HSB and then gather the FFT output data into the DSP memory where it can be displayed.

After you have loaded the correct bit stream into your HERON-IO2 module, the “DONE” LED should be switched off showing that the configuration was successful. Also The USER LED4 should flash about once per second, showing that the clocks are properly running in the FPGA. If the DONE LED is off, but the LED is not flashing it may be because the Delay Locked Loop used in the clock circuit of the FPGA needs to be reset. You

can do this using the utility under "programs → HUNT ENGINEERING → API board RESET" if you want, but such a reset should be made using the HUNT ENGINEERING Reset Plug in for Code Composer.

If you have a HEART based carrier (like an HEPC9) you will need to set the FIFO connections that will be used in your system. You will do this using the Heartconf program, which can be called from the HUNT ENGINEERING reset plug in for Code Composer Studio. This is probably the best way as it will re-configure your connections every time you reset the system. If you are not already comfortable doing this you should review the HEART movies and documentation again.

DSP example: setting it up

If you have an HEPC8 your FIFO connections will be determined by the position of the modules on your carrier board.

If you have an HEPC9 however the FIFO connections are determined by the settings that you made.

The example that we supply is a C file called fft_example.c. It needs to be changed to reflect your actual needs, and then built using Code Composer Studio.

The example is a HERON-API project that can be set up using the 'Create new HERON-API project' plug-in. To do this, choose "Tools→HUNT ENGINEERING→Create new HERON-API Project" from inside Code Composer Studio. This will guide you through setting up the project and as long as you choose the name "fft_example" for the project it will incorporate the **fft**_example.c source file.

This project will incorporate the correct HERON-API library for your module and Module carrier combination.

You must review some settings at the top of the source file, and change them to reflect your system setup.

The first line is

#define FIFONO 2 /* FIFO through which module communicates with FPGA */

This is the fifo that the C6000 uses to communicate with the HERON-IO2. For example if you have a DSP module in slot1 of an HEPC8, and the HERON-IO2 is in slot2, the setting needs to be 2. If you have an HEPC9 you must set whatever you have chosen, which is 0 in both of our HEPC9 configuration examples given above.

The next thing is to set the line

#define FPGA_TO_DSP 3

This is the fifo that the HERON-IO2 uses to communicate with the C6000. For example if you have a DSP module in slot1 of an HEPC8, and the HERON-IO2 is in slot2, the setting needs to be 3. If you have an HEPC9 you must set whatever you have chosen, which is 0 in both of our HEPC9 configuration examples given above.

Finally you need to change the line

#define FPGA_SLOT 2

to show which module slot the HERON-IO2 can be found in. This is used to address the HSB messages.

Then you can build the example using Code Composer Studio.

DSP Example: using it

To run the example, load the program onto the DSP using the File→Load command of Code Composer. At this stage it is advisable to open the HUNT ENGINEERING Reset Plug in, and set the option to "halt processors, reload them and then run to main". If you have a HEART based carrier and you will use HeartConf, select this also in the reset plug in. Then use this reset to reset the system, and reload it, which empties the FIFOs, and configures HEART if you need to.

But before you can run the example, you must have remembered to load the bit-stream onto the HERON-IO2. What bit-stream to choose is shown above. If you don't know how to load a bit-stream onto the HERON-IO2, please review example1 once more. Verify that after the loading process the "Done" LED goes off, and now the system has been reset (using the plug in) the USER LED4 should be flashing.

It doesn't really matter in what order the DSP and the HERON-IO2 is loaded. You may just as well first load the bit-stream and only then load the DSP. Finally, do a "Debug->Run" in Code Composer Studio to start the example.

The fft_example starts by configuring the FIFOs using the HSB. The fifo number that is programmed is set by the "#define FPGA_TO_DSP 3" near the top of the program. You should have changed this to reflect which FIFO you want to use.

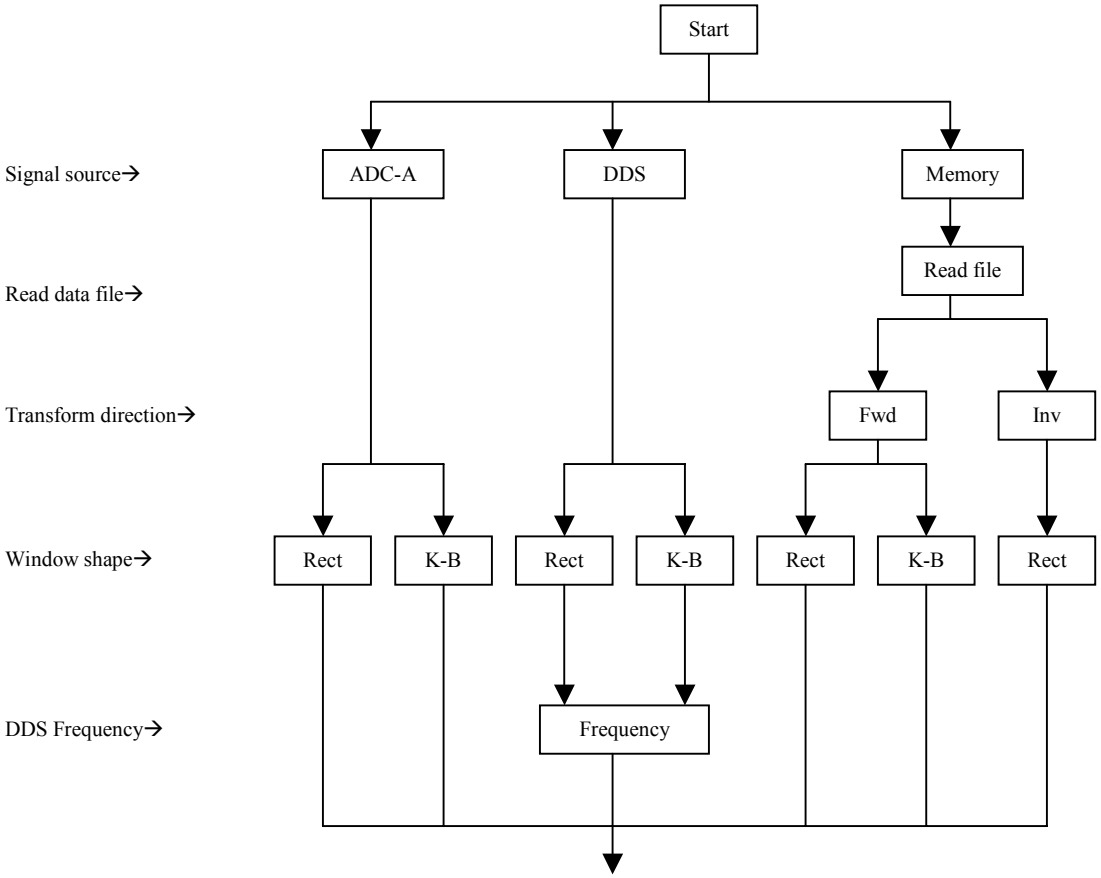


Figure 2 – Flow diagram

When you run the program there are set up options for the fft-example which are shown in the flow diagram, figure 2. The first choice is which signal source for the FFT and the options are between a signal generator connected to the ADC-A input of the HERONIO2V2, a Direct Digital Synthesiser (DDS) implemented in the Xilinx FPGA, and a block of memory within the Xilinx FPGA that can be written to over the HERON interface. If the Memory option is selected then the next option is the file that is loaded into this 1024x32 block of RAM, the least significant half of each word is the Phase component of the signal and the most significant half of each word is the Quadrature component. As the Memory option is the only source that allows complete control over the signal shape this is the only option that allows a choice of either forward or inverse Fourier Transforms. If the inverse transform is selected then only a rectangular window for the FFT data is available.
There are two window options available for the signal data in the fft-example, the standard rectangular window, and a Kaiser-Bessel window.
The last option, when the DDS source option is selected, is the required signal frequency, which can be resolved down to 1.5Hz .
Once the options have been selected and the relevant values downloaded over the HSB to the FPGA, the fft-example will grab 1024 word blocks of data from the output of the FFT over the HERON interface. Each block of data is synchronised to the output of the FFT, each word has the Phase component value in the least significant half word and the Quadrature component in the most significant half word.
This fft-example can now be used to demonstrate the characteristics/pitfalls of using an FFT, most of which apply however the FFT is implemented.

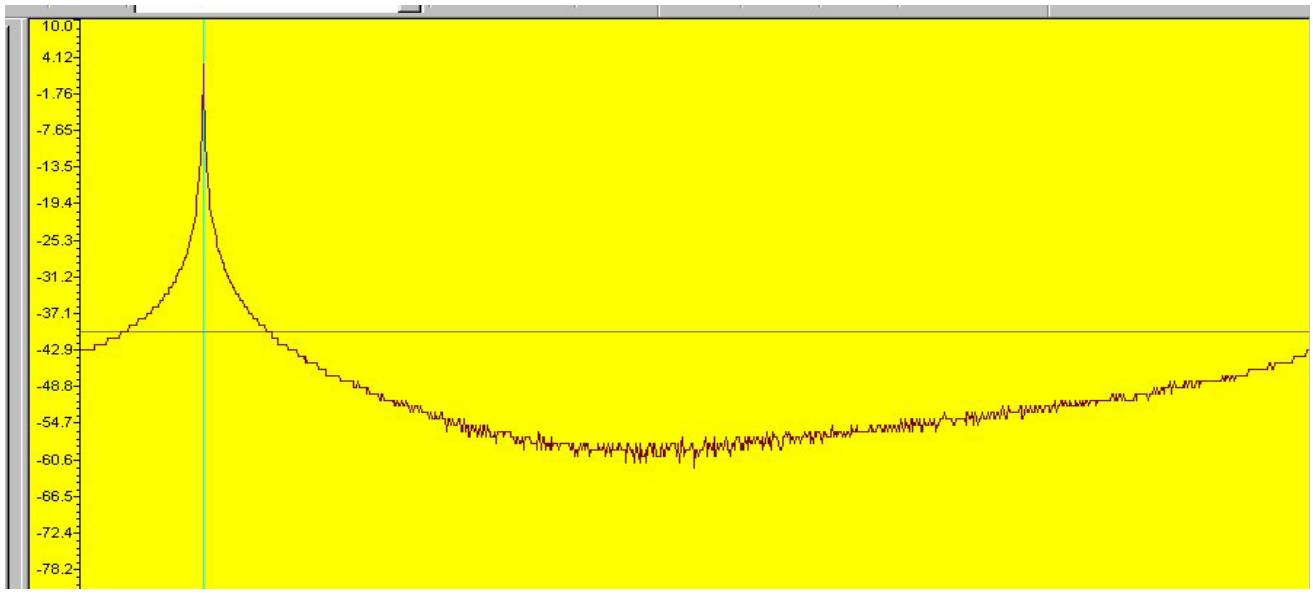


Figure 3 – DDS at 10MHz , Rectangular window

The Direct Digital Synthesiser (DDS) is a good source to start with as it is internal to the FPGA, has a high frequency resolution, and generates both Phase and Quadrature signals at maximum amplitude. Figure 3 shows the output of the FFT when a rectangular data window and a 10MHz DDS output frequency has been selected. The horizontal axis of the plot is the FFT output point number. There are 1024 samples on the input to the FFT which give 1024 complex points at the output of the FFT. The vertical axis is the magnitude of the FFT output and is on a logarithmic scale, it is in dB's. In the program 'fft_example.c' this is an integer array 'ampldata'.
. The first point to note is that the full 1024 points are displayed on the frequency axis so the range is from DC up to Fs (the sample frequency), but there is only one peak at output sample point 102. There is no negative frequency component at a position equivalent to (Fs-10MHz) = 90MHz because this is a full complex FFT and when the magnitude of the FFT output is calculated, by squaring and summing the Phase and Quadrature components for each point, the negative frequencies cancel out.
There are 1024 points at the output of the FFT, where the first is equivalent to DC and the last point one increment before Fs. With the sample frequency Fs of 100MHz the frequency increment is (100MHz/1024) = 97656.25Hz/point, which makes point 102 centred on 9.961MHz, while point 103 is centred on 10.058MHz. The 10MHz signal lies between these two points.
The peak of the FFT output is spread across more than one point, also the other points not in the peak of the plot do not go down to a noise floor but gently curve away from the peak. This is a function of the rectangular window shape. In effect each output point from the FFT is the amplitude spectral density obtained using a filter with a frequency response that is the Fourier transform of the window in the time domain, for a rectangular window this is (Sinx/x).
Figure 4 shows the Sinx/x responses for three adjacent FFT output points, the particular points to note are firstly the first side lobe in the point response is only 13 dB's down on the main lobe, and all further side lobe peaks only roll off at 6dB's per octave, secondly the intersection of the main lobes of adjacent points is 4dB's down on the peak. Thirdly all "other" point responses have a null in the centre of the main lobe

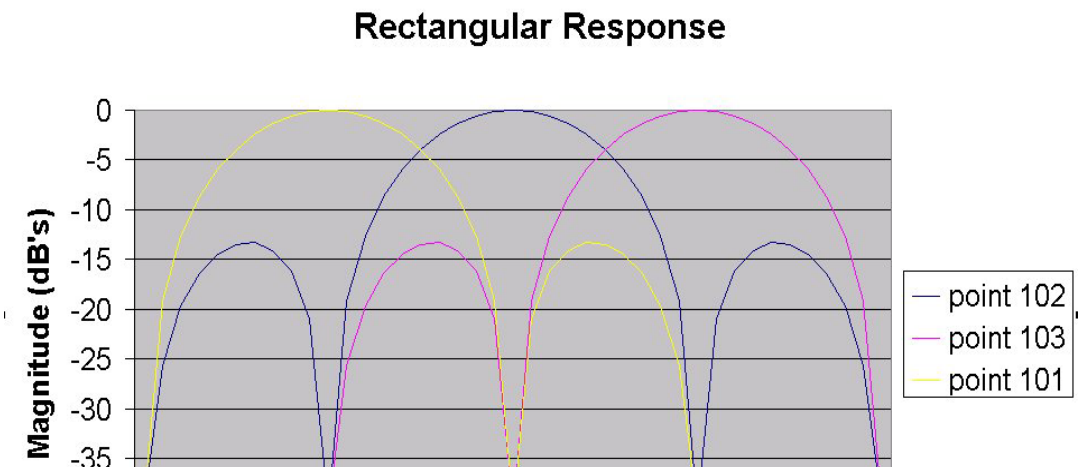


Figure 4 -- Sinx/x functions

The side lobe levels in the response is the reason that away from the main lobe, in figure 3, the FFT output does not drop down to the noise floor. The peak level measured at point 102 on the FFT output is not a true measure of the signal level as the roll off in the response of the main lobe can cause as much as 4dB's error, the intersection level of two adjacent main lobes.
One characteristic of the Sin(x)/x frequency response at the output of the FFT is that at the peak of the main lobe of any one of the points all the other points have a zero response. Figure 5 shows the FFT output for an input frequency from the DDS of (103 * 97656.25) = 10.05859375MHz, at the peak of the response for point 103. Notice that the skirts have disappeared, as all the other point's frequency responses have zeros at this frequency, and that point 103 is the only point in the peak and its level is greater than the level for point 102 in figure 3.

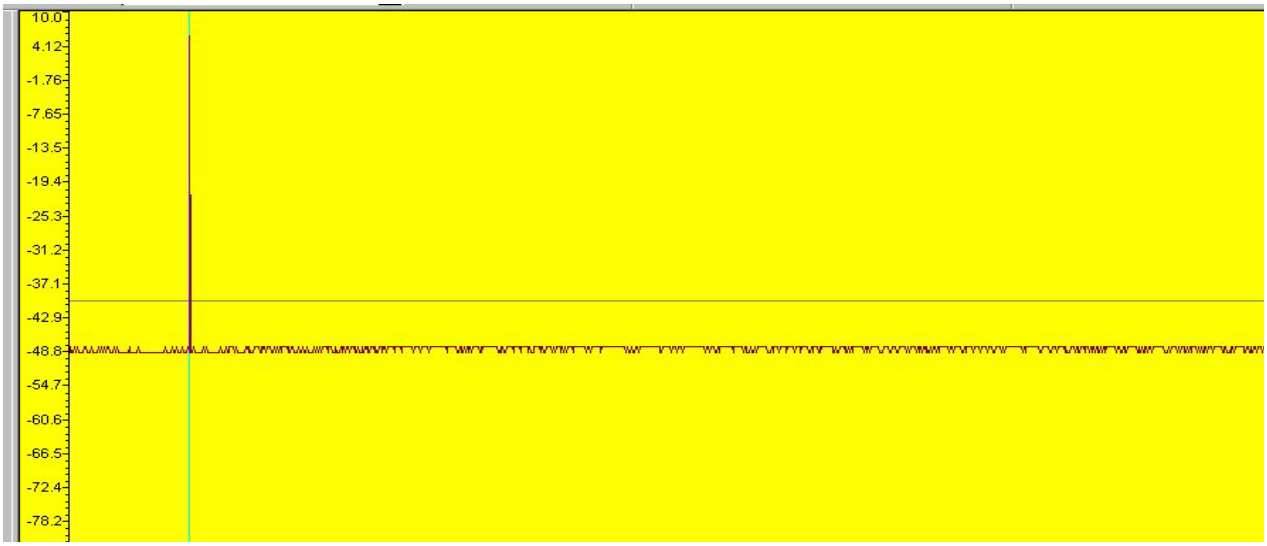


Figure 5 – DDS at 10.058MHz , Rectangular window

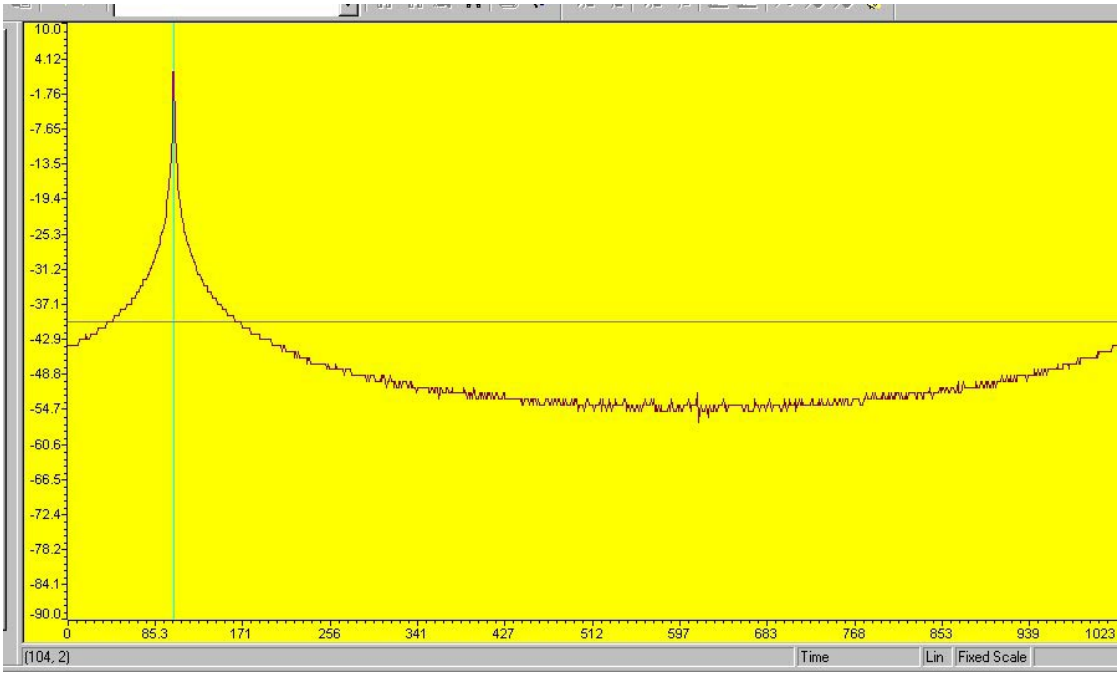


Figure 6 – DDS at 10.107MHz , Rectangular window

Figure 6 shows the FFT output for an input frequency from the DDS of (103.5 * 97656.25) = 10.10742188MHz, this half way between the adjacent points. The skirts are back, and the peak is shared equally between points 103 and 104 and both are 4dB's down from the peak level shown in figure 5.
The output frequency response of the FFT is just as expected for a rectangular window, the FFT block is working well, but what most applications require is to identify a frequency component measure its level and also identify any low level signals. The rectangular window located the frequency to the nearest point but the level was in a 4dB bracket, and depending on the signal frequency the skirts could mask any low level signals.
To overcome this problem a non rectangular data window can be used, in this example there are a set of shading coefficients for a Kaiser-Bessel window with alpha=3 stored in ROM on the FPGA. These coefficients can also be found in the 'k_b_window.coe' file in the Src directory in a form suitable for loading into a Xilinx ROM CoreGen block.

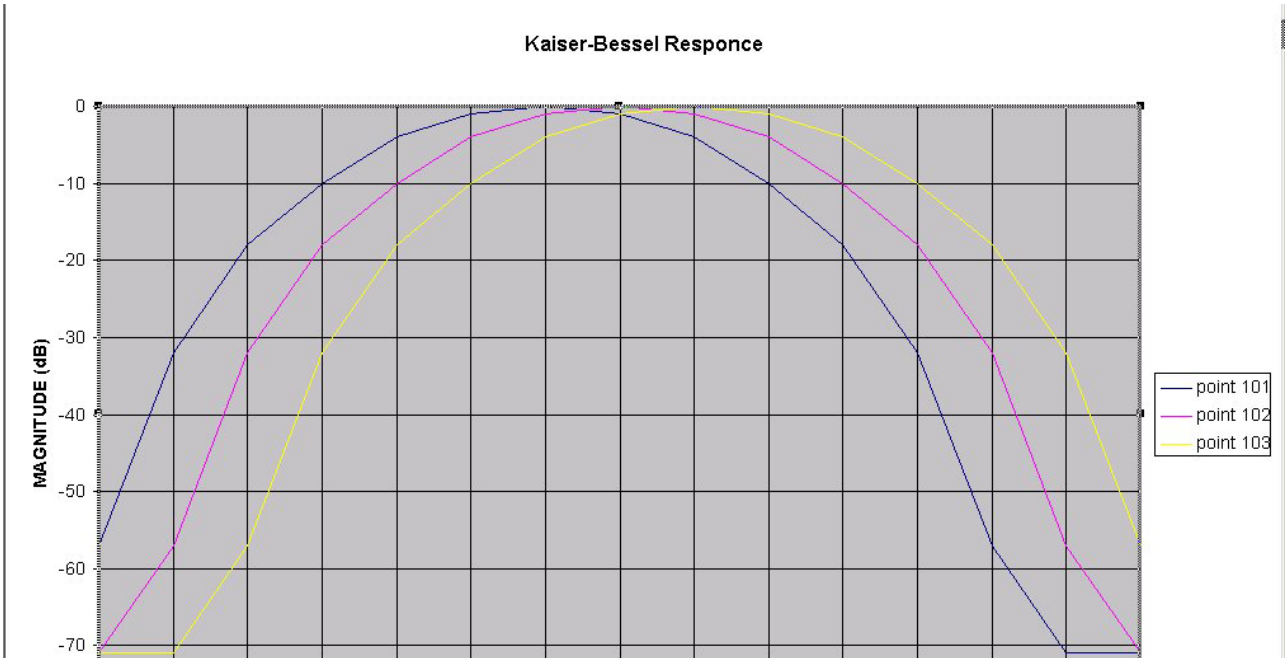


Figure 7 – Kaiser Bessel functions

Figure 7 shows the magnitude response for three adjacent points for a Kaiser Bessel window, the peaks are much broader than the Sinn(x)/x response, so the FFT output will be in more than one point but now the error in the measured level is down to about 1 dB. The first side lobe of the Kaiser-Bessel function is 69dB's down on the main lobe and successive peaks roll off at 6 dB's per octave so that the skirts are no longer noticeable.

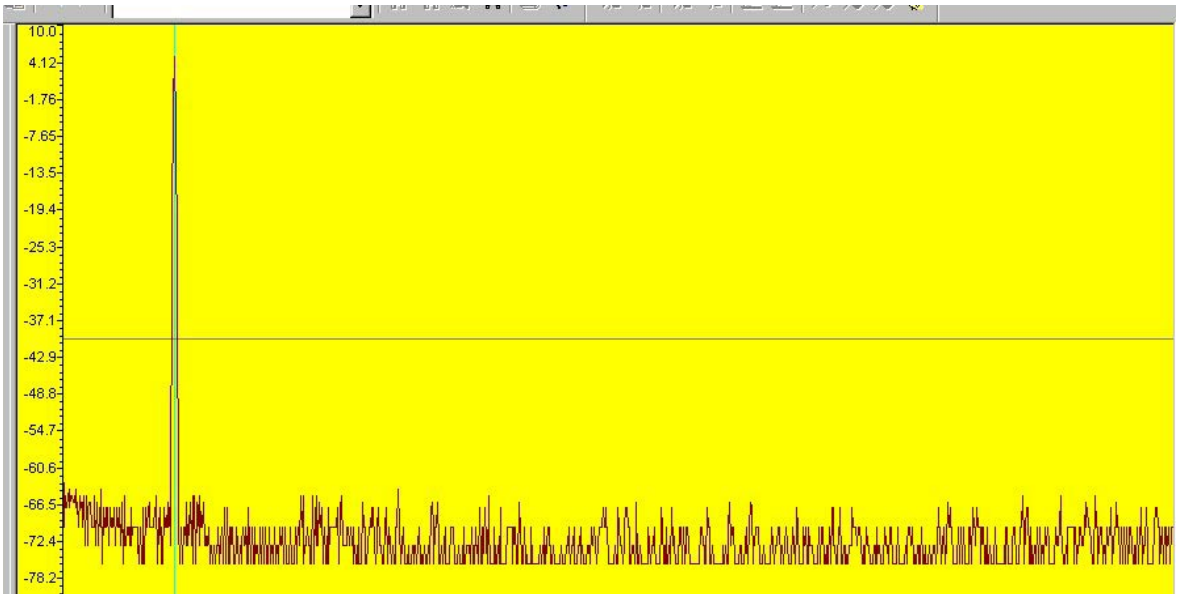


Figure 8 – DDS at 10.0MHz , Kaiser-Bessel window

Figure 8 shows the FFT output for an input frequency from the DDS of 10.0MHz, notice that the main lobe is broader but the skirts have gone.
At this point try increasing the DDS output frequency above Fs/2, the FFT still works with less than two samples per cycle because of the Phase and Quadrature nature of the data.

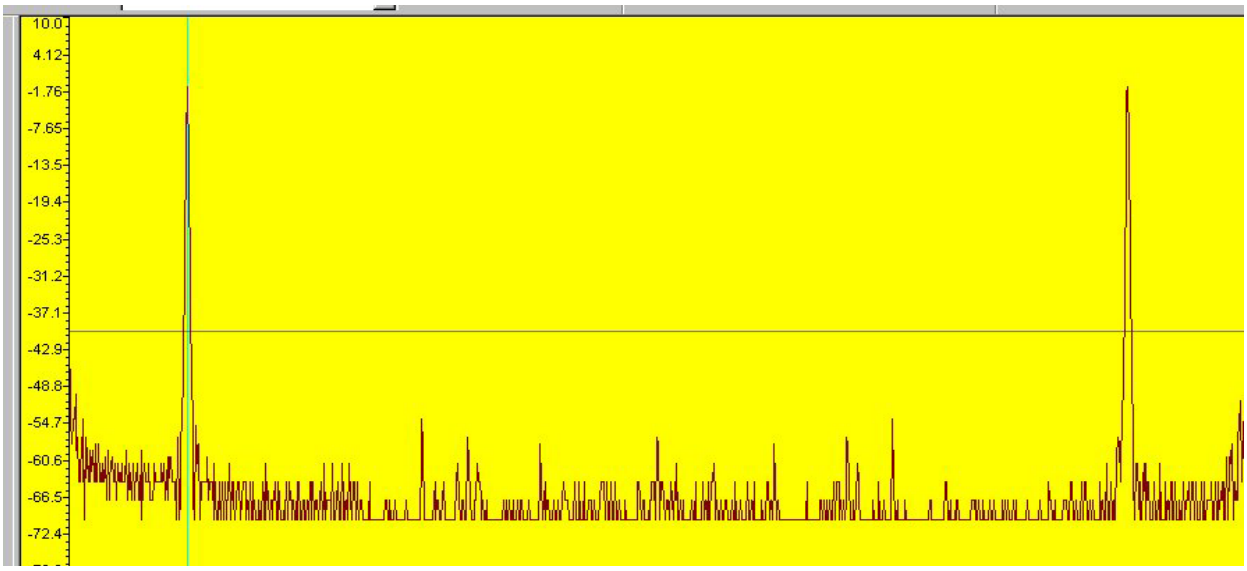


Figure 9 – ADC-A at 10.0MHz , Kaiser-Bessel window

Figure 9 shows the FFT output obtained by connecting a RF signal generator to the ADC-A input of the HERONIO2V2 with a frequency of 10.0MHz and an output level of +5dBm, this is equivalent to an input level of 2Volts peak to peak into the standard input impedance of the IO2V2 of 200 Ohms, and selecting the Signal

Generator input for the FFT with a Kaiser-Bessel window. The 10MHz lobe looks very like the output obtained with the DDS, but there are two main differences in the output. Both of these differences are because when the signal generator input for the FFT is selected the Quadrature input sample values are set to zero. The negative frequency components of the FFT output no longer cancel out when the magnitude is calculated so there is a second lobe at 90MHz. Setting the Quadrature input sample values to zero also lowers the effective input signal level to the FFT which is reflected in the output level being reduced by 6dB's.
Change the output frequency and level of the signal generator and observe the effect on the output of the FFT.

The last input option to the FFT is from RAM in the FPGA this allows any function to be used, the example software asks for a file name to download into the FPGA's memory block over the HERON interface. There are ***.dat files in the dsp directory that contain suitable data, more information on these files is at the end of the end of the 'FFT Example Specification' section. New files of this type can easily be generated for a forward FFT by calculating the values of the required waveform for both the Phase and Quadrature and saving them to a file, an example of this is shown in a commented out section of '**fft_example.c**'. To generate a file for use with the inverse transform it is easiest to capture the FFT output values when a forward transform is being performed. The Format of the data read and written over the HERON interface is the same, Phase in the least significant half word and Quadrature in the most significant half word, so by saving the forward FFT output directly to a file it can be loaded back into the memory for an inverse transform.
The signal energy for the frequency domain representation of a sine wave is low as it is effectively a single valued spike, so the sine output of the inverse FFT will have a relatively low amplitude. A gain of 64 has been implemented in '**fft_example.c**' when the inverse transform is selected to counter this. In the program '**fft_example.c**' the Phase data for an inverse transform is located in the integer array 'chadata', and the Quadrature data in the integer array 'chbdata'.
The forward FFT output is a direct representation of the time samples in the frequency domain, so if the output of a Kaiser-Bessel windowed forward FFT is used for an inverse FFT the sine wave output will have the window shading.
If the forward FFT is of the sum of two sine waves it will produce two peaks in the frequency domain, which when the inverse FFT is performed will produce a sum of sine waves time sequence, but if one of the peaks is removed in the frequency domain prior to being loaded into memory that frequency component will not be present at the output of the FFT in the time sequence. It has been filtered out.

Host based Example software

HEART carrier like HEPC9

If you have a HEART based module carrier like HEPC9, you can run the Host based example code with your HERON-IO2 module fitted to any slot. Then you need to configure the HEART connections between the module and the Host using HeartConf. This is done from inside the hegraph program for you using the network file that is in the same directory as the host example software.

HEPC8 carrier

If you have an HEPC8, then you must fit your HERON-IO2 module to the first HERON slot as this is the only slot that has a FIFO connection to the Host machine. In this case the FIFO number the module uses is FIFO #1.

Host example : What it does

We supply a pre-compiled Windows program for the PC, which can be used to communicate with the HERON-IO2, and to gather the data captured onto the PC where it is displayed.

The program is called 'hegraph.exe' and it is located in the 'host' sub-directory of this example. This example can be used both as a confidence check and a starting point for your development.

The 'hegraph.exe' program is supplied as an exe file, but we have included the Microsoft Visual C/C++ 6 project of this program as well. This allows you to change it and re-compile it as you want.

After you have loaded the correct bit stream into your HERON-IO2 module, the "DONE" LED should be switched off showing that the configuration was successful. Also The USER LED4 should flash about once per second, showing that the clocks are properly running in the FPGA. If the DONE LED is off, but the LED is not flashing it may be because the Delay Locked Loop used in the clock circuit of the FPGA needs to be reset. You can do this using the utility under "programs → HUNT ENGINEERING → API board RESET" if you want, but such a reset will also be made when you start the hegraph program.

When you run the hegraph program you will see a window appear, that has some control menus available. Using these menus you will be able to start the program (which will send HSB messages to correctly set the FIFO numbers that the FPGA should use, and also configure HEART if you have a HEART based module carrier). It will also capture and display the data coming from the IO2V2.

By selecting the 'Control" on the control menu this brings down a selection of four functions. Selecting one of these functions sends a control message to the FFT example on the IO2V2 via the HSB to select an input source for the FFT.

Function 1:- ADC output connected to the FFT.

Function 2:- DDS output connected to the FFT.(default frequency 10.0MHz)

Function 3:- ADC output connected to the FFT.

Function 4:- ADC output connected to the FFT.

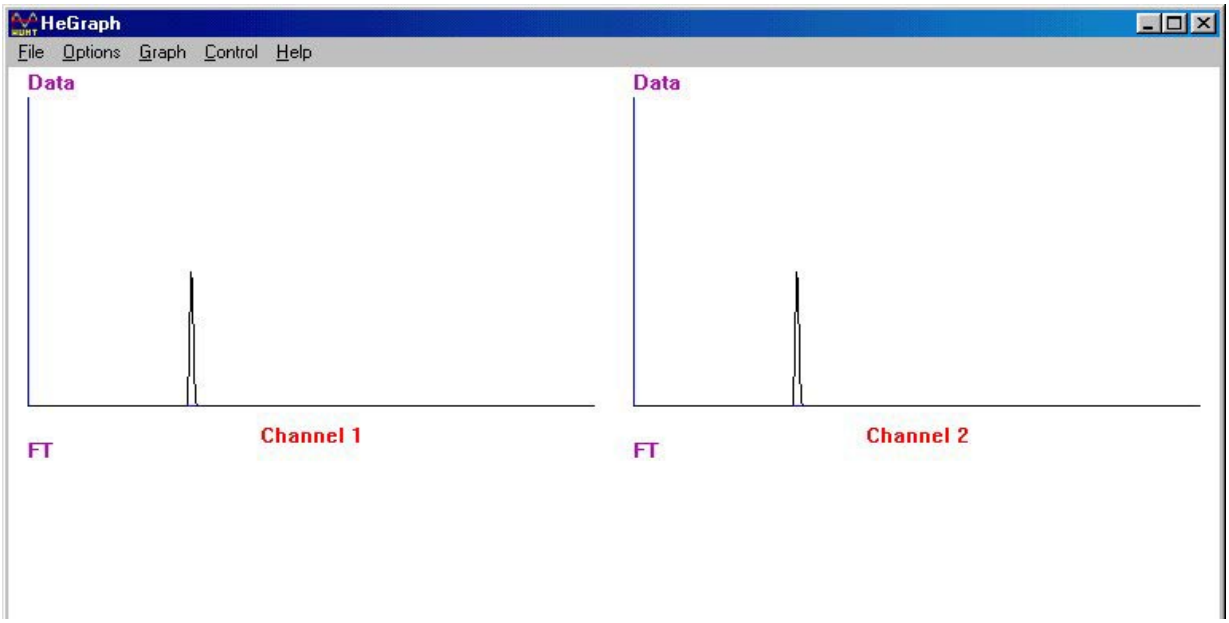
The program grabs blocks of data from the HERON interface. Each 32 bit word contains the output from the FFT in the form of the Phase component in the least significant half word, and the Quadrature component in the most significant half word.

The example2.c program loops round and gathers blocks of FFT output data from the FPGA over the HERON interface. This is in the form of Phase and Quadrature components. Example2.c also calculates the magnitude of each point of the FFT output by calculating the square root of the sum of the squares of the Phase and Quadrature components.

The hegraph program displays these FFT output values, with the output magnitude displayed in both channel 1 and channel 2. Figure 10 shows a output from the hegraph program for a signal generator source connected to the ADC-A input of the IO2V2 at a frequency of 10MHz and a level of +5dBm. The window used is Kaiser-Bessel.

If the signal source is changed to the DDS output, by selecting 'Function 2', the output level displayed will approximately double. This is because the DDS output has both Phase and Quadrature components while the ADC-A output has only a phase component. The Quadrature input to the FFT, when the ADC-A input is selected, is set to zero.

The hegraph program does give the option of taking a Fourier Transform of each channel of data but in this case it will not produce meaningful results.



FFT O/P MAGNITUDE FFT O/P MAGNITUDE

Figure 10 – Example of Host Software FFT Output

Implementation of the FFT

This example is based on a Xilinx CoreGen high performance 1024-point complex FFT/IFFT block, and uses other CoreGen blocks such as memory, DDS, etc to interface the FFT.

There is now another CoreGen FFT block available (28/03/03) which gives a complex FFT/IFFT for up to 16384 points with data sample precision from 8 to 24 bits. No attempt has yet been made to use this new core but from its data sheet it has three operating modes.

Radix-4 streaming I/O: This is the fastest mode but requires the most core space. Using the core space remaining after the HUNT ENGINEERING Transient Analysis Example 2 has been generated for a Virtex II XC2V1000 as a guide for the space still available for the FFT, a 2048 point by 16 bit would fit and would require 2048 clock cycles to complete the transform. This is equivalent to 20.48 micro Seconds with a 100MHz clock.

Radix-4 Burst I/O: This is a slower mode but uses less core space. Again using the core space left after the HUNT ENGINEERING Transient Analysis Example 2 has been generated as a guide, a 8192 point by 16 bit FFT should fit requiring 22581 clock cycles to complete data load and transform. This is equivalent to 225 micro Seconds with a 100MHz clock. Another option in the available core space would be a 2048 point by 24 bit FFT, which would take 5169 clock cycles, equivalent to 51.96 micro Seconds with a 100MHz clock.

Radix-2 Minimum Resources: This mode goes up to 1024 points by up to 24 bits and would fit in the available space. It would require 6215 clock cycles to complete data load and transform, equivalent to 62.15 micro Seconds with a 100MHz clock.

The above estimation of which of the FFT's would fit into the XC2V1000 on the HERON IO2V2 is based on the resources available in the FPGA relative to the FFT's requirements from the data sheet. The limiting factor tends to be the amount of block RAM available.

If the FFT was not limited by the XC2V1000 on the IO2V2, but was implemented in a module with larger FPGA's then the minimum size of FPGA, from the FFT plus Example2 requirements, would be:-

Streaming I/O

8192 x 16 FFT would require an XC2V3000

8192 x 24 FFT would require an XC2V6000

Burst I/O

8192 x 16 FFT would require an XC2V1000

8192 x 24 FFT would require an XC2V1500

The data sheet for the FFT claims a 16384 point transform in burst I/O mode but no resource requirements are given for this in the data sheet.

FFT Example Specification

Inputs: There are three possible signal inputs to the FFT:-

ADC-A

Sample Frequency: 100Msamples per Second

Sample Resolution: 12bit signed.

Analogue Input: +/- 1Volt Max into 200 Ohms

Direct Digital Synthesiser (DDS)

Sample frequency: 100Msamples per Second

Sample resolution: 16 bits full complex signal

Signal frequency: Default 10.0 MHz or as defined over the HSB

Signal amplitude: Always maximum (16bit signed).

FPGA Memory

Sample frequency: 100Msamples per Second

Sample resolution: 16 bits full complex signal

Signal: As defined by the values loaded over the HERON interface

Direct Digital Synthesiser(DDS):

DDS Frequency(default): 10.0MHz

Frequency Resolution(delF): 1.5Hz

Spurious Free Dynamic Range: 60dB

Sine and Cosine output: 16 bit signed

Frequency Control : 26 bits default to give 10MHz, the value is held in registers at HSB **addresses 5,6,7 and 8**.

How to calculate the 26 bit value to be loaded into these registers is described in the Xilinx DDS data sheet.

Window Multiplier:

Input Resolution: 16 bit signed on A input. (signal)

16 bit unsigned on B input.(window coefficients)

Output Resolution: 16 bit signed.

One multiplication per sample clock cycle (100MHz)

Kaiser-Bessel Window ROM:

ROM size: 1024x16

Window coefficient file: k_b_window.coe

The window is the same for both phase and quadrature channels so only one set of coefficients is required. For a rectangular window the coefficients become 0xffff across the whole window.

HSB interface registers:

Address 0: HERON input fifo number select.

Address 1: HERON output fifo number select.

Address 2 and 3: Number of output points to be transferred over the HERON interface.

HSB address	Description
HSB address 2	LS 8 bits (D7 – D0)
HSB address 3	MS 2 bits (D9 – D8) MS--XXXXXX(D9)(D8)--LS

HERON Interface number of output points

When address 3 is written to it initiates the transfer of the number of points defined.

Address 4: Bits written to this register perform three control functions.

D7,6,5,4	D3	D2	D1	D0
X	FORWARD FFT	WINDOW SELECT	INPUT SELECT	

Address 4 Control byte

Bits D0 and D1 control which signal source is selected for the FFT

D1	D0	SIGNAL SOURCE
0	**0**	**ADC-A**
0	1	DDS
1	0	MEMORY
1	1	ADC-A

Signal Source control bits

D2	WINDOW TYPE
0	**Kaiser-Bessel**
1	Rectangular

Window control bit

D3	FORWARD/INVERSE FFT
0	INVERSE
1	**FORWARD**

Forward/Inverse FFT control bit

The options denoted in **bold type** for address 4 are the default settings.

Address 5, 6, 7 and 8: DDS frequency control bits.

HSB address	Description
HSB address 5	LS 8 bits (D7 – D0)
HSB address 6	Next 8 bits (D15 – D8)
HSB address 7	Next 8 bits (D23 – D16)
HSB address 8	MS 2 bits (D25 – D24) MS--XXXXXX(D25)(D24)--LS

DDS FREQUENCY CONTROL REGISTERS

Refer to the section on the DDS CoreGen block or the Xilinx DDS data sheet as to how this value is calculated. The value is also calculated in 'fft_example.c'.

Heron Interface:

The FFT example can use the HERON interface both to read and write data to the FPGA. Writes to the FPGA load the (1024x32) signal source memory block.

The data read from the FPGA are 1024 point blocks of FFT output values.

The 32bit data words for both read and write across the HERON interface have the same format, the 16 bit Phase component in the Least Significant half word and the 16 bit Quadrature component in the Most Significant half word.

MS half word	LS half word
(16 bit Quadrature value)	(16 bit Phase value)

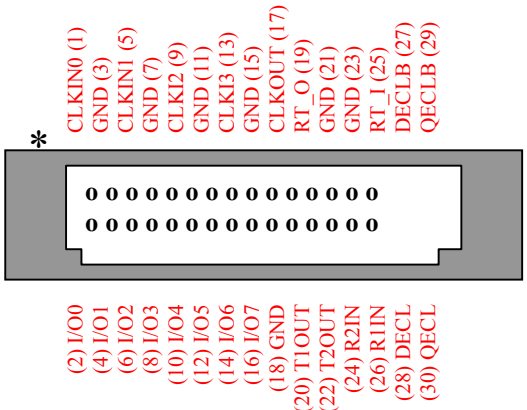
HERON Interface 32 bit Word Data Format

D/A Outputs:

Output Update Rate:	100MHz
Digital Input:	14 bit binary
Analogue Output:	+/-1Volt max

Digital IO Connector:

This is used in the example to output a pulse to synchronise the FFT output on DAC-A with an oscilloscope time base. The synchronisation pulse is output on (2)I/O0.



Files for Memory signal source:

There are five '.dat' files included in the dsp folder which contain data in a form suitable for down loading to the memory in the FPGA. The first two are for forward FFT's and the second three are for inverse FFT's.

Sine_5mhz.dat: Contains data representing a 5MHz complex sine wave in the time domain.
Two_tone.dat: Contains data representing the sum of a 5MHz and 18.5MHz complex sine waves in the time domain.

Fft_op_5mhz.dat: Contains data representing a 5MHz complex sine wave with a rectangular window in the frequency domain.
Fft_op_two_tone.dat: contains data representing the sum of a 5MHz and18.5MHz complex sine wave with a rectangular window in the frequency domain.
Fft_op_10mhz.dat: Contains data representing a 10MHz complex sine wave in the frequency domain, with a **Kaiser-Bessel** window.

FFT CORE GENERATOR BLOCKS

There are ten CoreGen blocks that have been generated particularly for this FFT example, to help show how each fits into the example figure 10 is a block diagram showing where each block fits into the signal path.

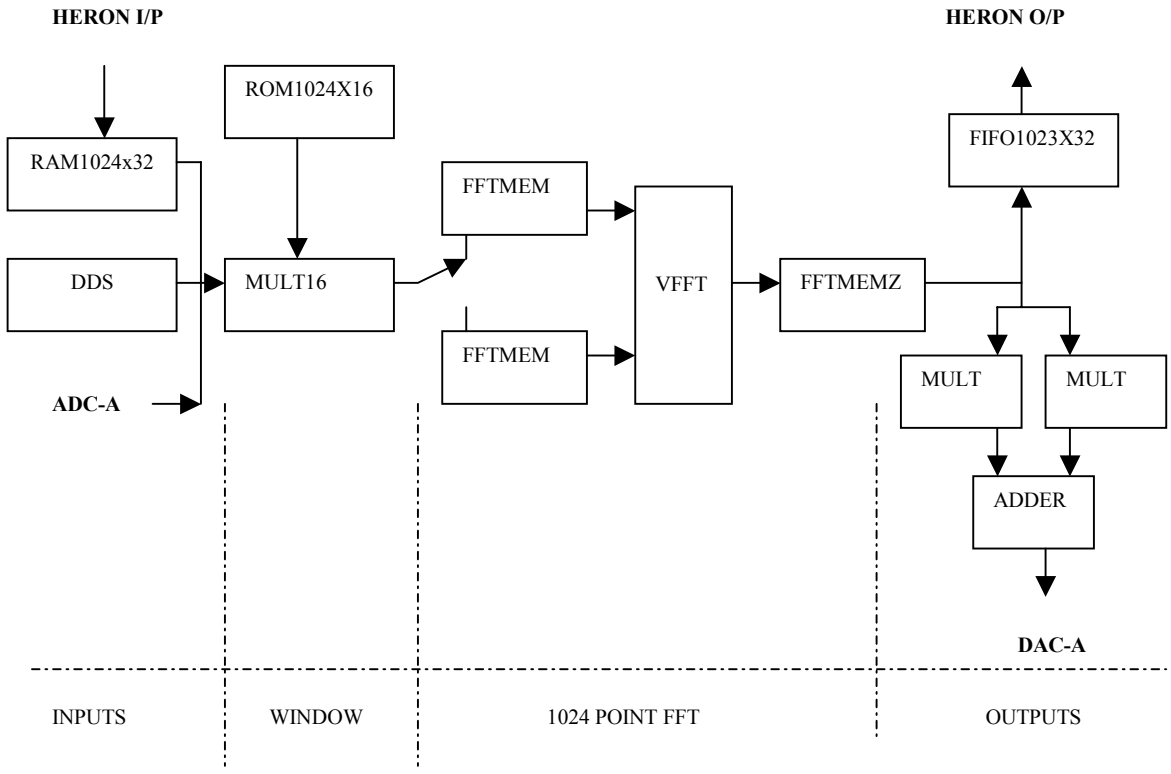


Figure 11 -- CoreGen blocks in the data path

Direct Digital Synthesiser

The DDS is a Xilinx 'CoreGen' block which has a two page GUI to define the operation of the built core. If further information is required there is a data sheet available by selecting the button at the bottom of the GUI.

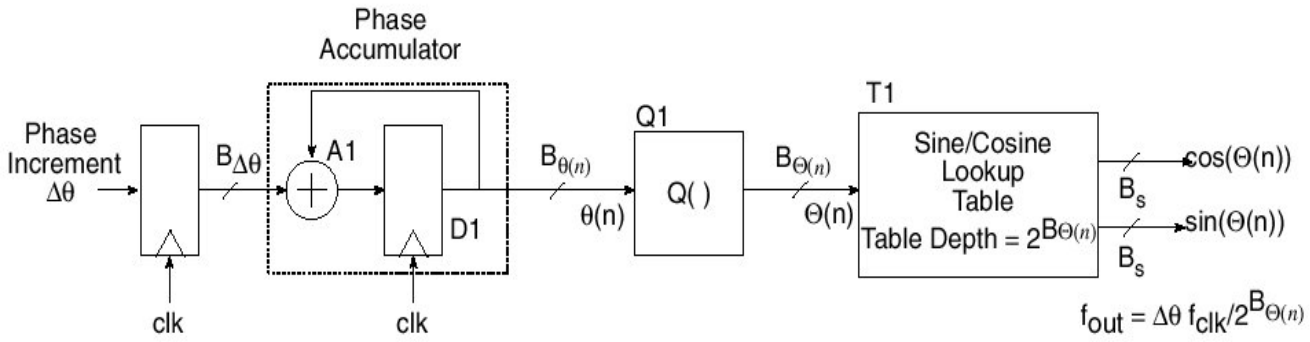


Figure 12 - Block Diagram of DDS

In this example the phase increment value is registered in and then its value is added into the accumulator on every cycle of the clock. The output value of the accumulator has a number of the least significant bits removed by the slicer Q1. The remaining bits address the sine/cosine look up table T1.

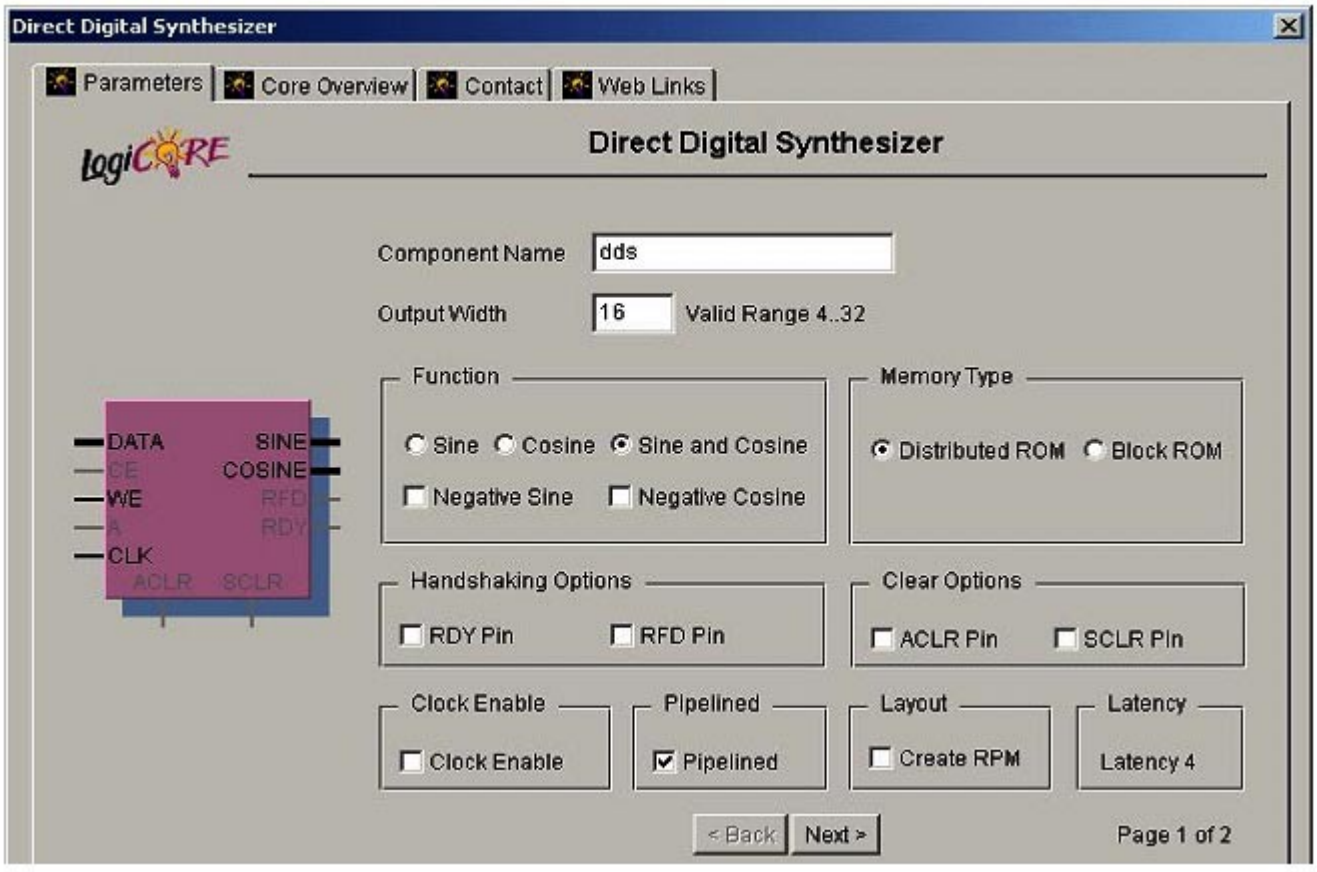


FIGURE 13 -- CoreGen DDS GUI Page 1

Output Width: 16Bits selected to match the FFT converter input width.

Function: 'Sine and Cosine' selected for full quadrature signal.

Negative Sine or Cosine options have not been selected for this example.

Memory Type: Distributed ROM selected.

Handshake Options: No handshake options are required for this example as the outputs are updating on every clock cycle.

Clock Enable: The clock enable is not selected as the DDS is running at 100MHz Sample clock frequency.

Clear Options: Clear options are not selected for this example.

Pipelined: Pipelined option selected.

Layout: Create RPM option not selected

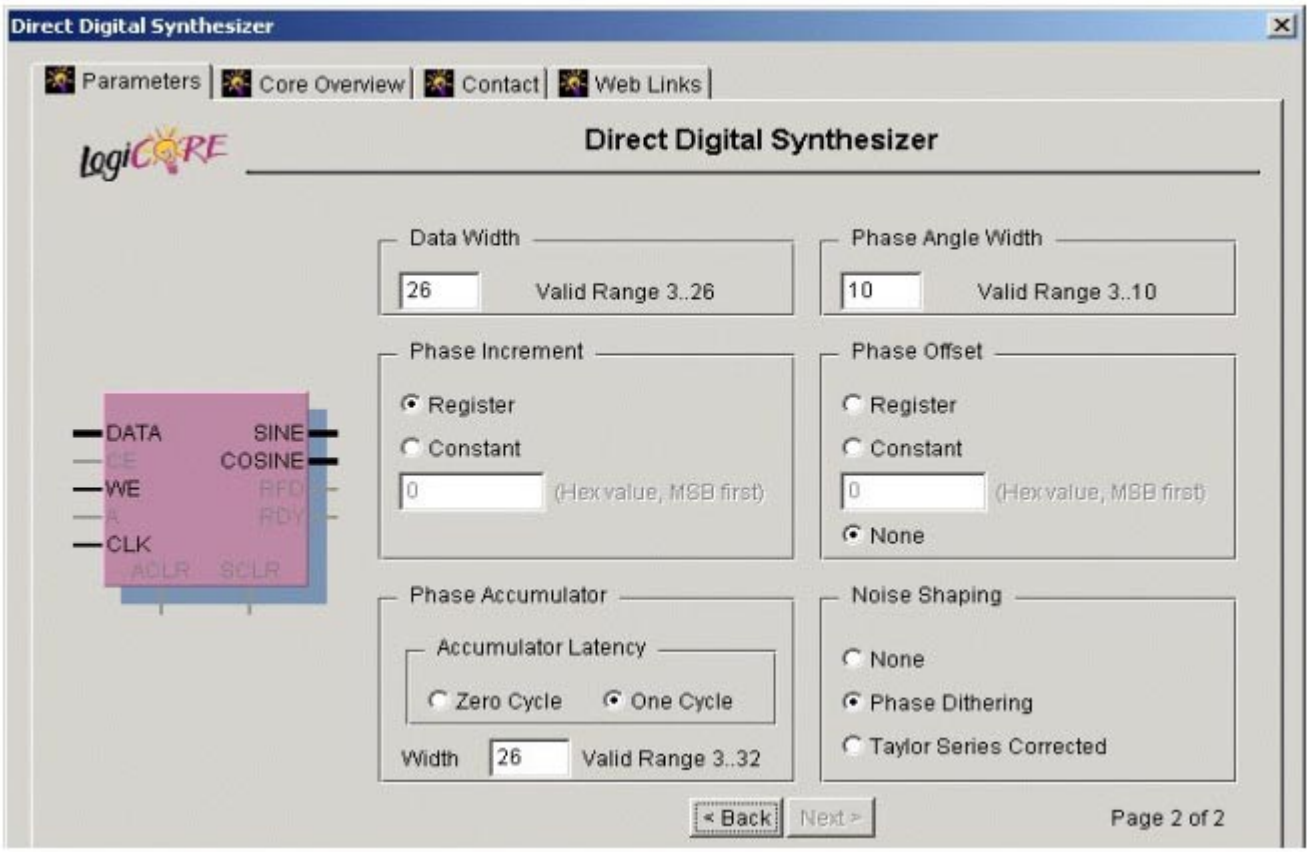


FIGURE 14 -- CoreGen DDS GUI Page2

Data Width: This is the width of the input data bus used to define the phase increment, and has been set to the full accumulator width of 26 bits.

Phase Angle Width(Blut): This is the number of bits used to address the sine/cosine look up table and is directly related to the Spurious Free Dynamic Range by 6dB's per bit. So 10 bits gives the required 60dB Spurious free dynamic range.

Phase Increment: The register option has been selected so that the Phase Increment value is loaded via the 26 bit input data bus, using the Write Enable(WE). In this example it allows the frequency to be updated via the HSB. The Phase Increment is defined by the relationship:-

DelPhi = (Fout / Fclk) * 2^(Blut)

With Fout = 10MHz; Fclk = 100MHz; and Blut = 10bits

Then DelPhi = 102.4

This real number now needs to be converted into a binary number that con be loaded into the DDS via the data input. The value at the output of the accumulator is 26 bits of which only the most significant 10 bits are used to address the sine/cosine look up table and are the integer part of 'DelPhi', the remaining 16 bits are the fractional part. To generate the binary number to represent 102.4 :-

Multiply 102.4 by 2^(16) = 6710886.4

This has in effect moved the point by 16 binary places

Convert to binary =1100110 01100110 01100110

But this is only 23 bits as the most significant bits are zero, so the full 26 bit value is:-

0001100110 01100110011001100110

Phase Offset: Not selected for this example.

Phase accumulator: 26 bit width with single cycle latency selected. The number of bits is determined by the frequency resolution relative to clock frequency:-

Bpa = log(Fclk / delF) / log(2)

With Fclk = 100MHz and delF = 1.5Hz

then Bpa = 26bits

Noise Shaping: Only 10 bits are used to address the Sine/Cosine look up table in this example out of the Phase accumulator width of 26 bits, this leads to an error between the ideal value and the value looked up in the Sine/Cosine look up table. At the output of the FFT this will look like noise with unwanted spectral lines at least 60dB down on the wanted signal, but this can be partially overcome by adding Phase Dither which has the effect of spreading the unwanted spectral lines over the whole band. Phase Dither has been selected for this example.

RAM 1024x32

This RAM is used to store a signal shape for input to the FFT. The Values stored in the memory have been transferred across the HERON interface.

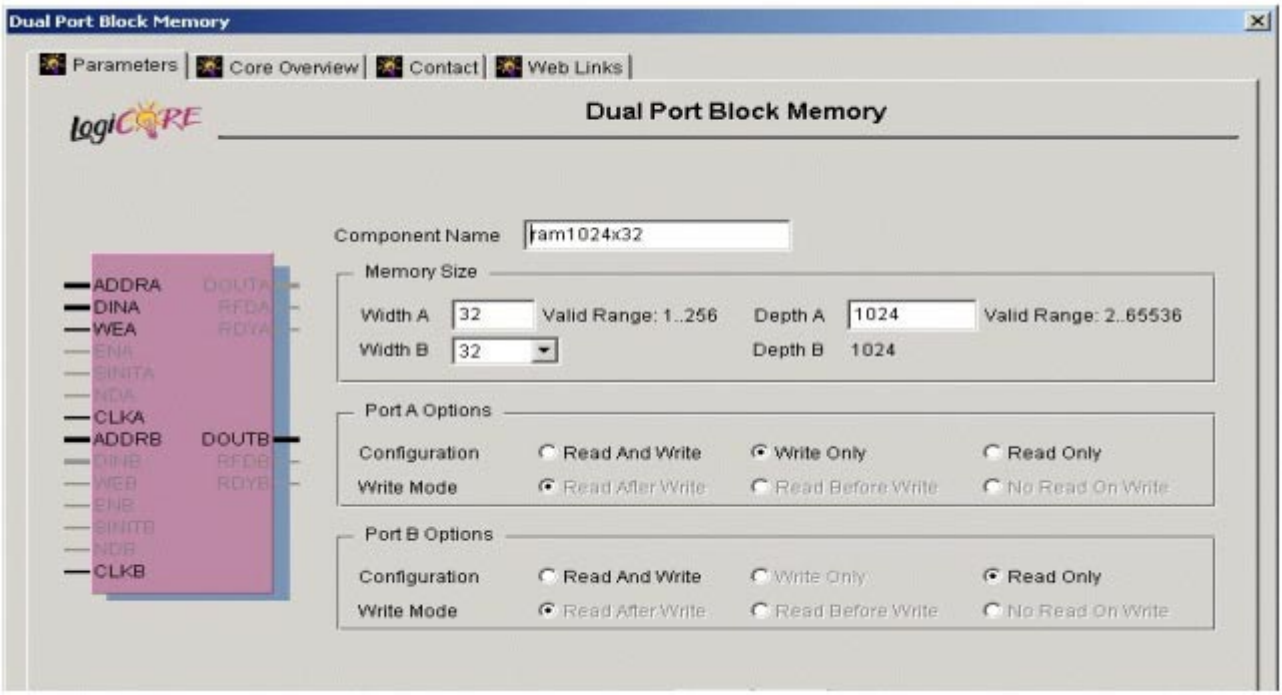


FIGURE 15 -- CoreGen Block RAM GUI page 1

Memory Size: The memory size used for this example is 1024, to match the number of points in the FFT, by 32 bits, to match the width of the HERON interface. In each memory location the Phase component is stored in the Least significant half word, and the Quadrature component in the Most significant half word.

Port A options: Port A is used to write the data into memory from the HERON interface.

Port B options: Port B is used to read the values for input to the FFT.

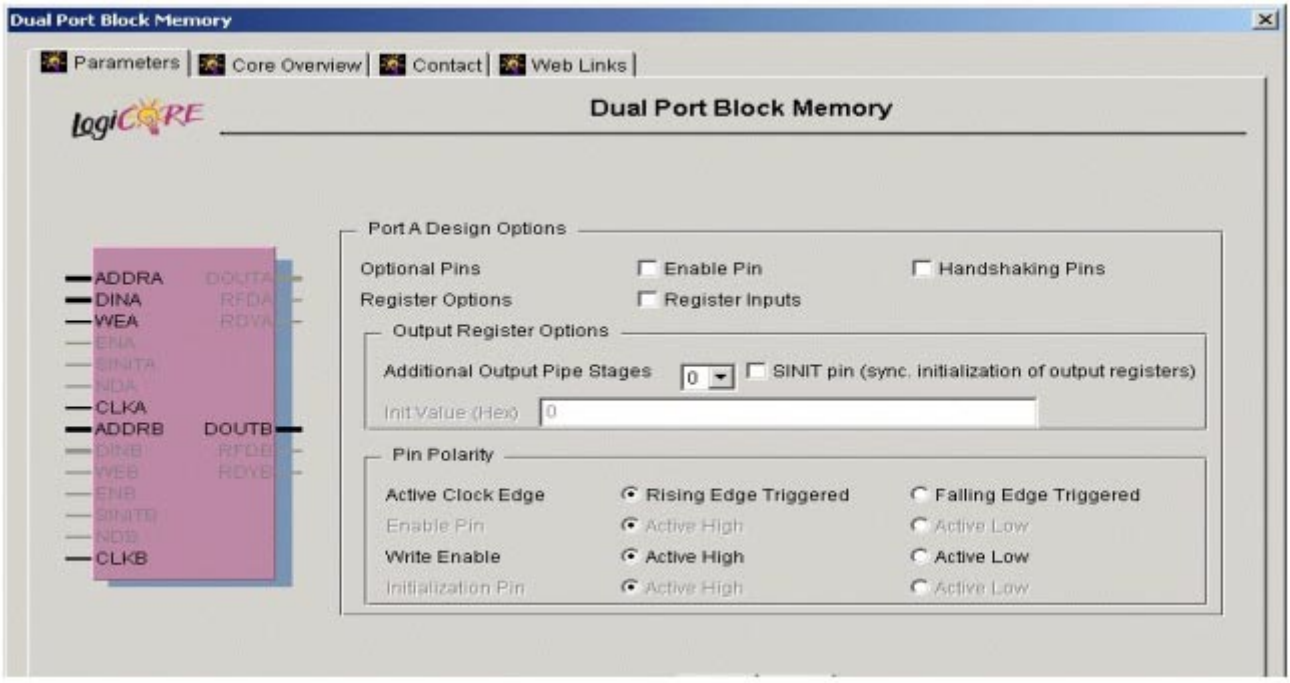


FIGURE 16 -- CoreGen Block RAM GUI page 2

Port A Options: None of the Port A options has been selected.

Port A Pin Polarity: Rising edge triggered for the active clock edge, and active high Write Enable.

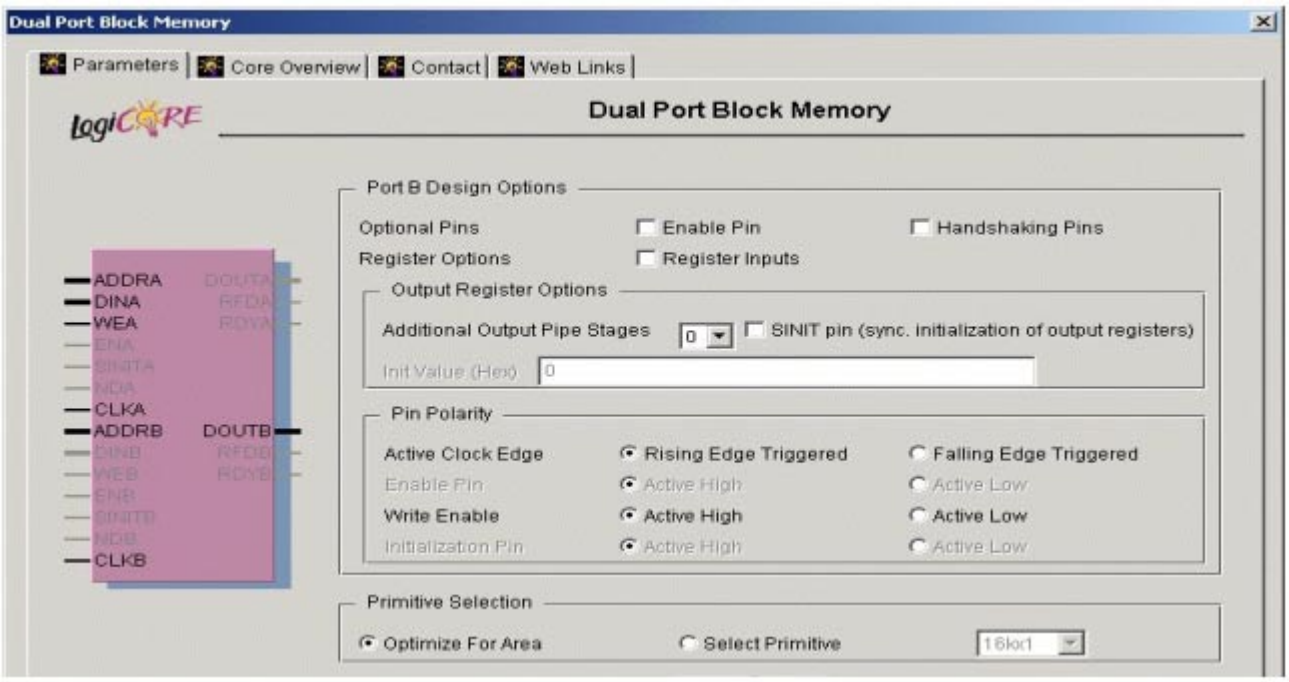


FIGURE 17 -- CoreGen Block RAM GUI page 3

Port B options: None of the Port B options has been selected.

Port B Pin Polarity: Rising edge triggered for the active clock edge, and active high Write Enable.

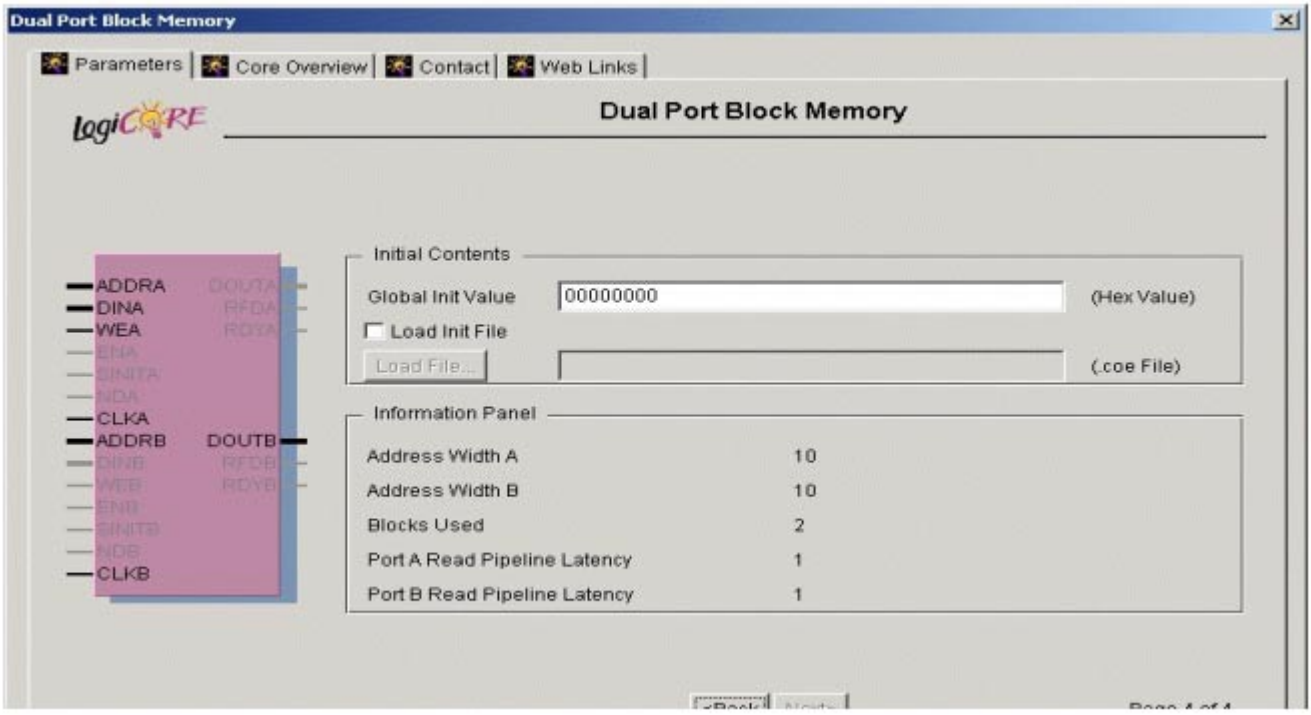


FIGURE 18 -- CoreGen Block RAM GUI page 4

Initial Contents: All of the RAM is initialised with zero's.

Mult16

This multiplier is used to apply the window function to the input blocks of data. Port A is connected to the signal data, which could come from either ADC-A , the DDS or from the Memory input. This is 16 bit signed data. The other port, Port B, is connected to the ROM that contains the window coefficients, this is 16 bit unsigned data. There are two of these multipliers used in the example, one for the Phase data and one for the Quadrature data.

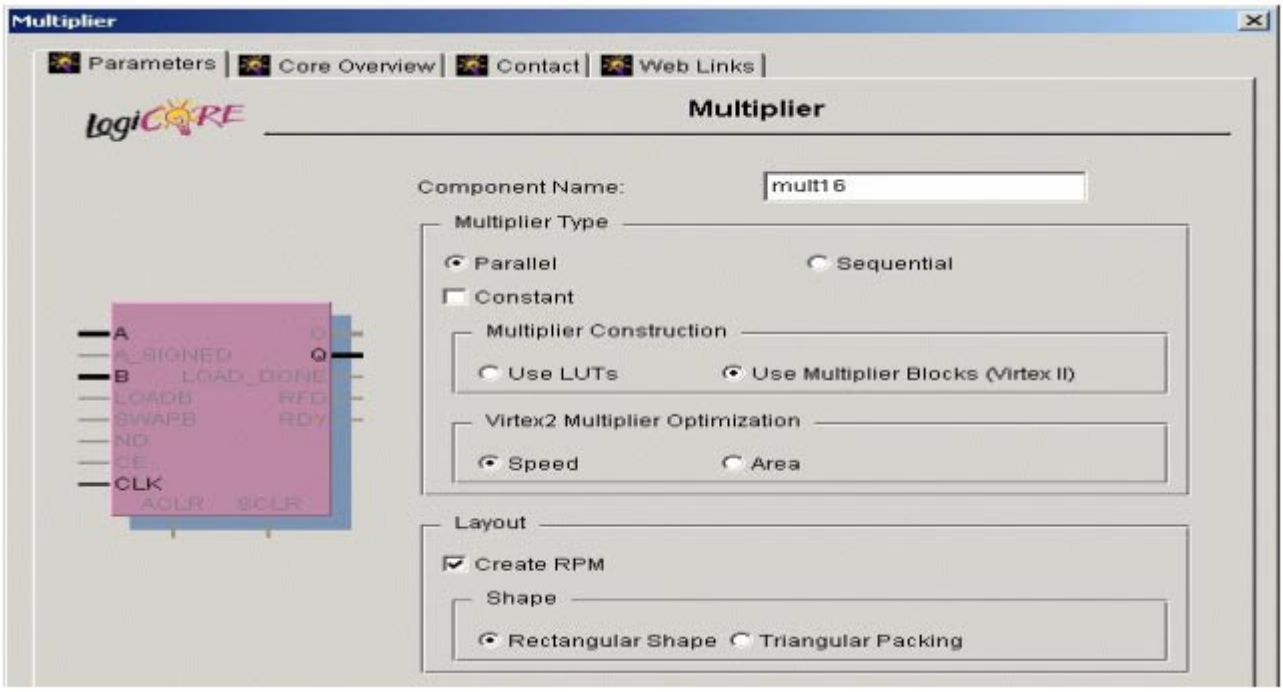


FIGURE 19 -- CoreGen Block Multiplier GUI page 1

Multiplier Type: This multiplier has to handle a clock rate of 100MHz as this is the rate at which signal data will be presented at its input, to achieve this the multiplier architecture is made parallel. As this example is using a IO2V2 with a Virtex II FPGA the dedicated multiplier blocks are used. The multiplier is optimised for speed.

Layout: Create RPM has been selected, with a rectangular shape.

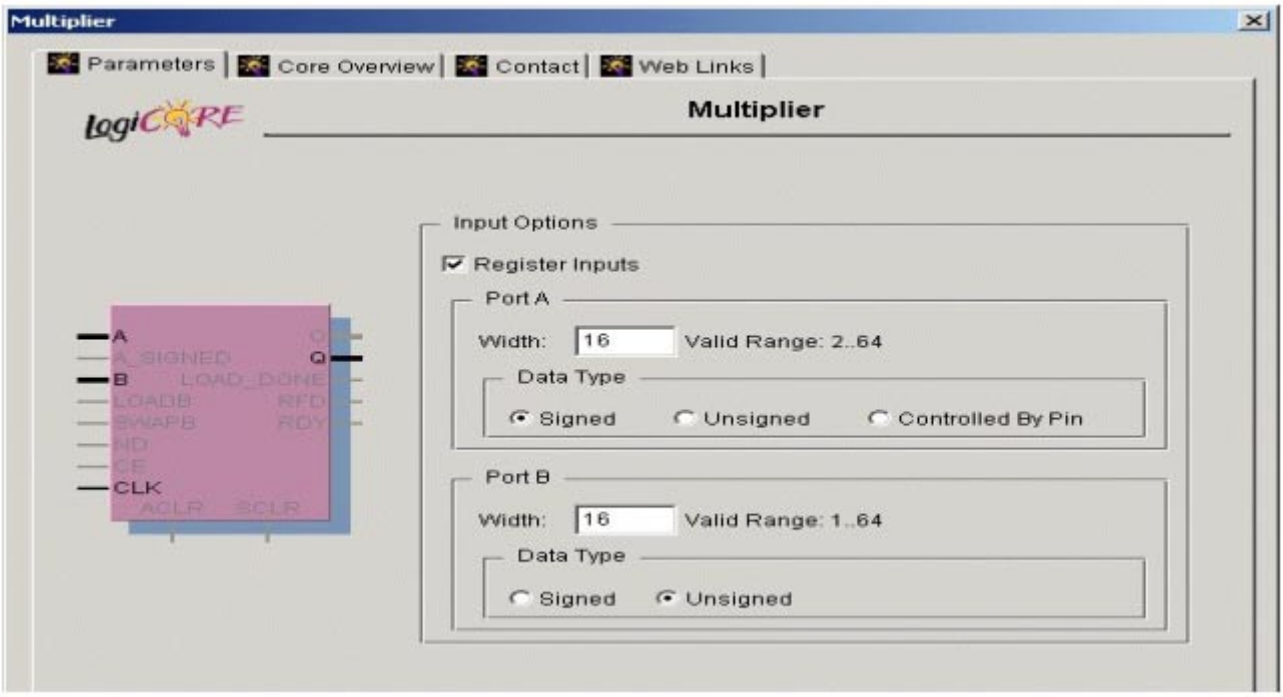


FIGURE 20 -- CoreGen Block Multiplier GUI page 2

Port A input options: Port A is 16 bits wide and expects signed data, this is the signal data. The input data is registered.

Port B input options: Port B is 16 bits wide and expects unsigned data, these are the window coefficients from the ROM. The input data is registered.

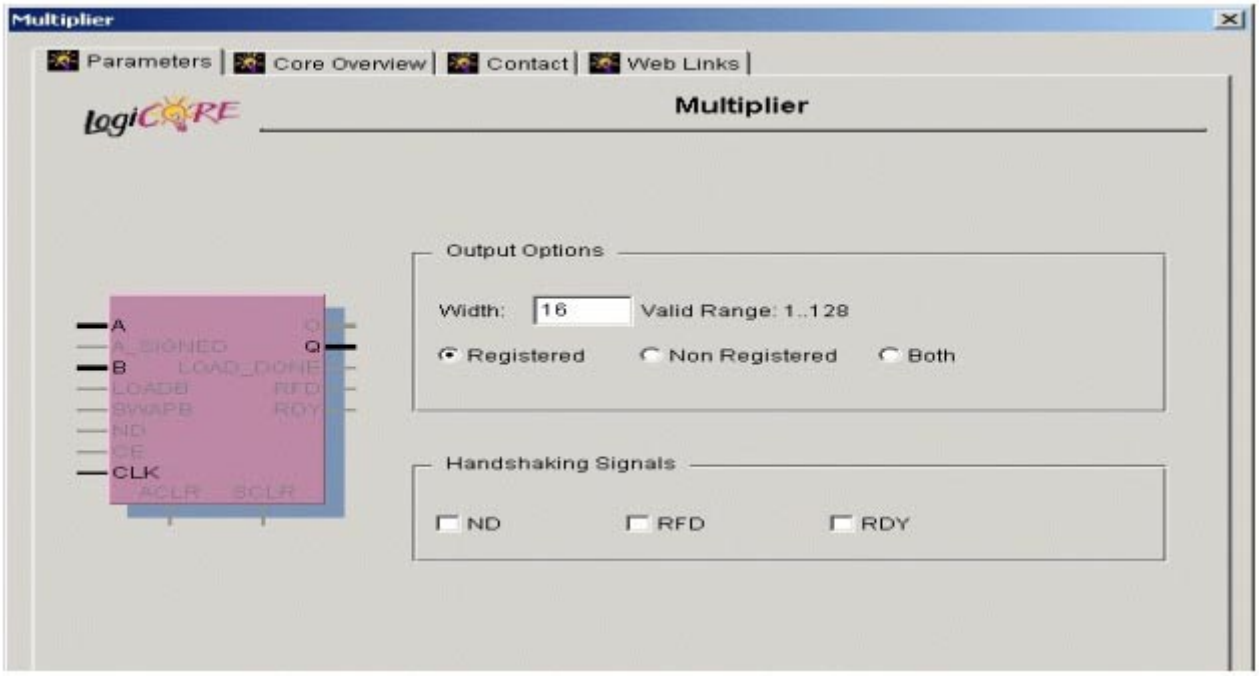


FIGURE 21 -- CoreGen Block Multiplier GUI page 3

Output options: The output width for the multiplier has been set to 16 bits to match the width of the FFT input data. The output is registered. There are no handshake options selected as the multiplication rate is one per clock cycle.

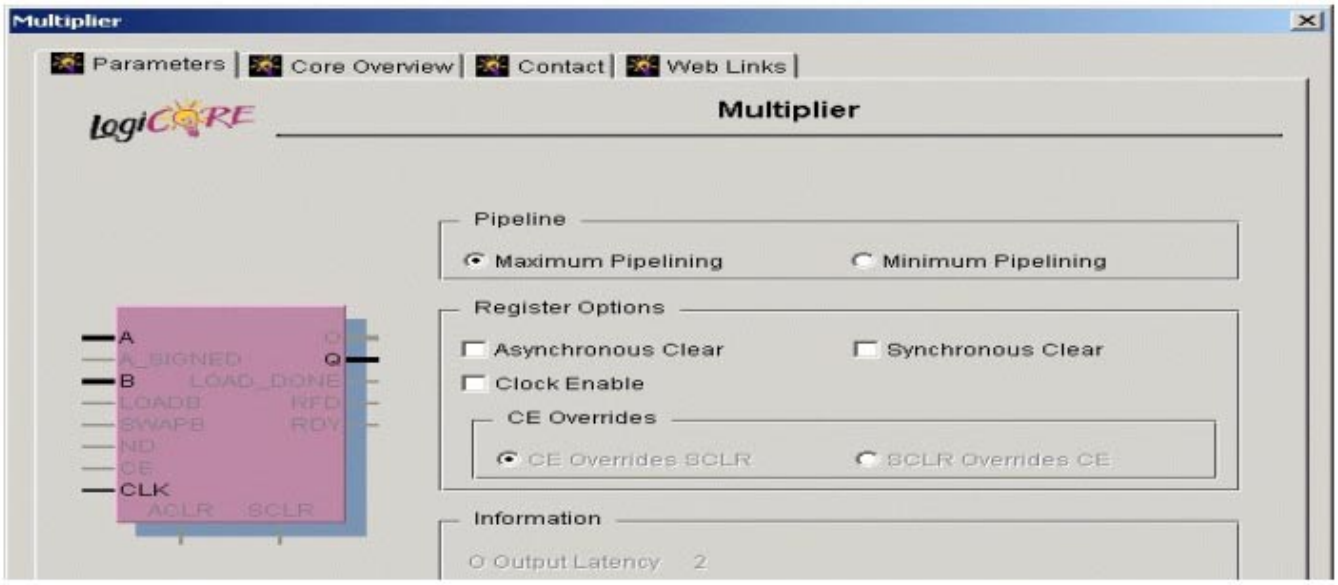


FIGURE 22 -- CoreGen Block Multiplier GUI page 4

Pipeline: Maximum pipelining has been chosen because of the speed at which the multiplier is to operate. Maximum pipelining helps with the speed but will increase the output latency.

Register options: There is no requirement for extra control signals for the registers in this example.

ROM 1024x16

This block of ROM is used to store the window coefficients for the Kaiser_Bessel Window. Only one block is required as the same coefficients are used for both Phase and Quadrature signals.

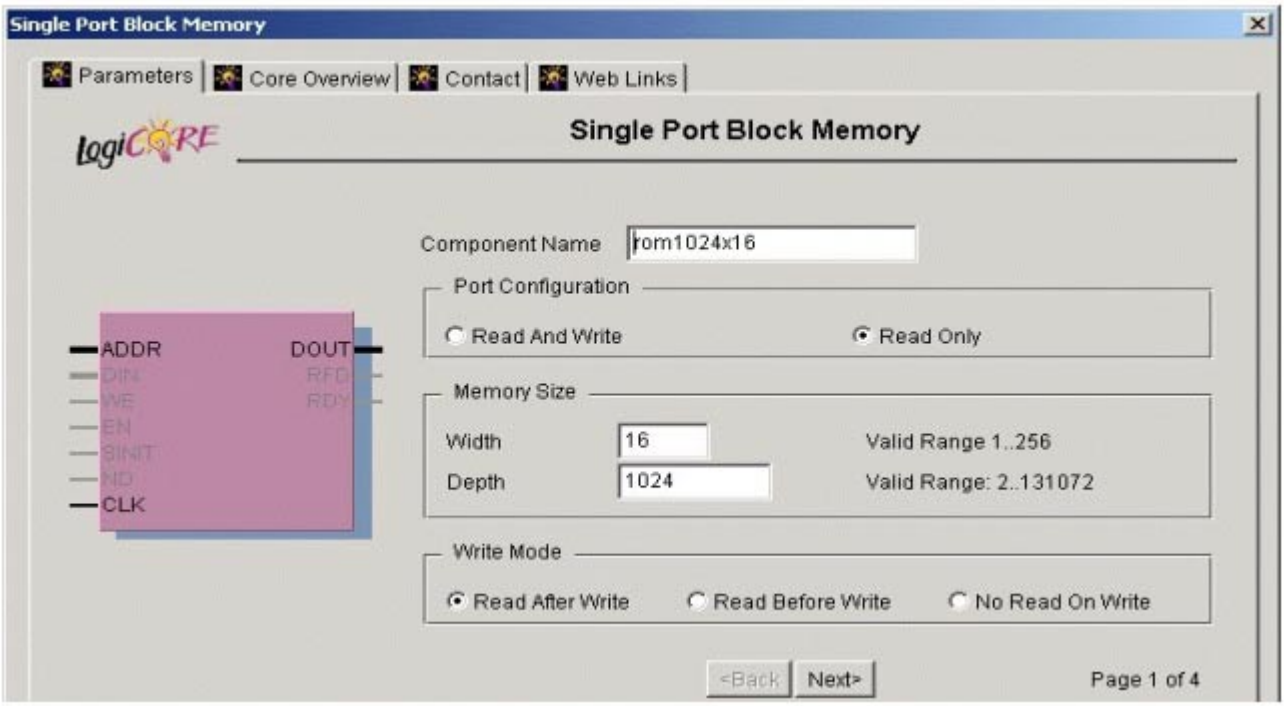


FIGURE 23 -- CoreGen Block ROM GUI page 1

Port Configuration: As this is to be a ROM block the port is configured as read only.

Memory Size: The data width and depth has been made to match the input of the FFT, 1024 by 16 bits.

Write Mode: Does not apply for Read only memory.

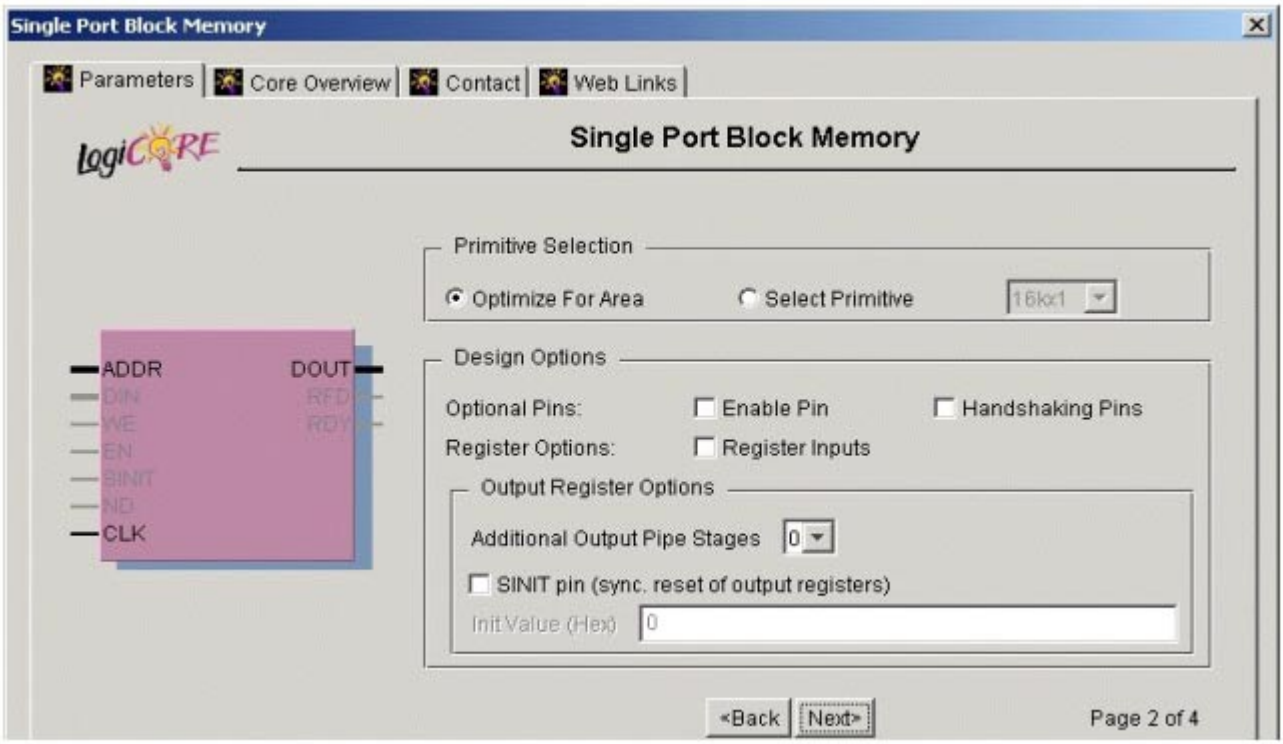


FIGURE 24 -- CoreGen Block ROM GUI page 2

Primitive Selection: Optimise for area selected.

Design Options: None of the options has been selected.

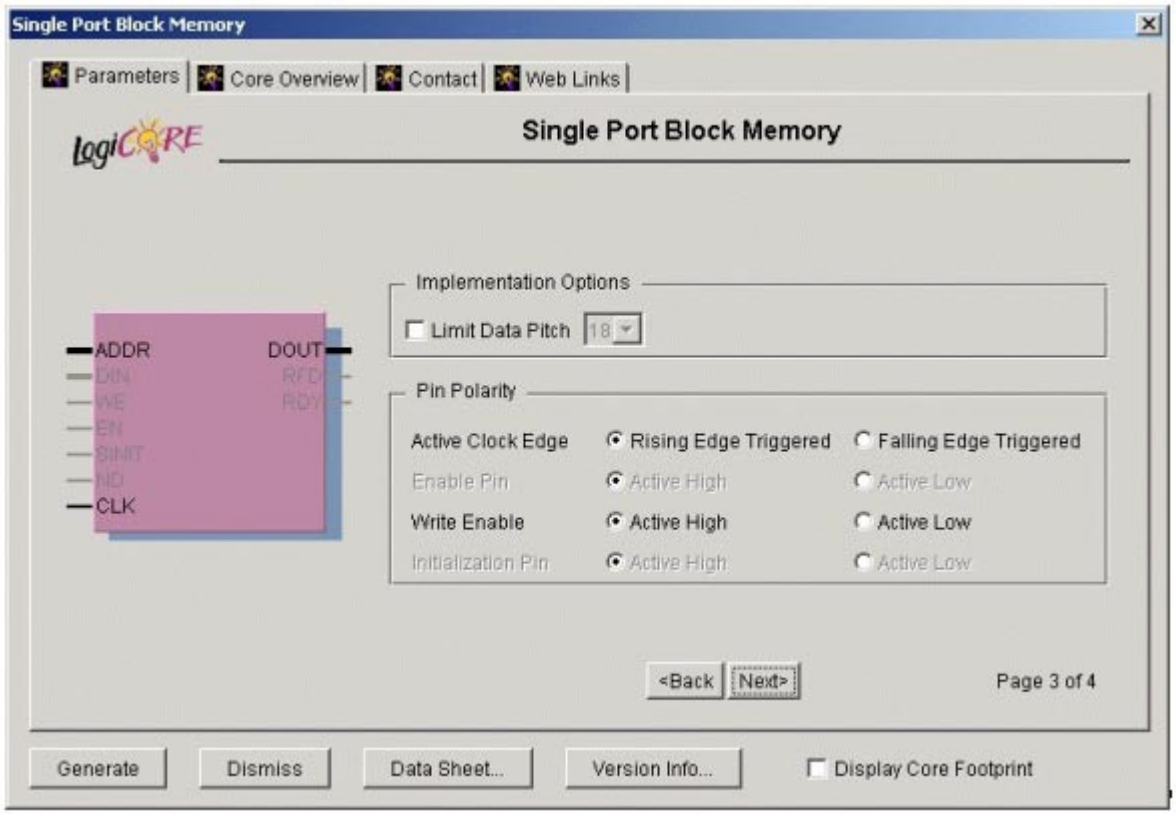


FIGURE 25 -- CoreGen Block ROM GUI page 3

Implementation Options: Limited data pitch not selected.

Pin Polarity: Clock rising edge triggered selected. The write enable options do not apply for ROM.

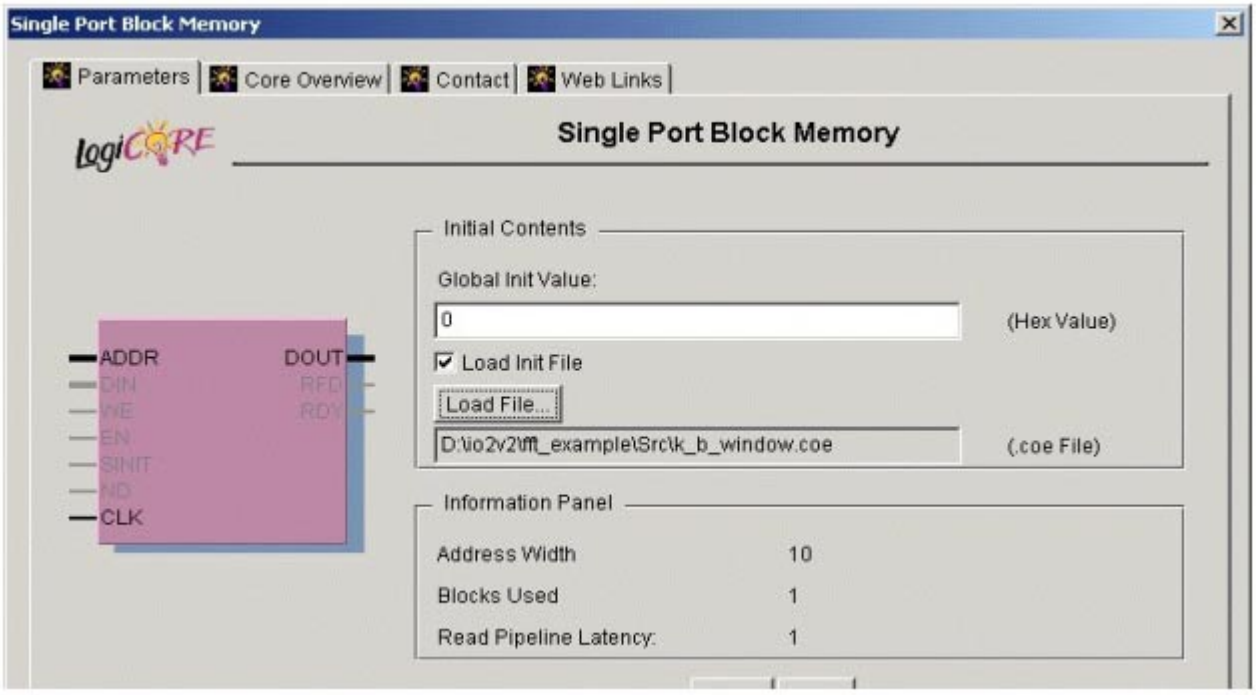


FIGURE 26 -- CoreGen Block ROM GUI page 4

Initial Contents: The initial values loaded into the ROM are defined by the file ‘k_b_window.coe’ that is located in the ..\Src directory

FFTMEM

This memory block is used for the input/working memory for the FFT CoreGen block in Triple Memory Space(TMS) configuration. In this mode one pair of blocks of memory, Phase and Quadrature, is used to capture data, while the other pair are used as working memory for the FFT. When the FFT is completed the rolls of the memories are swapped and the input memory now becomes the working memory and what was the working memory now captures a new block of input data.

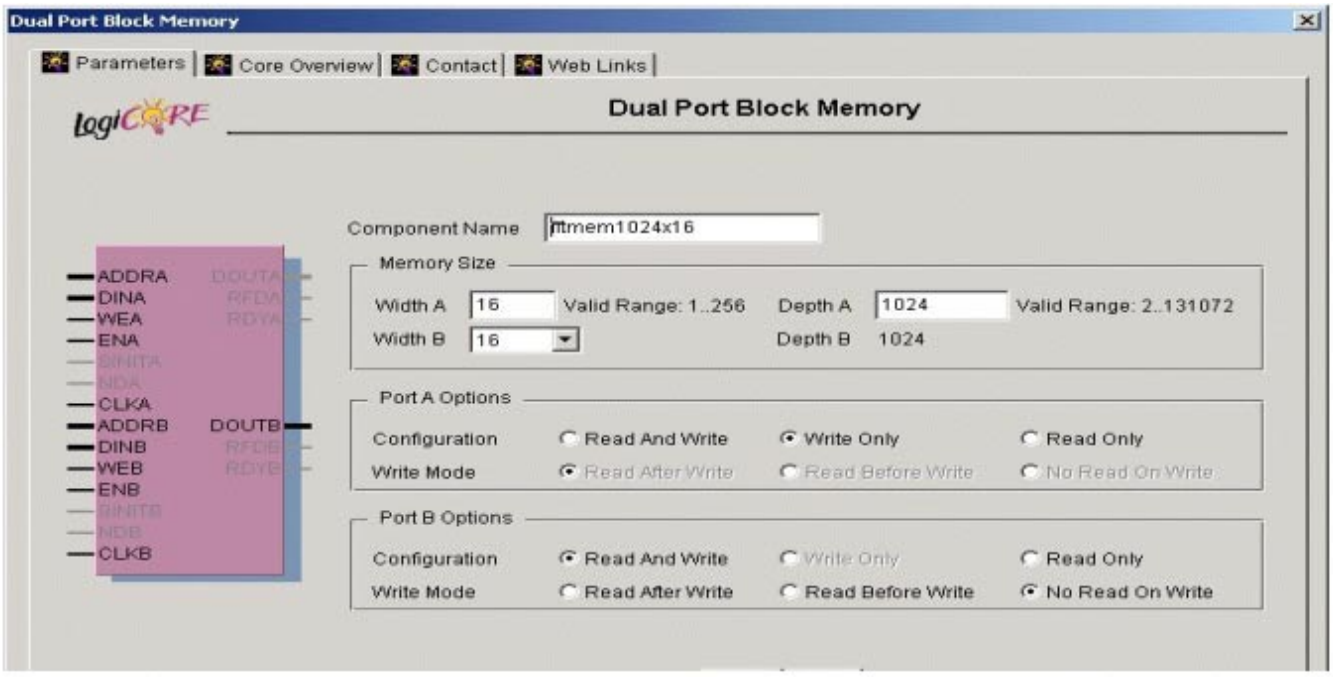


FIGURE 27 -- CoreGen Block fftmem GUI page 1

Memory Size: This is determined by the FFT Block and is 1024 by 16 bits.

Port A Options: Port A of the dual port memory is used by the FFT CoreGen block to write intermediate values back into the RAM when the block is being used as working memory. This port is write only.

Port B Options: Port B is used for both read and write. The FFT CoreGen block reads values from this Port to perform the FFT in the working mode, but when in the input mode the Port B is used to write the new block of data values into RAM. This Port is Read and Write, with the no read on write option selected.

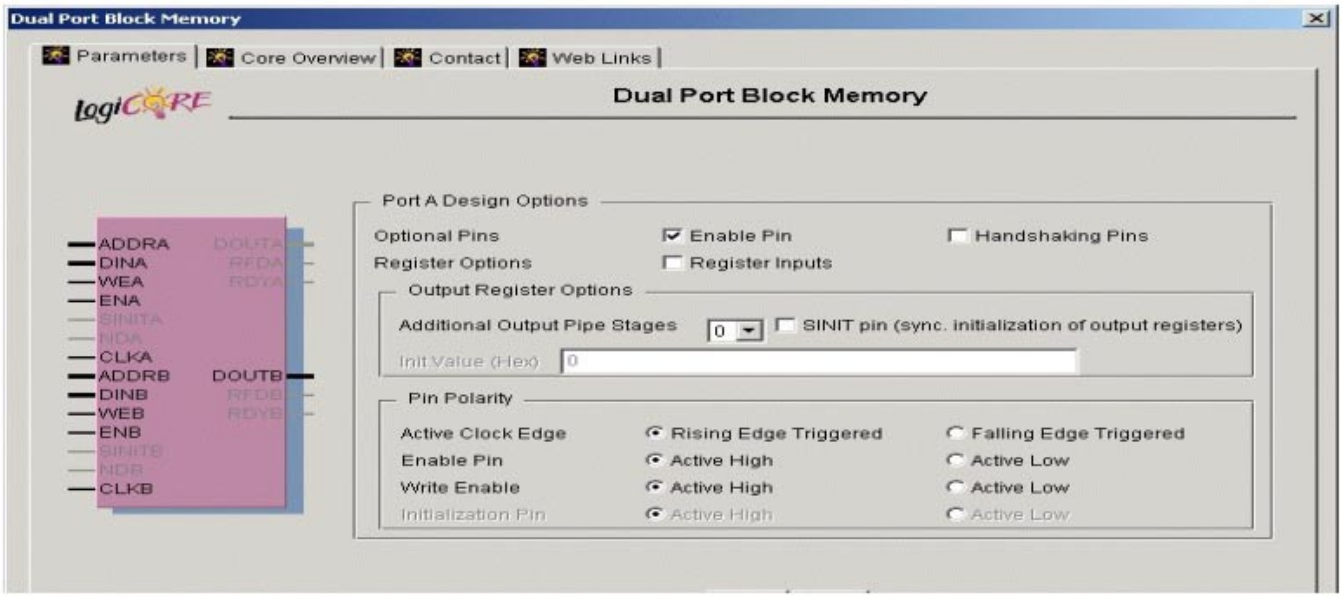


FIGURE 28 -- CoreGen Block FFTMEM GUI page 2

Port A Options: These are defined by the requirement of the FFT CoreGen block and an enable pin is required.

Pin Polarity: Rising clock edge triggered and both the ‘enable’ and ‘write enable’ with active high have been selected.

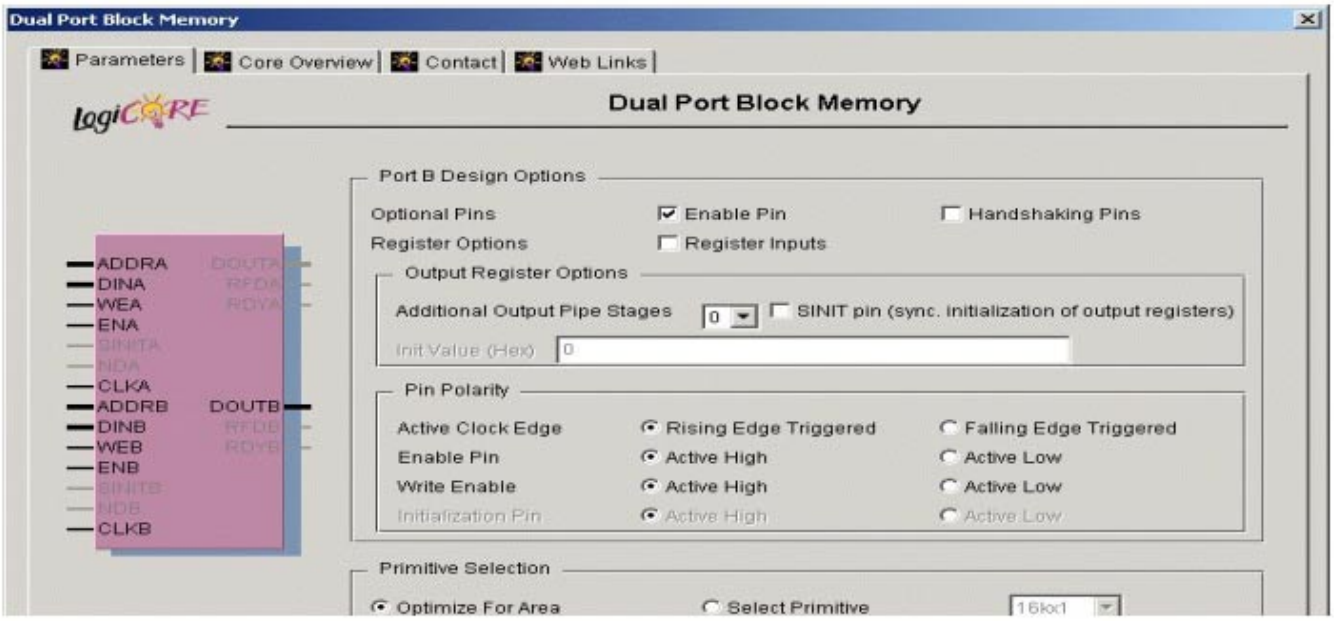


FIGURE 29 -- CoreGen Block FFTMEM GUI page 3

Port B Options: Only the ‘enable’ pin option has been selected.

Pin Polarity: As with Port A active rising edge clock and active high for the ‘enable’ and ‘write enable’ has been selected.

Primitive Selection: Optimise for area selected.

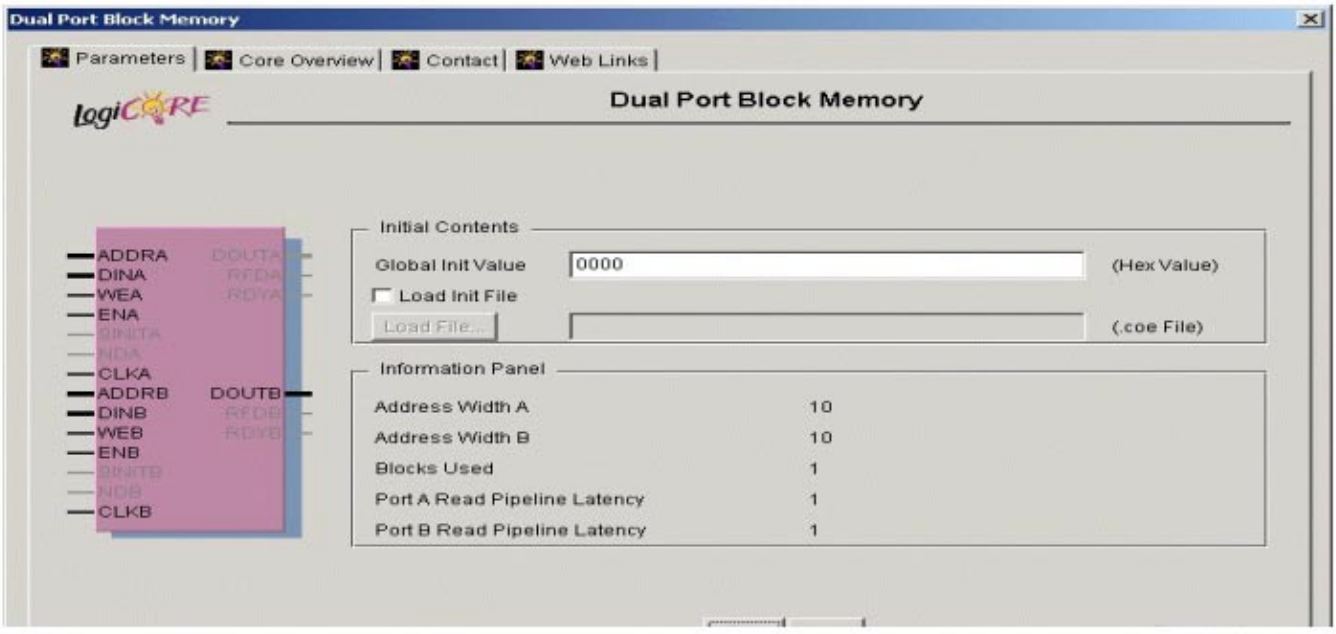


FIGURE 30 -- CoreGen Block FFTMEM GUI page 4

Initial Contents: The memory has an initial value of zero.

FFTMEMZ

This memory is used for the output values from the FFT, two blocks are used one for the Phase and the other for the Quadrature component.

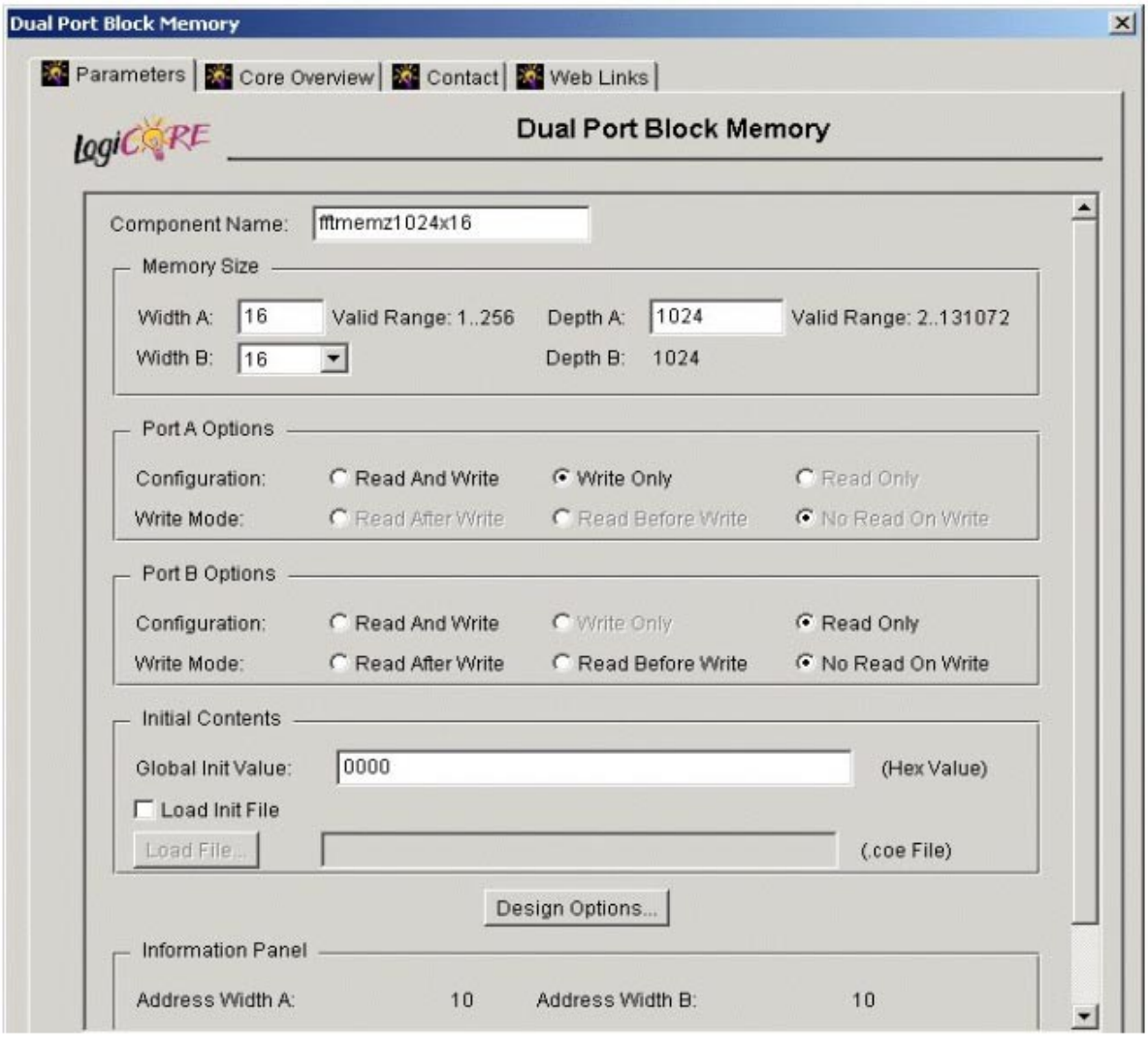


FIGURE 31 -- CoreGen Block FFTMEMZ GUI

Memory Size: Both port A and B are set up to match the Xilinx FFT CorGen block requirement of 1024 by 16 bits.

Port A Options: Port A is write only as this is the port that the FFT CoreGen block uses to output the results into the memory.

Port B Options: Port B is read only as this port is for the user to read the results from the memory.

Initial Contents: The contents of this memory block is initialised to zero.

VFFT

This is the 1024point complex FFT/IFFT CoreGen block. In this example it is used in the Triple-Memory-Space (TMS) configuration which allows the FFT core to be supplied with data 100% of the time and no potential computation cycles are wasted. This requires two banks of input/working memory each with both Phase and Quadrature components, the FFTMEM CoreGen blocks have been used for these areas of memory. The output memory, 'Output memory Z' is the same size as the input/working memory blocks but is configured differently, the FFTMEMZ coregen blocks have been used for this area of memory. Two FFTMEMZ blocks are required, one for the Phase and the other for the Quadrature compoment.
The VFFT CoreGen block does require the loading of the input data and the reading of the results to be synchronised to the core operation.

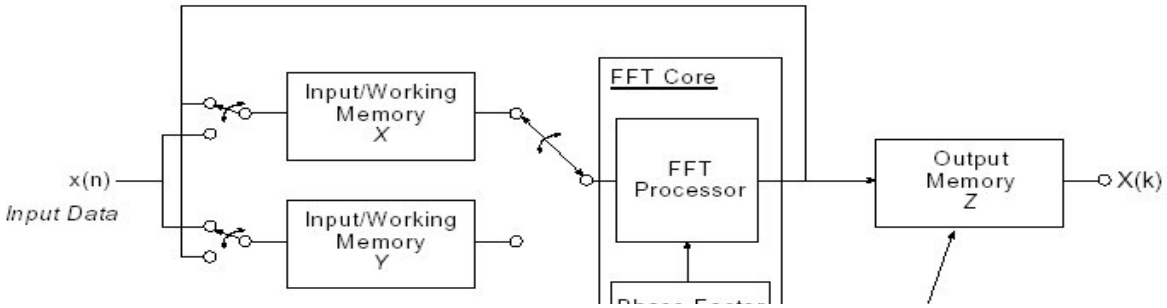


FIGURE 32 -- Block diagram of TMS structure

The VFFT core does not have any generate options other than the FPGA device that it is going to be fitted into. Once the core has been generated, how the core is connected will determine which mode it is going to operate in, figure 32 shows the connections required for a TMS configuration.

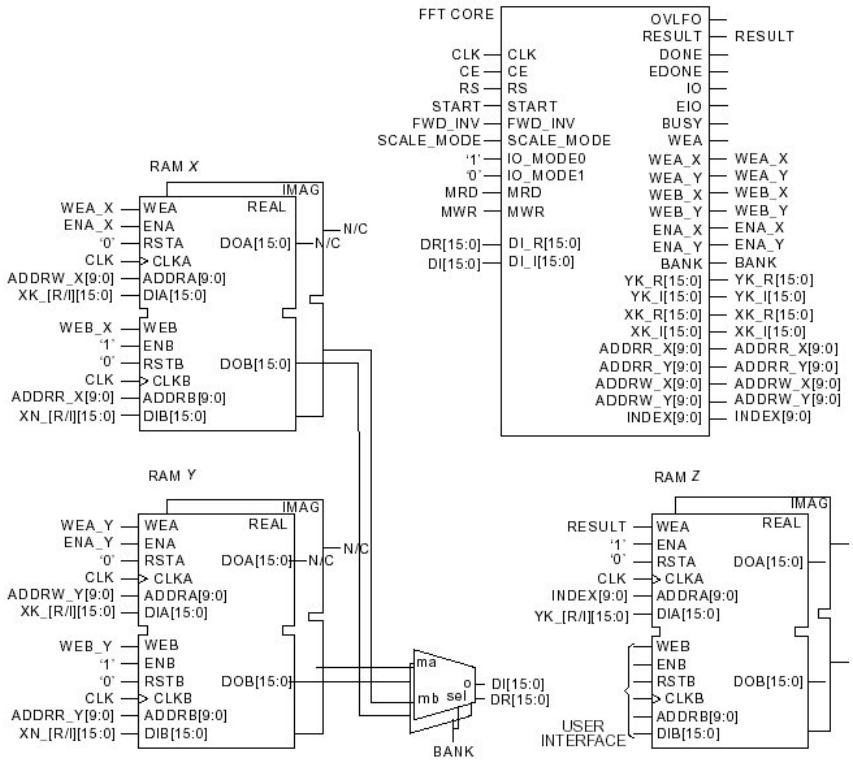


FIGURE 33 -- Block diagram of TMS connections

FIFO1023X32

This fifo block is used buffer the data between the output of the FFT output memory to the HERON output interface.

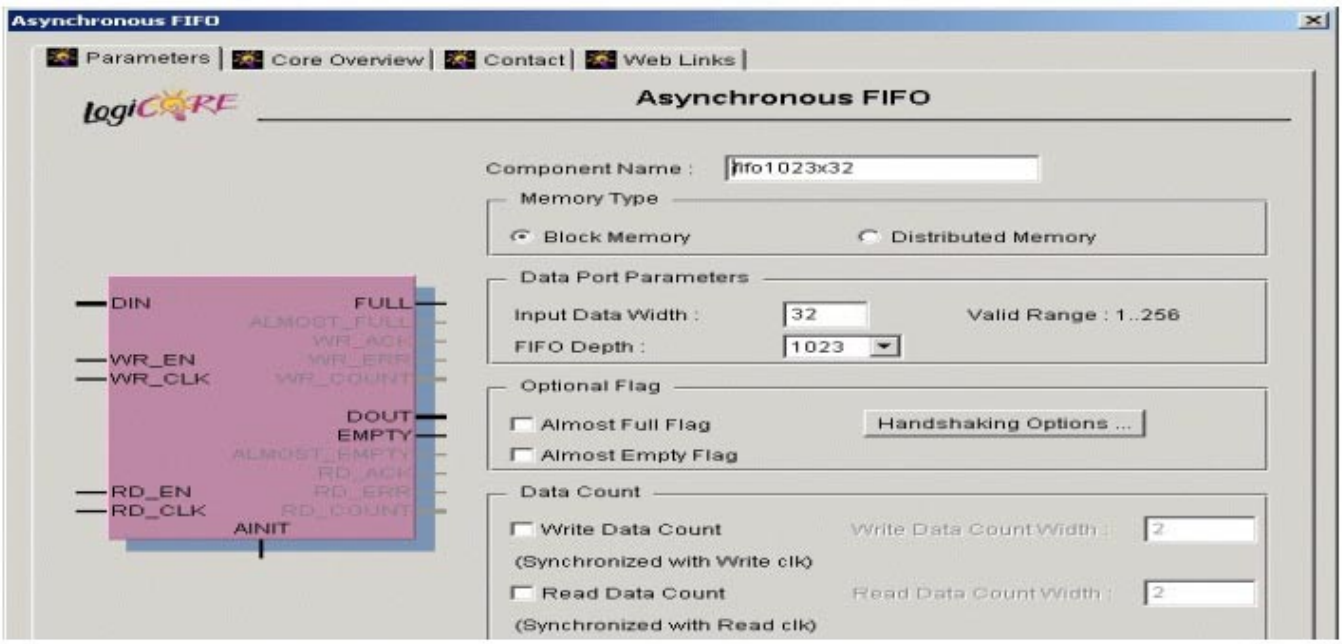


FIGURE 34 -- CoreGen Block fifo1023x32 GUI

Memory Type: The memory type selected for the fifo is block memory.

Data Port Parameters: The input data width is 32 bits. The complex output data from the FFT has 16 bit Phase data, which is fitted into the least significant half word, and 16 bit Quadrature data which is fitted into the most significant half word. The fifo depth selected from the options available is 1023 words.

Optional Flag: Almost full and almost empty are not selected.

Data Count: The read and Write data count options are not selected.

MULT

This CoreGen block is used to generate the squares of both the Phase and Quadrature components of the output of the FFT so that the magnitude squared can be calculated for output to DAC-A on the HERON IO2V2.

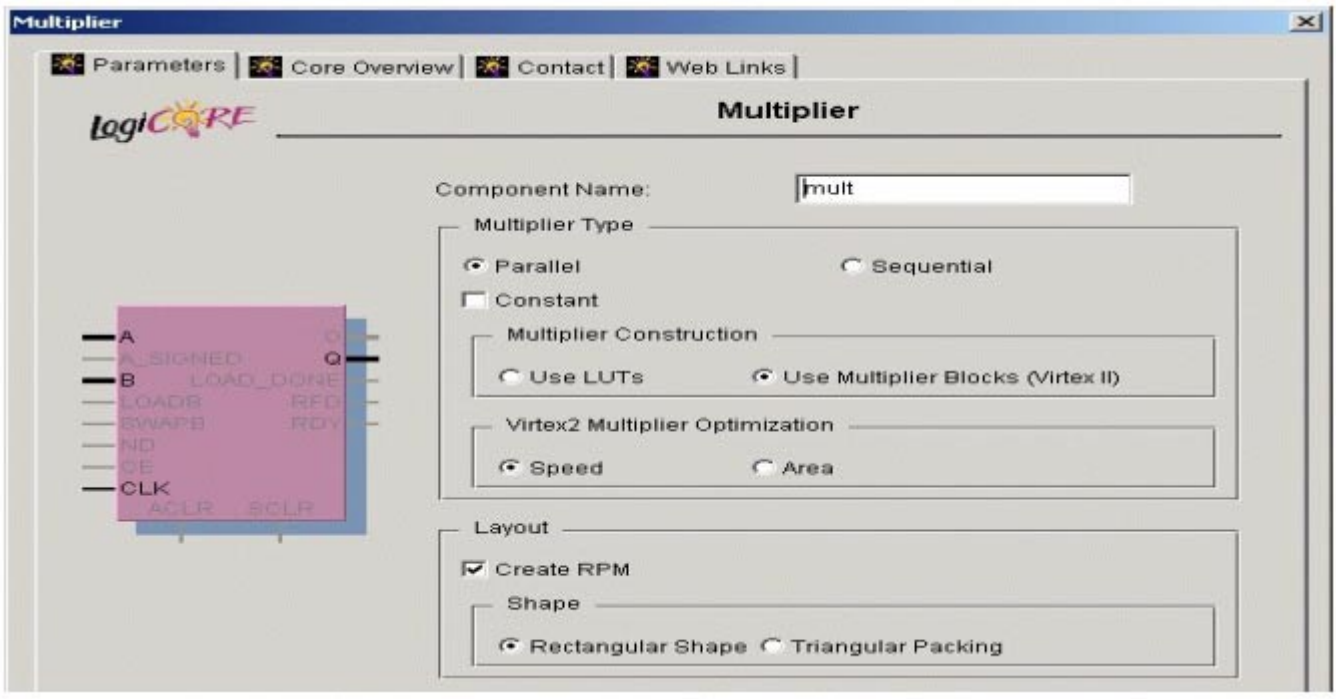


FIGURE 35-- CoreGen Block MULT GUI Page 1

Multiplier Type: The multiplier type selected is parallel as the rate required is one every 100MHz clock cycle. The multiplier blocks in the Virtex II have been selected to implement this multiplier, and the optimisation selected is for speed.
Layout: Create RPM has been selected with a rectangular shape.

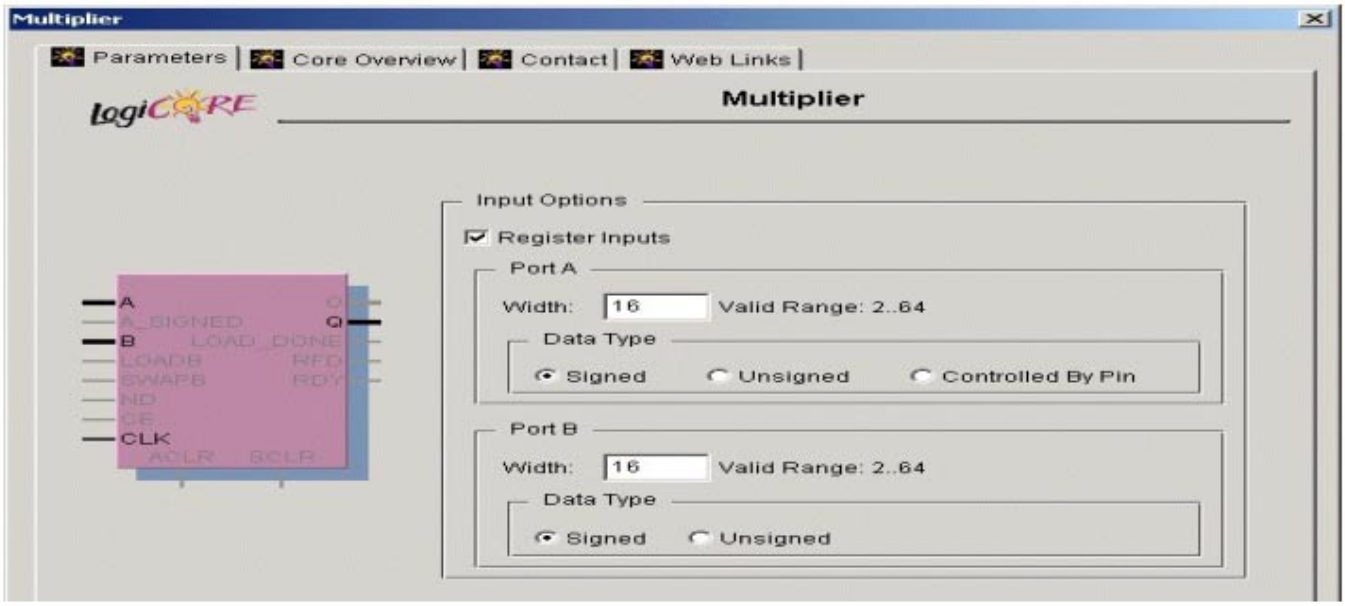


FIGURE 36 -- CoreGen Block MULT GUI Page 2

Input Options: Both the inputs to the multiplier are registered, and are set up identically for 16 bit signed inputs.

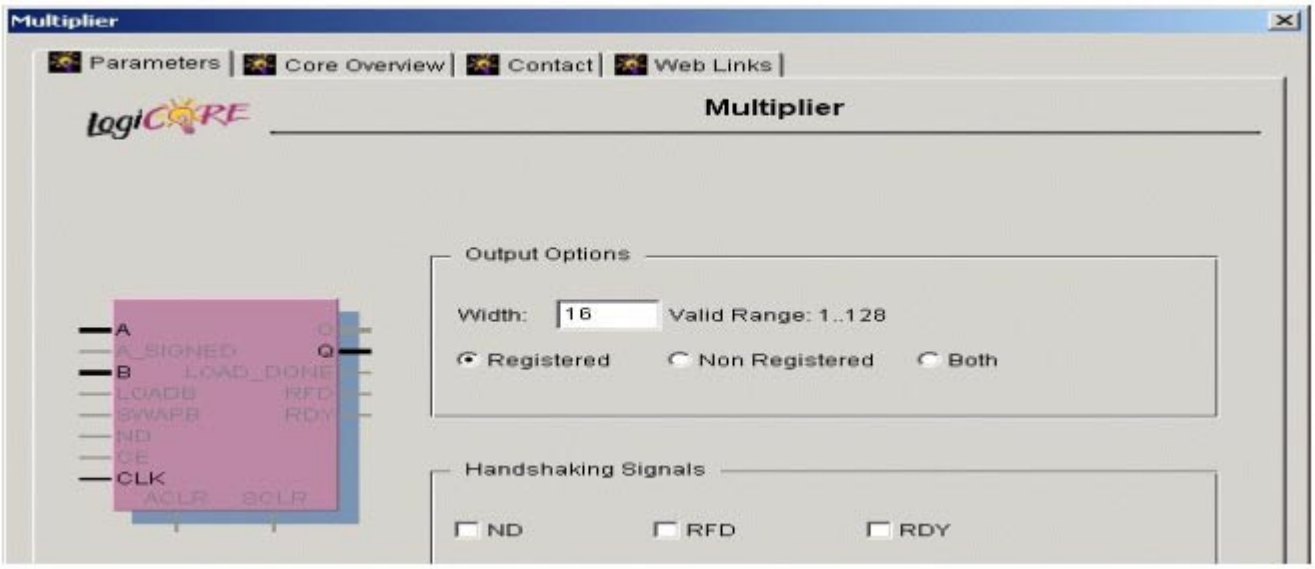


FIGURE 37 -- CoreGen Block MULT GUI Page 3

Output Options: The output width selected is 16 bits, and the output is registered.

Handshake Signals: No handshake signals have been selected as the multiplier will be multiplying at a rate of one every 100MHz clock cycle.

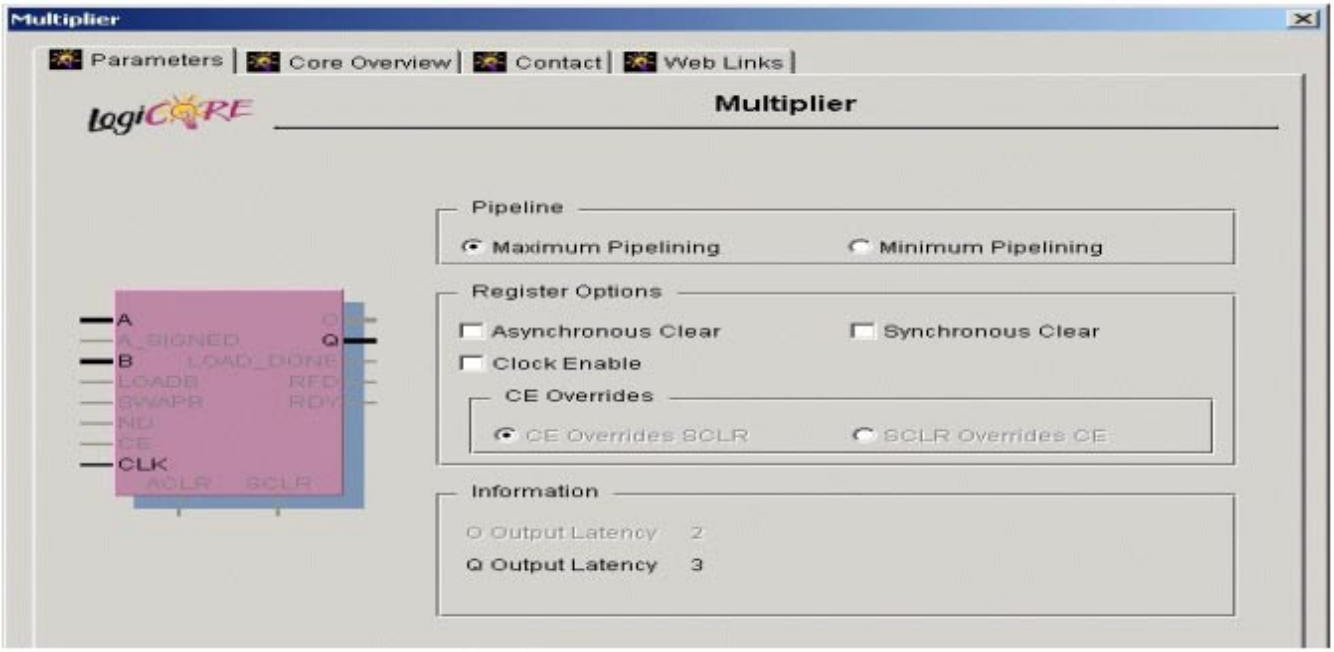


FIGURE 38 -- CoreGen Block MULT GUI Page 4

Pipeline: Maximum pipelining has been selected because of the speed at which the mutiplier is to operate. Increasing the pipelining tends to increase the speed at a cost of increased output latency.

Register Options: No register options have been selected.

ADDER

This CoreGen block is used combine the squared Phase and Quadrature components from the output of the FFT to produce a squared magnitude signal that is output on DAC-A.

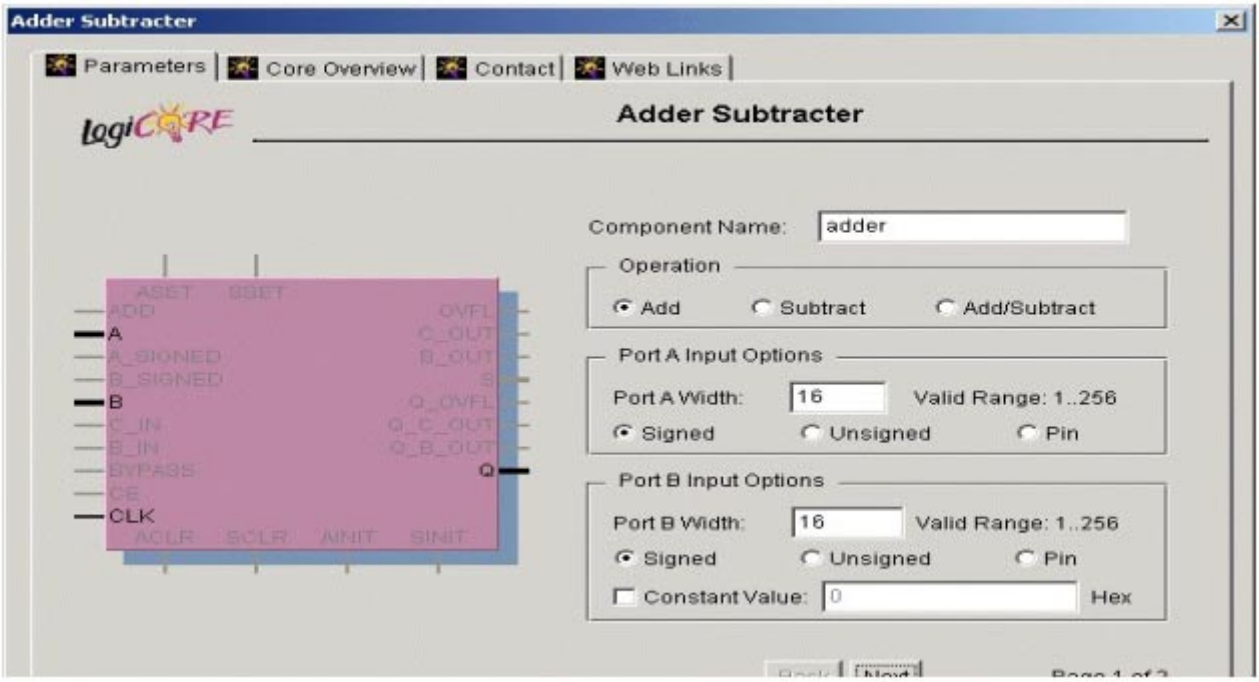


FIGURE 39 -- CoreGen Block ADDER GUI Page 1

Operation: The operation required of this CoreGen block is an adder.

Port A Input Options: The port A input width is 16 bits to match the output of the MULT CoreGen block output. Signed has been selected even though the output of the MULT block is a square and so should always be positive.

Port B Input Options: The port B input width is 16 bits to match the output of the MULT CoreGen block output. Signed has been selected even though the output of the MULT block is a square and so should always be positive

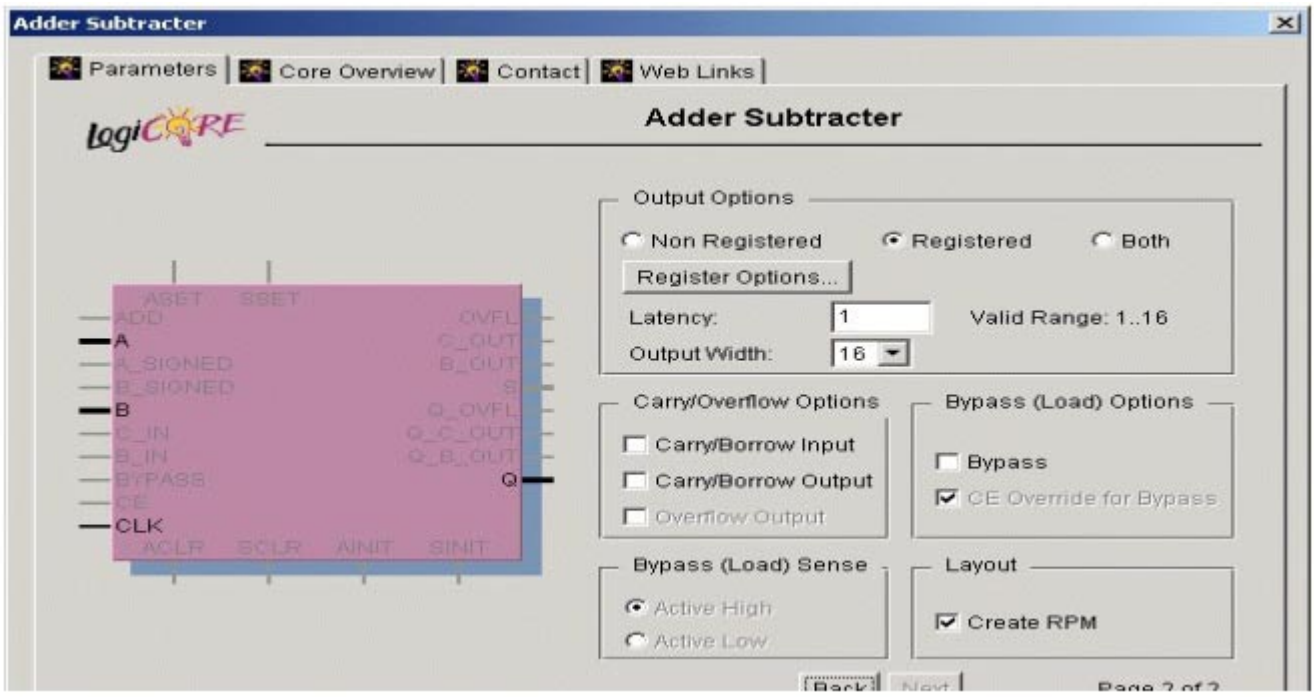


FIGURE 40 -- CoreGen Block ADDER GUI Page 1

Output Options: The options selected for the output of the adder block is registered with a latency of 1 and an output data width of 16 bits.