XAPP529 (v1.3) May 12, 2004

Connecting Customized IP to the MicroBlaze Soft Processor Using the Fast Simplex Link (FSL) Channel

Author: Hans-Peter Rosinger

Summary

MicroBlaze™ has the ability to use its dedicated FSL bus interface to integrate a customized IP core into a MicroBlaze™ soft processor-based system. This document describes possible methods to include customized IP cores into a Soft Core Processor (SCP)-based design.

The FSL interface is described in great detail, and a reference application involving a 1-dimensional Inverse Direct Cosine Transform (IDCT) is used to show how the implementation of a customized core can be done in software and hardware. The first part of this document deals with the different methods of integrating user IP cores into a soft processor-based system. The second part contains a short overview on MicroBlaze and the FSL interface. After that, the reference design, which can be downloaded from the Xilinx web site, is explained. The last point of this document contains the conclusion regarding the use of the FSL interface

Introduction

One advantage of a Soft Core Processor (SCP) is its flexibility: it uses only the processor features required for a specific application. Another advantage is its ability to integrate customized user Intellectual Property (IP) cores, which can result in a dramatic acceleration in software execution time due to algorithms being executed in parallel in hardware and not sequentially in software. MicroBlaze is a powerful and inexpensive SCP solution for the Virtex™ and Spartan™-II/3-based FPGA series. MicroBlaze combines all the flexibility advantages of SCP.

Generally, there are two ways to integrate a customized IP core into a MicroBlaze-based embedded soft processor system. One way is to connect the IP on the On-chip Peripheral Bus (OPB). The OPB is part of the IBM Core Connect™ on-chip bus standard. The second way is to connect the user IP to the MicroBlaze dedicated Fast Simplex Link (FSL) bus system. If the application is time-critical, the user IP should be connected to the FSL bus system; otherwise, it can be connected as a slave or master on the OPB. If the customized core is connected to the dedicated FSL interface, it is then possible to use predefined C functions to use the user core in the application software. This document deals primarily with the connection of a user IP on the MicroBlaze FSL interface. For more information regarding the connection of a user IP on the OPB, please refer to the user core template document:

www.xilinx.com/ise/embedded/edk_docs.htm

Integration of a User IP into a Soft Processor-Based System

There are different ways to connect a user IP into a soft microprocessor-based system. In general, every application can be realized and implemented either as software algorithm or as structural hardware. It is important to use the hardware implementation advantage (parallel execution), which allows the realization of strict timing-driven applications and the ability to control the user IP in software (e.g., C or C++). Figure 1 demonstrates how the parallel execution advantage can be used. The software routine needs 12 clock cycles to calculate the result G; however, in hardware it takes only 2 clock cycles to compute the same result.

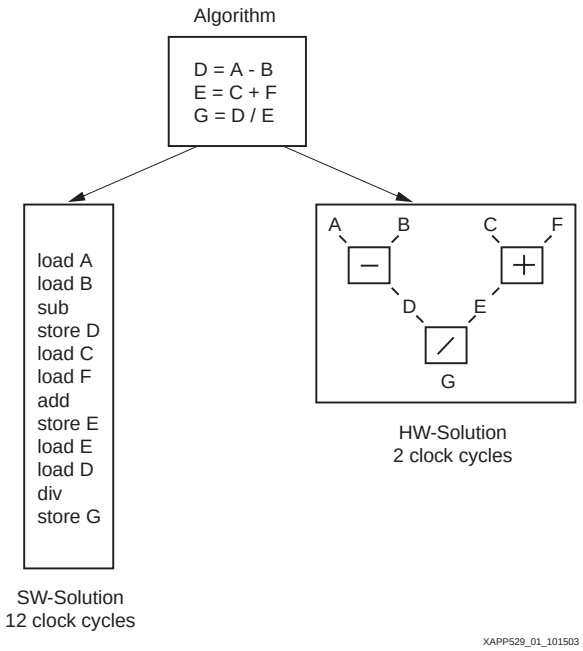


Figure 1: **Software versus hardware**

It is important to be able to use the customized IP core in the application program. Without an easy way to control the user IP, it doesn't make sense to include the core. One possible way is to integrate the user IP as an on-chip bus system (like the core connect bus). During the design stage of the core, the designer has to take the bus standard into account; that is, the designer has to make sure that he or she conforms to the bus specification if the core gets connected to the bus. This can be very time-consuming, a liability which no engineer can afford today. In some cases, templates, for example, the Xilinx IPIF user core template, exist. These templates make it easy and fast to connect an IP core on the Core connect bus. The next and more critical drawback to connecting the user IP to an on-chip bus is that most of the time the bus protocol overhead takes too much time and the speed advantage gets lost. Therefore, other different ways to include a customized user IP core are possible. One is to integrate the user IP as co-processor (if the processor has such a co-processor interface). If the soft processor has a special dedicated interface like the ARC Tangent, Tensilica, NIOS, or MicroBlaze soft processor, it is also possible to integrate a user IP. A soft processor is available as HDL source code or as a structural netlist. Therefore, it can be integrated into an ASIC or a FPGA.

Soft Processors Targeting ASIC Versus FPGA

It is important to understand the advantage of the flexibility made possible by using an FPGA design instead of an ASIC design. After the ASIC is manufactured, there is no way to reconfigure the logic inside the ASIC device. It would even be too costly to change the mask and manufacture a new ASIC. For a SCP, which targets the ASIC market, it is essential to be flexible in software. After the processor-based system with the customized instruction is mapped in an ASIC, it is possible to change the application only in software (changing the C-Code). Figure 2 shows an example on this. The first bar shows the execution time of a whole software program. Using a SCP with customized instructions reduces the overall execution time of the software program dramatically. If, for instance, the ASIC already exists but an aspect of the application changes for one customized instruction, then some modifications must be done.

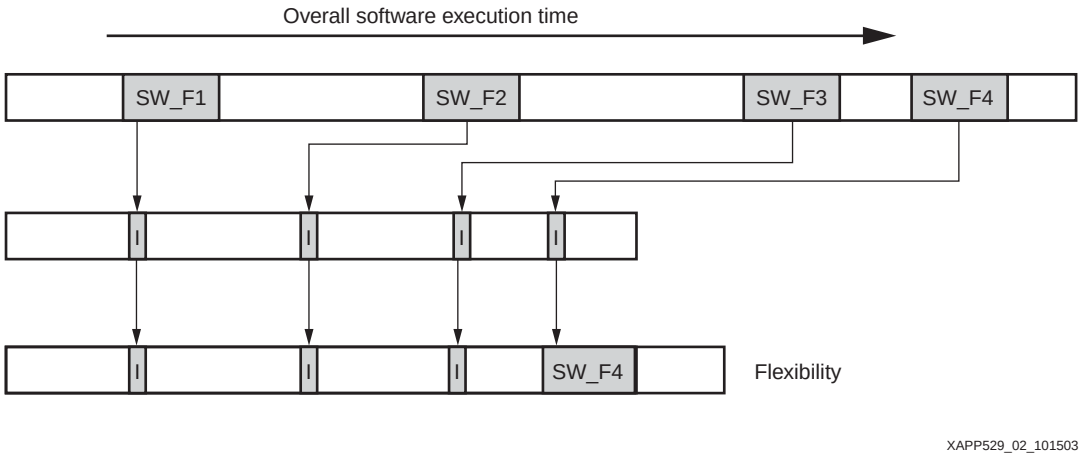


Figure 2: **Increasing software execution speed**

In an ASIC design, the only way is to not use the customized instruction but to address the new application requirement in software. Now, the overall software execution time will increase again; however, for an ASIC solution, it is the only way. Soft processors such as the ARC Tangent, the Tensilica, and the NIOS soft processor integrate the customized instruction completely in their execution unit and are useful if the target hardware is not changeable (ASIC).

FPGA devices instead allow for the reconfiguration of the internal logic very easily, quickly, and cheaply. Even in the last stages of the design, it is still possible to easily change the hardware inside the chip. If we look at the above example again, it is possible to change the new application requirement at any stage of the design in hardware, and it is not necessary to do the change in software. It is not that important to have the flexibility in software, because the flexibility in hardware has not been lost. For FPGA designs, it is not necessary or useful to include the customized user IP in the instruction set and inside the processor core, the RISC architecture. The next section details problems encountered when the customized IP core is included in the RISC architecture.

MicroBlaze FSL Interface Versus Customized Instruction

The integration of a customized IP core within the execution unit is very restrictive. One of the biggest restrictions is due to the nature of RISC processor architecture itself. Figure 3 shows a usual RISC processor architecture. Modern RISC architectures have a two-input and a one-output execution unit (ALU). Applications that require more than two input values and more than one output value are not optimal for these architectures, and several instructions have to be generated. Custom packet processing applications, for instance, require a lot of different dynamically changeable inputs (mask bits) and outputs. Those applications are not suitable for customized instructions because it is possible to use only two inputs (usually 64 bits) and one output (usually 32 bits).

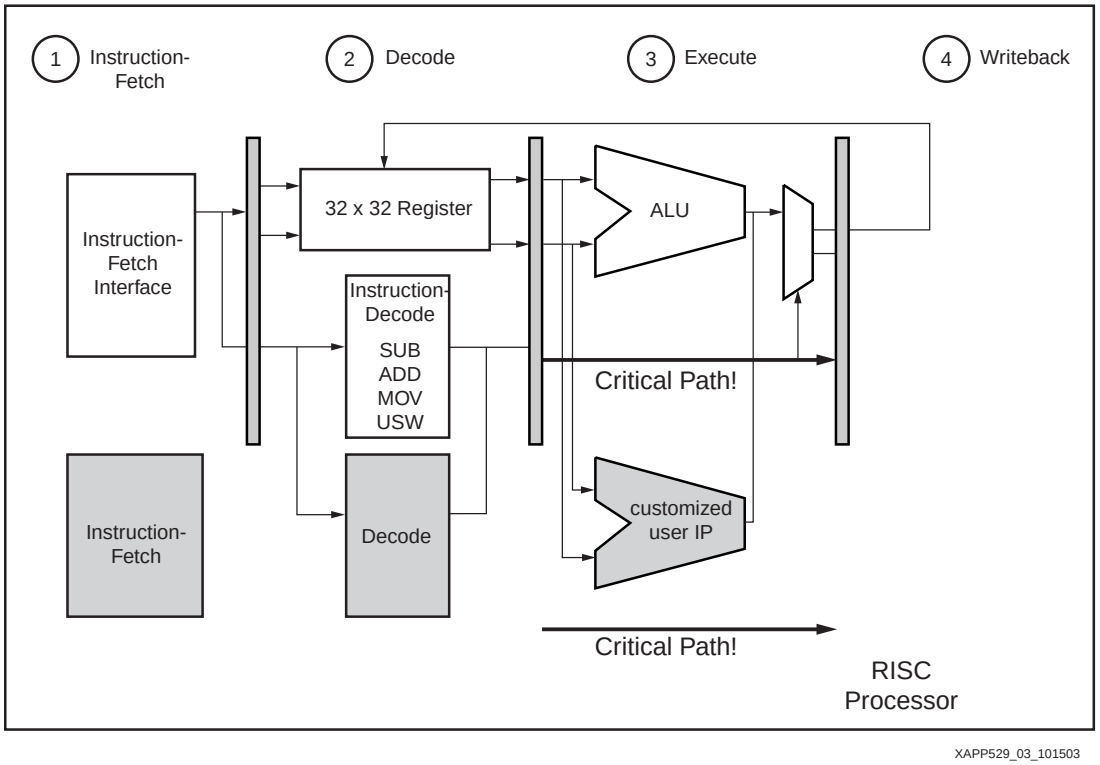


Figure 3: **Including a customized IP within the RISC architecture (Customized Instruction)**

Another bottleneck is the customized instruction itself. If the critical path of the whole system is through the user IP, the whole soft processor will decrease in performance (processor frequency), because the user IP is included within the soft processor architecture itself. If the RISC architecture doesn't allow the designer to stall the pipeline, the processor can't run at a higher frequency than the critical path would allow. The bigger the customized IP is, the more the designer must be careful not to decrease the whole processor performance. It is even not acceptable to cascade logic within one customized instruction, and several customized instructions have to be built. The software integration of customized instruction can't be handled directly from the compiler, thus the user has to use inline assembly to work with them. The customized instructions have to be implemented in software as inline assembler code and inline pragmas. This could produce a C application code, which is neither very clean nor portable.

Xilinx provides, with the MicroBlaze soft processor and the dedicated FSL interface, a very powerful, easy and flexible way to implement a customized user IP. Regarding the I/Os of the core, it is possible to use more than 2 dynamic inputs and more than 1 output because up to 16 FSL interface busses are provided. The user can use 8 inputs to the customized IP core and 8 outputs. Figure 4 shows the basic idea of connecting the customized user IP via the FSL interface onto the MicroBlaze. It is possible to provide the customized user IP core with many more inputs/outputs from another processor or external logic, and the big advantage is it is not necessary to change or extend the MicroBlaze core or the RISC architecture itself.

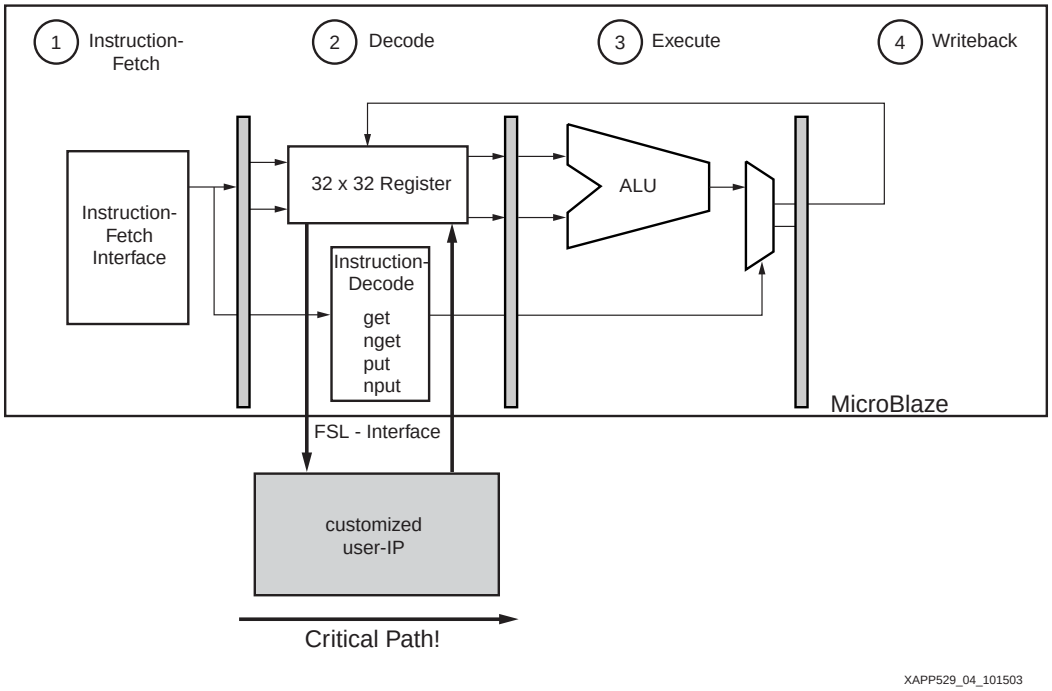


Figure 4: **Including a customized IP via the FSLinterface onto MicroBlaze**

Regarding the maximum frequency of the customized user IP, it won't decrease the clock frequency of MicroBlaze, because it is independent and the user core doesn't affect the internal MicroBlaze RISC architecture. It is no problem that the customized user IP is a subset of several IP cores that are cascaded together. The fact that the customized user core is implemented outside the processor architecture itself brings another advantage. If, for instance, the core takes 100 clock cycles to calculate a complex result, MicroBlaze can execute in the meantime a different application code and doesn't have to wait for the 100 clock cycles. The integration of the hardware in software doesn't require inline assembled code because the FSL interface has predefined C-macros that can be used for sending parameters to the hardware unit and to receive the result. Another powerful usage of the FSL is inter-processor communication. Two MicroBlaze processors have a very fast and clean way to communicate with each other. In the following sections, first MicroBlaze and then the FSL interface are discussed in greater detail. One example shows how a 1-dimension IDCT core gets connected in hardware within the EDK – XPS system builder and how to integrate the core in software.

General Description of the MicroBlaze Soft Processor

MicroBlaze is a standard 32-bit RISC Harvard-style Soft Processor, which is especially developed for the Virtex and Spartan-II/3-based FPGA architecture. The 32 by 32-bit registers are lookup table (LUT) RAM based. It guarantees a very short register access time. For memory, either the on-chip block RAM or off-chip memory can be used. The access time to the on-chip block RAM is minimal because there are dedicated routing resources to access them. Due to the fact that MicroBlaze is using the available FPGA resources very efficiently, it is possible to clock MicroBlaze up to 150 MHz. Thus, up to 125 Dhrystone MIPS can be reached. It is consequently the industry's fastest SCP for FPGAs. The MicroBlaze SCP can be customized for any application. Its barrel shifter, divide unit, data cache, instruction cache, and the FSL bus system are optional. The sizes of the caches are configurable from 2 to 64 Kbytes. Standard peripherals are provided as well and are Core Connect compatible. Consequently, they can be integrated in an embedded design very easy. These peripherals are either free, such as the memory controller, UART, interrupt controller, and timer, or commercial cores such as the Ethernet controller, gigabit Ethernet controller, PCI, HDLC, etc. All commercial IP Cores can be evaluated. For all free cores, the VHDL and the C-Code (TCP/IP stack) are readable. MicroBlaze is used in different areas such as

network applications, telecommunication applications, control, and consumer markets. Figure 5 shows a typical MicroBlaze SCP with its peripherals.

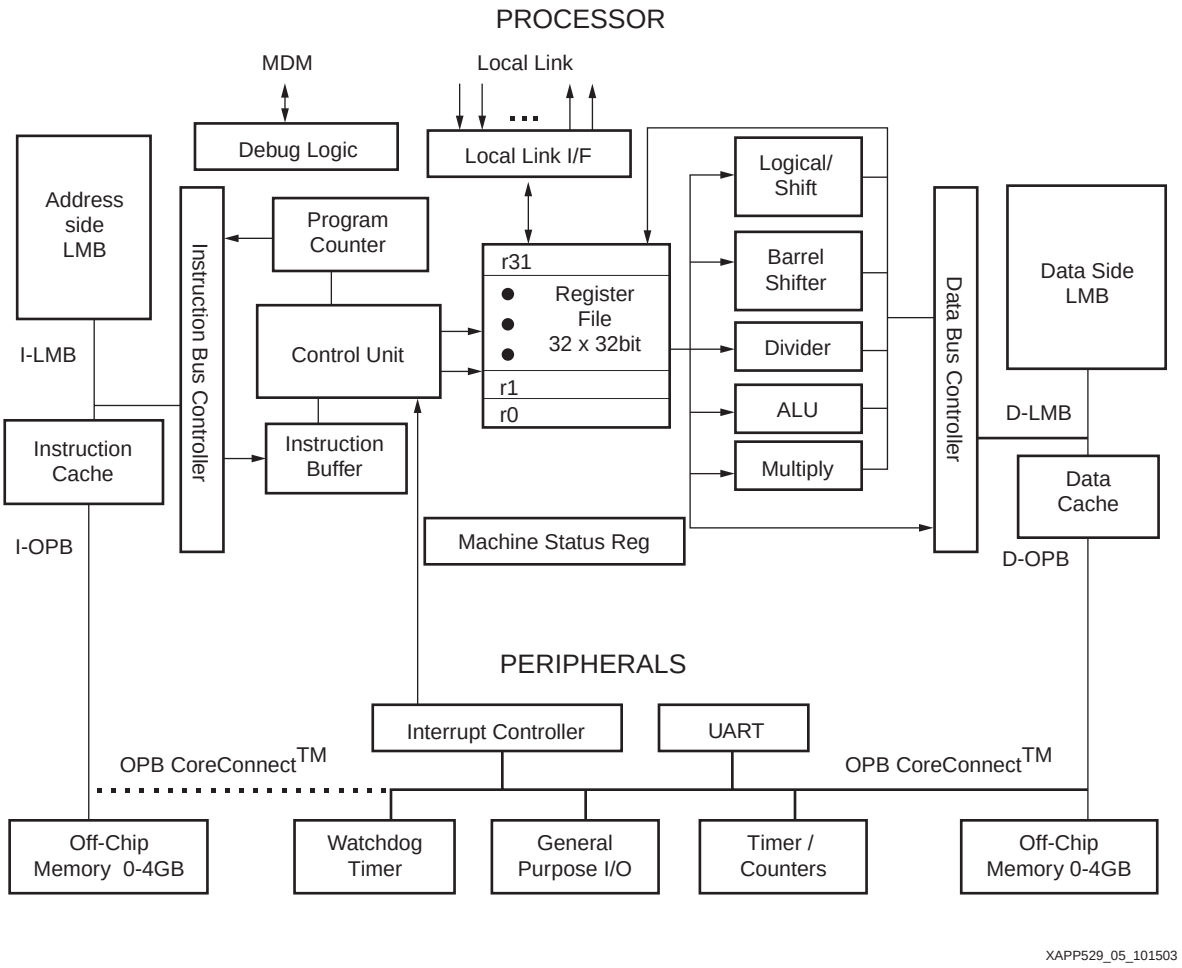


Figure 5: **MicroBlaze-based embedded processor system**

The Embedded Development Kit (EDK) includes the soft processor core and a standard set of peripherals and is available from Xilinx and its distribution partners. The kit includes a complete set of GNU-based software tools including the compiler, assembler, debugger, and linker. Variations of the kit include development boards that support the Virtex-E, Virtex-II, Virtex-II Pro, Spartan-II, Spartan-IIE, and Spartan-3 series of FPGAs.

For more information regarding MicroBlaze, please refer to the following link:

http://www.xilinx.com/ipcenter/processor_central/microblaze/index.htm

Detailed Description of the FSL Interface

This section describes the special FSL interface in detail. MicroBlaze contains eight input and eight output FSL interfaces. The FSL channels are dedicated unidirectional point-to-point data streaming interfaces. The FSL interfaces on MicroBlaze are 32 bits wide. Further, the same FSL channels can be used to transmit or receive either control or data words. A separate bit indicates whether the transmitted (received) word is control or data information. The performance of the FSL interface can reach up to 300 MB/sec. This throughput depends on the target device itself. The FSL bus system is ideal for MicroBlaze-to-MicroBlaze or streaming I/O communications.

The main features of the FSL interface are:

- Unidirectional point-to-point communication

- Unshared non-arbitrated communication mechanism
- Control and Data communication support
- FIFO-based communication
- Configurable data size
- 600 MHz standalone operation

The FSL bus is driven by one Master and drives one Slave. Figure 6 shows the principle of the FSL bus system and the available signals.

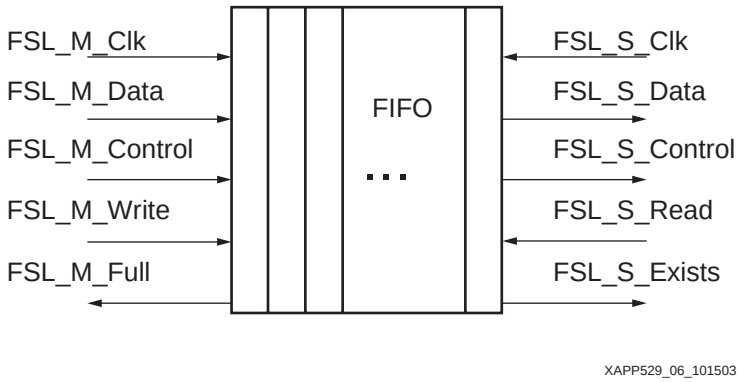


Figure 6: **FSL interface**

FSL peripherals may be created as a Master or a Slave to the FSL bus. A peripheral connected to the master ports of the FSL bus pushes data and control signals onto the FSL. All peripherals that act as a master to the FSL bus should create a bus interface of the type MASTER for the bus standard FSL in the Microprocessor Peripheral Description (MPD) file. A peripheral connected to the slave ports of the FSL bus reads and pops data and control signals from the FSL. All peripherals that are a slave to the FSL bus should create a bus interface of the type SLAVE for the bus standard FSL in the MPD file. The put and get instructions of MicroBlaze can be used to transfer the contents of a MicroBlaze register onto the FSL bus and vice-versa. The FSL bus configuration of MicroBlaze can be used in conjunction with any of the other bus configurations. Below is a brief overview of the FSL-related predefined C-functions available in EDK.

```
// Blocking Data Read and Write to Local Link no. id
microblaze_bread_datafsl(val, id)
microblaze_bwrite_datafsl(val, id)

// Non-blocking Data Read and Write to Local Link no. id
microblaze_nbread_datafsl(val, id)
microblaze_nbwrite_datafsl(val, id)

// Blocking Control Read and Write to Local Link no. id
microblaze_bread_cntlfsl(val, id)
microblaze_bwrite_cntlfsl(val, id)

// Non-blocking Control Read and Write to Local Link no. id
microblaze_nbread_cntlfsl(val, id)
microblaze_nbwrite_cntlfsl(val, id)
```

For more detailed information regarding the FSL bus information, please refer to the FSL bus data sheet (containing timing diagrams) and to the MicroBlaze user guide.

Description of the Application

As an application to demonstrate the use of the FSL interface, a 1-dimension IDCT is used. This DSP application highlights very well the performance win that could be reached. A 1-dimension IDCT realized in software would require a high execution time because the C- program would consist mainly of loops which get executed sequentially by the processor. If the application is implemented as its own hardware module, the execution time requires much fewer clock cycles. The used 1-IDCT core on the FSL interface is an example and needs approximately 150 LUTs and the latency of 64 clock cycles. Please note this IDCT core is used to show how to implement a user core on the FSL interface. The software application writes 8 values from memory to the FSL. The IDCT core gets the data and calculates the result. When the result is available, MicroBlaze reads the data (8 words) back from the FSL. The IDCT core is connected to the FSL interface as shown in Figure 7.

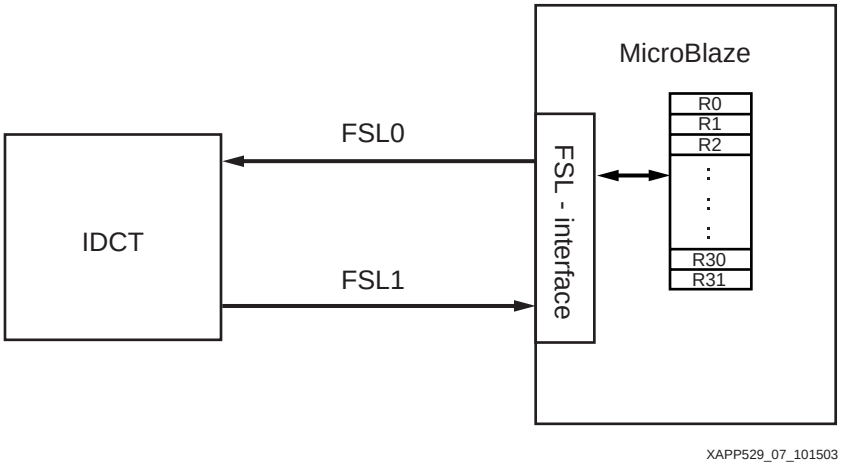


Figure 7: **Including the 1-dimensional IDCT IP via the FSL interface onto MicroBlaze**

For the FSL0 connection, the MicroBlaze is the Master on the FSL bus and the IDCT core is the Slave. Thus, MicroBlaze controls the data sent on the FSL0 bus to the IDCT core. For the FSL1 bus, it is vice versa, and the IDCT core is the Master and the MicroBlaze the Slave. The IDCT controls the data on the FSL1 bus.

By cascading the 1-dimensional IDCT core, it is possible to integrate a 2-dimensional IDCT core (Figure 8). The 1-D IDCT block will read from the FSL0 input and put the data out on the FSL1 bus. The corner-turn module also reads from FSL1 and puts out on FSL2. The last 1-D IDCT is also reading from the FSL2 and puts out the data on FSL3, which transfers the result back to MicroBlaze.

By doing this, the current 1-IDCT block can be used without any modification as a part in a 2-dimensional IDCT core. It also gives the user much more flexibility since it is possible to decide for another connection scheme at anytime.

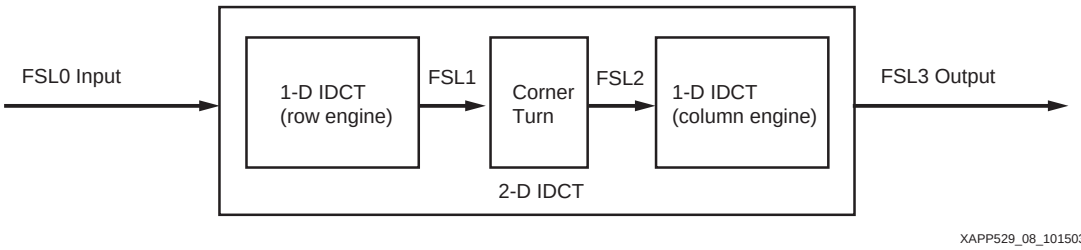


Figure 8: **Block Diagram for using a 1-D IDCT to implement a 2-D IDCT**

Integration in Hardware

This section describes how to integrate the user IP in a MicroBlaze-based embedded processor design. For this integration, the EDK 6.1 software tool is used. Figure 9 shows the embedded MicroBlaze design with the customized IDCT core and some OPB standard peripherals.

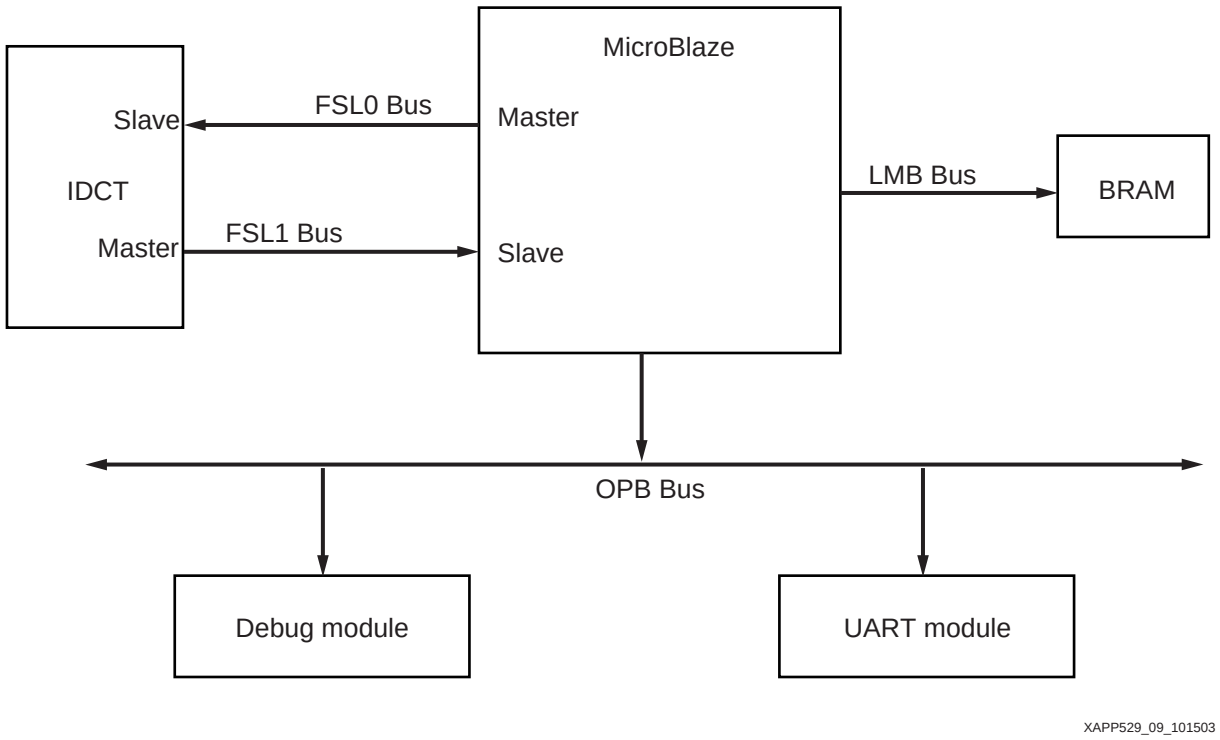


Figure 9: **Embedded processor system - hardware**

The whole embedded system consists of the MicroBlaze itself, two FSL bus systems, the user core, an OPB on-chip bus, two OPB peripherals (UART lite and the MicroBlaze Debug module), and the on-chip block RAM. The application program is stored in the on-chip block RAM.

Figure 10 shows in detail how the IDCT core is connected onto the MicroBlaze FSL interface. The IDCT is available in VHDL code. It is also possible to use a netlist instead.

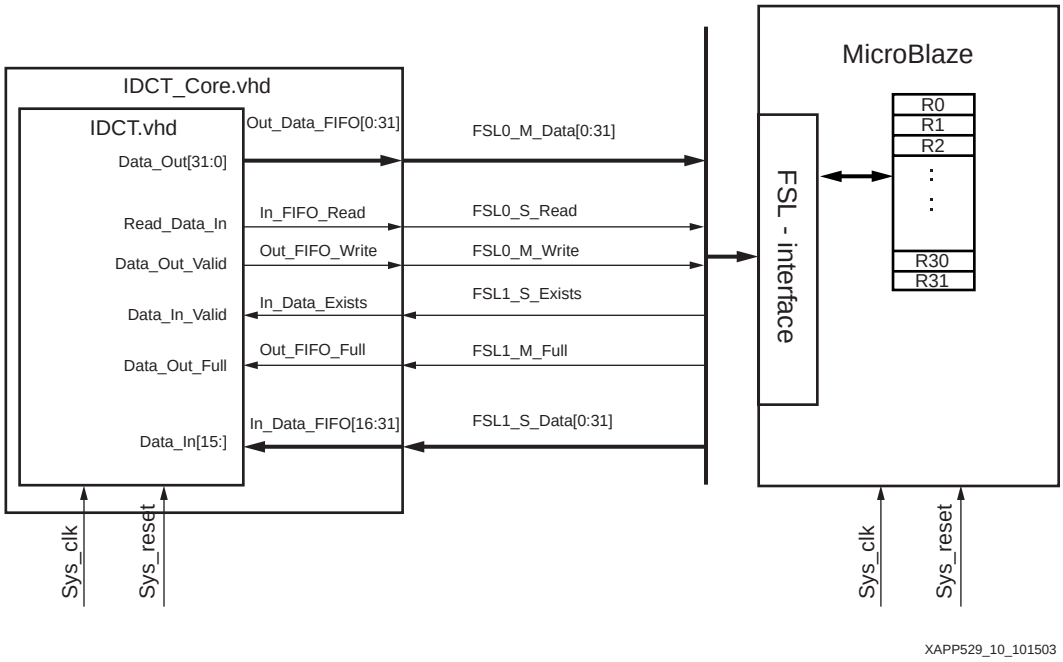
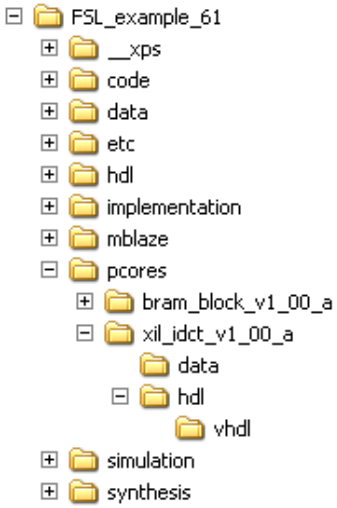


Figure 10: **Figure 10: Detailed connection of the 1-dimensional IDCT IP to MicroBlaze**

The control signals are provided from the FSL interface. External signals like the global system clock or the system reset can be integrated easily. In addition to the VHDL source code, a Microprocessor Peripheral Definition (MPD) file and a Peripheral Analyze Order (PAO) file are necessary. The MPD file defines the interface of the peripheral. The PAO file contains a list of HDL files that are needed for synthesis, and defines the analyze order for compilation. It is necessary to save all the files in a dedicated directory. The file structure in the XPS project should look like the following:



where the *xil_idct_v1_00_a/data* folder contains the MPD and the PAO file. The *vhdl* folder contains the VHDL source code of the user IP. If all the files are implemented correctly, the customized user core can be integrated in the Xilinx Platform Studio (XPS) and the bitstream of the hardware system can be generated.

Integration in Software

The next step is to integrate the user core into the software, the C application program. The application program is very simple and writes some data to the core and reads it back. The data block which will be written to the core consists of 8 input values. Before the next data block is written to the IDCT core, MicroBlaze waits for the resulting data block. Obviously, the resulting data block contains the 8 output values from the IDCT core. For writing into the user core, the predefined functions are used. For the example, the non-blocking write and read commands are used. The predefined functions are defined in the *mb_interface.h* file.

Verification of the Hardware

The verification of the hardware can be done in very different ways. The aim is to verify the FSL bus system and to be sure the data is transferred to the IP core and read back from the IP core correctly. It will be assumed that the customized IP core, in this case the IDCT core, already has the correct functionality. For the reference design, the verification was done with ModelSim 5.7e, and do script files are provided with the reference design. It can be seen from the output wave window that both the write to the core and the read from the core are successful.

Verification of the Software

To verify the software, the GNU debugger is used. The debugger can be started from XPS and is included in EDK. The *opb_mdm* debug module is used for the communication between MicroBlaze and the Xilinx Microprocessor Debugger (XMD) interface. On top of XMD, the GNU debugger GUI can be used.

Reference Design

The reference design targets the Memec Insight 2vp7 demo board (XC2VP7 -4, FG456 package). It has been implemented with the Xilinx EDK / ISE 6.2i software. The utilization values are completely device and implementation tool dependent. The total design requires 4 IOBs, 4 MULT18x18 elements, 4 RAMB16s and about 1300 slices, and the embedded soft processor design runs at a frequency of 100 MHz.

The MicroBlaze – FSL 1 dimension IDCT reference design can be downloaded from:

http://www.xilinx.com/bvdocs/appnotes/xapp529_6_1.zip

http://www.xilinx.com/bvdocs/appnotes/xapp529_6_2.zip

Conclusion

The MicroBlaze SCP with its powerful FSL interface can improve the performance of a whole application dramatically by outsourcing time-critical tasks into hardware. Besides the tremendous performance win, the solution is changeable until the last stage of the project by taking advantage of the flexibility of SCP and the FPGA architecture. By using customized instructions, the user is bounded to only two inputs and one output from the customized logic. With the FSL interface it is possible to have up to 8 inputs and 8 outputs, which allows much more flexibility, and cascaded logic within the customized core doesn't affect or lock the MicroBlaze RISC unit. The RISC architecture doesn't get manipulated and stays self-contained because it is not necessary to extend the processor RISC core. Predefined C functions are provided in EDK for integrating the customized user IP in a very easy and clean way in the C/C++ application program. If the target FPGA architecture is a Spartan-II, Spartan-IIE, or Spartan-3, it is even better, as these families are the most cost-effective solution that is available for high-performance embedded processor designs.

Revision History

The following table shows the revision history of this document.

Date	Version	Revision
11/30/03	1.0	Initial Xilinx release.
12/19/03	1.1	Corrected broken link.
3/16/04	1.2	Edit SCP to Soft Core Processor (1st use) and fix ref design link.
5/12/04	1.3	Edited links to reference designs.