COMPARISON BETWEEN FPGA CO-P ROCESSOR&TMS320C641X DSP FAMILY IN IMPLEMENTING DIF FFT ALGORITHM

Asst. Lecturer N. H. Abbas
Dept. of Elect. College of Eng.

Asst. Lecturer A. M. Ragib
Dept. of Mechatronic AL khawarizmy Eng. – University of Baghdad

ABSTRACT

The Decimation in Frequency Fast Fourier Transform (DIF FFT) is a computationally intensive digital signal processing function widely used in applications such as imaging and wireless communication. Historically, this has been a relatively difficult function to implement optimally in hardware, leading many software designers to use digital signal processors (DSPs) in soft implementations. Unfortunately, because of the function's computationally intensive nature, such an approach typically requires multiple DSPs within the system to support the processing requirements. This is costly from a device and board real-estate perspective as well as power intensive.

Field-programmable gate array (FPGA) co-processors have become an extremely cost-effective means of off-loading computationally intensive algorithms to improve overall system performance while reducing development time, cost and risks. This paper will describe two DIF FFT implementation approaches, one implemented as an FPGA co-processor and the other using only an external TMS320C641X DSP Family. It will then examine the advantages and disadvantages of these approaches from performance, cost, power consumption and ease of implementation perspectives.

الخلاصة

محول فورير السريع مقسم التردد (DIF FFT) كثيراً ما يستخدم في تطبيقات معالج الاشارة الرقمية مثل التصوير والاتصالات اللاسلكية. تاريخيا تكون هكذا وظائف صعبة البناء بصورة مثلى ككيان مادي ، لذلك تقدم عدد من مصممي البرامج من اجل استخدام معالج الاشارة الرقمية في بناء بسيط. ولكن لسوء الخط بسبب كون الوظائف المراد بنائها حسابيا مركزة لذلك نحتاج الى عدة معالجات داخل نفس النظام من اجل تحقيق متطلبات المعالجة. وهذا الشيء مكلف إضافة الى كونه يصرف قدرة اكثر .

المعالج المساعد الذي يستخدم ترتيب بوابة برمجة المجال (FDGA Co-processor) يستخدم من اجل تحسين أداء النظام بصورة شاملة ويقلل من زمن المعالجة والكلفة والمخاطر . هذه المنشورة تصف طريقتين لبناء محول فورير السريع مقسم التردد (DIF FFT) الاولى باستخدام ترتيب بوابة برمجة المجال (FPGA)

والثانية باستخدام معالج الاشارة الرقمية (TMS320C641X DSP Family). وبعدها نختبر فوائد ومساوئ كلا الطريقتين من ناحية الأداء ، الكلفة ، صرف الطاقة وسهولة البناء .

KEY WORDS
FFT, FPGA, DSPs, Simplest Implementation, Cost, H /W Parallelism, Power Consumption.

INTRODUCTION

FFTs are very common, computationally intensive signal processing functions found in a large number of signal processing systems. An example communication system that relies heavily on FFTs to perform a lot of its base signal processing is orthogonal frequency division multiplexing (OFDM). OFDM systems require large amounts of FFT computing ability to handle the required data rate of all transmit and receive channels passing through the network.

Typically, designers building OFDM transmitters and receivers have relied on DSPs as their device of choice for implementing these signal processing functions. DSPs typically come with a range of basic, assembly-optimized signal algorithms like FFTs and finite impulse response (FIR) filters, making these functions easier to implement compared to an ASIC or FPGA hardware-based approach. Unfortunately, the evolution in performance of DSPs has not been able to keep up with the demands of current and future communication system requirements. This forces the designer to implement arrays of DSPs simply to satisfy the data rates and vast number of channels needed by the system. A common challenge arising from this approach involves handling shared memory between the processors and ensuring that data does not get overwritten from one processor to another. These arrays also tend to take up a lot of board real estate and the large number of DSPs required can dramatically increase overall system cost.

An alternative approach to address this computational burden is to implement co-processors to speed up the computation of these functions using hardware accelerators. DSP vendors like Texas Instruments have started to add dedicated hardware co-processors to their DSP device offerings as seen in the C6416 DSP, which has dedicated on-chip Turbo and Viterbi co-processor hardware, primarily targeted at 3G wireless applications [TI 2001]. While this approach is beneficial for 3G applications, other users may not find these coprocessors particularly suitable for their needs. DSP vendors, on the other hand, are driven to implement co-processors suitable for a broad-based market that is relatively mature.

Altera's approach is to provide the user with the flexibility to implement their co-processor on an FPGA [Lim 2003]. These co-processors can be suitably designed to fit virtually any function or application the user is targeting owing to the flexible nature of the FPGA's device fabric. Additionally, the user is able to customize and construct their function in a way that fully exploits the parallel nature of a hardware implementation within the FPGA, enabling better channelization (useful in communication systems), and ultimately, greater data throughput.

The following sections of this paper serve to provide an understanding of the process of implementing an FFT algorithm in both a single DSP as well as an FPGA co-processor. Subsequently, both approaches will be evaluated and analyzed from the following design considerations and perspectives; ease of implementation, cost and performance, and power consumption. This comparison employs a TMS320C6416 DSP from Texas Instruments and a Stratix FPGA device from Altera, as well as their respective design tools and software.

IMPLEMENTING AN FFT ALGORITHM IN A DSP

DSP algorithms for functions like correlators, FFTs and FIR filters are generally supported by design tools and software that accompany the various DSP architectures. For example, Texas Instruments provides a selection of optimized, C-callable DSP library functions that is freely available for the C64x DSP family like DSP_fir_r8 and DSP_fft16x16r for FIR filtering and FFTs, respectively. These functions tend to run much faster than equivalent code written in ANSI C, since

they are hand-optimized by the processor vendor using assembly language for a targeted DSP architecture. These library functions are also preverified by the vendors, thereby reducing overall system development time and allowing the developer to focus on improving and differentiating their system [TI oct.2003].

Typical performance for a 1024-point 16-bit complex FFT running on a 720 MHz C64x family DSP is about 6526 cycles or about a 9.06 μs transform time [TI oct.2003].

IMPLEMENTING AN FFT ALGORITHM AS AN FPGA CO-PROCESSOR

The following actions describe a typical flow to develop an FPGA co-processor:

- • Profile the application to identify high-load software algorithms suitable for off-loading to a coprocessor.
- • Integrate co-processors from available off-the-shelf intellectual property (IP) functions or develop custom co-processor blocks.
- • Consider viable co-processor system architectures.
- • Select a suitable interface to the main processor.
- • System integrate the hardware and software components.
- • Verify system in hardware [Lim 2003], [Alt aug.2004].

For the purposes of this paper, we have selected an FFT algorithm for implementation as an FPGA coprocessor. The actions listed above, however, can be applied to any application that has computationally intensive functions. **Fig. (1)** shows the block diagram of an FFT co-processor implemented within an Altera Stratix FPGA and connected to a Texas Instruments TMS320C6416 DSP via the 32-bit external memory interface (EMIF).

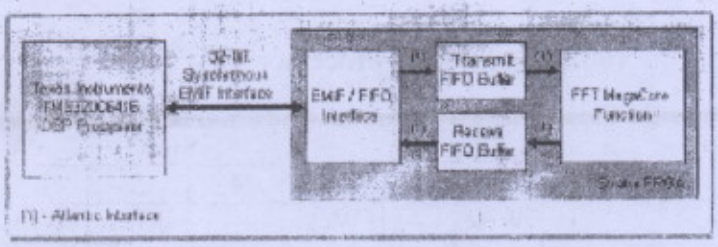

Fig.(1). FFT FPGA Co-processor block diagram

The FPGA co-processor circuitry consists of four main sections: the EMIF/FIFO interface, the transmit FIFO buffer, receive FIFO buffer and the FFT MegaCore function, which performs the FFT computation. The EMIF/FIFO interface handles the translation of the EMIF data and control signals to Atlantic data and control signals. All signaling between the various blocks within the FPGA co-processor is done using the Atlantic interface. The Atlantic interface is a flexible interface for high-throughput packet-based data transmission of arbitrary length. It provides a synchronous point-to-point connection between two blocks of logic with flexible flow control for master-to-slave and slave-to-master directions [Alt june2002]. The flexible nature of the Atlantic interfaces allows the designer to insert one or more co-processor modules between the Atlantic slave source and the Atlantic slave sink.

The transmit and receive FIFO buffers act as storage buffers that handle the transfer of the data packets between the EMIF and the FFT MegaCore. For a 1024-point FFT, the transmit and receive FIFO buffers are set to a depth of 2048 to avoid any overflow of data that might occur when writing to the FIFOs. The FFT MegaCore function is an optimized, parameterizable IP block available from Altera [Alt june2004]. The EMIF was chosen as the connection medium for the FPGA co-processor due to the data transfer rates available and the possibility of using the enhanced direct memory access (EDMA) controller integrated within the TMS320C6416 DSP. Table 1 shows the peak data-

transfer rates based on the given EMIF clock rates. In practice, lower overall rates will be achieved, e.g., when transferring data between two resources that share the EMIF.

Table (1). EMIF Peak Data Rates

EMIF Mode	Peak Transfer (MBps)		
	66MHz	100MHz	133MHz
32-bit Asynchronous	53	80	106
32-bit Synchronous	264	400	532
64-bit Synchronous	528	800	1,064

The DMA controller within the TMS320C6416 DSP transmits packets of data to be processed via a Synchronous EMIF to the EMIF/FIFO interface on the Stratix FPGA. The EMIF/FIFO interface translates the EMIF signals to Atlantic-based signaling so that appropriate data packets can be loaded into the transmit FIFO buffer. The control logic surrounding the transmit FIFO buffer monitors the fill level of the transmit FIFO to determine when a new input data packet of 1024 samples is available for processing by the FFT function. When the transmit FIFO buffer is almost full, the transmit FIFO can start sending data across the Atlantic interface to the FFT.
The output of the FFT is buffered by the receive FIFO buffer to allow for periods when the EMIF is busy. A DMA transfer is requested when a whole packet of processed data is available to be read from the receive FIFO buffer by the TMS320C6416 DSP. This occurs when the receive FIFO buffer is almost full. **Table (2)** shows how packets of data are scheduled through the hardware blocks in the system.

Table (2). FFT Co-Processor Data Scheduling

Action	Step							
	1	2	3	4	5	6	7	8
EMIF	Wr0	-	Wr1	-	Rd0	Wr2	Rd1	Wr3
Transmit FIFO Buffer	-	In0	-	In1	-	-	In2	-
FFT MegaCore Function	-	-	FFT0	-	FFT1	-	-	FFT2
Receive FIFO Buffer	-	-	-	Out0	-	Out1	-	-

The software that was implemented with the FFT co-processor design was built using the DSP functions and BIOS libraries included with the Texas Instruments Code Composer Studio software to configure the EDMA controller and interrupts. The software was used to stream blocks of data, generated using a sine wave generator algorithm, through the FFT co-processor. The sine wave generator is implemented using a double precision cordic algorithm on the DSP [Glob 2003]. The output values can be graphed within the Code Composer Studio workspace for visual verification.
Two DSP general purpose I/O (GPIO) pins are dedicated for use as event triggers to the EDMA: one for transmitting data from the DSP to the co-processor and one for receiving data from the co-processor to the EDMA on the DSP. The co-processor requests a new transmit DMA whenever the FFT function is free and when the transmit credit register (TX_CREDIT) is non-zero. The DSP must write to TX_CREDIT before the co-processor can begin operating. This way, the DSP can signal to the co-processor that the data buffers are ready.
When the FFT has completed processing the data and the data is available in the receive FIFO buffer, the co-processor control logic requests a receive DMA from the DSP. Each time a DMA is completed, the EDMA sends an interrupt request to the DSP. The software tracks the number of

packets transmitted and received. When a pre-defined number of packets have completed processing, the software calculates the average performance of the FFT co-processor across packets. The software that was created performs the following tasks within the main( ) routine:

- • Sets up timer0 for performance measurement.
- • Initializes the memory buffers for the EDMA.
- • Calculate the data values for the input sine wave.
- • Resets the FPGA co-processor and synchronous FIFO buffer
- • Initializes the DSP's chip support library.
- • Calls initEdma( ) to initialize the EDMA controller
- • Calls initHwi( ) to enable EDMA interrupts
- • Starts the timer
- • Increments TX_CREDIT
- • Waits until all blocks have been processed
- • Calculates average time to process one FFT

Each time a transmit or receive DMA interrupt occurs edmaHwi( ) is called to handle the interrupt.

DESIGN CONSIDERATIONS

The following subsections attempt to evaluate and analyze the advantages and disadvantages of both methods from an ease of implementation, cost and performance, and power consumption perspective. These are some of the main factors designers consider when deciding on the best implementation for their systems.

Ease of Implementation – Evaluation and Analysis

For most generic DSP functions like FIR filters, FFTs and correlators, the DSP-only approach significantly reduces the implementation effort with the availability of pre-built, assembly optimized, C-callable library functions. Even if the designer requires a custom function that is not available with generic library functions, complex algorithms are generally easier to implement in a high-level language like C or C++. Challenges may arise while trying to optimize the performance of the function for a particular DSP, often requiring an in-depth knowledge of the processor architecture and assembly instructions. Nevertheless, the designer remains within the same familiar development environment, and is not required to build additional hardware functions to complement the desired system.

On the other hand, the FPGA co-processor approach currently requires a certain amount of hardware knowledge to assemble the various components of the FPGA co-processing system, consisting of the EMIF/FIFO interface, transmit and receive FIFO buffers, and the actual co-processing function itself. The transmit and receive FIFO buffer sizes will have to be individually parameterized to suit each co-processor within the system. The availability of parameterizable, architecturally optimized pre-built IP functions from FPGA vendors like Altera, however, aids in the implementation of the co-processor, reducing the overall design and verification time of the system. With the FPGA co-processor approach, familiarity with DSP software development environments, as well as hardware design methods, is necessary to successfully integrate both elements of the system. Despite this apparent hurdle, the future of FPGA co-processing looks bright with the continuous evolution of better and better system integration tools like Altera's SOPC Builder which may eventually support built-in interfaces to DSPs like Texas Instruments' EMIF and Analog Devices' Link Port interface.

Cost and Performance – Evaluation and Analysis

It was established in section 2, that a 720 MHz TMS320C6416 DSP is capable of completing a 1024-point 16-bit fixed-point complex FFT in about 6526 cycles or about 9.06 μs. A similar FFT configuration running on a Stratix FPGA is capable of achieving transform times of up to 4.64 μs (1291 clock cycles at 278 MHz), while consuming about 15% of the entire FPGA.

The FPGA co-processor example used here is capable of completing a 1024-point, 16-bit complex FFT in about 13.6 µs at 133 MHz on a 64-bit synchronous EMIF configuration (9.7 µs transform time + 3.9 µs data transfer time). Taking the 10,000 unit volume price of the Stratix 1S25F672C8 device at $33 per unit and multiplying that by the resource usage gives an effective cost of $4.95 against the $115 per unit cost of the TMS320C6416 DSP at the 10,000 unit volume price [DSP 2004]. Overall, the FPGA co-processor shows a 15.5 times improvement from a relative price/performance perspective.

Running the EMIF at higher clock rates (133 MHz rather than 100 MHz) or at a higher bandwidth (64-bit synchronous rather than 32-bit synchronous) could potentially increase the performance of the FPGA coprocessor assuming that the performance bottleneck is caused by the latency in data transfer. The use of FIFO buffers in the transmit-and-receive paths enables the FFT (or any other co-processing function) to run at higher clock speeds from the EMIF. This is useful for system optimization if the co-processor function is found to be the performance bottleneck. The co-processor function can then be run at a higher speed compared to the EMIF to decrease the processing time. **Table (3)** shows the average throughput achieved with pipelining when running the EMIF and co-processor and different clock rates.

Table (3). FFT Co-Processor Computation Time at Varying Clock Rates

Co-processor Block	Synchronous Clock Rates (us)		
	32-bit 100MHz	64-bit 133 MHz	64-bit 266 MHz
EMIF Interface&TxFIFO Buffer	10.2	3.9	1.9
FFT MegaCore Function	12.9	9.7	4.8
EMIF Interface & Rx FIFO Buffer	10.2	3.9	1.9
Average Throughput	23.1	13.6	6.7

Power Consumption – Evaluation and Analysis

Power consumption on a DSP and FPGA is a function of the underlying CMOS process used to fabricate the devices. The basic formula used to evaluate power for a CMOS circuit is Power, P (Watts) =1/2*C*V2*f, where C is the load capacitance in Farads, V is the supply voltage in volts and f is the operational clock frequency in Hertz. From this formula it is easy to see that by reducing any one of the

three main components, capacitance, voltage or operational frequency, we are able to reduce the overall

power consumed by the device.

Power consumption in a DSP is attributed to a few or all of the following factors: the type of instructions being implemented by the processor (This affects the switching from a logic 0 to a logic 1 and vice versa on the device, which in turn, affects power consumption, since the more frequently switching occurs, the larger the capacitance on a signal path, causing more power to be consumed.), the clock frequency the processor is running at, the operational voltage of the processor, and the number of peripherals being used or enabled [DSP 2004]. The power consumption reported for the C64x DSP family running at 720 MHz for typical activity is approximately 1.2 Watts (for internal logic only) . Based on the information provided in [DSP 2004], this estimate can and will vary depending on the type of application being performed on the DSP.

Not unlike a DSP, the power consumed by an FPGA is also subject to similar factors such as operating voltage, operational clock frequency, and the number of logic 0 to logic 1 and logic 1 to logic 0 crossings (also known as toggle rate). Along with operational voltage, the FPGA also has an additional source of power consumption known as leakage or core power. This is the amount of power required to "turn on" or enable the underlying FPGA fabric and configuration memory to

maintain proper FPGA operation. This core power consumption occurs whether or not the FPGA has been configured and is performing its programmed tasks. The power consumed by the operation of an application on the FPGA is subject to the same factors and conditions as a DSP or any other application running on a CMOS-based technology.

The Quartus II software by Altera that supports the Stratix device family has a built-in power estimator that reports the approximate power consumed by a particular application for a given set of stimuli. The stimuli or vectors used for this power estimation are important, since the toggle rate of data is a key component in power consumption. The designer has to be careful that the vectors used in the estimation are indicative of the signals that will be passing through the system during actual operation to get more accurate representations of the power that will be consumed in the field.

Approximate power estimates for a 1024-point 16-bit complex FFT built using the FFT MegaCore function was obtained using the Quartus II software version 4.1. **Table (4)** shows the power estimation numbers for operating clock frequencies of 100 MHz, 133 MHz and 275 MHz (the highest clock rate achievable by the FFT MegaCore on a Stratix device with these parameters).

Table (4). FFT MegaCore Function Power Consumption

Function	Power Consumption (mW)		
	100 MHz	133 MHz	275 MHz
FFT MegaCore Function	520.74	596.08	883.84

From a function to function comparison, we can see that an FPGA FFT implementation generally utilizes less power than a similar FFT function implemented on a DSP. Even at the maximum operating clock frequency of 275 MHz, the FPGA FFT implementation still consumes less power than the DSP processor. In this case, the FPGA is not fully utilized (only about 15% in this case) and can handle more functions and larger amounts of data in parallel compared to the DSP. Increased functionality added to the FPGA will most likely increase the toggle rate, the operating frequency and the amount of logic utilized within the FPGA, all of which will increase the total amount of power consumed by the FPGA. In return, the designer gets the benefit of a higher performance system, a trade-off the system designer should take into account. Additionally, with the FPGA co-processor approach, the designer has to factor in the addition power consumed by the DSP (now running in a reduced function, since a large portion of the processing has been off-loaded to the FPGA co-processor), in addition to the power consumed by the FPGA co-processor.

SUMMARY AND CONCLUSIONS

DSP and FPGA co-processor solutions provide designers with a myriad of implementation options and solutions for today's system designers. Along with these solutions comes a variety of design factors and considerations that need to be evaluated to select the best approach, depending on system requirements like ease of implementation, cost and performance as well as power consumption.

DSPs can provide the simplest implementation for a wide range of DSP algorithms and applications, but the cost/performance, implementation flexibility and hardware parallelism provided by an FPGA coprocessor cannot be overlooked. A simple example of this is a basic OFDM communication system, which requires many FFT operations to be computed for a large number of channels at once. Today's DSPs are unable to keep up with the load required of these systems unless an approach requiring large arrays of DSPs is employed. Integration of these DSPs within the system is not a trivial task. The coordination of shared memory between processors is especially complex.

FPGAs, on the other hand, are capable of parallelizing the operation of these functions, reducing the overall computation time of each operation and are, therefore, able to support a larger number of

channels within a single device. Additionally, the densities of FPGAs have grown significantly over the last two years to the point that multiple instantiations of these functions can be implemented within a single FPGA, reducing the total number of devices required and ultimately board real estate.

From a price/performance comparison, FPGA co-processors provide better performance for lower cost compared to a single DSP approach. Additionally, since the FPGA is not fully utilized, more functionality and parallelism could be added to the FPGA co-processor to increase the amount of processing the FPGA is capable of without impacting the cost of the system. Also, from a function-to-function power comparison, we see that for the same function, an FPGA implementation is capable of consuming less power than a DSP. For comparable performance to an FPGA, a designer may be required to implement an array of multiple DSP processors, something which could possibly increase the cost and power consumption of a system beyond that of an FPGA co-processor implementation.

REFERENCES

TMS320C6414, (2001),TMS320C6415, TMS320C6416 Fixed-Point Digital Signal Processing Data Sheet, Texas Instruments, February

Lim S. Lim, and P. Ekas, (2003), Design Methodology for Hardware Acceleration for DSP, Proc. International Signal Processing Conference, GSPx,

TMS320C64x (2003), DSP Library Programmer's Reference, Texas Instruments, October.

Application Note 352: (2004), FPGA Peripheral Expansion & FPGA Co-Processing with a TI TMS320C6000 DSP Processor, Altera Corporation, August.

Atlantic Interface Functional Specification, (2002), version 3.0, Altera Corporation, June.

FFT MegaCore Function User Guide, (2004), version 2.1.0, Altera Corporation, June.

Fast, Continuous, (2003), Sine Wave Generator, GlobalDSP, December.

Stratix FPGA Device Handbook, (2004), version 3.0, Altera Corporation, April.

TI Moves 'C64x to 90 Nanometers, (2004), 1GHz, BDTi's DSP Insider, Vol. IV, No. 2, http://www.bdti.com/dspinsider/archives/dspinsider_040218.html