Dynamically Reconfigurable FFTs for Cognitive Radio on a Multiprocessor Platform

Qiwei Zhang, Karel H.G. Walters, Andre B.J. Kokkeler and Gerard J.M. Smit
Department of Electrical Engineering, Mathematics and Computer Science
University of Twente, Enschede, The Netherlands
Email: q.zhang@utwente.nl

***Abstract*— Multiprocessor platforms have been proposed as an enabling technology for Cognitive Radio. In this paper, we explore various FFT implementations on a multiprocessor prototype platform as building components for OFDM based Cognitive Radio. The results show that our FFT implementations on the multiprocessor platform are fast and energy efficient. Moreover, the FFT implementations can be dynamically reconfigured with relatively small reconfiguration overheads.**

***Index Terms*— Cognitive Radio, Dynamically reconfigurable FFT, Multiprocessor System-on-Chip, Montium.**

I. Introduction

RECENT studies have shown that most of the assigned radio spectrum is underutilized. On the other hand, the increasing number of wireless multimedia applications lead to increasing spectrum scarcity. Cognitive Radio [1] is considered as a promising technology to address the paradox of spectrum scarcity and spectrum under-utilization. In Cognitive Radio, a spectrum sensing process locates the unused spectrum segments in a targeted spectrum pool. These segments will be used optimally without harmful interference to licensed users. This technology is called *spectrum pooling* [2]. In spectrum pooling, orthogonal frequency division multiplexing (OFDM) is used as the baseband transmission scheme. The cognition is realized by nullifying those subcarriers which cause interference to the licensed user (a user who has the legal license for the spectrum). The remaining frequency segments will be used optimally by Cognitive Radio. Cognitive Radio has to operate in different bands under various data rates and combat adversary channel conditions. Therefore, the reconfigurability of the physical layer has to be supported by a Software Defined Radio (SDR) platform. General Purpose Processors (GPPs) and Digital Signal Processors (DSPs) based SDR platforms are inadequate for future high data rate wireless communications in terms of throughput and energy efficiency for battery-operated terminals. With the advance of semiconductor technology, wireless baseband processors are moving toward Multiprocessor System-on-Chips (MPSoCs) which integrate heterogeneous processing elements tailored for different processing tasks. MPSoCs offer high performance, flexibility and energy efficiency. Therefore, we proposed a tiled MPSoC architecture to support Cognitive Radio in [3]. To demonstrate Cognitive Radio on an MPSoC platform, we implement various dynamically reconfigurable algorithms for Cognitive Radio on a prototype MPSoC platform called BCVP (Basic Concept Verification Platform) [4]. The paper is organized as follows. Section II introduces various FFTs as building components for our Cognitive Radio system. Section III presents how the implementation is done from the algorithm level to the platform level. Results of the platform implementation will be shown in section IV. Future work and conclusions are in the last two sections.

II. FFTs for Cognitive Radio

The FFT is considered as a major component in our OFDM based Cognitive Radio for both energy based spectrum sensing and OFDM transmission [3]. The FFT is the most computationally intensive part in the whole streaming baseband system of Cognitive Radio. Table I shows the computational complexity, in terms of complex multiplications, of our AAF OFDM receiver [5] with 512 subcarriers. Therefore we choose to implement the FFT on our prototype platform to demonstrate how the MPSoC platform can support the streaming DSP processing of Cognitive Radio. Two types of FFT algorithms are used in our OFDM based Cognitive Radio system: normal radix-2 FFTs and sparse FFTs, proposed in [6].

N	Freq. cor.	FFT	Channel eq.	Phase offset cor.
512	512	2034	512	385

TABLE I
The number of complex multiplications of major receiver tasks for one AAF OFDM symbol

A. Reconfigurable radix-2 FFT

Cognitive Radio needs reconfigurable radix-2 FFTs for the following reasons. First, Cognitive Radio may adapt to different transmission modes by changing the number of subcarriers in OFDM. Second, multi-resolution spectrum sensing can be supported by FFTs with various size. Moreover, the reconfiguration has to be done at run-time and with a minimum overhead since the frequency occupancy and channel conditions are constantly changing for Cognitive Radio without any predetermined patterns. The basic idea to make radix-2 FFTs reconfigurable is to reuse the computation structure and the twiddle factors of larger FFTs for smaller FFTs. An example is shown in figure 1 where a 4 point FFT can be reconstructed from an 8 point FFT by skipping the last stage

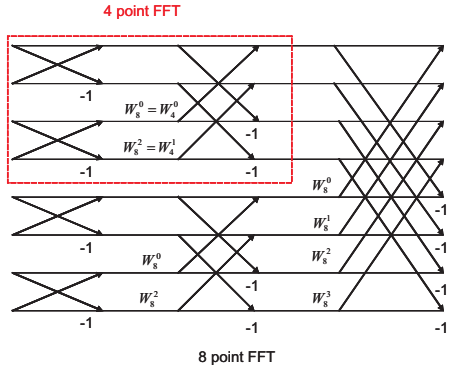

Fig. 1. An example of reconfigurable radix-2 FFT

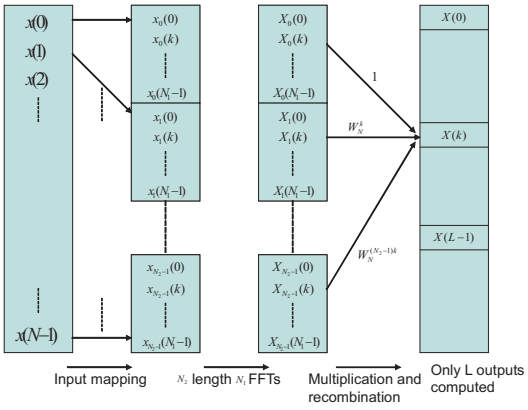

Fig. 2. Computational structure of transform decomposition

butterfly operations. The same rule applies for reconstructing any smaller size FFTs from a larger FFT. Therefore, in our implementation we load the configuration of a large size FFT during the initialization and switch to small FFTs by adapting a small part of the configuration. In this way the reconfiguration overhead is reduced to a minimum since there is no need to reload the whole configuration during reconfigurations.

B. Sparse FFT

The Sparse FFT has been introduced in [6] for OFDM based Cognitive Radio where a large number of subcarriers may be switched off to avoid interference to the licensed user. In this scenario, there are many zero inputs/outputs for the IFFT/FFT of the OFDM system. If normal radix-2 FFTs are applied, a large fraction of the computations will be wasted on zeros which are known to the transmitter and the receiver. The algorithm takes advantage of the zero values and only computes the sparsely distributed nonzero values. The algorithm is based on transform decomposition described in [7] but tailored for our Cognitive Radio system. The mathematical derivation of the algorithm can be found in [6]. In this paper, we focus on the implementation on the multiprocessor platform. Figure 2 shows the computational structure of a sparse FFT which decomposes the DFT into two major parts: N_2 blocks of N_1-point DFTs which can be implemented as radix-2 FFTs (N_1 and N_2 are chosen to be power-of-two integers) and the multiplications with twiddle factors together with the recombination of the multiplications. The reduction of computation comes from the second part where only L twiddle factors are multiplied with each $X_{n_2}(\langle k \rangle_{N_1})$ ($\langle \rangle_{N_1}$ denoting modulo N_1) for $n_2 = 1, 2, ..., N_2$. Based on the computational structure, we perform a quantitative analysis on the computational complexity by counting the number of complex multiplications. The number of multiplication for the sparse FFT is $(N_2 - 1) * L + \frac{N}{2} \log_2 N_1$, which is less than the number of multiplications for a radix-2 FFT ($\frac{N}{2} \log_2 N$) when $L < N/2$.

We use a sparse FFT only when a fraction of subcarriers is used for the OFDM system, therefore the system has to be reconfigured from a radix-2 FFT to a sparse FFT and vice versa. From the computational structure of the sparse FFT in figure 2, we can see that a sparse FFT can re-use the radix-2 FFT configuration in the first part and only add an additional part for the multiplication and recombination of the twiddle factors.

III. Implementation on a Multiprocessor Platform

A. BCVP

We use a platform named BCVP (Basic Concept Verification Platform) for verification purposes. This prototype multiprocessor platform is constructed with off-the-shelf components. It contains two ARM9 cores, a Xilinx FPGA, DRAM, two DDCs (Digital Down Converter), a Viterbi decoder and peripheral I/O. The major components considered in this paper are shown in figure 3. The Xilinx Virtex-II 8000 FPGA emulates three Montium tile processors [8] connected by a circuit switched NoC (Network-on-Chip) [9]. The Montium tile processor (see figure 4), developed at the University of Twente, is a Domain Specific Reconfigurable Hardware building block which targets the 16-bit fixed point DSP algorithm domain. In our previous work [10], several DSP algorithms used for wireless communications have been mapped onto the Montium. Although there exist dedicated hardware blocks or even reconfigurable hardware blocks for a specific algorithm such as FFT (e.g. [11]), they are limited to only one algorithm. We could embed many such different dedicated hardware blocks into a SoC, however this will increase the area and complexity of the system. Moreover, the design efforts for dedicated hardware blocks are considerably large since each block has to be optimized for one algorithm. Therefore, our approach is to use one type of reconfigurable hardware cores, such as the Montium, for many DSP processing tasks.

B. The Montium Processor

Figure 4 shows the architecture of the Montium tile processor. The upper part is the computing part that can be configured to implement a particular algorithm. The ALUs can do basic DSP operations like multiplications and additions and they can also perform basic logic functions. The five identical ALUs (ALU1...ALU5) in a tile can exploit spatial concurrency to enhance performance. This parallelism demands a very high memory bandwidth, which is obtained by having 10 local memories (M01...M10) in parallel. The compiled configurations are decoded by instruction decoders. The CCU

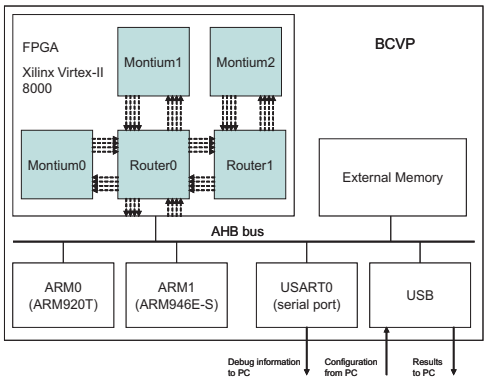

Fig. 3. The major components in the BCVP

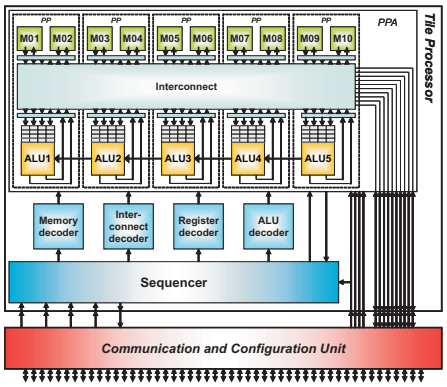

Fig. 4. The Montium tile processor

(Communication and Configuration Unit) is the interface for off-tile communication. With minimum control overhead for concurrent processing, the Montium can achieve reconfigurability with energy efficiency. The Montium has a low silicon cost, as the core is very small. For instance, the silicon area of a single Montium TP with 10 KB of embedded SRAM is 2.4 mm^2 in 0.13μm CMOS technology. The power consumption in this technology is only 577 μW/MHz (including all memory accesses).

C. Design method

Mapping applications onto a multiprocessor system is a challenging task. Designers have to deal with the low level interfaces for the inter-component communication and synchronization which become a bottleneck from a performance and an energy point of view. Opportunities for reuse of hardware and software modules are limited. In [12], we propose to use a task transaction level interface (TTL) approach [13] to design Cognitive Radio on a multiprocessor platform by raising the abstraction level. In the TTL approach, an application is modelled as a task graph. A task is an entity that performs computations. One task may communicate with other tasks via channels. Communications are invoked by calling TTL interface functions. A task implementation is the implementation on a particular tile, e.g. object code for an ARM or configuration data for the Montium processor. Figure 5 shows the functional reference of the reconfigurable FFT in the TTL model. The FFT task can be reconfigured to a radix-2 FFT

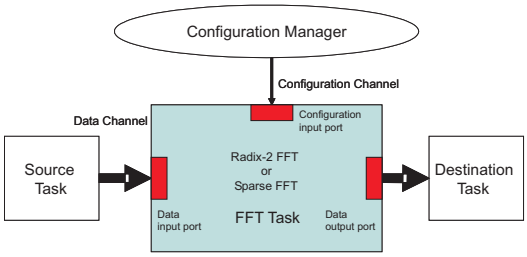

Fig. 5. The TTL model for the reconfigurable FFT task

with various sizes or to a sparse FFT where many subcarriers are switched off. The configuration is done by a configuration manager. The functionality of the algorithm has been verified by running the software C/C++ implementation of the TTL model on a Linux PC. High level profile information can be obtained in terms of annotated instruction counts and the number of tokens sent through channels.

The next design stage is mapping the functional reference to the platform. The FFT task is performed by one of the Montium processors. One ARM (only the ARM946E-S is used throughout the paper) processor will send the data to be processed to the Montium and retrieve the results from the Montium. A real-time operating system called BasOS [14] has been developed for the BCVP running on the ARM processor. It can implement the TTL interface functions for the inter-processor communications between the Montium and the ARM. The Montium configuration for the reconfigurable FFT is generated by using a Montium compiler. The input for this compiler consists of assembly-line statements. We made a demo by connecting a Linux PC to the BCVP via USB and a serial port. First, the PC loads the Montium initialization configuration to the BCVP via a USB port. A graphical user interface on the PC can control the reconfiguration and send new reconfigurations and data to the BCVP and visualize the result retrieved from the BCVP via the USB port.

D. Mapping the reconfigurable FFT onto the Montium

The major design task is mapping the reconfigurable FFT onto the Montium. Mapping the radix-2 FFT has been described in [8]. An FFT algorithm consists of a number of stages depending on the size of the FFT. Each stage consists of several butterfly operations. One butterfly operation can be computed in one clock cycle using four Montium ALUs. The butterfly computation is done sequentially on a stage by stage basis. Since the butterfly operation has a repeating pattern, it can be implemented in a loop. A stage requires two sequencer instructions which are looped for the amount of input values that are needed. For the reconfigurable radix-2 FFT, during the initialization stage we load the configuration of the largest FFT needed into the sequencer. Run-time reconfiguration is achieved by altering the configuration memory. Each entrance in this memory defines an instruction and contains a control part which is used to implement a sequencing state machine. By reducing the loop counter values in the sequencer instructions and only using the stages needed, the large FFT can be reconfigured to a small FFT. Switching back to a large FFT

can also be done by adjusting the loop counters and using more stages in the sequencer. The configuration is nothing more than bytes of data that adjust the sequencer. Table II

Seq PC	ALU	AGU	jump	loop	description
[0]	[0]	[3]	[1]	1	Start of stage 1
[1]	[1]	[4]	[2]	8 (4)*	
[2]	[1]	[4]	[3]	1	Start of stage 2
[3]	[0]	[3]	[4] ([7])*	8 (4)*	
[4]	[0]	[3]	[5]	1	Start of stage 3
[5]	[1]	[4]	[6]	8	
[6]	[1]	[4]	[7]	1	Start of stage 4
[7]	[0]	[3]	[8]	8 (4)*	

TABLE II

A SIMPLIFIED SEQUENCER PROGRAM FOR A RECONFIGURABLE FFT CHANGING FROM A 16 POINT TO AN 8 POINT FFT WHERE THE ADAPTION HAS BEEN SUGGESTED

shows a simplified version of what a 16 point FFT sequencer program would look like. The numbers below the AGU and ALU column represent indices of specific Address Generation Unit (AGU) or ALU instructions. In order to change this 16 point FFT sequencer program to an 8 point FFT only the loop counter values have to be adjusted and a stage has to be skipped. By adjusting the jump target in sequencer instruction 3 from 4 to 7 and adjusting the loop counters from 8 to 4, an 8 point FFT is created. The values changed are indicated by an asterisk in table II.

The method of mapping a sparse FFT onto the Montium has been indicated in [6]. During the input mapping and the recombination stage, the memory addressing is constantly hopping from a position in one block to the same position in another, see figure 2 (e.g. from $x_0(0)$ to $x_1(0)$). Each memory of the Montium has an Address Generation Unit (AGU) which can generate the required addressing pattern. The radix-2 FFT configuration can be re-used for each memory block in the first part. In the second part where multiplications with twiddle factors and recombination are done, we use the 5th ALU to calculate the indices to be multiplied and used as a memory address for AGU. This address generation costs 3 extra clock cycles per each nonzero value, which is the efficiency bottleneck of the sparse FFT on the Montium as will be shown in the next section.

IV. RESULTS

The performance of the FFT on the Montium has already been benchmarked with other processors in [15]. Here, we intend to use real measurements to verify the result in [15]. Due to the limitation of the FPGA, the emulated Montium on the BCVP only runs at 6.8MHz. However, the Montium can run as fast as 100MHz on a SoC [16]. Table III shows the measured execution time of radix-2 FFTs on the Montium running at 6.8MHz in comparison with the fixed point implementations on the ARM946E-S running at 86MHz on the BCVP. Although the Montium runs at 12 times lower frequency than the ARM, it still outperforms the ARM by more than 4 times in processing speed. The power consumption of the Montium, in 0.13μm technology, is estimated at 0.577 mW/MHz. Therefore the largest reconfigurable FFT (FFT-512)

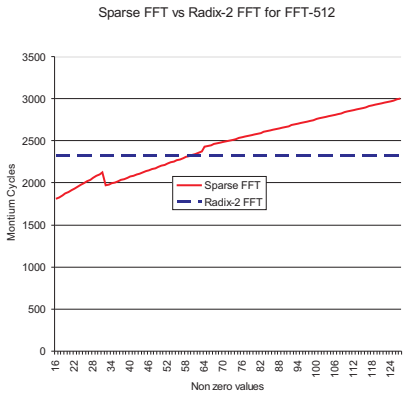

Fig. 6. The performance of the sparse FFT vs radix-2 FFT for FFT-512 on the Montium

on the Montium only cost 1.35 μJ which is comparable to an ASIC implementation.

FFT size	512	256	128	64
Montium (ms)	0.35	0.15	0.07	0.03
ARM (ms)	1.31	0.59	0.27	0.12

TABLE III

THE EXECUTION TIME OF RADIX-2 FFTS ON THE MONTIUM TILE PROCESSOR AND THE ARM ON THE BCVP

Figure 6 shows the number of clock cycles of the sparse FFT vs. the radix-2 FFT for 512 samples on the Montium as a function of the amount of non-zero subcarriers. Due to the address generation bottleneck mentioned in Section III, the final sparse FFT implementation on the Montium turns out not as efficient as the estimation in [6]. As shown in figure 6, the complexity of the sparse FFT exceeds the radix-2 FFT when there are more than 64 non-zero outputs. However, it is still more efficient than the radix-2 FFT when a large number of subcarriers is switched off (e.g. 480 out of 512). An additional benefit of the sparse FFT is that only the non-zero outputs are sent through the on-chip communication architecture. Therefore it results in less communication cost which is considered to be the bottleneck for smaller feature sized SoCs in the future.

Table IV shows the costs of the reconfiguration for both radix-2 FFT and sparse FFT (less than 64 non-zeros) in terms of bytes. When using reconfigurable code, a large configuration is sent to the Montium at initialization and changing FFT size involves small pieces of reconfigurable code. When using static code, changing FFT size requires a complete reconfiguration. Dynamic reconfiguration is about 10 times more efficient than reloading the complete static configuration. Reconfiguration time depends on the I/O and NoC speed.

V. FUTURE WORK

Samples of an MPSoC based on the BCVP are now available. It contains one ARM926 core, boot memory, Viterbi decoder, two DDCs (Digital Down Converters), two A/D converters, peripheral I/O, and four Montium tiles. Our future

	reconfigurable code	static code
Initialization	1792	0
to 512-point	104	1138
to 256-point	82	1054
to 128-point	78	970
to 64-point	80	886
to 512-sparse (8 * 64 point)	78	–
Total	2214	4048

TABLE IV

BYTES THAT NEED TO BE SENT FOR RECONFIGURATION

plan is to demonstrate a Cognitive Radio system including OFDM transmission and spectrum sensing on this SoC.

VI. CONCLUSION

In this paper, we presented reconfigurable FFTs, as building components for OFDM based Cognitive Radio on a prototype multiprocessor platform. The Montium [8] tiled processor on the platform is configured to perform the reconfigurable FFTs. The performance of radix-2 FFTs on the Montium emulated on a FPGA is 4 times faster than the ARM on the BCVP. The performance of sparse FFT [6] on the Montium shows its computation efficiency when there are many zero subcarriers in the OFDM system of Cognitive Radio. The FFT reconfiguration proves to be very efficient with only small overheads, which is crucial to Cognitive Radio for its fast adaption to the environment. Therefore speed, reconfigurability and energy efficiency make multiprocessor platforms ideal radio platforms to support future wireless applications such as Cognitive Radio.

ACKNOWLEDGMENT

The work is sponsored by the Dutch Ministry of Economic affairs Freeband AAF project. We acknowledge the support for the TTL interface approach by NXP and for the Montium development tool by the Recore Systems. The authors would like to thank their colleagues from the International Research Centre for Telecommunications-transmission and radar (IRCT) of the Technical University Delft (TUD) and the Signal and System (SAS) group at the University of Twente (UT) for the fruitful discussions in the AAF project.

REFERENCES

[1] J. Mitola III. *Cognitive Radio: An Integrated Agent Architecture for Software Defined Radio*, PhD Thesis, Royal Institute of Technology, Sweden, May. 2000.

[2] T.A. Weiss and F.K. Jondral, "Spectrum pooling: An innovative strategy for the enhancement of spectrum efficiency," *IEEE Commun. Mag.*, Mar. 2004

[3] Qiwei Zhang, Andre B.J. Kokkeler and Gerard J.M. Smit, "A reconfigurable radio architecture for Cognitive Radio in emergency networks," *European Conference on Wireless Technology*, September 2006, Manchester, UK

[4] Smit, G.J.M. and Schuler, E. and Becker, J.E. and Quevremont, J. and Brugger, W. "Overview of the 4S project," *International Symposium on System-on-Chip*, Nov 2005, Tampere, Finland

[5] Hoeksema, F.W., Heskamp, M., Schiphorst, R. and Slump, C.H. "A node architecture for disaster relief networking," *Proceedings of the first IEEE Symposium on New Frontiers in Dynamic Spectrum Access Networks (DySPAN2005)*, November 2005, USA

[6] Qiwei Zhang, Andre B.J. Kokkeler and Gerard J.M. Smit "An efficient FFT for OFDM based Cognitive Radio on a reconfigurable architecture," *IEEE International Conference on Communications (ICC2007)*, 2007, UK

[7] Henrik V. Sorensen and Sidney Burrus "Efficient computation of the DFT with only a subset of input or output points," *IEEE Trans. on Signal Processing*, Mar. 1993

[8] Paul Heysters *Coarse-Grained Reconfigurable Processors; Flexibility meets Efficiency*, PhD Thesis, University of Twente, Sep. 2004

[9] Wolkotte, P.T. and Smit, G.J.M. and Rauwerda, G.K. and Smit, L.T. "An energy-efficient reconfigurable circuit switched Network-on-Chip," *12th Reconfigurable Architecture Workshop (RAW 2005)*, Apr 2005, USA

[10] Gerard K. Rauwerda, Paul M. Heysters and Gerard J.M. Smit "Towards software defined radios using coarse-grained reconfigurable hardware," *IEEE Transcation on Very Large Scale Integration Systems*, Jan. 2008

[11] Yutian Zhao, Ahmet T. Erdogan and Tughrul Arslan, "A low-power and domain-specific reconfigurable FFT fabric for System-on-Chip applications," *IEEE International Parallel and Distributed Processing Symposium (IPDPS05)*, 2005, USA

[12] Qiwei Zhang, Andre B.J. Kokkeler and Gerard J.M. Smit "Cognitive Radio design on an MPSoC platform," *International Conference on Cognitive Radio Oriented Wireless Networks and Communications (CrownCom 2007)*, Aug 2007, USA

[13] Pieter van der Wolf et al, "Design and programming of embedded multiprocessors: An interface-centric approach," *In Proceedings of ISSS+CODES*, Sept. 2004

[14] Bas van Sisseren *Design of a Lightweight Real-Time Streaming Kernel*, Master Thesis, University of Twente, 2007

[15] Heysters, P.M. and Smit, G.J.M. and Molenkamp, E., "Energy-efficiency of the montium reconfigurable tile processor," *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA'04)*, 2004, USA

[16] Smit, G.J.M. and Kokkeler, A.B.J. and Wolkotte, P.T. and van de Burgwal, M.D., "Multi-core Architectures and Streaming Applications," *International Workshop on System-Level Interconnect Prediction (SLIP 2008)*, 2008, UK