A Coarse-Grained Array based Baseband Processor for 100Mbps+ Software Defined Radio

Bruno Bougard, Bjorn De Sutter, Sebastien Rabou, David Novo, Osman Allam, Steven Dupont, Liesbet Van der Perre

IMEC, Kapeldreef 75, B-3001 Leuven, Belgium
E-mail: bruno.bougard@imec.be

ABSTRACT

The Software-Defined Radio (SDR) concept aims to enabling cost-effective multi-mode baseband solutions for wireless terminals. However, the growing complexity of new communication standards applying, e.g., multi-antenna transmission techniques, together with the reduced energy budget, is challenging SDR architectures. Coarse-Grained Array (CGA) processors are strong candidates to undertake both high performance and low power.
The design of a candidate hybrid CGA-SIMD processor for an SDR baseband platform is presented. The processor, designed in TSMC 90G process according to a dual-VT standard-cells flow, achieves a clock frequency of 400MHz in worst case conditions and consumes maximally 310mW active and 25mW leakage power (typical conditions) when delivering up to 25,6GOPS (16-bit). The mapping of a 20MHz 2x2 MIMO-OFDM transmit and receive baseband functionality is detailed as an application case study, achieving 100Mbps+ throughput with an average consumption of 220mW.

1. INTRODUCTION

Wireless technology is considered as a key enabler of many future consumer products and services. To cover the extensive range of applications, future handhelds will need to concurrently support a wide variety of wireless communication standards. The growing number of air interfaces to be supported makes traditional implementations based on the integration of multiple specific radios and baseband ICs cost-ineffective and claims for more flexible solutions. Software Defined Radios (SDR), where the baseband processing is deployed on a programmable or reconfigurable hardware, has been introduced as the ultimate way to achieve flexibility and cost-efficiency [1].

Several SDR platforms have already been proposed in academia and industry [1,2,3]. Most of these platforms support the execution of current wireless standards such as WCDMA (UMTS), IEEE 802.11b/g, IEEE 802.16. However, a key challenge still resides in the instantiation of such programmable architectures capable to cope with the 10x increase both in complexity and in throughput required by emerging standards relying on multi-carrier and multi-antenna processing (IEEE 802.11n, LTE), still being cost effective. Leveraging on the sole technology scaling is not sufficient anymore to sustain the complexity increase. In order to achieve the required high performance at an energy budget acceptable for handheld integration (~300mW), architectures must be revisited keeping in mind the key characteristics of wireless baseband processing: *high data level parallelism* (DLP) and *data flow dominance.*

In nowadays SDR platforms, Very Long Instruction Word (VLIW) processors with SIMD (Single Instruction – Multiple Data) functional units are often considered to exploit the data level parallelism with limited instruction fetching overhead [2,3]. In other approaches, data flow dominance is sometime exploited in coarse-grained reconfigurable arrays (CGA) [4,5]. The first class of architectures have tighter limitations in achievable throughput for a given clock frequency while the seconds have as main disadvantage to require very low level programming.

In this paper, we present the design, based on the ADRES/DRESC framework [6], of a hybrid CGA-SIMD SDR processor fully programmable from C-language. The core of the processor is made of 16 densely interconnected 64-bit 4-way SIMD functional units with global and distributed register files. The CGA is associated with a 4-bank data scratchpad (L1) and provides an AMBA2 interfaces for configuration and data exchange. Besides, three functional units, operating as VLIW and sharing the global register file, can execute C-compiled non-kernel code fetched through a 32K 128-bit wide instruction cache. When in array mode, C-compiled DSP kernels are executed while keeping configurations in local memories (one context per scheduled loop cycle) that are configured through direct memory access (DMA). The DRESC framework is used to transparently compile a single C language source code to both the VLIW and the CGA machines.

We focus on the design and the implementation of the aforementioned processor in TSMC 90nm technology and demonstrate its utilization as baseband engine for a 20MHz 2x2 MIMO-OFDM modem as in IEEE802.11n applications. In section 2, the principal architecture level characteristics are reviewed, both at the processor and at the core level. In section 3, the design methodology and results are presented. Importantly, the selection of process and standard-cells library options is discussed as well as the approach followed to minimize the processor power consumption. The goal is to achieve a minimum total energy per task, assuming that the processor will be embedded in a platform providing power management and standby leakage control support [7]. The processor performance and power consumption when executing MIMO-OFDM baseband processing are discussed in section IV. Conclusions are drawn in section V.

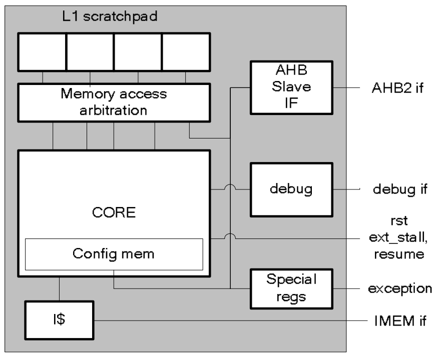

Fig. 1 Processor Top level Architecture

2. ARCHITECTURE

A. Processor architecture

The processor is designed to serve mainly as slave in multi-core SDR platforms [7]. The top level block diagram is depicted in Figure 1. The processor has an asynchronous reset, a single external system clock and a half-speed (AMBA) bus clock. Instruction and data flow are separated (Harvard architecture). A direct-mapped instruction cache (I$) is implemented with a dedicated 128-bit wide instruction memory interface. Data is fetched from an internal 4-bank 1-port-per-bank 16Kx32-bit scratchpad (L1) with 5-channel crossbar and transparent bank access contention logic and queuing. The L1 is accessible from external through an AMBA2-compatible slave bus interface. The CGA configuration memories and special registers are also mapped to the AMBA bus interface via a 32-bit internal bus. After reset and as soon as the external stall signal is de-asserted, the processor start fetching VLIW instruction, resulting in a series of cache misses leading to the load of the I$.

Besides, the processor has a level-sensitive control interface with configurable external *endianness* and *AHB priority* settings (settable priority between core and bus interface to access L1), *exception signaling, external stall* and *resume* input signals. Because of the large state, CGA-based processors are typically non-interruptible. The *external stall* and *resume* signals provide however an interface to work as a slave in a multi-processor platform. The first is used to stop the processor while maintaining the state (e.g. to implement flow control at SOC level). Internally, a special stop instruction can be issued that sets the processor in an internal sleep state, from which it can recover at assertion of the *resume* signal. The data scratchpad and special register bank stay accessible through the AHB interface in sleep mode

Finally, for the sake of prototyping, a dedicated data debug interface is implemented in the current design.

B. Core architecture

The core-level architecture is depicted in Figure 2. The CGA module is further detailed in Figure 3. The core is mainly made of a Global Control Unit (CGU), 3 predicated VLIW Functional Units (FU), the CGA module and a 6-read/3-write ports 64x64-bit Central Data and 64x1-bit Predicate Register File (CDRF/CPRF).

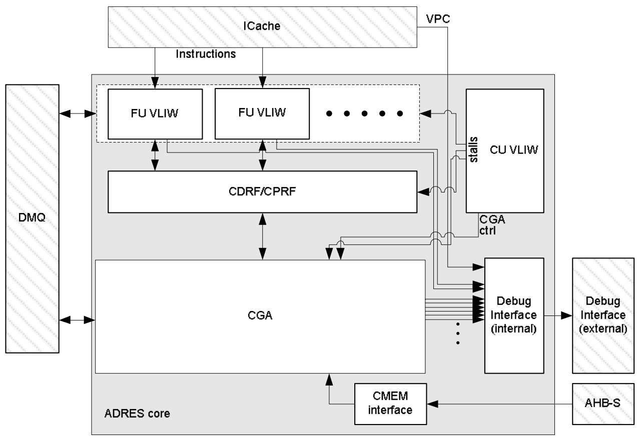

Fig. 2 Processor core Architecture

VLIW and CGA operate the CDRF/CPRF in mutual exclusion and hence its ports are multiplexed. This shared register file naturally enables the communication between the VLIW and the CGA working modes. The two modes often need to exchange data as the CGA executes data-flow dominated loops while the rest of the code is executed by the VLIW.

The CGA is made of 16 interconnected units from which 3 have a two-read/one-write port to the global data and predicate register files. The others have a local 2-read/1-write register file. This local registers are less power hungry than the shared one due to their reduced size and number of ports. The execution of the CGA is controlled by a small size ultra wide configuration memory. The latter extends the instruction buffer approach, so common in VLIW architectures, to the CGA. On this way the CGA instruction fetching power is importantly reduced.

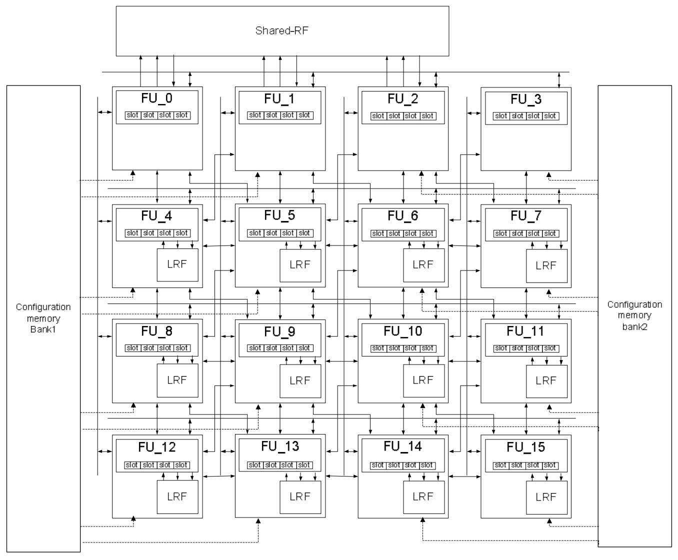

Fig. 3 CGA unit interconnection

TABLE 1

INSTRUCTION SETS

Op Group	*Instuction*	*Semantic*	*# FUs*	*WW [bits]*	*Delay [cycles]*
Arith	add, add_u	dst = src1 + src2	0-15		1
	sub, sub_u	dst = src1 - src2			
	or	dst = src1 *or* src2			
	nor	dst = src1 *nor* src2			
Logic	and	dst = src1 *and* src2	0-15		1
	nand	dst = src1 *nand* src2			
	xor	dst = src1 *xor* src2			
	xnor	dst = src1 *xnor* src2			
Shift	lsl	dst = src1 << src2	0-15		1
	lsr, asr	dst = src1 >> src2			
	eq	dst = src1 == src2			
	ne	dst = src1 != src2			
	gt, gt_u	dst = src1 > src2 (signed, unsigned)			
Comp	lt, lt_u	dst = src1 < src2 (signed, unsigned)	0-15		1
	ge, ge_u	dst = src1 => src2 (signed, unsigned)			
	le, le_u	dst = src1 =< src2 (signed, unsigned)			
	pred_clear	dst = 0			
	pred_set,	dst = 1			
	pred_eq	dst = (scr1 == src2)		32	
	pred_ne	dst = (scr1 != src2)			
Pred	pred_lt, pred_lt_u	dst = (scr1 < src2) (signed, unsigned)	0-15		
	pred_le, pred_le_u	dst = (scr1 =< src2) (signed, unsigned)			
	Pred_gt, pred_gt_u	dst = (scr1 > src2) (signed, unsigned)			
	Pred_ge, pred_ge_u	dst = (scr1 => src2) (signed, unsigned)			
Mul	mul, mul_u	dst = src1 * src2 (32-bit)	0-15		2
	jmp	PC = src2			
Branch	jmpl	PC = src2; R9 = PC + Y	0		3
	br	PC = PC + X +imm<<2			
	brl	PC = PC + X + imm<<2; R9=PC+Y			
	lu_uc	dst = zext$_{8-32}$(mem8[Rsrc1+src2])			
	ld_u	dst = sext$_{8-32}$(mem8[Rsrc1+src2])			
Ldmem	ld_uc2	dst = zext$_{16-32}$(mem16[Rsrc1+src2]); dst = zext$_{16-32}$(mem16[Rsrc1+imm<<1])	0-3		5/7
	lc_c2	dst = sext$_{16-32}$(mem16[Rsrc1+src2]); dst = sext$_{16-32}$(mem16[Rsrc1+imm<<1])			
	ld_i	dst = mem32[Rsrc1+Rsrc2) ; mem32[Rsrc1+imm<<1]			
	st_c	mem8[Rsrc1+imm] = Rsrc3$_{0-8}$			
Stmem	st_c2	mem16[Rsrc1+imm<<1] = Rsrc3$_{0-15}$	0-3		1
	st_i	mem32[Rsrc1+imm<<2] = Rsrc3			
Control	cga	Execute loop in CGA mode	-	-	-
	halt	Drop to sleep mode ; waiting for resume signal			
	c4add	dst = | src1a + src2a | src1b + src2b | src1c + src2c | src1d + src2d |			
SIMD1	c4sub	dst = | src1a - src2a | src1b - src2b | src1c - src2c | src1d - src2d |	0-15		1
	c4shiftR	dst = | src1a >> src2 | src1b >> src2 | src1c >> src2 | src1d >> src2 |			
	c4shiftL	dst = | src1a << src2 | src1b << src2 | src1c << src2 | src1d << src2 |		64	
SIMD2	d4prod	dst = | src1a * src2a | src1b * src2b | src1c * src2c | src1d * src2d |	0-15		3
	c4prod	dst = | src1a * src2b | src1b * src2a | src1c * src2d | src1d * src2c |			
Div	div	dst = src1 / src2	0-1	24	8

VLIW and CGA functional units have 64-bit data-paths. The supported functionality is distributed over several different instruction groups. Table 1 lists the different groups detailing some of the instructions comprised, the functional units that implement such a group (see Fig. 3), the bit-width of the operated word and the group execution latency in cycles. It must be noticed that the basic instruction group (arith, logic, shift, comp, pred, mul, branch) operates only on the 32 LSB (Least Significant Bits) of the datapaths and registers. Similarly, load/store instructions assume 32-bit physical storage. Hence, 64-bit registers contents can only be loaded/stored with two instructions. Only the special instructions (group SIMD1, SIMD2), which are the hottest in utilization, operate in 64-bit according to a 4x16-bit SIMD alignment. Finally, the architecture also includes 2 hardwared dividers which operate on the 24 LSB.

3. IMPLEMENTATION

A. Process and library selection

The architecture described above is implemented to reach 400MHz clock rate in worst-case condition when implemented in 90nm technology. Hence it delivers up to 16 units x 4 way SIMD x 400MHz = 25.6GOPS (16-bit) as foreseen to be sufficient to implement 2x2 20MHz MIMO-OFDM at 100Mbps+ [8]. To achieve such a clock frequency at maximum energy efficiency, TSMC90G process has been selected. Although it is leakier, the 90G process has better power-delay product that the 90LP process usually considered for embedded application. Leakage in operation mode is tackled with multi-VT design and, in standby, with third-party substrate biased standard cell library and memory macros.

The current design is done with multiple VT standard cell libraries with substrate-biasing support and single port register file and SRAM macros. Multi-ported register files are synthesized from RTL descriptions.

B. *Power-aware Design and Verification Methodology*

The micro-architecture implementation started with the RTL description of the architectural components. To minimize active power, we focused on clock gating. The RTL descriptions of the functional units and multi-ported register files have been written in such a way that automated fine-grained clock gating was enabled during synthesis. Furthermore, operand isolation was manually implemented in the functional units to avoid the toggling of unused operators.

The RTL was validated by means of simulating the execution of a functional regression test suite. The RTL execution is compared with a STRL reference model. All instructions, including all SIMD instructions, were covered with specific test loops. The overall operation was tested with the execution of a MIMO-OFDM baseband program covering both basic instruction set and SIMD instructions.

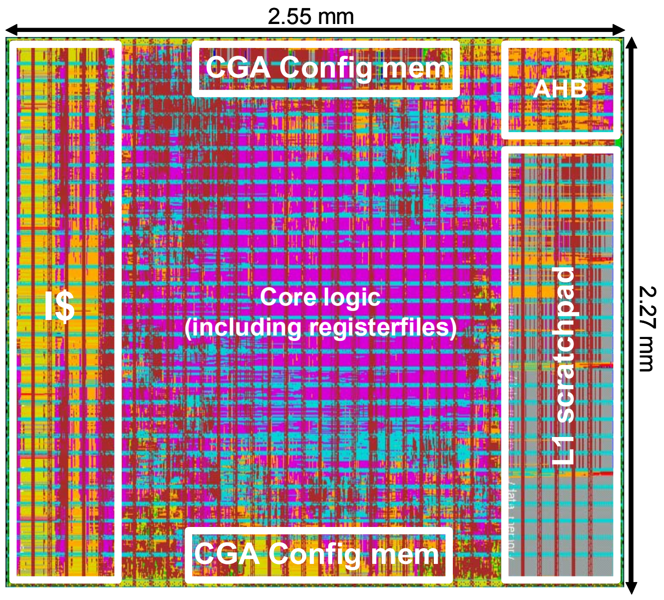

Fig. 4 Processor layout and main building blocks

The RTL was then synthesized with Synopsys Design CompilerTM without wire-load model but with 20% timing over-constraint. Physical synthesis tools such as Synopsys Physical CompilerTM have been excluded because not leading to significant area or power benefit compared to the proposed strategy. This results from the dominance of the timing constraint. The drawback is that the timing can only be reported after place & route. Clock gating is inserted and optimized during synthesis. 95% of the flip-flops turn out to be clock-gated with the appropriate activation signal. Finally, scan test support and memory BIST logic were inserted.

The resulting netlist was simulated at gate level with the same regression suite, plus specific testbench for scan test and BIST behavior validation. Then, it was used as input for physical design with Cadence SOC EncounterTM. Macros are placed at the periphery as illustrated in Figure 4. Standard cells placement is then optimized, followed by clock tree synthesis and final place & route. After parasitic extraction from the resulting layout, timing and power estimation was carried out. Timing was checked with Synopsys PrimeTimeTM. Power in VLIW and CGA mode was estimated with Synopsys PrimePowerTM based on switching activity traced during gate-level simulation.

C. *Design Results*

The final layout achieves a timing of <2.5 ns in worst case conditions, which makes the operation of the processor at 400MHz possible. The critical path is located in the execution stage of the functional units implementing the pipelined multiplier. The die area reaches 5.79 mm2 including L1, I$ and configuration memories. The silicon occupation in the standard cells area reaches 60% (9-layer back end). The area breakdown is given in Figure 5. One can observe that the memories occupy roughly 50% of the area. The CGA functional units take 29%, followed by the VLIW units (8%) and the global and distributed registerfile (5% and 3%).

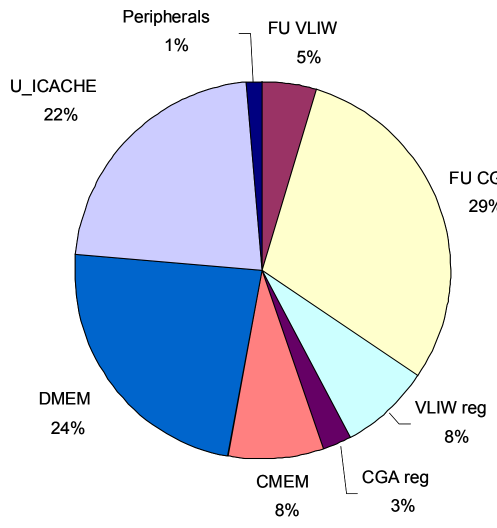

Fig. 5 Processor Area Breakdown

4. PERFORMANCE AND POWER ANALYSIS

In order to evaluate the processor performance and power consumption in its targeted operation conditions, the execution of a MIMO-OFDM inner modem was profiled. Table 2 presents compilation and profiling data on the kernels of both the preamble processing and the data processing. Notice that the whole program is written in ANSI-C and compiled with the DRESC framework. To exploit the 4-way SIMD capabilities, intrinsic functions were introduced in the C code. In total, the preamble processing takes 15,3us, which is higher than the preamble elapsed time (8us). This introduces a latency of 7.3us but do not hamper the throughput. For packet data processing, loop merging is used so that two symbols are processed in parallel, resulting in a processing time of 3.8us per symbol, which is lower than the symbol elapsed time; hence, guaranteeing real-time processing of a packet.

The processor executes the MIMO-OFDM running mostly in CGA mode: for the data processing about 60% of the time is spent in CGA mode, and for the preamble processing, it is about 72% of the time. For the different kernels, the main modes in which they are running are indicated in Table 2, as well as the IPC obtained by running the kernel in that mode. When a kernel is presented as running in CGA mode, this means that the computations are all performed in a loop that is mapped onto the ADRES CGA mode. Of course there is still some VLIW code present in those kernels. This VLIW code takes care of the stack frame setup and cleaning in the C procedures implementing the kernel, and of setting up the data for the CGA loop. Some kernels are divided over two loops, and in that case some VLIW code also glues the two CGA loops together. When a kernel is presented as being run in VLIW mode, no loops were present that can be mapped onto ADRES's CGA mode. And in case the kernel is presented as "mixed", this means that it contains a loop that is mapped to CGA mode, but that there is also a significant part of VLIW mode preprocessing or postprocessing going on in the kernel.

On our ADRES core, the number of instructions executed per cycle (IPC) should be much higher in CGA mode, which has 16 functional units at its disposal for executing software-pipelined loops, than in VLIW mode, which has only 3 issue slots available to execute largely sequential code. The IPC numbers in Table 2 confirm this. On average, CGA-mode kernels obtain an IPC of 10.31. So a utilization factor of 10.32/16 = 64.5% is obtained in CGA mode. In pure VLIW mode code, the average IPC is 1.94, totalling a utilization of 1.94/3 = 64.6%. While it is coincidental that these two utilization factors are so close together, the fact that they are in the same range demonstrates that our ADRES core is a well balanced mix of VLIW and CGA resources.

The processor power consumption is estimated based on gate-level activity profile obtained from the gate-level simulated execution of the aforementioned program. Static and dynamic power consumptions are distinguished, as well as dynamic power in non-kernel (VLIW) mode and in kernel (CGA) mode. Results are depicted in Table 3. Power are given for typical design corner (V=1V, nominal process, T=25C). Leakage is extrapolated to typical leakage corner (V=1V, nominal process, T=65C). The average power when executing the reference program is 220mW.

TABLE 3
PROCESSOR POWER CONSUMPTION

	Active (typical)	*Leakage (typical)*	*Leakage (T=65C)*
VLIW	75 mW	12.5 mW	*25 mW*
CGA	310 mW	12.5 mW	*25 mW*
Average	220 mW	12.5 mW	*25 mW*

TABLE 2
PROFILING OF THE SDM-OFDM CODE

	kernel	*mode*	*IPC*	*cycles*
Preamble Processing	acorr	mixed	3.47	122
	fshift	CGA	12.16	211
	xcorr	CGA	9.15	280
	acorr	mixed	3.47	194
	fshift	CGA	12.16	678
	fft	CGA (2x)	10.36	712
	remove zero carriers	VLIW	1.10	76
	freq offset estimation	CGA	6.32	314
	freq offset compensation	mixed	4.48	424
	sample ordering	VLIW	1.61	210
	SDM processing	CGA (2x)	9.90	1540
	sample reordering	VLIW	2.69	256
	equalize coeff. calc.	CGA	8.38	636
	non-kernel code	VLIW	1.69	452
	total		8.05	6105
				= 15.3 us
Data Processing	fshift	CGA	13.33	378
	fft	CGA (2x)	11.46	493
	data shuffle	VLIW	2.60	100
	tracking	VLIW	1.83	117
	comp	CGA	9.00	219
	demod QAM64	CGA	12.04	224
	total		10.34	1531
				= 3.8 us

A further breakdown of the active power consumption identifies which parts of the design consume most. In both non-kernel and kernel mode, a significant share (respectively 28% and 38%) goes to the interconnect sub-system which include buffers, multiplexers and pipeline registers between the CGA functional units. In non-kernel, 22% and 21% further go to the VLIW functional units and global register file respectively; 13% to the L1, 10% to the I$. The CGA units that are idle consume 2%. In kernel mode, after the interconnect, the CGA functional units, configuration memories and L1 dominates with respectively 25%, 13% and 10% of the power. The global and distributed register files counts for 8% and 2%. The idle VLIW units and I$ consume a remaining 5%.

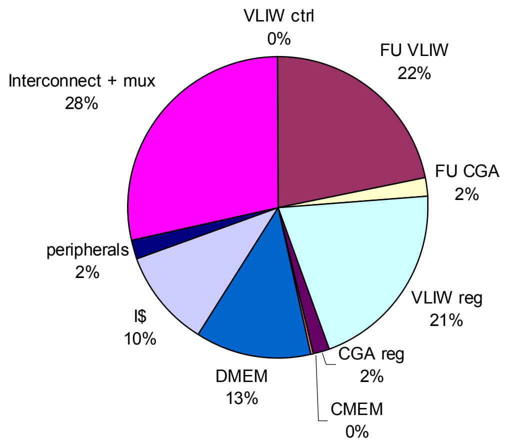

Fig. 6a Power consumption breakdown in VLIW mode

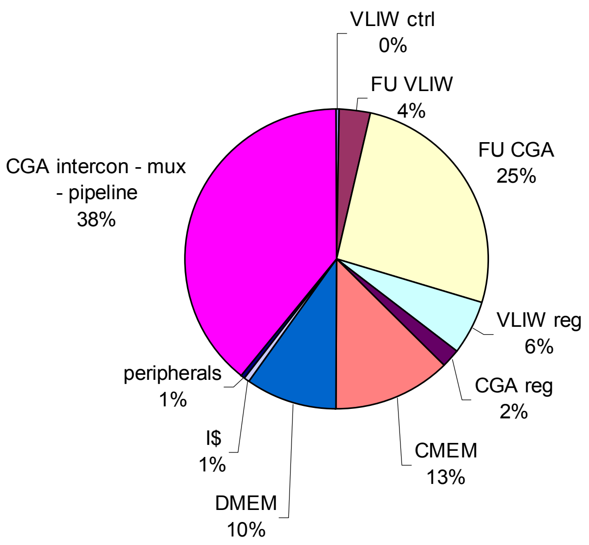

Fig. 6b Power consumption breakdown in CGA mode

5. CONCLUSIONS

The design of a hybrid CGA-SIMD baseband processor for SDR is presented. The processor, designed in TSMC 90G technology according to a dual-VT standard-cells flow, achieves a clock frequency of 400MHz in worst case conditions (corresponding to 25,6GOPS) and occupies 5.79 mm^2 including L1, I$ and configuration memories. Two operations modes are foreseen: non-kernel VLIW and kernel CGA mode. In non-kernel mode, active power consumption is estimated to 75.4mW while it reaches 310mW in CGA-mapped loops. Static power consumption is 25mW at 65C, which can be reduced in standby thanks to the substrate-biasing support of the considered standard-cells library. The processor is shown to be able to execute 20MHz 2x2 SDM-OFDM baseband processing, achieving 100Mbps+ throughput, consuming 220 mW.

6. REFERENCES

[1] J. Glossner, D. Iancu, L. Jin, E. Hokenek, M. Moudgill, "A software-defined communications baseband design," *Communications Magazine, IEEE*, Vol. 41, pp. 120-128, 2003.

[2] K. Van Berkel, H. F., P. Meuwissen, K. Moeren, M. Weiss, "Vector Processing as an Enabler for Software Defined Radio in Handsets for 3G+WLAN Onwards," *SDR Forum Technical Conference*, 2004, pp. 125-130.

[3] L. Yuan, L. Hyunseok, M. Woh, Y. Harel, S. Mahlke, T. Mudge, C. Chakrabarti, K. Flautner, "SODA: A Low-power Architecture For Software Radio," pp. 89-101, 2006.

[4] A. Lodi, A. Cappelli, M. Bocchi, C. Mucci, M. Innocenti, C. De Bartolomeis, L. Ciccarelli, R. Giansante, A. Deledda, F. Campi, M. Toma, R. Guerrieri, "XiSystem: a XiRisc-based SoC with reconfigurable IO module," *Solid-State Circuits, IEEE Journal of*, Vol. 41, pp. 85-96, 2006.

[5] H. Singh, L. Ming-Hau, L. Guangming, F. J. Kurdahi, N. Bagherzadeh, E. M. Chaves Filho, "MorphoSys: an integrated reconfigurable system for data-parallel and computation-intensive applications," *Computers, IEEE Transactions on*, Vol. 49, pp. 465-481, 2000.

[6] B. Mei, S. Vernalde, D. Verkest, H. De Man, R. Lauwereins, "Exploiting loop-level parallelism on coarse-grained reconfigurable architectures using modulo scheduling," *Computers and Digital Techniques, IEE Proceedings-*, Vol. 150, pp. 255-261, 2003.

[7] L. Van der Perre, B. Bougard, e. al., "Architectures and Circuits for Software defined Radios: Scaling and Scalability for Low Cost and Low Energy," *International Solid State Circuits Conference (ISSCC)*. San Francisco, CA, 2007.

[8] B. Bougard et al., “Energy Efficient Software Defined Radio Solutions for MIMO-based Broadband Communication”, Proc. European Signal Processing Conference, Poznan, Sept. 2007