RDSP: A RISC DSP based on Residue Number System

Ricardo Chaves
IST/INESC-ID
R. Alves Redol,9, 1000-029 Lisboa, Portugal
rjfc@sips.inesc-id.pt

Leonel Sousa
IST/INESC-ID
R. Alves Redol,9, 1000-029 Lisboa, Portugal
las@inesc-id.pt

Abstract

This paper is focused on low power programmable fast Digital Signal Processors (DSP) design based on a configurable 5-stage RISC core architecture and on Residue Number Systems (RNS). Several innovative aspects are introduced at the control and datapath architecture levels, which support both the binary system and the RNS. A new moduli set $\{2^n-1, 2^{2n}, 2^n+1\}$ is also proposed for balancing the processing time in the different RNS channels. Experimental results, obtained trough RDSP implementation on FPGA and ASIC, show that not only a significant reduction in circuit area and power consumption but also a speedup may be achieved with RNS when compared with a binary DSP.

1 Introduction

In the last few years signal processing has assumed a growing importance on our way of living. Signal processing applications demand cheaper, faster and especially low power processors, due to the emerging use of portable devices. With this goal in mind, a new programmable DSP is developed in this paper. This is the first proposal of a completely programable DSP based on RNS arithmetic units, after the simplified architecture reported in [7]. RNS allows the usage of smaller, parallel and carry free and, consequently, faster arithmetic units than the binary system [8]. On the other hand, RNS structures are less regular, more complex and additional conversion units are required for conversion between RNS and the commonly used binary system.

The main characteristics of the proposed processor, named RDSP, are: *i)* the existence of an accumulator and a 2-stage multiplication unit, in order to support MAC instructions, commonly found in signal processing; *ii)* a data path that supports both the binary and the RNS systems and efficient internal converters/instructions for conversion between the two systems; *iii)* it comprises configurable static/dynamic branch prediction techniques for control hazards; *iv)* the conditioned execution of all instructions by any of the processor's flags, without penalty, along with a complete forwarding mechanism. The processor core has also been developed with a modular approach, allowing an easy insertion of additional arithmetic units.

In order to analyze the performance of the proposed fixed-point DSP, three different versions were developed: *i)* one using a 32-bit binary system; *ii)* the second one using 32-bit dynamic range RNS; *iii)* a third hybrid version using both RNS and a 32-bit binary system. A new moduli set, $\{2^n-1, 2^{2n}, 2^n+1\}$ ($n = 8$), is also proposed for calculation time balancing between the three RNS arithmetic channels.

Behavioral and structural descriptions of the RDSP were made in VHDL considering two different target technologies, namely Field Programmable Gate Arrays (FPGA) and Application Specific Integrated Circuits (ASIC). Experimental results show the significance in using RNS to design efficient DSPs, namely to reduce cost and power consumption and, simultaneously, increase pipeline frequency.

This paper is organized as follows. Section 2 describes the main features of the RDSP architecture. The new arithmetic units for the proposed RNS moduli set are shown in section 3. Experimental results obtained by implementing the RDSP in different technologies are provided in section 4. Finally, conclusions are presented in section 5.

2 The RDSP

The RDSP is a pipeline RISC processor with 5 stages: Instruction Fetch (IF); Instruction Decode (ID); Instruction Execution 1 (EX1); Instruction Execution 2 (EX2); and Write back (WB). Moreover it has a modular microarchitecture that allows the easy insertion and removal of arithmetic units in the two execution stages. The RDSP data path has been developed in order to support both a 32-bit binary system and an equivalent RNS system with 33 bits, independently of the used moduli set, or any other numbering system with at most 33 bits (see figure 1). All possible data forwarding mechanisms were implemented in the

Proceedings of the Euromicro Symposium on Digital System Design (DSD'03)
0-7695-2003-0/03 $17.00 © 2003 IEEE

RDSP, and the few remaining data hazards are solved by introducing a single stall. An assembly language was defined for the processor and an assembler was developed to simplify its programming.

2.1 RDSP General Structure

Three versions of the RDSP, all with specialized memory addressing and MAC instructions, were developed in order to evaluate the architecture performance with different arithmetic units. The first version only uses the binary numbering system, like commercial programmable DSP. The second one uses the RNS for arithmetic operations. Finally the third version of the RDSP not only uses RNS arithmetic units and instructions but it also supports a subset of the 32-bit binary version operations, accomplished by the reutilization of the 2^{2n} RNS channel hardware. All three versions have binary addition and logical instructions (with flag updating) for control and memory addressing.

Since arithmetic units in the processor depend on the implemented version, an arithmetic Address Generation Unit (AGU) is always present in the RDSP. This arithmetic unit is used, not only to generate memory addresses, but also to compute addition and subtraction instructions that do not use the accumulator. The only arithmetic instructions that update flag registers are the ones performed in the AGU. The flag register can also be updated by the Logical Unit (LU). This unit is also an intrinsic part of the RDSP, performing all the logic operations and constant manipulations. The LU is able to perform bit oriented operations, bit wise logical operations and word shifting. Flags updated by these two units can be used to control program flow, either by controlling branch behavior or by conditioning instructions execution.

A data register bank with a dual input port is required since the RDSP has two execution stages, which can write in the register bank in the same clock cycle. When both execution stages try to write the same register, the WAW hazard is solved by writing the most recent value generated in the first stage. To minimize the number of memory accesses, the data register bank was generated with 32 registers of 32 or 33 bits, depending on the adopted number system.

Having in mind the usage of the RDSP in real and autonomous signal processing applications, a configuration register bank was introduced and can be accessed in parallel with the data register bank. Configuration registers are used for setting the RDSP operation, for example the branch predictor or the interruption register. The Input/Output (I/O) ports are also mapped in this register bank to access external peripherals devices.

2.2 Memory Addressing

The RDSP memory, located in the second execution stage, can only be accessed by register addressing. However, the value provided by the register bank can be manipulated in the AGU located in the first execution stage. Since in signal processing it is usual to have data sequentially stored in memory, the RDSP provides memory addressing with pre and post-increment (or decrement). Register addressing with an 8-bit offset and bit-reversed addressing is also implemented, in order to enhance the execution speed of FFT algorithms.

2.3 Conditional Execution

In most processors the only way to conditionally execute instructions is by using conditional branch instructions or, more recently, by conditioning the execution of an instruction to the value of a register [4].

All the instructions executed in the RDSP can be conditioned to the value of a flag. For example, to calculate the absolute value of a sum one can compute the symmetric value subjected to the condition of the flag that indicates whether or not the addition result is negative ($R_{10} = |R_2 + R_3|$):

$$\begin{array}{ll} & add\ R_{10}\ R_2\ R_3 \\ [\text{neg}] & sub\ R_{10}\ R_0\ R_{10} \quad ; \text{with } R_0 = 0 \end{array}$$

Taking into account that an instruction is only considered executed when the result is written, which corresponds to the commit step in speculative processing, conditioning is accomplished simply by disabling the write on the internal registers or on the memory when the considered flag is false. This conditioning method is valid for all instructions, except for branch instructions.

2.4 Branch Instructions

In the RDSP, branch addresses can be given by an offset to the current program counter or by providing the full address through a data register.

The new program memory address is obtained, either by loading the value read from the data register bank or by adding the desired 16-bit offset. The offset is added to the program counter by a dedicated adder placed in the ID stage, in order to obtain the target address value as soon as possible in the pipeline. However, when the new program counter is loaded there is already a new instruction in the IF stage. A branch delay, of one instruction, is then present in the RDSP.

In order to link branch instructions that are used as routine calls, all branch instructions can save the program counter in any of the data registers when a branch occurs.

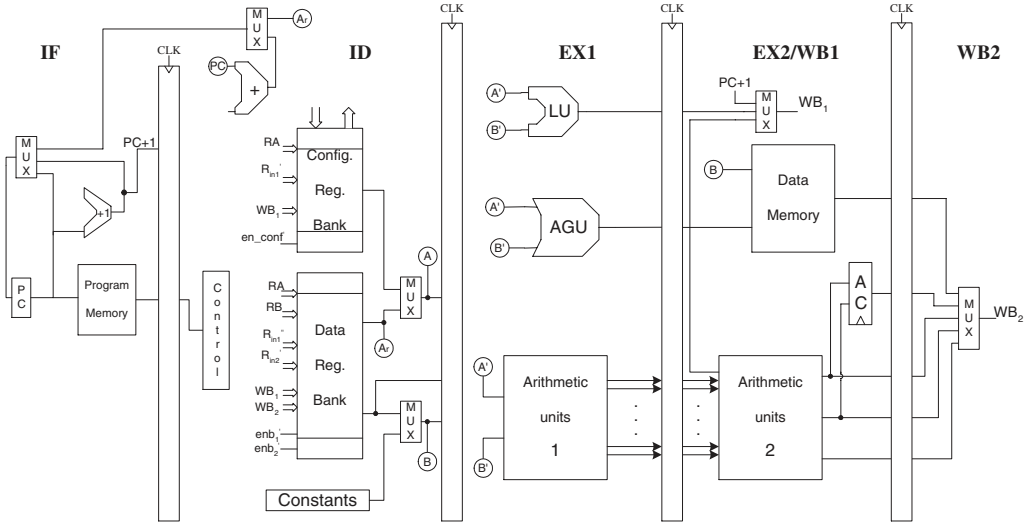


Figure 1. RDSP architecture.

This feature is very useful, since this architecture does not have a stack pointer nor a stack memory.

Because flags are only updated in the second execution stage (for efficiency reasons), control hazards may occur when a conditional branch instruction is preceded by an instruction that updates the tested flag. To minimize these hazards the RDSP has branch prediction mechanisms, which can be of the type branch taken, branch not taken or dynamic prediction performed by a four state Branch Prediction Buffer (BPB). The choice of the perdition type is selected via the PERD_REG register located in the configuration register bank.

2.5 Execution Stages and Arithmetic Units

With the modular structure of the two execution stages, all three versions of the RDSP were created simply by inserting the arithmetic units in the execution stages, connecting them to the data path and to the write back multiplexer and adjusting the data forwarding mechanisms. The binary version consists of a 32-bit adder and multiplier with or without accumulation, in which products are represented with 64 bits. Two MAC instructions exist, one that accumulates the 32 MSB while the other accumulates the 32 LSB. The RNS version also supports addition and multiplication operations, with and without accumulation, and conversion units between the numbering systems are also included. The mixed version provides all the RNS instructions and hardware, plus some additional hardware in the 2^{2n} channel in order to obtain 32-bit binary results.

In the Binary version of the RDSP, the signed multiplication of 32 by 32 bits can be performed by generating partial products and by applying a Wallace-tree compressor with at most 8 FA in the critical path, followed by a full 64-bit adder to combine the Sum and Carry-bit vectors. To add the value of the accumulator, a $3-2$ compressor is used before the final adder. The accumulator circuit is located in the second execution stage.

Execution stages and arithmetic units for the RNS and mixed versions of the RDSP are presented in the next section.

3 Arithmetic Units for RNS

The residue number systems compute the result in parallel in the several channels of the base [2]. The value of each channel is given by the remainder of the division of the number X by one of the elements of the co-prime set $\{m_1, \ldots, m_j\}$, that constitute the RNS moduli set:

$$x_i =< X > m_i; \quad i = 1, \ldots, j \tag{1}$$

The value of the binary number X can by uniquely represented by $(x_1, \ldots, x_j)$, with:

$$X \epsilon [0, M[\ ;\ M = \prod_{1}^{j} m_i \tag{2}$$

In RNS, addition and multiplication of numbers, represented in a common moduli set, can be independently performed, in each one of its channels, and thus by separate arithmetic units. This distribution originates carry free operations that can be performed in parallel by smaller arithmetic units, which also increases system modularity and fault tolerance [6].

A number represented in RNS $(r_1, \ldots, r_j)$ can be reconverted to binary representation by [2]:

$$X = \left\langle \sum_{i=1}^{3} \hat{m}_i \left\langle \frac{r_i}{\hat{m}_i} \right\rangle m_i \right\rangle_M \quad (3)$$

3.1 The new moduli set $\{2^n - 1, 2^{2n}, 2^n + 1\}$

A new moduli set $\{2^n - 1, 2^{2n}, 2^n + 1\}$ is proposed, instead of the traditional and most frequently used moduli set $\{2^k - 1, 2^k, 2^k + 1\}$ [2, 8].

In order to obtain a dynamic range equivalent to a binary system with 32 bits, it is required that $k = \lceil \frac{32}{3} \rceil = 11$. With the new moduli set $\{2^n-1, 2^{2n}, 2^n+1\}$, only $n = \lceil \frac{32}{4} \rceil = 8$ is required, which is a round number. In fact, the value of n is always a round number and a power of 2, considering that the binary dynamic range is also given by a power of two. This is an important issue since most fast adder structures are optimized for operands with a number of bits which are powers of two [10, 4]. Therefore, the resulting RNS adder structures are also optimized, which does not occur with the traditionally used moduli set.

In this new moduli set the binary channel (2^{2n}) has twice as much bits as the other two channels. However this is not a disadvantage since the $2^n \pm 1$ arithmetic units are more complex. Using fast parallel-prefix adders, the time for a binary addition with 2n bits, $T_0 = T_p + \log_2(2n)$, is approximately the same as an n-bit modulo adder for the $2^n \pm 1$ channels, $T_0 \simeq T_p + \log_2(n) + 1$ [11, 10], thus originating balanced structures with respect to time. Note that the $2^n + 1$ channel still requires a CSA structure in order to perform the $2^n + 1$ addition compensation [2].

This adders optimization is not enough to consider the new moduli set better than the traditional one, since the critical path is usually on the multipliers. However, considering the usage of Wallace-tree structures for compressing the partial products, the difference between the 2^{2n} channel compressor and the $2^n \pm 1$ channels compressors only corresponds to two full adders (see able 1) [9].

Table 1. Number of FA stages in a Wallace-tree structure.

j	3	4	5-6	7-9	10-13	14-19	...
W(j)	1	2	3	4	5	6	...

Multipliers for modulo $2^n \pm 1$ need an additional Carry Save Adder (CSA), also designated as a $3 - 2$ compressor, between the compression matrix and the final adder, and comprise some extra complexity for the processing of the n^{th} bit of the operands [2, 11, 5]. Thus, the RNS multiplication may also be considered as being balanced. Even more, considering that with the old moduli set $\{2^k-1, 2^k, 2^k+1\}$ the number of FA in a Wallace-tree multiplication matrix of the $2^n \pm 1$ channels would be 5 (with k=10 or 11), instead of 4 (n=8), for the new moduli set.

3.2 Conversion Units

Although the addition and multiplication units of the old moduli set can be directly used for the new moduli set, new conversion units between RNS and binary systems have to be developed.

For signal processing it is necessary to support signed values. This can be easily accomplished by adjusting the signed binary value to the RNS representation [8]:

$$x_i =< X >_{m_i} \text{ for } X > 0 \quad (4a)$$

$$x_i =< M + X >_{m_i} \text{ for } X \leq 0 \quad (4b)$$

which can be used to represent signed numbers in RNS belonging to the set $[-M/2, M/2[$. In the binary to RNS conversion it is also necessary to take in consideration the fact that the RNS dynamic range is only $M = \prod_1^j m_i = 2^{4n} - 2^{2n}$ while the binary system has a dynamic range of $2^{4n} - 1$. Consequently this conversion requires a saturation mechanism, given by:

$$\Gamma = X \text{ for } X \epsilon [-M/2, M/2[\quad (5a)$$

$$\Gamma = M/2 - 1 \text{ for } X \geq M/2 \quad (5b)$$

$$\Gamma = -M/2 \text{ for } X < -M/2 \quad (5c)$$

The resulting conversion units, described below, were designed bearing in mind that they are to be inserted in the two execution stages of the RDSP.

3.2.1 Binary to Residue Converter

The first operation to be performed in the binary to RNS conversion is the addition of the constant M when the binary number is negative, i.e., the most significant bit assumes the value '1' (equation 4). After this step, equation 5 is used to compute Γ by applying two comparators and one multiplexer. The RNS adapted number Γ can thus be represented as:

$$\Gamma = \sum_{i=0}^{4n-1} \gamma_i 2^i = N_3 2^{3n} + N_2 2^{2n} + N_1 2^n + N_0 \quad (6)$$

Using an identical methodology as the one in [2], it results, for $m_2 = 2^{2n}$:

$$\begin{aligned} x_2 &= < N_3 2^{3n} + N_2 2^{2n} + N_1 2^n + N_0 >_{2^{2n}} \\ &= N_1 2^n + N_0 \end{aligned} \quad (7)$$

for $m_1 = 2^n - 1$:

$$\begin{aligned} x_1 &= < N_3 2^{3n} + N_2 2^{2n} + N_1 2^n + N_0 >_{m_1} \\ &= < N_3 + N_2 + N_1 + N_0 >_{m_1} \\ &= << N_3 + < N_2 + N_1 + N_0 >_{m_1} >_{m_1} >_{m_1} \end{aligned} \quad (8)$$

and for $m_3 = 2^n + 1$:

$$\begin{aligned} x_3 &= < N_3 2^{3n} + N_2 2^{2n} + N_1 2^n + N_0 >_{m_3} \\ &= < \overline{N_3} + 2 + N_2 + \overline{N_1} + 2 + N_0 >_{m_3} \\ &= < 2 + < 1 + \overline{N_3} + < 1 + N_2 + \overline{N_1} \\ &\quad + N_0 >_{m_3} >_{m_3} >_{m_3} \end{aligned} \quad (9)$$

This conversion from the adapted binary value to RNS can thus be efficiently performed with $3-2$ compressors and full adders modulo $2^n \pm 1$, as depicted in figure 2. Note that the last adder modulo $2^n + 1$ adds 2 units instead of the value 1, typically added by modulo $2^n + 1$ adders.

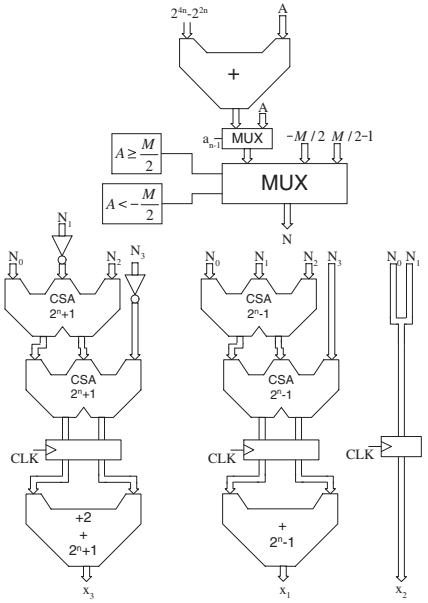


Figure 2. Signed RNS encoder.

3.2.2 Residue to Binary converter

The RNS to binary conversion is performed based on the Chinese Remainder Theorem (CRT) [1]:

$$X = \left\langle \sum_{i=1}^{3} \hat{m}_i \left\langle \frac{x_i}{\hat{m}_i} \right\rangle m_i \right\rangle_M \quad (10)$$

where $\hat{m}_i = M/m_i$ and $\left\langle \frac{1}{\hat{m}_i} \right\rangle_{m_i}$ is the multiplicative inverse of $\hat{m}_i$. Equation 10 can be written as [2]:

$$X + MA(X) = \sum_{i=1}^{3} \hat{m}_i \left\langle \frac{1}{\hat{m}_i} \right\rangle_{m_i} x_i \quad (11)$$

For the proposed moduli set, it can be proved that the multiplicative inverse of $\hat{m}_i$, $i = 1, 2, 3$, are [3]:

$$\left\langle \frac{1}{\hat{m}_1} \right\rangle_{m_1} = 2^{n-1} \quad (12a)$$

$$\left\langle \frac{1}{\hat{m}_2} \right\rangle_{m_2} = 2^{2n} - 1 \quad (12b)$$

$$\left\langle \frac{1}{\hat{m}_3} \right\rangle_{m_3} = 2^{n-1} \quad (12c)$$

Replacing these values in equation 11:

$$\begin{aligned} X + MA(X) = \quad & 2^{2n}(2^n - 1)(2^{n-1})x_3 \\ & + (2^{2n} - 1)(2^{2n} - 1)x_2 \\ & + 2^{2n}(2^n + 1)(2^{n-1})x_1 \end{aligned} \quad (13)$$

and dividing X by 2^{2n}, it is possible to obtain:

$$X = 2^{2n} \left\lfloor \frac{X}{2^{2n}} \right\rfloor + x_2 \quad (14)$$

where:

$$\begin{aligned} \left\lfloor \frac{X}{2^{2n}} \right\rfloor = \Bigg\langle & \underbrace{\left\langle (2^{2n-1} + 2^{n-1}) x_1 \right\rangle_{2^{2n}-1}}_{A} \\ & - 2^n x_3 + \underbrace{\left\langle (2^{2n} - 2) x_2 \right\rangle_{2^{2n}-1}}_{B} \\ & + \underbrace{\left\langle (2^{2n-1} + 2^{n-1}) x_3 \right\rangle_{2^{2n}-1}}_{C} \Bigg\rangle_{2^{2n}-1} \end{aligned} \quad (15)$$

Finally, simplifying the partial expressions A, B, C, $-2^n x_3$:

$$\begin{aligned} A &= x_{1,0} x_{1,n-1} \ldots x_{1,1} x_{1,0} x_{1,n-1} \ldots x_{1,1} \\ B &= \overline{x}_{2,2n-1} \ldots \overline{x}_{2,0} \\ C &= x_{3,x} x_{3,n-1} \ldots x_{3,1} x_{3,x} x_{3,n-1} \ldots x_{3,1} \\ -2^n x_3 &= \overline{x}_{3,n-1} \ldots \overline{x}_{3,0} 1 \ldots 1 \overline{x}_{3,n} \end{aligned} \quad (16)$$

the upper 2^{2n} bits of X can thus be calculated with two $3-2$ compressors and one full adder modulo $2^{2n}-1$:

$$\left\lfloor \frac{X}{2^{2n}} \right\rfloor = \left\langle \langle C + B + A \rangle_{2^{2n}-1} + (-2^n x_3) \right\rangle_{2^{2n}-1} \quad (17)$$

To convert the obtained binary value, X, to a signed representation, M should be subtracted from the result, which is the same as adding 2^{2n}, when the signed binary value is negative. Since the signal of the final result is only known after this addition has been performed, the multiplexer is located after the adder and its operation is controlled by the MSB of sum (see figure 3).

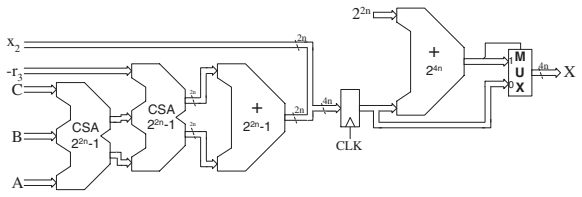


Figure 3. Signed RNS decoder.

3.3 Addition and Multiplication

The new moduli set uses the same addition and multiplication structures as the traditional one. Only the existence of accumulation instructions requires the introduction of specialized circuitry. For the 2^{2n} and 2^n-1 channels, the accumulation can be implemented with a multiplexer and a $3-2$ compressor before the final modulo adder.

For the 2^n+1 channel, the accumulation is more complicated, since the $3-2$ compressor adds one extra unit and only has an n-bit input for another $n+1$-bit operand. When the value of the accumulator is less than 2^n, or no accumulation is required (when the input corresponding to the accumulator takes the 0 value) the accumulator value is placed in the third input of the $3-2$ compressor, as illustrated in figure 4. Whenever the accumulator value is equal to 2^n ($< -1 >_{2^n+1}$) the carry-in signal is complemented ($C_0 = \overline{c_{out}} \oplus ac$). If the carry-in is equal to 1, then complementing it (to 0) is the same as subtracting 1 ($-1 =< 2^n >_{2^n+1}$). If the carry-in signal is equal to 0, then complementing it is equivalent to add 1; in this case, the value -2 ($=< 2^n - 1 >_{2^n+1}$) is placed in the extra input of the $3-2$ compressor and thus the value -1 ($=< 2^n - 1 - 2^n >_{2^n+1}$) is added. This algorithm is represented in the equation 18 and in the table 2. Like with all other modulo 2^n+1 adders, an extra 1 is added to the final result.

$$\begin{aligned} \langle a + b + acumul + 1 \rangle_{2^n+1} &= \langle a + b + val \rangle^*_{2^n+1} \\ &= \left\langle S + 2\, C_{n-2:0} + \overline{C_{n-1} \oplus ac} \right\rangle_{2^n+1} \end{aligned} \quad (18)$$

Table 2. Inputs for the 2^n+1 acumulator.

accumulation	acumul	C_{in}	val	ac
no	–	–	0	0
yes	$\neq 2^n$	–	acumul	0
yes	$= 2^n$	0	0	1
yes	$= 2^n$	1	2^n-1	1

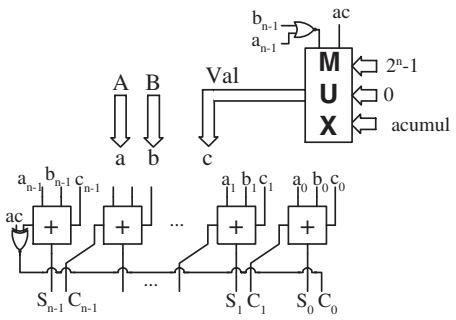


Figure 4. Modulo 2^n+1 accumulation.

In the RNS version of the RDSP, all calculations in the RNS are performed in independent arithmetic units for the three channels. The arithmetic units in the binary channel (2^{2n}) are identical to the ones described for the binary version except, it uses an unsigned multiplier of 16-bit operands with the result also expressed in 16 bits. A similar structure is used for the 2^n-1 channel, with modulo 2^n-1 arithmetic units.

The 2^n+1 channel can also be structured in a similar way as the binary channel. However, since each adder adds an extra a compensation factor in the addition and in the multiplication [2] has to be adjusted to the number of adders. The obtained architecture in shown in figure 5.

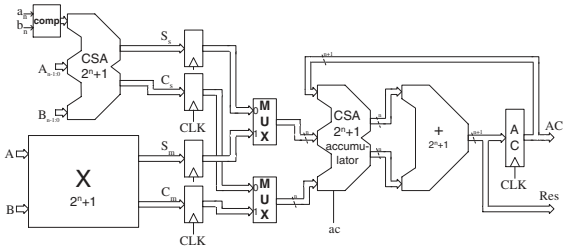


Figure 5. Addition and multiplication modulo 2^n+1.

The channels are disposed according to the order ($2^n+1, 2^n-1, 2^{2n}$), which allows the usage of the lower 16 bit as binary unsigned numbers, disregarding the value of the

upper 17 bits. The RNS adders are shared with the RNS encoder.

The mixed version of the RDSP is similar to the RNS version, but the 2^{2n} channel is expanded. Both the adder and the 2^{2n} modulo multiplier are extended to 32 bits. However, the multiplier only accommodates 32 bits by 16 bits. Thus, the number of FA per column does not increase, as well as the compression delay that is identical to the one in the RNS version. The 16 lower bits of the 32-bit operand can be optionally set to zero. Thus, 32 by 32 bits multiplications can be performed with two instructions, i.e., $A \times B = A \times B_L + B_H \times A_L$. In this version, the RNS coexists with the binary system without increasing the RDSP machine cycle.

4 Experimental Results

The RDSP was developed having in mind two distinct implementation technologies: a programmable logic device (a FPGA VirtexE2000 from Xilinx in a RC 1000-PP Celoxica development board) and an ASIC (based on Standard Cell technology, using $0.25\mu m$ CMOS from UMC).

The FPGA, in the development board, has been especially chosen to implement a prototype of the RDSP, as a fast and reliable form to test the processor. The Celoxica development board includes four memory banks, with 4×512 Kbytes, and a PCI protocol to communicate with the PC through DMA access. The external memory banks were used for both the data memory and the program memory.

The FPGA occupation rate and the maximum operating frequency achieved in the RDSP implementation are shown in table 3. In all three versions of the RDSP the critical path

Table 3. FPGA Implementation.

	RNS	Mixed	Binary
Max. Freq. (MHz)	29	29	27
FPGA Occupation (%)	15	20	24

was located in the data register bank. Due to the location of the critical path, no speedup is obtained with RNS. However, it is noticeable a significant reduction in the FPGA occupation when using RNS. As expected the occupation of the mixed version of the RNS is half way between the RNS and the binary version.

Having in mind low cost programmable devices usage, a FPGA VirtexE300 is sufficient to implement the RDSP with only RNS, while a VirtexE400 is required to implement the binary version of the RDSP.

Considering a technology without configurable pre-optimized components, the three versions of the RDSP were developed in a $0.25\mu m$ Standard Cell CMOS technology. This kind of technology allows a more controlled translation from a VHDL structural description. Experimental results are shown in table 4. In this table, the maximum frequency achieved, the circuit area and the power consumption values are shown for all the three versions of the RDSP.

Table 4. CMOS results.

	RNS	Mixed	Binary
Frequency (MHz)	257	256	224
Area (mm^2)	1,52	1,65	2,18
Power @100Mhz (mW)	285,5	315,8	363,7
Δ Frequency	+15%	+14%	0%
Δ Area	-30%	-25%	0%
Δ Power @100Mhz	-22%	-13%	0%

In this technology the RNS version has a clear advantage in comparison to the binary version. Not only the circuit area is reduced by 30%, but also the frequency obtained with RNS is 15% higher than its binary equivalent. The reduction in power consumption is approximately 22%. This percentage is less than the one obtained for the area, mainly because of the presence of conversion units and the fact that the arithmetic units perform calculations in parallel and consequently, more switching activity is registered.

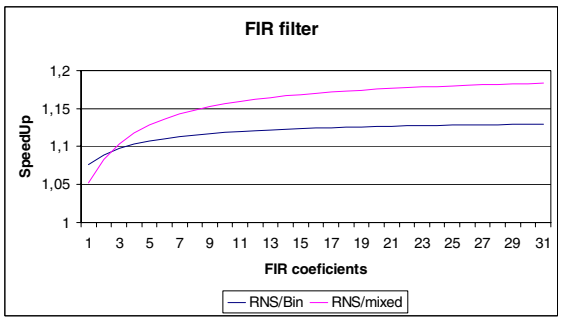
The mixed version of the RDSP is able to have approximately the same performance as the RNS version concerning the working frequency, and an intermediate area and power consumption between those obtained for the binary and RNS versions.

The frequency values are not conclusive enough to compare the performance of the three versions of the RDSP, since the usage of the RNS arithmetic units also implies the usage of conversion instructions. The performance of the different versions of the RDSP was then comparatively evaluated by programming two different signal processing algorithms: FIR filters and matrix multiplications. For the mixed version of the RDSP results were obtained using only the binary units, since the performance obtained when using the RNS units is the same as the one obtained for the RNS version.

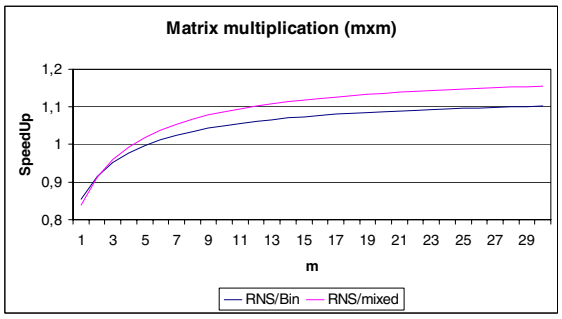
Figure 6 illustrates the speedup achieved by the RNS version, compared with the binary and mixed versions. Even with the conversion instructions, the RNS version is able to achieve significant speedups, up to 12% in FIR filters. As FIR filters dimensions or matrix size increases the conversion instructions become less relevant, wich allow higher speedup values for the RNS version.

5 Conclusions

In this paper a new RISC DSP based RNS is proposed, with a new moduli set $\{2^n - 1, 2^{2n}, 2^n + 1\}$. The modu-



(a) FIR



(b) Matrix multiplication

Figure 6. RNS versus binary Speedup.

lar structure of this 5 stage DSP allowed a straightforward development of three versions of the processor, which were used to analyze the benefits of using an RNS system in a programmable DSP. Experimental results show that the usage of the RNS system allows an increase of the maximum operating frequency with a significant reduction in the processors dimension and power consumption. With the proposed moduli set RNS channels are more balanced and a better reutilization of the hardware when binary and RNS systems coexist in the same processor is achieved.

References

[1] S. Andraos and H. Ahmad. A new efficient memoryless residue to binary converter. *IEEE Transactions on Circuits and Systems*, 35(11), November 1988.

[2] A.S.Ashur, M.K.Ibrahim, and A. Aggoun. Novel RNS structures for the moduli set $(2^n - 1, 2^n, 2^n + 1)$ and their application to digital filter implementation. *Signal Processing*, 46, 1995.

[3] R. Chaves. RDSP: A digital signal processor with suport for residue arithmetic. Master's thesis, Instituto Superior Tecnico, Lisboa, 2003. written in portuguese.

[4] J. Hennessy and D. Patterson. *Computer Architecture : a Quantitative Approach*. Morgan Kaufmann Publishers, 3 edition, 2002.

[5] Y. Ma. A simplified architecture for modulo $(2^n + 1)$ multiplication. *IEEE Transactions on Computers*, 47(3), March 1998.

[6] A. B. Premkumar. An RNS converter in $2^n + 1, 2n, 2^n - 1$ moduli set. *IEEE Transations on Circuits and Systems–II: Analog and Digital Signal Processing*, 39(7), July 1992.

[7] J. Ramrez, A. Garca, P. G. Fernndez, and A. Lloris. FPL implementation of a SIMD RISC RNS-enabled DSP. *Proc. of the 4th World Multiconference on Circuits, Systems, Communications and Computers*, July 2000.

[8] M. A. Soderstrand, W. K. Jenkins, G. A. Jullion, and F. J. Taylor. *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*. IEEE Press, 1986.

[9] Z. Wang, G. Jullien, and W. Miller. An efficient tree architecture for modulo $2^n + 1$ multiplication. *Journal of VLSI Signal Processing*, August 1996.

[10] R. Zimmermann. *Binary adder architectures for cell-based VLSI and synthesis*. PhD thesis, Swiss Federal Institute of Technology, 1997.

[11] R. Zimmermann. Efficient VLSI implementation of modulo $(2^n \pm 1)$ addition and multiplication. *Proc. IEEE Symp. on Computer Arithmetic*, pages 158–167, April 1999.