Defence Research and Development Canada
Recherche et développement pour la défense Canada

DEFENCE R&D DÉFENSE

RTL Design of a Generic Pseudonoise Generator

Jean-François Beaumont

Defence R&D Canada – Ottawa
TECHNICAL MEMORANDUM
DRDC Ottawa TM 2004-176
September 2004

Canada

RTL design of a generic pseudonoise generator

Jean-François Beaumont

Defence R&D Canada – Ottawa

Technical Memorandum

DRDC Ottawa TM 2004-176

September 2004

Abstract

Pseudonoise (PN) generators have many applications in the field of digital signal processing, especially in wireless communication systems. They are usually implemented in hardware through the use of Linear Feedback Shift Registers (LFSR) in either Application Specific Integrated Circuits (ASICs), Field Programmable Gate Array (FPGAs) or Complex Programmable Logic Devices (CPLDs).

Although the methods and techniques to code the Register Transfer Level (RTL) algorithm with a Hardware Description Language (HDL) are currently well known in the design community, this technical memorandum presents an innovative code implementation. This new algorithm is a generic PN generator that takes as input the polynomial and its associated degree and, using current synthesis tools, generates a compact area circuit. Results show that the generated circuit does not consume more logic resources than a traditional hard-coded PN generator. The net advantage of using this new algorithm over its predecessor is significant savings in terms of design time, verification time and hence, development costs.

Résumé

Les générateurs de bruit pseudoaléatoire ont plusieurs applications dans le domaine du traitement de signal numérique, particulièrement dans les systèmes de communication sans-fil. Ils sont généralement mis en oeuvre sous forme matérielle par l'entremise de registres à décalage à rebouclage linéaire soit dans des circuits intégrés spécifiques (ASICs), des matrices prédiffusées programmables (FPGAs) ou des réseaux logiques programmables complexes (CPLDs).

Bien que les méthodes et techniques pour coder l'algorithme au niveau du transfert de registre à registre (RTL) avec un langage de description matériel (HDL) sont actuellement bien connus dans le milieu de la conception matérielle, ce mémorandum technique présente une implémentation par codage innovatrice. Ce nouvel algorithme est celui d'un générateur de bruit pseudoaléatoire générique prenant comme données d'entrées le polynôme et le degré de celui-ci et qui, en utilisant des outils de synthèse actuels, génère un circuit à espace compact. Les résultats montrent que le circuit généré ne consomme pas plus de ressources logiques qu'un traditionnel générateur de bruit aléatoire codé en dur. L'avantage résultant de l'utilisation de ce nouvel algorithme par rapport à ses prédécesseurs se traduit en économies appréciables au niveau du temps pour la conception et la vérification ainsi qu'au niveau des coûts de développement.

This page intentionally left blank.

Executive summary

Digital signal processing for wireless communication systems is an area where DRDC Ottawa has developed considerable expertise over the past years and still continues to expand. Part of the ongoing research activity in this area focuses on the development of algorithms for the major wireless communication systems on software radio platforms. Since the communication standards for these systems are constantly evolving, the current trend in the industry is to move towards the software radio approach. This technical memorandum presents one of these algorithms: a Register Transfert Level (RTL) generic pseudonoise generator.

Pseudonoise (PN) sequences are widely used in the field of digital signal processing, especially in wireless communication systems. Typically, PN generators are implemented in hardware through the use of Linear Feedback Shift Registers (LFSR) in either Application Specific Integrated Circuits (ASICs), Field Programmable Gate Array (FPGAs) or Complex Programmable Logic Devices (CPLDs). The algorithm description of the circuit is coded at the RTL via a Hardware Description Language (HDL), such as Verilog or VHDL, and the synthesis is performed with the synthesis tool generally provided with the vendor's software toolkit of the targeted device.

Although the methodology to code the PN generator RTL algorithm with an HDL language is currently well known in the design community, this technical memorandum presents an innovative way to code this algorithm. This new algorithm is one of a generic PN generator where the specific polynomial and its associated degree are presented as inputs to generate the correct circuit in a very compact form, when synthesized with the modern existing synthesis tools. Simulations were run with different polynomials to verify that the circuit generates the correct expected sequence. To verify the logic resource utilization in the device, a Xilinx Spartan II FPGA was targeted to synthesize the circuit using polynomials of various degree and complexity. The results show that the generic circuit generates the correct sequence in all cases and consumes the same amount of logic resources as the traditional hardcoded implementation.

The advantage of this algorithm over its predecessors is that a single VHDL component is required to produce many PN generators with different polynomials and degree within the same design. The user instantiates the same component for all the different PN generators, provides to each of them their specific polynomial and associated degree, and is limited only by the available device ressources. The VHDL compiler and the synthesis tool generate the correct circuits, significantly reducing design, verification and maintenance time.

Jean-François Beaumont. 2004. RTL design of a generic pseudonoise generator. DRDC Ottawa TM 2004-176. Defence R&D Canada – Ottawa.

Sommaire

Le traitement des signaux numériques pour les systèmes de communication sans-fil est un secteur où RDDC Ottawa a développé une expertise considérable au cours des dernières années et qui continue sans cesse de croître. Une partie des activités de recherche en cours dans ce secteur se concentre sur le développement d'algorithmes pour les principaux systèmes de communications sans-fil sur des plateformes de radio réalisée logiciel. Puisque les normes sur les communications pour ces systèmes sont en constante évolution, la tendance actuelle dans l'industrie est de se diriger vers l'approche radio réalisée logiciel. Ce mémorandum technique présente l'un de ces algorithmes : un générateur RTL de bruit pseudoaléatoire (BP) générique.

Les séquences pseudoaléatoires sont largement utilisées dans le domaine du traitement de signal numérique, particulièrement dans les systèmes de communication sans-fil. Généralement, les générateurs BP sont mis en oeuvre sous forme matérielle par l'entremise de registres à décalage à rebouclage linéaire soit dans des circuits intégrés spécifiques (ASICs), des matrices prédiffusées programmables (FPGAs) ou des réseaux logiques programmables complexes (CPLDs). La description du circuit sous forme algorithmique est codée au niveau du transfert de registre à registre (RTL) via un langage de description matériel (HDL), tel que Verilog ou VHDL, et la synthèse est effectuée à l'aide d'un outil de synthèse généralement fourni avec la trousse de logiciels du fabricant pour le composant sélectionné.

Bien que la méthodologie pour coder l'algorithme du générateur BP avec un langage HDL est actuellement bien connue dans le milieu de la conception matérielle, ce mémorandum technique présente une façon innovatrice de coder cet algorithme. Ce nouvel algorithme est celui d'un générateur BP générique où le polynôme spécifique et son degré sont les données d'entrées servant à générer le bon circuit sous forme compact, lorsque synthétisé avec les outils modernes de synthèse existants. Des simulations furent effectuées avec différents polynômes afin de vérifier que le circuit génère la séquence prévue exacte. Afin de vérifier l'utilisation des ressources logiques du composant, un FPGA Spartan II de Xilinx fut choisi pour effectuer la synthèse du circuit en utilisant des polynômes de degré et de complexité différents. Les résultats ont montré que le circuit générique générait dans tous les cas la séquence exacte et utilisait la même quantité de ressources logiques que l'implémentation tradionnelle codée en dur.

L'avantage de cet algorithme sur ses prédécesseurs est qu'un seul constituant VHDL est requis pour produire plusieurs générateurs BP de polynôme et de degré différents dans le même design. L'utilisateur instancie le même constituant pour tous les différents générateurs BP, fournit à chacun leur polynôme spécifique et le degré y étant associé, et est limité uniquement par les ressources disponibles dans le composant. Le compilateur VHDL et l'outil de synthèse génère les circuits exacts, réduisant ainsi significativement le temps de conception, de vérification et d'entretien.

Jean-François Beaumont. 2004. RTL design of a generic pseudonoise generator. DRDC Ottawa TM 2004-176. R & D pour la défense Canada – Ottawa.

Table of contents

List of figures

List of tables

This page intentionally left blank.

1 Introduction

Pseudonoise (PN) sequences are widely used in many areas of Digital Signal Processing. A few examples are in radar, error correction, cryptography, Global Positioning Systems (GPS), Very Small Aperture Satellite Terminals (VSAT) and digital communication systems. One good example of a PN sequence application in digital communications is in the CDMA IS-95 system where they are use to "spread" and "despread" the signal in the spread spectrum modulation and demodulation operations [1, 2, 3]. They are also very often used to simply generate gaussian noise for any kind of digital device testing.

The most typical way of implementing PN sequences in hardware (PN generators) is through the usage of Linear Feedback Shift Registers (LFSR) [4] in either Application Specific Integrated Circuits (ASICs), Field Programmable Gate Array (FPGAs) or Complex Programmable Logic Devices (CPLDs). The usual method to build such circuits in those devices is to use a Hardware Description Language (HDL), like Verilog or VHDL, to code them at the Register Transfer Level (RTL) and synthesize them with a synthesis tool, provided usually with the device vendor's tools. Another easier alternative is to directly use a predesigned core provided with the device vendor's tools, instead of HDL coding, and perform the synthesis.

Both of these methods have their pros and cons. The advantage of manually coding the circuit with a HDL language is that it is device independent, implying that the same code can be ported to any appropriate device regardless of vendor. However, this method is more time consuming than directly using a vendor specific predesigned core. In addition to coding the circuit, the designer must also verify expected functionality.

On the other hand, the use of a predesigned core is more straightforward, as they are provided for free with most vendor's development tools, and they have been verified, ensuring reliable output. The designer simply has to provide the correct information (like the polynomial) to generate an area and speed optimized circuit for the targeted device. However, the disadvantage of this method is that the design is not device independent. If changes are required during simulations, the core must be regenerated again with a different software tool, external to the simulator. Since the core generator software is external to the HDL simulator, simulation with different polynomial values causes changes to be made in two steps: regenerating the core and recompiling. This is more time consuming than simply changing the HDL code and recompiling. Additionally, the predesigned core approach cannot be use to create a generic circuit.

The development of a device independent RTL design includes production and testing. The reduction of development time gives rise to the desire to have a generic circuit that fits any polynomial. Such generic HDL based designs have already been proposed in the literature [5]. However, these circuits hardcode a finite number of polynomials within the design entity. This type of design, requires some overhead, consumes extra logic resources and limits the user in his choice of polynomials. Some of these require modification of the design to get the desired polynomial, which can introduce errors.

This technical memorandum presents a novel architecture for a RTL based PN generator. Unlike the existing designs, this new architecture does not hardcode the polynomials within the circuit entity but takes the polynomial and its associated degree as inputs to generate a compact area circuit (without extra overhead) when synthesized with the current synthesis tools. The circuit is designed in VHDL and is based on the Modular Shift Register Generator (MSRG) configuration, also known as the "Galois" configuration.

The net advantage of this approach is that a single VHDL component can be instantiated many times in a design for PN generators having different polynomials of various degree (from 2 and up to the limit imposed by the silicon size). This approach reduces design, debug and maintenance time for systems having many PN generators with different polynomials of various degree. The user is merely required to instantiate the same component for all the different PN generators. The polynomial and its associated degree are provided as inputs to each instantiated component. The VHDL compiler and the synthesis tool generates the correct circuit.

Also, because of the generic circuit's architecture and the logic circuit optimization performance of today's HDL synthesis tools, the synthesis results obtained in this technical memorandum show that the silicon utilization area is the same as if the circuit was manually coded and customized for the given polynomials.

This design can be implemented in CPLD, FPGA and ASIC devices. For the purpose of this technical memorandum, simulations were run with different polynomials to verify its correct functionality. To verify the silicon utilization area in a component, synthesis was performed, targeting a Xilinx Spartan II FPGA, with polynomials of various degree and complexity.

2 Pseudonoise generator implementation

2.1 Theory

An LFSR based PN generator with n stages will always produce a periodic output sequence. The initial condition of the shift register will eventually be reproduced after a finite number of clock cycles. The period cannot be greater than 2^n (maximum number of combinations of n binary digits). However, due to the architecture of this type of generator, the all-zero condition does not occur, unless set as the initial condition. In this case, the generator will remain in the same state forever. Therefore, the maximum number of possible states is $2^n - 1$. A generator output with such a period is called a "maximal length sequence" or "m-sequence". The higher the number of stages, the longer the sequence will be and the more random it will appear. For that reason, they are also called "pseudorandom noise" generators.

Each PN sequence can be described by a binary irreducible primitive polynomial of the form:

$$f(x) = 1 + c_1 x + c_2 x^2 + \cdots + c_i x^i + \cdots + c_{n-1} x^{n-1} + x^n, \quad c_i \in GF(2) \tag{1}$$

Where $GF(2)$ is a binary Galois Field (two elements, 0 and 1).

There are two different ways of implementing in hardware the irreducible (primitive) polynomial as an LFSR: The Simple Shift Register Generator (SSRG), also known as the "Fibonacci configuration" and the Modular Shift Register Generator (MSRG), also known as the "Galois configuration". Figure 1 and Figure 2 illustrate both configurations. The scope of this technical memorandum is limited to the generic PN generator circuit based on the MSRG configuration.

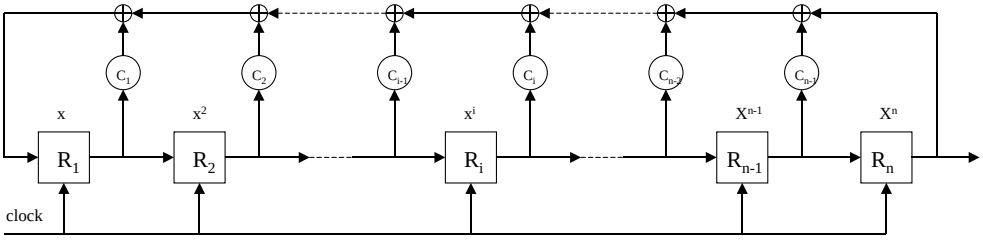

Figure 1: *SSRG configuration*

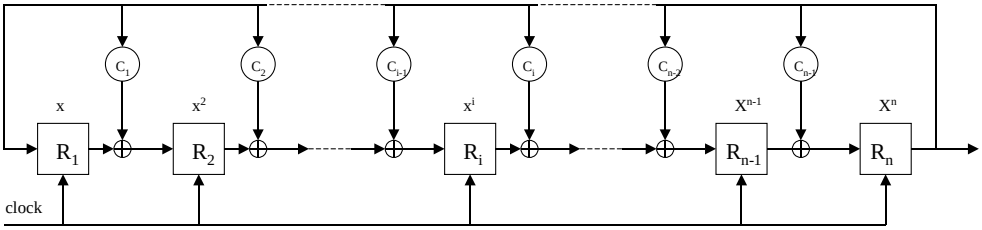

Figure 2: *MSRG configuration*

2.2 General mechanization of the irreducible polynomial

Considering only the MSRG configuration, the "mechanization" of the irreducible polynomial is accomplished using binary storage elements and modulo-2 adders [1]. Referring to Figure 2, the binary storage elements of the LFSR are designated as R_1, R_2, …, R_n. The coefficients of the polynomial, designated as C_1, C_2, …, C_{n-1}, play the roles of "switches" that dictate whether or not the output of the last stage R_n is modulo-2 added to the stage R_i. If the value of C_i is "0", the output of the stage R_i is simply shifted to the next stage R_{i+1}. If the value of C_i is "1", the output of the stage R_i is modulo-2 added to R_n and fed to the next stage R_{i+1}. Only the input of the first stage R_1 is connected directly to the output of the last stage R_n. Each register stage R_i is associated with the power term x^i of the irreducible polynomial.

The operation of the LFSR is initiated by the loading of a n-tuple vector in the n-stage register. Once the initial vector is loaded, the LFSR starts generating the sequence at the rate specified by the clock connected to the binary storage elements. A non-zero n-tuple loading vector will produce $2^n - 1$ distinct vectors in the registers, periodically, as the clock runs. The sequence resulting from the output, taken from the last stage R_n, is a PN sequence of length $2^n - 1$, which is one version (phase shift) out of $2^n - 1$ possible sequences. This unique sequence depends on the choice of the n-tuple loading vector, which is again a one out of $2^n - 1$ possible choices.

2.3 Conventional hardware implementation

Usually, PN generators are utilized within complex systems that are implemented in either custom silicon devices like ASICs or in programmable devices like FPGAs and CPLDs. As a result, their implementation is typically done using an HDL language such as VHDL which makes them device independant. The logic circuit described in VHDL is synthesized using software tools taking the code as input and mapping it to gates in the silicon or programmable device.

To better illustrate how an irreducible polynomial is implemented in hardware, let's take the following simple polynomial of degree 5:

$$f(x) = 1 + x^3 + x^5 \tag{2}$$

where $C_1 = C_2 = C_4 = 0$ and $C_3 = C_5 = 1$. This polynomial can also be represented by its set of binary coefficients ($C_5C_4C_3C_2C_1$) as 10100.

The optimized hardware implementation of this polynomial is straightforward and illustrated on Figure 3. All binary storage elements are implemented using D flip-flops (for a total of 5) and the modulo-2 adder is implemented with an XOR gate. The VHDL code associated with it is also quite simple.

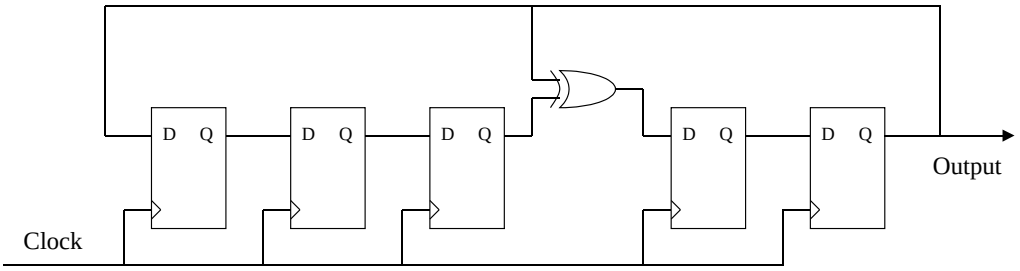

Figure 3: *Optimized hardware implementation of polynomial* $f(x) = 1 + x^3 + x^5$

VHDL code excerpt of conventional PN generator:

```
LFSR : process (Clock, Reset_n) is
  begin  -- process LFSR
    if (Reset_n = '0') then         -- asynchronous reset (active low)
      R <= "00001";
    elsif rising_edge(Clock) then  -- rising clock edge
      R(1) <= R(5);
      R(2) <= R(1);
      R(3) <= R(2);
      R(4) <= R(3) xor R(5);
      R(5) <= R(4);
    end if;
  end process LFSR;
```

For designs having few PN generators with simple polynomials like the one presented above, the designer can design, debug and maintain the circuits very easily. The tasks are manageable. However, if a design has many PN generators with different and more complex polynomials of various degrees, it could become time consuming to perform these tasks. The VHDL code must have unique entities and architectures for each PN generator and each must be designed, debugged and maintained separately.

It would be much simpler for the designer if a single generic component could be used for all the different PN generators in the design. This component would only need as input the polynomial and its degree (supplied at the instance level by the user) and the VHDL compiler and synthesis tool would generate automatically the circuit described by the polynomial.

To give an idea, it would take a few hours for an experienced HDL designer to manually code and verify the three code generators of the CDMA IS-95 system. Using a generic PN generator, these tasks are performed in a fraction of the time with the added benefit of a HDL implementation. This time savings is directly proportional to the number and the size of the PN generators in the system.

Also, knowing that PN generators have applications in many other areas (like in radar, cryptography, error correction and 3G cellular systems), a generic circuit like the one proposed would be beneficial, especially if these systems use more PN generators in their future designs.

2.4 Proposed architecture

Looking at the generic polynomial in Equation 1 on page 3 and at Figure 2 on page 4, one can notice the repeatability of the structure. The block diagram of the generic hardware implementation can be seen as a group of basic cells, each one made of a binary storage element, a “switch” and a modulo-2 adder, repeated as many times as the degree of the polynomial, and connected together in a specific way. The coefficient C_i determines the relevance of the modulo-2 adder, except for the first cell where the input is connected directly to the last stage. However, it is easy to model the first cell as a particular case of the other cells.

From these observations, a generic cell (Figure 4) could be designed. The logic inside the cloud must satisfy the requirements of the following truth table (Table 1).

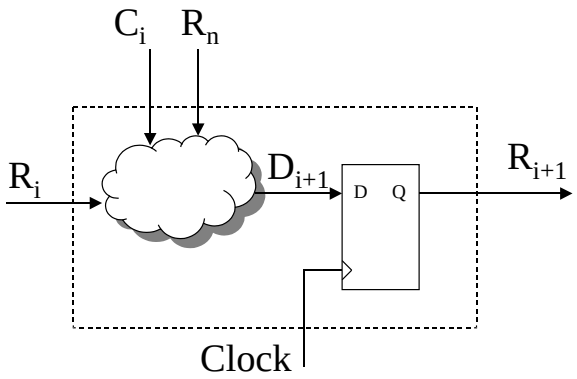

Figure 4: *Generic Cell (GC_i) with unknown combinatorial logic cloud*

C_i	R_n	R_i	D_{i+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Table 1: *Truth table for logic circuit inside the Generic Cell's cloud*

The logic circuit that satisfies the truth table can be synthesized easily using a Karnaugh map, as shown on Figure 5:

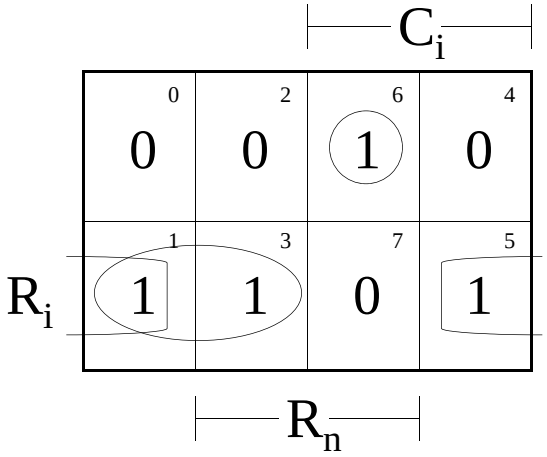

Figure 5: *Karnaugh map for logic circuit synthesis of Generic Cell's cloud*

From the variable grouping of the Karnaugh map, the following logic equation (Equation 3) is extracted. This equation is then further optimized using some basic Boolean algebra to get the final result:

$$D_{i+1} = \overline{C_i} R_i + \overline{R_n} R_i + C_i R_n \overline{R_i} \tag{3}$$

$$D_{i+1} = \overline{C_i} R_i + \overline{R_n} R_i + \overline{R_i}(C_i R_n) \tag{4}$$

$$D_{i+1} = R_i(\overline{C_i} + \overline{R_n}) + \overline{R_i}(C_i R_n) \tag{5}$$

$$D_{i+1} = R_i(\overline{C_i R_n}) + \overline{R_i}(C_i R_n) \tag{6}$$

$$\boxed{D_{i+1} = R_i \oplus C_i R_n} \tag{7}$$

After optimization, it simplifies to the point where only an AND gate needs to be added to the XOR gate of the modulo-2 adder to incorporate the effect of the polynomial constant (C_i) to the generic cell (Figure 6). This is confirmed by examining the truth tables of the AND and the XOR gate (Table 2 and Table 3).

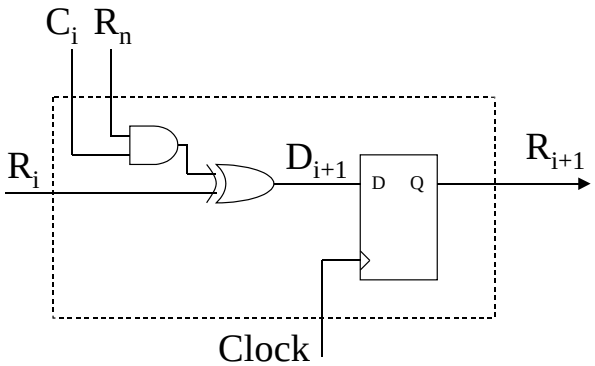

Figure 6: *Generic Cell (GC_i)*

C_i	R_n	R_nC_i
0	0	0
0	1	0
1	0	0
1	1	1

Table 2: *AND gate truth table*

R_nC_i	R_i	D_{i+1}
0	0	0
0	1	1
1	0	1
1	1	0

Table 3: *XOR gate truth table*

If the value of the polynomial constant (C_i) is “0”, the value of R_nC_i will be “0”, regardless of the value R_n. Therefore, if the value of R_nC_i is “0”, the output value of the XOR gate (D_{i+1}) will be equal to the value of R_i, which is like having no XOR gate at all (or a straight connection between R_i and D_{i+1}). On the other hand, if the value of polynomial constant (C_i) is “1”, the value of R_nC_i will be equal to the value of R_n, which is like having no AND gate at all (or a straight connection to one of the inputs of the XOR gate). Therefore, the input of the D flip-flop will be the XOR value between the previous D flip-flop output (R_i) and the feedback from the last flip-flop of the LFSR (R_n).

For a polynomial of degree 5, the generic circuit of the PN generator would be as illustrated in Figure 7. The realization of such a circuit in VHDL is done using two blocks. The first block is a simple “for-generate” statement loop in which each pass generates the combinatorial logic feedback from the last cell of the shift register, the appropriate polynomial coefficient, and the previous cell’s output (Q) to be used as input to the next cell (D) within the shift register. The second block is the shift register itself that connects to the inputs and outputs of the logic feedback block. One notices the special case where the first cell of the shift register is connected directly to the last cell. All the other shift register cells are connected to the feedback combinatorial logic array. The nice thing about this approach is that

the VHDL compiler and the synthesis tool take care of generating the correct circuit, based on the polynomial and the degree associated with it, supplied as inputs to the component (see following code excerpt of the VHDL entity). The available silicon area of the device being used is the only factor that limits the maximum value of the polynomial's degree. The minimum value is 2.

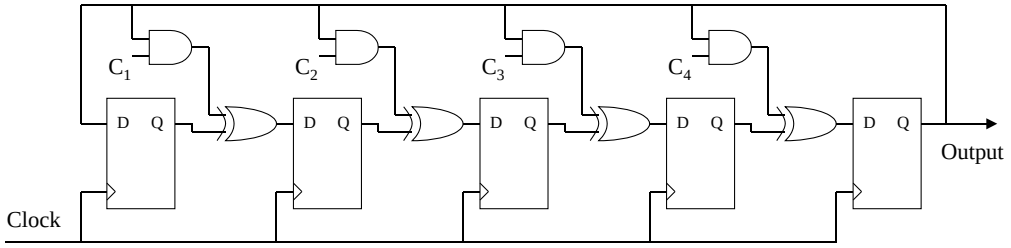

Figure 7: *Generic PN generator circuit for a polynomial of degree 5*

If the polynomial ($C_5C_4C_3C_2C_1$) is equal to 10100, the circuit should have the same PN sequence as the one illustrated in Figure 3 on page 5, given the same loading vector.

VHDL code excerpt of generic PN generator:

```
entity PN_generator is
  generic (
    Poly_degree : integer);
  port (
    -- Input clock
    Clock      : in  std_logic;
    -- Reset input (active low)
    Reset_n    : in  std_logic;
    -- Polynomial
    Polynomial : in  std_logic_vector(Poly_degree downto 1);
    -- Generator output sequence
    PN_out     : out std_logic
    );
end entity PN_generator;
```

2.5 Circuit optimizations

Compared to the non-generic version of the PN generator, the generic one has more logic. One would be tempted to think that this version would require more gates and therefore, would not be as good in terms of performance and silicon area utilization. This would probably have been true in the early days of synthesis tools, but not anymore.

Today's synthesis tools are more efficient and can now optimize away unnecessary gates used in a design. This is especially true in the case where the polynomials are defined as constants, like in the generic PN generator. In this case, the input values of the "AND" and the "XOR" gates of each generic cell are static. Depending on their input values, some of these gates can be removed from the circuit without affecting its functionality. Therefore, the synthesis of a PN generator with a defined polynomial should lead to the same result whether it has been designed using the conventional way or with the generic circuit. It is easy to verify by using the RTL schematic viewer (in most of today's synthesis software tools) and comparing both results. Another, (and better) way to verify this is to consult the tool's synthesis and mapping reports to verify the quantity of logic resources that have been utilized.

It is very important to understand that the optimization relies entirely on the fact that the polynomial values are defined as constant at the instance level. If the polynomial values are not defined as constants, this optimization will not occur.

3 Verification and results

3.1 Simulations

To verify the correct functionality of the generic PN generator, polynomial values needed to be chosen. Since the sequence length is equal to $2^n - 1$ for a maximal length sequence, the verification takes greater time if the polynomial degree is high. Also, based on the technique used to generate the circuit (for-generate loop), a high polynomial degree is not required to verify the circuit functionality. A polynomial of degree four or five is sufficient. Any problem within the circuit be apparent early in the verification process, since a single incorrect bit in the sequence will fail the test. Therefore, the following two simple polynomials (f(x)) were used to verify the functionality of the generic PN generator.

The first objective was to verify that the selected polynomials could produce the expected sequences (S) given specific initial loading vectors (L)[1]. The second objective was to demonstrate that only a single VHDL component of the generic PN generator was needed in a system having more than one PN generator with different polynomials of various degrees.

Polynomial #1:

$f_1(x) = 1 + x^3 + x^5$
$S_1 = 000010101110110001111001101001$
$L_1 = 00001$

Polynomial #2:

$f_2(x) = 1 + x^3 + x^4$
$S_2 = 000111101011001$
$L_2 = 0001$

To achieve these two objectives, a VHDL testbench with two instances of the generic PN generator component was created (see following VHDL excerpt code). The first instance used polynomial #1 and the second instance polynomial #2. Both generators were run with a 1 MHz clock for a period of 10 ms to permit the periodicity of the sequences. A reset signal was periodically set to see the behavior of the generator during and after the reset. The reset was 10 us long and repeated every 100 us. Both instances of the PN generator behaved as expected. Figure 8 shows the results obtained from simulations.

[1] $L = \{R_n, R_{n-1}, \ldots, R_2, R_1\}$, where n is the polynomial degree and R_i a register of the LFSR

VHDL code excerpt of testbench:

```
  -- Polynomials definition

  -- First polynomial
  -- F1(x) = x^5 + x^3 + 1 = 1*x^5 + 0*x^4 + 1*x^3 + 0*x^2 + 0*x^1 + 1
  constant F1_degree : integer := 5;    -- Polynomial degree
  constant F1 : std_logic_vector(F1_degree downto 1) := "10100";

  -- Second polynomial
  -- F2(x) = x^4 + x^3 + 1 = 1*x^4 + 1*x^3 + 0*x^2 + 0*x^1 + 1
  constant F2_degree : integer := 4;    -- Polynomial degree
  constant F2 : std_logic_vector(F2_degree downto 1) := "1100";

...

  PN_generator_1 : component PN_generator
    generic map (
      Poly_degree => F1_degree)
    port map (
      Clock   => Clock,
      Reset_n => Reset_n,
      Polynomial => F1,
      PN_out  => PN_out_1);

  PN_generator_2 : component PN_generator
    generic map (
      Poly_degree => F2_degree)
    port map (
      Clock   => Clock,
      Reset_n => Reset_n,
      Polynomial => F2,
      PN_out  => PN_out_2);

...
```

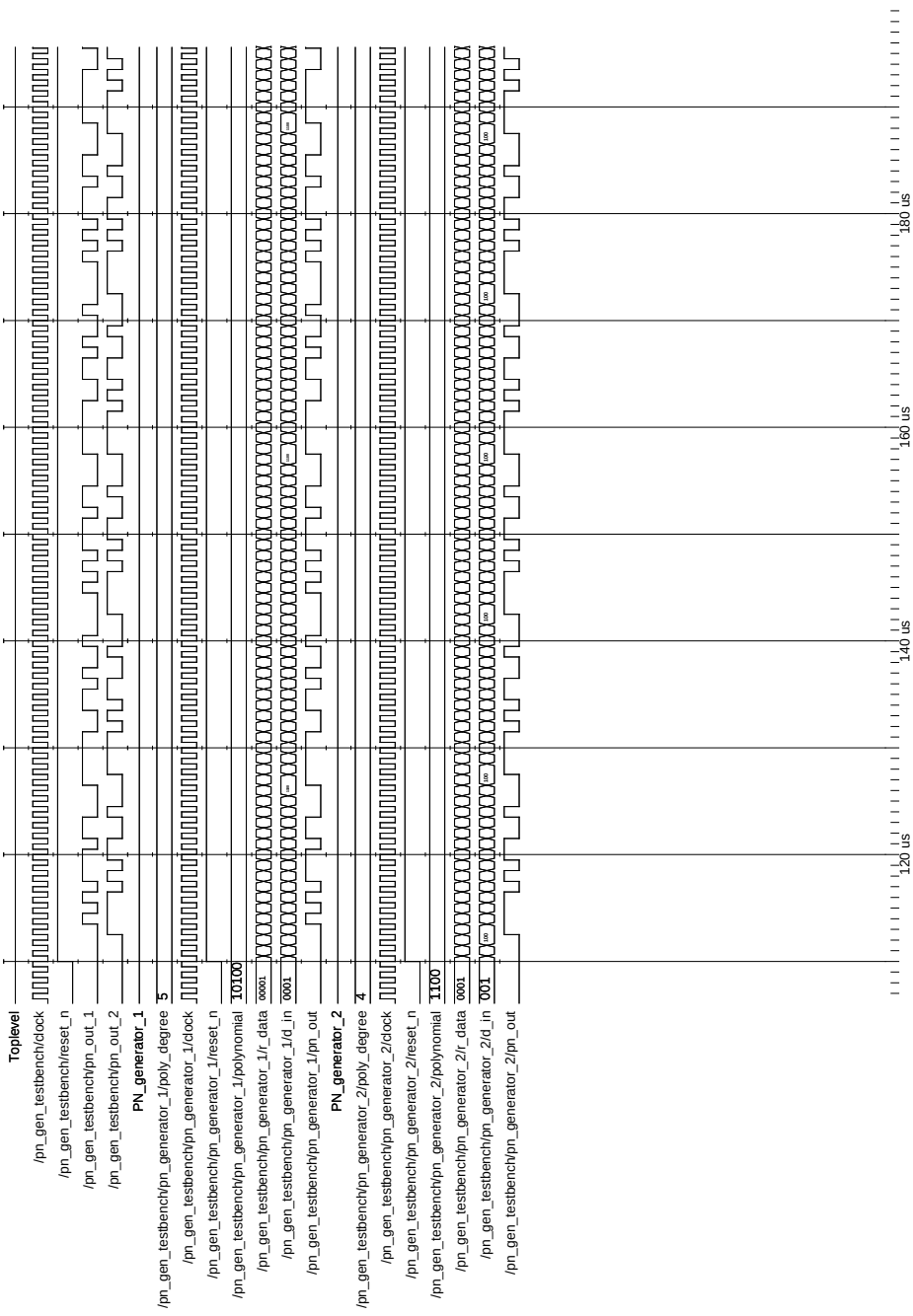

Figure 8: *Simulation results for polynomials* $f_1(x)$ *and* $f_2(x)$

3.2 Synthesis (hardware implementation)

Most of today's synthesis tools for ASIC or FPGA devices perform logic optimizations in the HDL source code. These tools are particularly good for performing the logic optimizations when some of the circuit inputs are constants.

Since the generic PN generator is essentially built around a logic circuit having a constant input (the polynomial value), it is expected that the synthesis tool can optimize the circuit such that the end result would be very close to one where the logic circuit was already optimum, such as a manually coded and custom PN generator. Also, because of the nature of the logic optimization (constant based), the result should be independent of the PN generator size (polynomial). Another perspective, is to envision the circuit as a number of the same generic cell, all connected together and each being optimized the same way (depending on the constant polynomial value).

To verify this assumption, the Xilinx ISE FPGA development toolkit was used. Unlike the functional verification where a relatively small polynomial was sufficient to verify the algorithm, it beneficial to have polynomials of various sizes to verify the logic resource allocation in the device. The device targeted for these synthesis experiments was the Xilinx Spartan II XC2S50E-6TQ144. The following four polynomials were used to create the PN generators. In order to compare the synthesis results, these PN generators were created with the three different design methods: the Xilinx core generator, the custom HDL code and the generic HDL code .

Synthesis verification polynomials:

- $f_1(x) = 1 + x^3 + x^5$
- $f_2(x) = 1 + x^5 + x^7 + x^8 + x^9 + x^{13} + x^{15}$
- $f_3(x) = 1 + x^1 + x^2 + x^3 + x^5 + x^6 + x^7 + x^{10} + x^{16} + x^{17} + x^{18} + x^{19} + x^{21} + x^{22} + x^{25} + x^{26} + x^{27} + x^{31} + x^{33} + x^{35} + x^{42}$
- $f_4(x) = 1 + x^1 + x^3 + x^4 + x^{64}$

The following tables provides a comparative summary of the results obtained for each of the polynomials using the three different methods. These results are taken from the "Map Report" of the Xilinx ISE software toolkit.

	Xilinx Core	Custom HDL	Generic HDL
Logic Utilization			
Number of Slice Flip Flops	5 out of 1,536	5 out of 1,536	5 out of 1,536
Number of 4 input LUTs	1 out of 1,536	1 out of 1,536	1 out of 1,536
Logic Distribution			
Number of occupied Slices	3 out of 768	3 out of 768	3 out of 768
Total equivalent gate count for design	46	46	46

Table 4: *Synthesis results for polynomial #1 ($f_1(x)$) using the three design methods*

	Xilinx Core	Custom HDL	Generic HDL
Logic Utilization			
Number of Slice Flip Flops	15 out of 1,536	15 out of 1,536	15 out of 1,536
Number of 4 input LUTs	5 out of 1,536	5 out of 1,536	5 out of 1,536
Logic Distribution			
Number of occupied Slices	8 out of 768	8 out of 768	8 out of 768
Total equivalent gate count for design	150	150	150

Table 5: *Synthesis results for polynomial #2 ($f_2(x)$) using the three design methods*

	Xilinx Core	Custom HDL	Generic HDL
Logic Utilization			
Number of Slice Flip Flops	42 out of 1,536	43 out of 1,536	43 out of 1,536
Number of 4 input LUTs	19 out of 1,536	19 out of 1,536	19 out of 1,536
Logic Distribution			
Number of occupied Slices	22 out of 768	22 out of 768	22 out of 768
Total equivalent gate count for design	450	458	458

Table 6: *Synthesis results for polynomial #3 ($f_3(x)$) using the three design methods*

	Xilinx Core	Custom HDL	Generic HDL
Logic Utilization			
Number of Slice Flip Flops	64 out of 1,536	64 out of 1,536	64 out of 1,536
Number of 4 input LUTs	3 out of 1,536	3 out of 1,536	3 out of 1,536
Logic Distribution			
Number of occupied Slices	33 out of 768	32 out of 768	32 out of 768
Total equivalent gate count for design	530	530	530

Table 7: *Synthesis results for polynomial #4 ($f_4(x)$) using the three design methods*

These results show that there are no differences for the logic utilization and distribution in the device between a custom HDL design and the generic one. The Xilinx core however provides results that are slightly different, but not significantly. If the SRL16 macro was used for the design with the Xilinx core, the synthesis results would have been better. However, the circuit would have been different from the custom and the generic design since the usage of the SRL16 macro generates a design with serial loading and without a reset. For this reason, core design with the SRL16 macro was not used in the comparison.

4 Conclusion

With the availability of today's high density silicon devices, HDL designers are faced with designs that are always growing in terms of size and complexity. In order to meet their tight schedules and milestones, HDL designers are always seeking design methodologies that allow:

1. shorter design time
2. shorter verification time
3. optimized design (logic resources utilization)

This technical memorandum presents the design of a generic PN generator that helps achieve the first two goals. This HDL block is easy to use and has been proven to work, and as a result, will reduce the development and verification time, upon implementation. Designers who need many different implementations of PN generators in a single RTL based design will find it has several advantages. For example, a designer building a CDMA system (such as IS-95) where different pseudonoise sequences are needed [1] would find it quite useful. As proposed, only one piece of HDL code needs to be maintained.

Although the architecture cannot guarantee an optimized design (in terms of logic resources utilization), the results obtained in this technical memorandum show that one is likely.

The generic PN generator presented here is a basic one. It is possible to add to it other features like fill vector loading, masking and enabling/disabling capabilities and still keep it sufficiently generic such that all the advantages of using it remain. This is an area of work.

Finally, it is also possible to be able to implement this circuit using HDL languages other than VHDL. Verilog 2001 and System C are examples of alternatives but testing these was beyond the scope of this technical memorandum.

References

1. Jhong Sam Lee, Leonard E. Miller (1998). CDMA Systemes Engineering Handbook, Artech House.

2. ir.J.Meel (1999). Spread Spectrum. Technical Report. De Nayer Instituut. Jan De Nayerlaan, 5, B-2860 Sint-Katelijne-Waver, Belgium.

3. Schwarz, Richard (2001). An Introduction to Linear Recursive Sequences in Spread Spectrum Systems. Technical Report. Filtronic Sigtek Inc.

4. Mutagi, R.N. (1996). Pseudo noise sequences for engineers. *Electronics & Communications Engineering Journal*, pp. 79–87.

5. Smith, Douglas J. (2001). HDL Chip Design, Doone Publications.

List of acronyms

ASIC	Application Specific Integrated Circuit
CDMA	Code Division Multiple Access
CPLD	Complex Programmable Logic Device
FPGA	Field Programmable Gate Array
GF	Galois Field
HDL	Hardware Description Language
IS	Interim Standard
LFSR	Linear Feedback Shift Register
MSRG	Modular Shift Register Generator
PN	Pseudo Noise (or Pseudonoise)
RTL	Register Transfer Language
SSRG	Simple Shift Register Generator
VHDL	Very high speed integrated circuit (VHSIC) Hardware Description Language

DOCUMENT CONTROL DATA

(Security classification of title, body of abstract and indexing annotation must be entered when document is classified)

1. ORIGINATOR (the name and address of the organization preparing the document. Organizations for whom the document was prepared, e.g. Centre sponsoring a contractor's report, or tasking agency, are entered in section 8.)

Defence R&D Canada – Ottawa
3701, Carling avenue, Ottawa, Ontario, K1A 0Z4

2. SECURITY CLASSIFICATION (overall security classification of the document including special warning terms if applicable).

UNCLASSIFIED

3. TITLE (the complete document title as indicated on the title page. Its classification should be indicated by the appropriate abbreviation (S,C,R or U) in parentheses after the title).

RTL design of a generic pseudonoise generator

4. AUTHORS (Last name, first name, middle initial. If military, show rank, e.g. Doe, Maj. John E.)

Beaumont, Jean-François

5. DATE OF PUBLICATION (month and year of publication of document)	6a. NO. OF PAGES (total containing information. Include Annexes, Appendices, etc).	6b. NO. OF REFS (total cited in document)
September 2004	31	5

7. DESCRIPTIVE NOTES (the category of the document, e.g. technical report, technical note or memorandum. If appropriate, enter the type of report, e.g. interim, progress, summary, annual or final. Give the inclusive dates when a specific reporting period is covered).

Technical Memorandum

8. SPONSORING ACTIVITY (the name of the department project office or laboratory sponsoring the research and development. Include address).

Defence R&D Canada – Ottawa
3701, Carling avenue, Ottawa, Ontario, K1A 0Z4

9a. PROJECT OR GRANT NO. (if appropriate, the applicable research and development project or grant number under which the document was written. Specify whether project or grant).	9b. CONTRACT NO. (if appropriate, the applicable number under which the document was written).
15bl12	

10a. ORIGINATOR'S DOCUMENT NUMBER (the official document number by which the document is identified by the originating activity. This number must be unique.)	10b. OTHER DOCUMENT NOs. (Any other numbers which may be assigned this document either by the originator or by the sponsor.)
DRDC Ottawa TM 2004-176	

11. DOCUMENT AVAILABILITY (any limitations on further dissemination of the document, other than those imposed by security classification)

(X) Unlimited distribution
( ) Defence departments and defence contractors; further distribution only as approved
( ) Defence departments and Canadian defence contractors; further distribution only as approved
( ) Government departments and agencies; further distribution only as approved
( ) Defence departments; further distribution only as approved
( ) Other (please specify):

12. DOCUMENT ANNOUNCEMENT (any limitation to the bibliographic announcement of this document. This will normally correspond to the Document Availability (11). However, where further distribution beyond the audience specified in (11) is possible, a wider announcement audience may be selected).

Full unlimited announcement

13. ABSTRACT (a brief and factual summary of the document. It may also appear elsewhere in the body of the document itself. It is highly desirable that the abstract of classified documents be unclassified. Each paragraph of the abstract shall begin with an indication of the security classification of the information in the paragraph (unless the document itself is unclassified) represented as (S), (C), (R), or (U). It is not necessary to include here abstracts in both official languages unless the text is bilingual).

Pseudonoise (PN) generators have many applications in the field of digital signal processing, especially in wireless communication systems. They are usually implemented in hardware through the use of Linear Feedback Shift Registers (LFSR) in either Application Specific Integrated Circuits (ASICs), Field Programmable Gate Array (FPGAs) or Complex Programmable Logic Devices (CPLDs).

Although the methods and techniques to code the Register Transfer Level (RTL) algorithm with a Hardware Description Language (HDL) are currently well known in the design community, this technical memorandum presents an innovative code implementation. This new algorithm is a generic PN generator that takes as input the polynomial and its associated degree and, using current synthesis tools, generates a compact area circuit. Results show that the generated circuit does not consume more logic resources than a traditional hardcoded PN generator. The net advantage of using this new algorithm over its predecessor is significant savings in terms of design time, verification time and hence, development costs.

14. KEYWORDS, DESCRIPTORS or IDENTIFIERS (technically meaningful terms or short phrases that characterize a document and could be helpful in cataloguing the document. They should be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location may also be included. If possible keywords should be selected from a published thesaurus. e.g. Thesaurus of Engineering and Scientific Terms (TEST) and that thesaurus-identified. If it not possible to select indexing terms which are Unclassified, the classification of each should be indicated as with the title).

Pseudo Noise, Pseudo-noise, Pseudonoise generator, PN generator, Pseudonoise sequences, LFSR, Generator, ASIC, FPGA, RTL, Xilinx, generic

Defence R&D Canada

Canada's leader in defence
and national security R&D

R & D pour la défense Canada

Chef de file au Canada en R & D
pour la défense et la sécurité nationale

www.drdc-rddc.gc.ca