PN-GENERATORS EMBEDDED IN HIGH PERFORMANCE SIGNAL PROCESSORS

Ulrich Walther and Gerhard P. Fettweis

Dresden University of Technology
Mannesmann Mobilfunk Chair for Mobile Communications Systems
Mommsenstr. 13, D-01062 Dresden, Germany

ABSTRACT

Wireless systems based on CDMA such as proposed in the 3GPP standard utilize pseudo random number sequences for generation of the spreading codes. These sequences are usually created by the use of finite field arithmetic. A pure software implementation onto a digital signal processor (DSP) is very inefficient and would result in unreasonable high computational load. In this paper a hardware-based scheme is proposed which allows for an efficient implementation on a high performance DSP as well as into an ASIC. With the configurable approach a binary PN-sequence with an arbitrary primitive polynomial can be generated. The unit was specially designed for and embedded into a domain specific processor, which supports W-CDMA and alike baseband processing.

1. INTRODUCTION

The wireless communication standard 3GPP represents a Wideband CDMA system. At the transmitter the original data is spread by means of a special sequence, the spreading code. The spreading operation increases the signal bandwidth and enables user and cell separation.

At the receiver side the sampled data is fed into a RAKE receiver to cope with multi-path propagation channels. The RAKE embodies a set of "fingers" each of them performing a de-spread operation: the incoming data will be multiplied by the conjugate code sequence. The fingers need to correlate the data and the code with different input delays according to the channel impulse response. This can be achieved by buffering the input data and feeding the fingers appropriately. However, a large buffer memory might result which is rather unfavorable for a DSP-solution. Another approach eliminates the need for this by starting the code sequences for each finger with a different delay, i.e. the code phase for each finger differs.

A software solution in a standard processor would increase the algorithms complexity by about 100% raising the computational requirements to an infeasible measure. Therefore, we are looking for a hardware-based code generation which free the processor for further computations. However, since this hardware is required for the spreading operation only the extensions should add as little to the core size as possible.

In this paper we present an architecture with a minimum additional hardware area which is independent from the generating polynomial over GF(2^m), $m \leq 32$. Simple control matched to the instruction set of the processor is accomplished by memory mapped configuration registers accessible via standard move instructions.

Section 2 explains the generation of PN-sequences and introduces a mathematical formulation in order to derive different forms of such sequence generators and modifications thereof. The implementation and embedding of the proposed circuit in a DSP-data-path is outlined in Section 3. Finally some results and a comparison of different realizations will be given.

2. PN-SEQUENCE GENERATION

The 3GPP spreading operation is divided into two parts as indicated in Fig. 1. First the data sequence is multiplied with a set of orthogonal channelization codes $c_{ch,i}$ for separating different users. The In-phase and Quadrature-phase parts are joined to form a complex signal, which is then scrambled with a complex Gold-code [1][2]. A Gold-sequence is derived from two real M-sequences of length 2^m-1 which are constructed using a primitive polynomial of degree m

$$p(x) = x^m + p_{m-1}x^{m-1} + \cdots + p_1 x + p_0 \qquad (1)$$

with $p_0 = 1$, $p_i \epsilon \{0,1\}$, $0 < i < m$. The initial state is determined by the first *m* symbols $y_0 \ldots y_{m-1}$. The subsequent symbols are given by the recurrence[1]

$$y_{k+m} = \sum_{i=0}^{m-1} p_i \, y_{k+i} \pmod 2 \qquad (2)$$

A linear feedback shift register (LFSR) of length m can be used to carry out this operation as depicted in Fig. 2 (also proposed in [1]). However, the straight forward implementation may have two draw-backs: The series of adder stages in the feedback forms a

[1] for binary sequences all sums are modulo 2, i.e. addition becomes *XOR*, multiplication is performed as *AND*

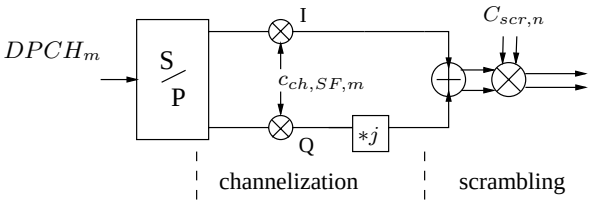

Figure 1: Downlink spreading operation in 3GPP

This work was supported by Deutsche Forschungsgemeinschaft contract SFB 358-A6

long critical path and, more serious, the initialization of the register becomes difficult.

The initial state of the register is determined by two parameters: the code number and the delay/offset with respect to other RAKE fingers. In 3GPP uplink mode the binary representation of the code number directly determines the initial state and the offset tells how many times the shift register need to be pulsed before the first valid symbol is available at the output. In the downlink case the sum of code number and offset defines how many times the register need to be pulsed starting with the initial vector $\mathbf{s}^{(0)}$. For large offsets or code numbers it would take a long time to "preset" the generator. Alternatively one could use a look-up table where the states of the register are stored for different offset values. Again for large offsets this solution is not practicable since the memory size becomes unreasonably large. A much smarter solution would calculate the initial state within a few cycles - finite field arithmetic need to be implemented.

One way to produce an offset in the sequence output is the use of the LFSR with a mask that multiplies the contents of the memory elements with the masking coefficients and adding the results up (see Fig. 3). This procedure is proposed in the standard [1] to create a half-period phase shifted version of the original sequence for the quadrature parts. However, in order to produce an arbitrary offset the masking value $\mathbf{b}$ needs to be re-calculated for each initialization.

Mathematically the state of the LFSR at time k can be described by the vector $\mathbf{s}^{(k)} = [s_0^{(k)}\ s_1^{(k)}\ s_2^{(k)} \ldots\ s_{m-1}^{(k)}]^T$. After one clock pulse the state changes into

$$\mathbf{s}^{(k+1)} = \mathbf{M}\,\mathbf{s}^{(k)} \tag{3}$$

with the transition matrix

$$\mathbf{M} = \begin{pmatrix} p_{m-1} & p_{m-2} & \ldots & p_1 & 1 \\ 1 & 0 & \ldots & 0 & 0 \\ 0 & 1 & \ldots & 0 & 0 \\ \vdots & \vdots & \ldots & \vdots & \vdots \\ 0 & 0 & \ldots & 1 & 0 \end{pmatrix} \tag{4}$$

The output y_k equals $s_{m-1}^{(k)}$. The shifted output caused by the mask $\mathbf{b}^{(q)} = [b_0^{(q)}\ b_1^{(q)} \ldots\ b_{m-1}^{(q)}]^T$ results in

$$y_{q+k} = (\mathbf{b}^{(q)})^T\ \mathbf{s}^{(k)} \tag{5}$$

where q is the offset between the sequences. The special case of $\mathbf{b}^{(0)} = [0\ 0\ \ldots\ 0\ 1]^T$ produces zero offset. In order to determine the mask $\mathbf{b}^{(q)}$ we substitute the recursion (3) and the output yields

$$y_q = s_{m-1}^{(q)} = [0\ 0 \ldots\ 0\ 1]\ \mathbf{M}^q\ \mathbf{s}^{(0)} \tag{6}$$

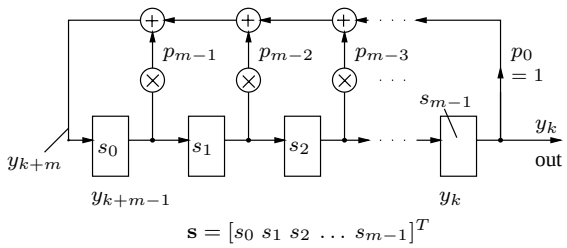

Figure 2: Linear feedback shift register

Setting $k = 0$ in (5) and merging with (6) generates

$$\mathbf{b}^{(q)} = (\mathbf{M}^T)^q\ \mathbf{b}^{(0)} \tag{7}$$

Hence a circuit for exponentiation of the matrix $\mathbf{M}$ is required [3].

A popular solution to that problem is the use of a Galois Field (GF) multiplier. Consider the transposed form of the LFSR shown in Fig. 4. That circuit is also known as modular feedback shift register (MFSR) [4].

The state of the MFSR at time q should be defined with $\mathbf{b}^{(q)}$. Than the subsequent state is given by

$$\mathbf{b}^{(q+1)} = \mathbf{M}^T\ \mathbf{b}^{(q)} \tag{8}$$

with $\mathbf{M}$ being the same matrix as for the LFSR. The recursion can easily be transformed into expression (7). Thus the state of the MFSR is identical to the mask value $\mathbf{b}$ if the initial state $\mathbf{b}^0$ holds the above mentioned condition. Furthermore the outputs of the MFSR and the LFSR are equivalent if $\mathbf{s}^{(0)} = [1\ 0\ \ldots\ 0]^T$, hence

$$b_0^{(q)} = s_{m-1}^{(q)}. \tag{9}$$

In terms of Galois Field arithmetics each state of the MFSR represents an element of the field[2]. Furthermore every field element can be expressed as a power of a primitive element α, i.e. $\tilde{\mathbf{s}}^{(0)} = \alpha^0$, $\tilde{\mathbf{s}}^{(1)} = \alpha$, $\ldots$, $\tilde{\mathbf{s}}^{(i)} = \alpha^i$. Each clock cycle the register advances its state to the next higher power of the primitive element. Therefore the MFSR is referred to as α-multiplier.

If the MFSR is extended according to Fig. 5, the circuit could multiply arbitrary numbers of GF(2^m) in a serial manner, hence it is known as GF-multiplier [5]. First the shift register is initialized with the all-zero state. Each clock cycle the partial product vector $x_i\mathbf{y}$ is added to the actual state. The x_i are the subsequent components of the x-operand starting with the most significant bit. After m cycles the product is available.

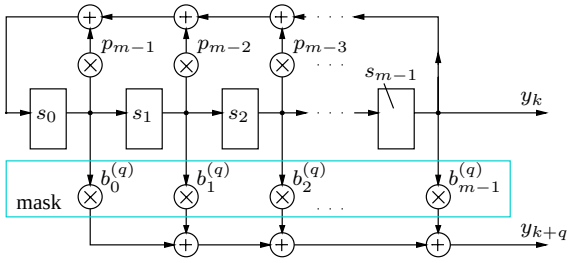

Figure 3: Linear feedback shift register with mask

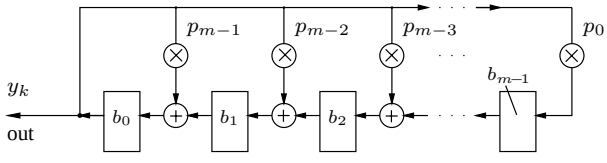

Figure 4: Modular feedback shift register

[2] please refer to [5][6][7] for a detailed description of GF(2^m)

The multiplier circuit is used to create an element α^n by forming it from previously calculated and stored values α^{n_i} using the relation

$$\alpha^n = \prod_{i=0}^{L-1} \alpha^{n_i} \quad \text{with} \quad n = \sum_{i=0}^{L-1} n_k, \tag{10}$$

i.e. the power is generated as a series of multiplications. Example: Assume the binary representation of the number $n = \sum_{i=0}^{L-1} n_i 2^i$, $n_i \in \{0, 1\}$. Now the product calls always for values of the form α^{2^i}. In this case $L = m$ holds, which equals the total number of α-powers to be stored. However, the number of multiplications increases to m, hence the total number of cycles for the initialization becomes m^2. That value can be reduced at the cost of an increased memory size if the number n is represented with a higher order number system (e.g. octal, $O = 8$) and a group of bits defines the value of a digit n_i (e.g. 3=ld(O)=ld(8)). The total storage for an m-bit number becomes mem $= \lceil m/\mathrm{ld}(O) \rceil \times O$ with O being the order of the system. Now only $m/\mathrm{ld}(O)$ multiplications are required. The tradeoff between memory size and intitialization time should be considered in the final implementation.

Now the overall PN-generator becomes a combination of a LFSR (Fig. 3) and a GF-multiplier according to Fig. 5 which generates the mask. After the initialization phase the GF-multiplier is not used any more. Therefore this implementation seems rather inefficient. A simpler yet powerful structure shall be proposed here.

Consider the LFSR and add the partial product circuitry from the GF-multiplier giving the circuit shown in Fig. 6. Now we claim the following:
In order to calculate $\mathbf{s}^{(q+k)}$ from $\mathbf{s}^{(k)}$ we reset the shift register, load the value $\mathbf{s}^{(k)}$ into the y-operand register and the value $\mathbf{b}^{(q)}$ (the corresponding α-power from GF(2^m) in the x-operand register and clock the circuit m times.

Proof: Without loss of generality we set $k = 0$ and the m-cycle operation yields the state

$$\begin{aligned} \mathbf{s}^{(q)} &= b_0^{(q)}\, \mathbf{M}^{m-1}\, \mathbf{s}^{(0)} + b_1^{(q)}\, \mathbf{M}^{m-2}\, \mathbf{s}^{(0)} \\ &\quad + \ \ldots \ + b_{m-1}^{(q)}\, \mathbf{s}^{(0)} \end{aligned} \tag{11}$$

Using the LFSR property $s_{m-1-i}^{(q)} = s_{m-1}^{(q+i)}$ ([3]) and the identity (9) leads to the LFSR state

$$\tilde{\mathbf{s}}^{(q)} = \begin{pmatrix} b_0^{(q+m-1)} \\ b_0^{(q+m-2)} \\ \vdots \\ b_0^{(q)} \end{pmatrix} = \begin{pmatrix} [1\,0\,\ldots\,0]\ (\mathbf{M}^T)^{m-1}\, \mathbf{b}^{(q)} \\ [1\,0\,\ldots\,0]\ (\mathbf{M}^T)^{m-2}\, \mathbf{b}^{(q)} \\ \vdots \\ [1\,0\,\ldots\,0]\ (\mathbf{M}^T)^{0}\ \ \mathbf{b}^{(q)} \end{pmatrix} \tag{12}$$

For both equations (11) and (12) we calculate the i-th element

$$s_i^{(q)} = \sum_{k=0}^{m-1} b_k^{(q)}\, M_{(i+1),1}^{m-1-k} \quad (i = 0\ \ldots\ m-1) \tag{13}$$

$$\tilde{s}_i^{(q)} = \sum_{k=0}^{m-1} b_k^{(q)}\, M_{(k+1),1}^{m-1-i} \quad (i = 0\ \ldots\ m-1) \tag{14}$$

where $M_{ij}^{(m)}$ is element i, j of the matrix $\mathbf{M}^m$. Equivalence of (13) and (14) is assured if

$$M_{i,1}^{(m-k)} = M_{k,1}^{(m-i)} \qquad (i, k = 1\ \ldots\ m) \tag{15}$$

that is the i-th row of $\mathbf{M}^l$ equals the $i+1$-st row of $\mathbf{M}^{l+1}$. This property is satisfied by the special structure of the matrix $\mathbf{M}$ and could be shown with ease.

Thus the new circuit from Fig. 6 enables the creation of a sequence with arbitrary phase offset without using an additional GF-multiplier.

3. IMPLEMENTATION

Based on the considerations above a hardware implementation was carried out. Since the processor should work for all uplink and downlink modes the circuit has to be reconfigurable. Hence the control/configuration unit is accomplished via memory mapped registers, which are accessible by the normal instruction set of the processor.

The main blocks of the proposed LFSR-based PN-sequence generator are the logic for shifter and masks and the 4 configuration registers as shown in Fig. 7. The register based design allows the construction of sequences with arbitrary polynomials ($m \leq 32$) and phase shifts.

[3] the subsequent m outputs are already stored in the register

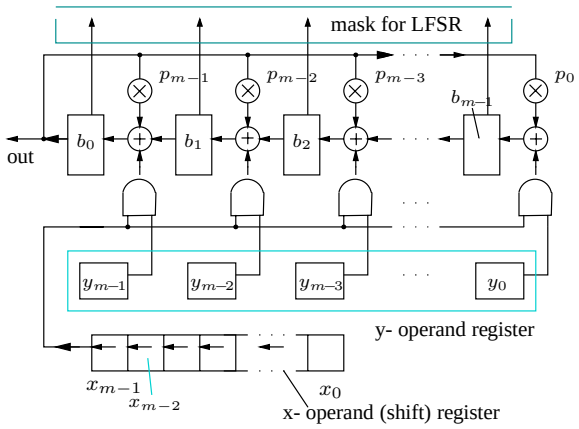

Figure 5: Serial GF-multiplier

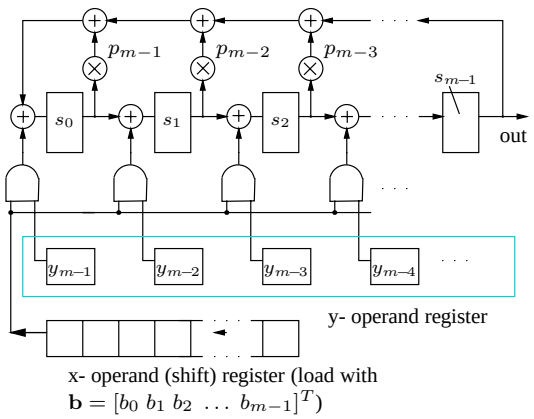

Figure 6: Proposed PN-generator

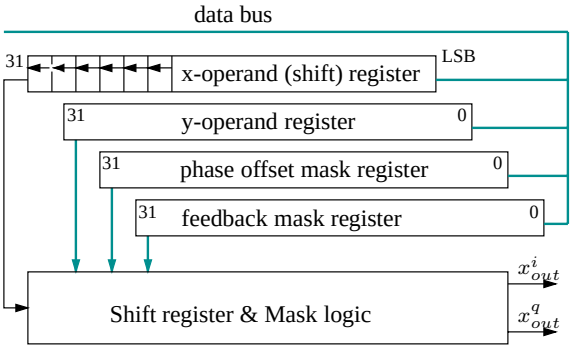

Figure 7: PN-generator blocks

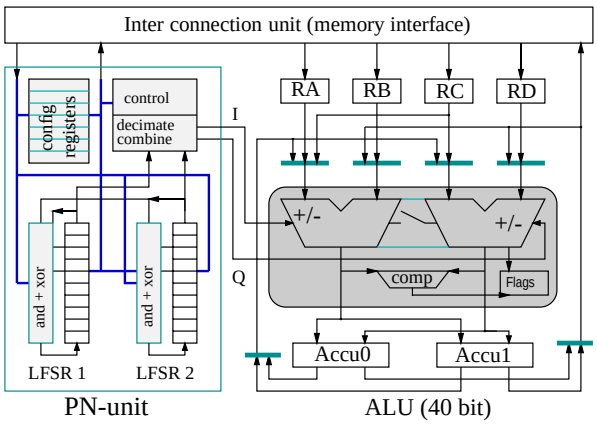

Figure 8: Data-path unit with embedded PN-generator

In order to generate a Gold-code two M-sequences are added. Hence two modified LFSRs and a mapping/decimation circuitry are needed for the complete generator (see Fig. 8).

Note: A special short scrambling mode is defined for the uplink. Its is possible to implement the complete set of 3 short-code generators of GF(2^8) and GF($2^{2\cdot8}$) within the described LFSRs with only minor modifications.

4. RESULTS

The circuit described above as well as a GF-multiplier based version have been simulated in VHDL, and synthesized for a 0.25 μm technology. The results are shown in Table 1. The synthesis

	Parameter	GF-Mul + LFSR	modified LFSR
1.	Area [KGates]	3.595	2.726
2.	Speed [MHz]	110	125
3.	# 16bit regs	20	16

Table 1: Complexity comparison of different implementations

reveals a superiority of the modified LFSR to the combined GF-multiplier-LFSR in terms of area ($\approx 25\%$ savings) and speed.

As initially mentioned, the unit is part of a data-path in a DSP, which is especially suited to handle algorithms for W-CDMA (including Forward Error Control etc.) This processor contains a highly parallel architecture [8][9] with a scalable number of data-path units. For 3GPP application, 8 parallel datapath units as depicted in Fig. 8, are needed. The outputs of the code-generator (a combination of the PN-code and the channelization code) is directly fed into the arithmetic logic unit where the despreading operation is carried out.

The area of the whole data-path unit will be increased by about 10%. Considering a speed-up by at least a factor of 2 over the software approach the realization reveals an improvement of the efficiency by more than 45% implying power savings of the same magnitude. It should be noted that the registers can be used for general purposes as well. Thus the actual application specific "overhead" reduces to the shift registers and mask circuitry.

5. CONCLUSIONS

A special PN-generator was designed which can be included into the datapath of a digital signal processor. The approach was compared to the GF-multiplier based method with respect to area/power consumption and its embedding properties into the target processor architecture.
Equipped with such hardware extensions the parallel device is capable of performing the computations of a CDMA spreading/despreading operation. This is a further step towards the complete realization of the 3GPP baseband processing algorithms within a single DSP device.

6. REFERENCES

[1] 3GPP-RAN-TS-25.213, "Spreading and Modulation (FDD)," *Technical Specification*, Jun 2000.

[2] E. H. Dinan and B. Jabbari, "Spread Codes for Direct Sequence CDMA and Wideband CDMA Cellular Networks," *IEEE Comm. Mag.*, vol. 36, no. 09, pp. 48–54, Sep 1998.

[3] D. Burshtein, "An efficient way to produce a delayed version of a maximum length sequence," *WO Patent 99/35564*, 1999.

[4] R. Pickholz, D. Schilling, and L. Milstein, "Theory of Spread-Spectrum Communications - A Tutorial," *IEEE Trans. Comm.*, pp. 855–884, May 1982.

[5] S. Lin and D. J. Costello, *Error Control Coding: Fundamentals and Applications*, Prentice Hall, 1983.

[6] R. E. Blahut, *Theory and Practice of Error Control Codes*, Addison-Wesley, 1984.

[7] M. Simon, J. Omura, R. Scholtz, and B. Levitt, *Spread Spectrum Communications Handbook*, McGraw-Hill, 1994.

[8] U. Walther, F. Tischer, and G. Fettweis, "New DSPs for Next Generation mobile Communications," in *Proc. GLOBECOM'99*, 1999, vol. V, pp. 2615–2619.

[9] G. Fettweis et al., "Breaking new grounds over 3000 MAC/s," in *Proc. ICSPAT'98*, 1998, vol. II, pp. 543–547.