Among the different emerging technologies surveyed previously in Chap. 1, crossbars are a very promising approach to integrate silicon nanowires and molecular switches into functional circuits. Chapter 2 proposed a fabrication framework for crossbars, the multi-spacer patterning technique, which has the advantage of being CMOS compatible and using only photolithography steps. Other approaches to fabricate crossbars reported in literature include nanomolds and self-assembly.

The fundamental difference between crossbars lies in the way nanowires are fabricated. As explained in Chap. 2, nanowires can be fabricated with top-down and bottom-up techniques. Top-down techniques generally use a certain kind of patterning, such as photolithography, while in bottom-up approaches nanowires are grown from a seed and need to be transferred onto the functional substrate. This transfer is generally carried out by means of fluid-assisted deposition and self-assembly of the nanowires.

The motivation towards the crossbar architecture has many reasons. First, the ability to manipulate and place single nanowires is limited. Some methods exist, such as the use of atomic force microscope (AFM) to manipulate single nanowires, but they cannot be deployed in mass production of VLSI systems. Second, the nanowire technology is immature and consequently unreliable. Then, it is highly desirable to organize the circuits into fault-tolerant structures with a large regularity and redundancy levels. Finally, crossbars can implement some circuit families in a more area-efficient way than CMOS, such as random access memory (RAM), look-up tables (LUT) or two-level logic circuits. They promise to reach the ultimate physical limit of memory and computation with the actual electron-based VLSI paradigm.

The ability of crossbars to perform computation and memory in an area-efficient way with a low reliability level motivates their implementation with a hybrid architecture including both CMOS and crossbar parts. The CMOS part is larger in area than the crossbar part, while it is more reliable. The signals coming from the CMOS part are routed to every crossbar. Hence, it is necessary to link the CMOS part, defined with standard photolithography, to the crossbar part, defined on the
sub-photolithographic scale. The interface between these two parts is the decoder, which has been the core of many research works, and which represents to topic of this chapter.

The decoder is a critical part that highly depends on the underlying nanowire technology [1–5]. The design of the decoder needs to carefully address the variability of the nanowires. In previous approaches, no decoder design technique has been suggested for the MSPT technology; and for other technologies, only binary codes have been used. The scope of this chapter is to enlarge the design space for the decoder, by developing new code families that can enhance the decoder fault-tolerance, save area and reduce the fabrication complexity. The introduced novel codes are based on a multi-valued logic (MVL), and some of them are derived from well known codes by arranging their elements in an optimized way.

Parts of this chapter have been published in [6, 7]. It is organized as follows. The crossbar architecture is first introduced, showing the different parts of the circuit and highlighting the decoder part. Then, previously proposed decoder and encoding schemes are surveyed. The construction of new families of MVL codes is then introduced, and the ability of these codes to uniquely address nanowires and improve the decoder defect-awareness while saving area is demonstrated. Thereafter, the MSPT decoder methodology is presented and its defect tolerance and technology complexity are optimized by a set of codes, some of which are specifically developed for this purpose. Finally, the chapter is concluded by a discussion of the obtained results and an assessment of the contributions of this part of the work.

3.1 Crossbar Architecture

The baseline organization of a nanowire crossbar circuit is depicted in Fig 3.1a. An arrangement of two orthogonal layers of parallel nanowires defines a regular grid of intersections called crosspoints. The separation between the two layers can be filled with a phase change material or molecular switches at the crosspoints. Information storage, interconnection or computation can be performed with these crosspoints [8, 9]. A set of contact groups is defined on top of the nanowires. Every contact group makes an ohmic contact to a corresponding distinct set of nanowires, which represents the smallest set of nanowires that can be contacted by the lithographically defined lines, called mesowires (MWs).

This configuration bridges every set of nanowires within a contact group to the outer CMOS circuit. In order to fully bridge the scales and make every nanowire within this set uniquely addressable by the outer circuit, a decoder is needed. It is formed by a series of transistors along the nanowire body, controlled by the mesowires and having different threshold voltages $V_T$ (Fig. 3.1b). The distributions of $V_T$’s is called the nanowire pattern. Depending on this pattern and the pattern of applied voltages in the decoder ($V_A$’s), one single nanowire in the array can be made conductive (Fig. 3.1c). In this case, this nanowire is said to be addressed by the applied voltage pattern.
It is possible to think of replacing each transistor at the diagonal crosspoints by an ohmic contact and to eliminate all other transistors; thus, mapping each horizontal wire onto a vertical. However, this method is technologically difficult, because the nanowire pitch is defined below the photolithographic limits. Moreover, when the number of horizontal wires (i.e., the NWs) becomes large and their size small, while that of the addressing wires (i.e., the MWs) remains at the lithographic scale, the decoder size increases and it becomes larger than the size of the crossbar part of the circuit. Then, the area gain given by the compactness of the crossbar part can be canceled by the large area of the MWs in the decoder part. By encoding the NWs, however, the number of MWs scales only logarithmically with the number of NWs. Table 3.1 illustrates the area of crossbars realized with different lithography pitches \( L_\text{l} \) and sub-lithographic or nanoscale pitches \( L_\text{n} \), and for different crossbar densities \( D \). The number of NWs \( N \) was derived from the crossbar density, whereas the number of MWs \( M \) was scaled logarithmically with \( N \).

### 3.2 Decoder and Encoding Types

The decoder is the fundamental element of the crossbar circuit that bridges the scales. Its design highly depends on the underlying nanowire fabrication technology. Yield, area and complexity of the decoder are highly depending on the

<table>
<thead>
<tr>
<th>D (kB)</th>
<th>N</th>
<th>M</th>
<th>( L_\text{l} = 65 ) nm</th>
<th>( L_\text{n} = 10 ) nm</th>
<th>( L_\text{n} = 5 ) nm</th>
<th>( L_\text{l} = 32 ) nm</th>
<th>( L_\text{n} = 10 ) nm</th>
<th>( L_\text{n} = 5 ) nm</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>512</td>
<td>9</td>
<td>129</td>
<td>38</td>
<td>117</td>
<td>32</td>
<td></td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>724</td>
<td>10</td>
<td>247</td>
<td>71</td>
<td>228</td>
<td>62</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
encoding scheme as well, making the encoding scheme an important system-level design parameter. In this section, previous design and fabrication approaches of nanowire decoders and usually used encoding schemes are surveyed.

3.2.1 Decoder Design and Fabrication

Even though the structure of the decoder circuit is simple, its reliable fabrication and design are challenging. The need to use different transistors necessitates different doping levels in specific regions on the nanowires whose location cannot be controlled precisely because the nanowire scale is below the lithographic limit. Thus, nanowires that are already doped during the fabrication process may simplify the task. When it comes to the decoder, it is fundamental to consider the nanowire fabrication technique and distinguish between differentiated and undifferentiated nanowires. Differentiated nanowires are those having a certain doping profile; they are generally fabricated in a bottom-up approach and the doping profile is defined during the nanowire growth. Undifferentiated nanowires have no specific doping profile; they are identical to each other, and they are generally fabricated in a top-down approach. Today, different techniques have been proposed to fabricate and design the decoder for both differentiated and undifferentiated nanowires.

3.2.1.1 Decoders for Differentiated Nanowires

Differentiated NWs have an axial or a radial doping profile which was defined during the NW growth process. An axial decoder was presented in [1], in which the distribution of the $V_T$'s is fully random. The NWs are dispersed parallel to each other and they are addressable when they have different $V_T$ patterns. The probability that their addresses are different may be increased by increasing the number of addressing wires. On the other hand, the radial decoder [2] relies on NWs with several radial doping shells. The remaining shells after a sequence of etchings depends on the etching order in every region. The suite of shells along the NW after all etching steps defines the NW patterns. While both axial and radial decoder give the same estimate of the number of MWs needed to address the available NWs; the radial decoder has the advantage of being less sensitive to misalignment of NWs.

Assuming that the doping regions have the same width but different patterns, then the NWs laid out parallel to each other can be addressed by the crossing perpendicular MWs. If each NW is chosen with a known probability of having a given code, then the probability that it has a unique code (i.e., sequence of doping regions) increases with the number of codes. Depending on the NW code, a certain sub-set of MWs will prohibit the conduction in almost all other NWs except the one with this given code. To address $N$ NWs, $M$ MWs are needed, $M = \left\lceil 2.2 \cdot \log_2(N) \right\rceil + 11$. With these dimensions, the axial decoder works properly,
i.e., it uniquely addresses the nanowires, with a probability greater than 99%, but it does not guarantee that the doping regions will lie directly under the MWs. The misalignment induces a slight decrease in the effective number of addressed NWs [1].

However, in the radial decoder [2], the radially doped NWs are self-aligned, and then exposed to different etchants in a certain order. The horizontal position of the etched regions is precisely defined by lithography, and the MWs are laid out at these positions. The remaining shell in each region depends on the etching order and the suite of shells along the NW after all etching steps defines the NW code. Therefore, the etching process takes care of precisely defining the sub-lithographic vertical dimensions of the contact regions. The radial decoder is technologically different from the axial decoder, but gives the same estimate of the number of MWs needed to address the available NWs, and is less sensitive to misalignment of NWs.

### 3.2.1.2 Decoders for Undifferentiated Nanowires

On the other hand, for undifferentiated nanowires, namely those fabricated in a top-down process, a mask-based decoder was presented in [10] and its ability to control undifferentiated NWs was proven. The MWs are separated from the NWs by a non-uniform oxide layer: in some locations a high-\(\kappa\) is used, in the others a low-\(\kappa\) dielectric. The high-\(\kappa\) dielectric amplifies the electric field generated by the MWs relatively to the low-\(\kappa\) dielectric. Consequently, the field effect control by the MWs happens only at the NW regions lying under the high-\(\kappa\) dielectric. The oxide mask is lithographically defined; making the decoder depending on the lithography limits. In order to address \(N\) nanowires, the mask-based decoder necessitates the use of \(M = 2 \cdot \log_2(N) + \epsilon\) mesowires, with \(\epsilon\) a small constant \(\geq 1\), which depends on the fabrication technique and the degree of redundancy to be achieved.

For undifferentiated NWs, a random contact decoder has been presented in [3, 4]. Unlike the other decoders for which the NW codes are among a known set of codes, the connections established between MWs and NWs for this decoder are fully random. It results from a deposition of gold particle onto the NWs, where the only controlled parameter is the density of particles. In order to control each of the \(N\) NWs uniquely with a high probability, \(M = 4.8 \cdot \log_2(N) + C\) mesowires are needed, with \(C\) a large constant that depend on the design parameters.

Recently, a decoder called Micro to Nano Addressing Block, MNAB, has been presented in [5] for undifferentiated NWs. Due to its analogue working principle, it needs only two MWs to address any number of NWs within a certain range depending on the technology used. The two MWs are laid out parallel to all the NWs and create an electric field in the NW array by means of the voltages applied at them. Depending on the voltage applied at each MW, the minimum of the electric field can be set at any NW which will conduct; the resistances of all other NWs will highly increase.
3.2.2 Encoding Schemes

Various code types have been investigated for decoding nanowire arrays, all of them are defined with binary logic. The binary hot code, or simply, hot code (HC), is defined with two parameters \((M, k)\), and it spans the space of code words with the length \(M\) having \(k\) occurrences of the bit ‘1’ and \((M - k)\) occurrences of the bit ‘0’ in every code word \((k \leq M)\). It is also known as the \(k\)-out-of-\(M\) code; which was first used as a defect/tolerant encoding scheme [11]. For instance, the code words 001111 and 010111 belong to the same hot code space with \((M, k) = (6, 4)\).

The binary tree code, or simply tree code (TC), with the length \(M\) is a 2-to-2\(^M\) encoder representing the \(2^M\) binary numbers 0...0 to 1...1. For instance, 000, 001 and 101 are elements of the TC space with the length \(M = 3\). However, in order for tree codes to uniquely address nanowires, it is necessary to make them reflexive, i.e., to append to every code word its \(n\)-complement. For instance, 000, 001 and 101 become respectively 000111, 001110 and 101010. These reflected (binary) tree codes are called, binary reflexive codes BRC.

Gray codes (GC) are known for their interesting properties that enhance the fault tolerance in many applications [12]. However, there has been no attempt to use Gray codes to design nanowire decoders. Gray codes are an arrangement of tree codes where successive code words differ in only one digit. For instance, the sequence of code words 010 \(\Rightarrow\) 010 taken from the binary tree code is not allowed in Gray codes, because it has two transitions, at the second and the third digits. In a GC, 010 can be only followed by 110, 000 or 011, in order to keep the transition count qual to one.

There are various types of Gray codes. The Gray code (GC) means in this work the first patented Gray code [13]. In addition, we consider also the balanced Gray code (BGC) [14], where digit changes are distributed as equally as possible among all digit positions. In this work, we assume that the digit change is 2 or less whenever we refer to the BGC. The way such a code can be derived was explained in [14].

The Gray code and its various versions are special arrangements of the tree code. Thus, in order to make nanowires addressable by these codes, we need to use their reflected form by appending to every code word its complement. We will omit to specify that the used TC, GC and BGC are reflected but we will assume it since all the codes cited in the work are used to address nanowires.

3.3 Multi-Valued Logic Encoding

Previously explored nanowire encoding schemes, i.e., codes, are binary. The code length impacts the decoder size and the overall crossbar area. It is therefore interesting to investigate the benefits of reducing the code length by using MVL
codes. The generalization of the usual codes to MVL produces novel code families that have not been explored before. In this section, the construction rules for new code families are presented. Defects that can affect them are modeled. Then, the fault-tolerance of the considered codes and their impact on the crossbar circuit in terms of reliability and area are investigated.

3.3.1 Circuit Design with Multi-Valued Logic

The research areas for multi-valued logic can be summarized in three categories: MV algebra, MV semiconductor circuits and MV network synthesis. A review of the background of MV algebra was presented in [15] and [16]. The implementation of algebraic notions into real circuits was motivated from one side by the exponential growth of interconnects in digital circuits and their limited scaling abilities [15], and from the other side by the need for higher density of information storage. The use of MVL encoding of data reduces the area needed for MVL buses and memories.

Many circuit design techniques were used to implement MVL: different current-mode MVL circuits were reviewed in [17]; while in [18] a voltage-mode MVL full adder was demonstrated. A charge- and voltage-based approach for MVL Flash memories was described in [19] and MVL SRAM memory and logic blocks fabricated with the same technology were introduced in [20].

Design tools for mapping MVL functions onto FPGA systems were presented in [21]. Optimized design methodologies for MVL PLA were introduced in [22] and [23] while considering the MVL encoding problem of inputs and outputs and the benefits of encoding in terms of number of products in the minimized function. Simplification and minimization of EXOR-sum-of-product expressions for MVL functions were presented in [24] and [25] respectively. The use of MV functional decomposition algorithms based on MV decision diagrams was investigated in [26] for logic synthesis. Efficient minimization of MVL networks with don’t – cares was presented in [27] as a way to implement MVL hardware, and also as an optimization opportunity for binary functions at the MV stage, which cannot be discovered in the binary domain.

3.3.2 Semantic of Multi-Valued Logic Addressing

In this section, we generalize the notion of encoding to multiple-valued bits by first defining some basic relations needed to identify possible codes. Some basic concepts used in encoding theory are generalized from the binary definitions stated in [28] to the multiple-valued logic. The matching between a code and its pattern corresponds here to conduction. Before introducing the impact of defects, we
consider the code (Ω) and pattern (A) spaces to be identical, realizing a 1-to-1 mapping between each other. Algebraic operations are performed as defined in the ring of integers.

**Definition 1** A multiple-valued *pattern* a, or simply a pattern a, is a suite of M digits $a_i$, in the n-valued base $\mathbb{B}$; i.e., $a = (a_0, \ldots, a_{M-1}) \in \mathbb{B}^M$, $\mathbb{B} = \{0 \ldots n - 1\}$.

A pattern represents a serial connection of M transistors in the silicon nanowire core; each digit $a_i$ of the code word represents a threshold voltage $V_{T,i}$, with the convention $a_i < a_j \iff V_{T,i} < V_{T,j}$ for all $i, j = 0 \ldots M - 1$. An analogue equivalence holds for $a_i = a_j$ and consequently for $a_i > a_j$. This convention is equivalent to discretizing the M values of $V_T$ and ordering them in an increasing order. In Fig. 3.2a, b we illustrated the pattern 002120 representing the $V_T$ sequence (0.2 V, 0.2 V, 0.6 V, 0.4 V, 0.6 V, 0.2 V).

**Definition 2** A multiple-valued *code word* c, or simply a code word c, is, similarly to a pattern, a suite of M digits $c_i$, in the n-valued base $\mathbb{B} = \{0, \ldots, n - 1\}$; i.e., $c = (c_0, \ldots, c_{M-1}) \in \mathbb{B}^M$.

A code word represents the suite of applied voltages $V_A$ at the M mesowires. These are defined such that every $V_{A,i}$ is slightly higher than $V_{T,i}$, and lower than $V_{T,i+1}$. Hence, a similar convention holds for the order of $V_{A,i}$ with respect to that of $c_i$. In Fig. 3.2c, d we illustrated the code word 202111 representing the sequence of applied voltages (0.7 V, 0.3 V, 0.7 V, 0.5 V, 0.5 V, 0.5 V).

**Definition 3** A *complement* of digit $x_i$ in a code word or pattern x is defined as: $\text{NOT}(x_i) = \overline{x}_i = (n - 1) - x_i$. The operator NOT can be generalized to the vector x, acting on each component as defined above. Notice that $\text{NOT}(\text{NOT}(x)) = x$.

**Definition 4** A pattern a is *covered* by a code word c if and only if the following relation holds: $\forall i = 0 \ldots M - 1, c_i \geq a_i$. By using the sigmoid function

$$\sigma(x) = \begin{cases} 0 & x \leq 0 \\ 1 & x > 0 \end{cases}$$
generalized to vectors: \(\sigma(x) = (\sigma(x_0), \ldots, \sigma(x_{M-1}))\), the definition above becomes: \(a\) is covered by \(c\) \(\Leftrightarrow \|\sigma(a - c)\| = 0\). Alternatively, we can define the order relations on vectors \(c\) and \(a\):

\[
\begin{align*}
c < a & \Leftrightarrow \forall i, \quad c_i < a_i \\
c > a & \Leftrightarrow \forall i, \quad c_i > a_i
\end{align*}
\]

The relation becomes relaxed (i.e., \(\leq\) or \(\geq\)) if exists \(i\) such that \(c_i = a_i\). Then, a pattern \(a\) is covered by a code word \(c\) if and only if \(a \leq c\). The same definition for covering can be generalized to two patterns or two code words.

Covering a given pattern with a certain code is equivalent to applying a suite of gate voltages making every transistor conductive. Then, the nanowire is conducting and we say that it is controlled by the given sequence of gate voltages. Figure 3.3a illustrates the case in which the code covers the pattern and the nanowire is conducting, while Fig. 3.3b illustrates the opposite case.

**Definition 5** A pattern \(a\) implies a pattern \(b\) if and only if \(\|\sigma(b - a)\| = 0\); i.e., \(b\) is covered by \(a\). We note this as follows: \(a \Rightarrow b\). Since a 1-to-1 mapping between the patterns and codes was assumed, we generalize this definition to code words: 

\[
(a^a \Rightarrow b^b) \Leftrightarrow \|\sigma(b^b - c^a)\| = 0; \text{ i.e., } c^b \text{ is covered by } c^a.
\]

This means that if a nanowire with the pattern \(a\) corresponding to the code \(c^a\) is covered by a code \(c^*\), then the nanowire with the pattern \(b\) corresponding to the code \(c^b\) is also covered by the same code \(c^*\). Applying the voltage suite \(c^*\) will result in turning on the nanowires with either pattern (see Fig. 3.4).

**Definition 6** The code words \(c^a\) and \(c^b\) are independently covered if and only if \(c^a\) does not imply \(c^b\) and \(c^b\) does not imply \(c^a\).

---

Fig. 3.3 Example of conducting and non-conducting nanowires.
(a) Conducting nanowire (code covers pattern).
(b) Non-conducting nanowire (code does not cover pattern)

Fig. 3.4 Example of implication between patterns:
\(c^a\) covers \(a\); since \(a \Rightarrow b\), then \(c^c\) covers also \(b\).
This definition means that there exists a voltage suite that turns on the nanowire with the pattern \( a \) corresponding to \( c^a \) but not that with the pattern \( b \) corresponding to \( c^b \) (see Fig. 3.5a, b). Reciprocally, there exists a second voltage pattern that turns on the nanowire with the pattern \( b \) corresponding to \( c^b \) but not that with the pattern \( a \) corresponding to \( c^a \) (see Fig. 3.5c, d).

**Definition 7** The code word \( c^a \) belonging to the set \( X \) is **addressable** if and only if
\[
\text{it does not imply any other code in } X_{\text{nf}c^a}.
\]
We define the set \( X \) to be addressable if and only if every code word in \( X \) is addressable.

Assuming that there is a 1-to-1 mapping between the code space \( \Omega \) and the pattern space \( A \), then saying that a code \( c^a \) implies no other code in \( \Omega \setminus \{c^a\} \) is equivalent to saying that it covers only the pattern \( a \) but no other pattern in \( A \setminus \{a\} \).

Thus, there exists a voltage sequence that activates only the nanowire with the pattern \( a \) but no other nanowire having its pattern in \( A \setminus \{a\} \).

**Proposition 1** A set \( \Omega \) of code words is addressable if and only if every code word in \( \Omega \) is independently covered with respect to any other code word in \( \Omega \).

**Proof** This follows directly from Definitions 6 and 7.

Consequently, an admissible set of applied voltages that uniquely addresses each nanowire corresponds to the set of code words \( \Omega \) that independently covers every pattern in \( A \). This set of patterns can be simply taken as \( \Omega \) itself, if \( \Omega \) is addressable.

### 3.3.3 Code Construction

#### 3.3.3.1 Hot Encoding

In binary logic, the \((k, M)\) **hot code space** is defined as the set of code words with the length \( M \) having \( k \) occurrences of the bit ‘1’ and \((M - k)\) occurrences of the bit
3.3 Multi-Valued Logic Encoding

‘0’ in every code word \((k \leq M)\). It is also known as the \(k\)-out-of-\(M\) code; which was first used as a defect tolerant encoding scheme \([11]\). This definition can be generalized to the \(n\)-valued logic. We first define \(k\) as an \(n\)-dimensional vector \((k_0, \ldots, k_{n-1})\), such that \(\sum_i k_i = M\). Then, the multi-valued \((k, M)\)-hot encoding is defined as the set of all code words having the length \(M\) such that each \(k_i\) represents the occurrence of the digit \(i\), \(i = 0, \ldots, n - 1\). We consider for instance the ternary logic \((n = 3)\), and we set \(k = (4, 3, 1)\) and \(M = 8\). Then, every code word in the considered \((k, M)\)-hot space contains 4× the digit ‘0’, 3× the digit ‘1’ and 1× the digit ‘2’. The considered code space includes for instance the code words 00001112 and 00210110.

**Proposition 2** The code space defined by a multi-valued \((k, M)\)-hot encoding is addressable.

**Proof** Consider two different code words \(c^a\) and \(c^b\) in the code space defined by the \((k, M)\)-hot encoding. Both codes are identical except at \(P\) different digits lying at the positions \(p_0, \ldots, p_{P-1}\). \(c^b\) is obtained by a permutation of \(\{c_{p_0}^a, \ldots, c_{p_{P-1}}^a\}\). Hence, there is at least one position \(p_i\) for which \(c_{p_i}^a > c_{p_i}^b\) holds and at least one position \(p_j\) for which \(c_{p_j}^a < c_{p_j}^b\) holds. This proves that \(\|\sigma(c^b - c^a)\| \neq 0\) and \(\|\sigma(c^a - c^b)\| \neq 0\) and that every two code words are independently covered. Then, Proposition 1 states that the whole code space is addressable.

**Example 1** For instance the code words \(a = 00001112\) and \(b = 00210110\) differ at the 3rd, 4th, 5th and last digits, in positions \(\{2, 3, 4, 7\}\). The third digit of \(a\) is smaller than the third digit of \(b\), which proves that \(b \nleq a\); and the fifth digit of \(a\) is bigger than the fifth digit of \(b\); which proves that \(a \nleq b\). Thus, both codes are independently covered.

**Proposition 3** The size of the code space defined by a multi-valued \((k, M)\)-hot encoding is maximal for \(k_i = M/n\forall i = 0, \ldots, (n - 1)\). The size of the maximal-sized space is asymptotically proportional to \(\propto n^M/M^{(n-1)/2}\) for a given \(n\).

**Proof** The number of code words is given by \(M!/(\prod k_i!);\) which can be maximized by using the Gamma function \(\Gamma(k_i + 1) = k_i!\). The Stirling formula yields the asymptotic size for large \(M\).

In this work, it is implicitly meant that the \((k, M)\)-hot code with the maximal-sized space are used, even if just \((k, M)\)-hot code are mentioned.

**3.3.3.2 N-ary Reflexive Code**

The binary tree code with the length \(M\) is a 2-to-2\(^M\) encoder representing the \(2^M\) binary numbers 0...0 to 1...1. Similarly, a \(n\)-ary tree code with the length \(M\) is defined as the set of \(n^M\) numbers ranging from 0...0 to \((n - 1)\)...(n - 1). For instance: the ternary \((n = 3)\) tree code with the length \(M = 4\) includes all
ternary logic numbers ranging between 0000 and 2222. As one can easily see, some code words imply many others from the same space: for instance 2222 implies all other codes. It is possible to prevent the inclusive character of the \( n \)-ary tree code by attaching the complement of the code word (i.e., 2222 becomes 22220000). The as-constructed code is the \( N \)-ary Reflexive Code (NRC).

**Proposition 4** The code space defined by the NRC is addressable.

**Proof** The first (non-reflected) halves of any two code words \( \mathbf{c}^a \) and \( \mathbf{c}^b \) having the total length \( M \) (\( M \) is even) differ by at least one digit at say position \( i \) (\( i = 0 \ldots (M/2 - 1) \)). Let \( \mathbf{c}^a \) be the code word such that \( c^a_i < c^b_i \). The reflection implies that \( c^a_{M/2+i} > c^b_{M/2+i} \). This proves that \( \|\sigma(\mathbf{c}^b - \mathbf{c}^a)\| \neq 0 \) and \( \|\sigma(\mathbf{c}^a - \mathbf{c}^b)\| \neq 0 \) and that \( \mathbf{c}^a \) and \( \mathbf{c}^b \) are independently covered. Then, Proposition 1 states that the whole code space is addressable.

**Example 2** We consider for instance the ternary \((n = 3)\) reflexive code words with the length \( M = 8 : \mathbf{c}^a = 22220000 \) and \( \mathbf{c}^b = 00122210 \). The codes differ at some positions; we consider for instance the first one. The first digit of \( \mathbf{c}^a \) (‘2’) is bigger than the first digit of \( \mathbf{c}^b \) (‘0’), proving that \( \mathbf{c}^a \not\leq \mathbf{c}^b \). The complement of the first digit is the fifth one (since \( M = 8 \)). Because of the complementarity of the values of the digits, the fifth digit of \( \mathbf{c}^a \) (‘0’) is smaller than the fifth digit of \( \mathbf{c}^b \) (‘2’), proving that \( \mathbf{c}^b \not\leq \mathbf{c}^a \). Consequently, \( \mathbf{c}^a \) and \( \mathbf{c}^b \) are independently covered.

In a similar way, the reflection principle works for any other code (e.g., Hamming code), making the whole code space addressable. However, in return it doubles the code length. Although the binary hot code is denser than the binary reflexive code, this statement holds for the multi-valued logic only if the codes are defect-free. This aspect is analyzed in the following sections.

### 3.3.4 Defect Models

The control of the silicon nanowires is based on the modulation of the threshold voltage of the control transistors. The proposed encoding schemes impose a distribution of the applied control voltages between the successive threshold voltages. The main issue with the threshold voltage is its variability and process-dependency: several aspects of insufficient controllability of the technology are expressed on the device level as a random variation of the threshold voltage (e.g., variation of the doping level, oxide thickness, mechanical stress of the nanowires, etc). The independent nature of these random effects and their superposition generally justify the assumption of a normal distribution of the threshold voltage (see Fig. 3.6). The MVL encoding only depends on the value of \( V_T \)’s; however other random defects, such as the nanowire breakage and the nanowire-to-metal contact quality impact the array yield (but not the addressable code space size). These random effects were modeled by a statistical factor explained in Sect. 3.3.7.
Many random errors can affect the molecular switches (bad switching, stack-at-defects,...) are not modeled, because they do not affect the decoder yield; so that their impact on the overall crossbar yield can be modeled by including a multiplication factor that reflects the statistical defects affecting the molecular switches.

### 3.3.4.1 Basic Error Model

Figure 3.6 illustrates the main assumptions for basic error models. We assume that the threshold voltages $V_{T,i}$ are equidistant; i.e., $V_{T,i+1} - V_{T,i} = 2zV_0$, $V_0$ being a given scaling voltage and $z$ is given by the technology. The applied voltages $V_{A,i}$ are set between every two successive threshold voltages $V_{T,i}$ and $V_{T,i+1}$, not necessarily in the middle, rather shifted by $vV_0$ towards $V_{T,i}$; where $v$ is a design parameter.

If the variability of $V_{T,i}$ is high or the spacing between two successive $V_{T,i}$’s is low due to the large number of doping levels, then $V_{T,i}$ may exceed a voltage $V_{X,i}$ given by $V_{A,i} - dV_0$; where $d$ will be derived in the following. While $V_T$ increases, the sensed current while $a_i$ is applied on the digit $c_i$ decreases ($a_i = c_i$) and the sensed current while $a_i + 1$ is applied on the same digit increases. The voltage $V_{X,i}$ is defined as the gate voltage which results in the decrease of the sensed current for $a_i$ by the factor $q$ from its value at $V_{T,i}$. The higher $q$, the more accurate is the sensing. Thus, $q$ is also considered as a design parameter. Assuming that the transistors are saturated, then the current in the saturation region is proportional to $(V_{A,i} - V_T)^2$, where $V_T$ is the actual threshold voltage. Consequently,
the following condition on \( V_X \) must hold: \((V_{A,i} - V_{T,i})^2/(V_{A,i} - V_{X,i})^2 = q; \) which gives: \( \delta = (x + v)/\sqrt{q} \) for long channel transistors.\(^1\) This fixes the values of \( V_{X,i}; \) when \( V_{T,i} \) exceeds \( V_{X,i}, \) the digit \( a_i \) acts as \( a_i + 1; \) its address becomes \( c_i + 1 \) and we call this case the flip-up defect.

Now, consider the case when \( V_{T,i} \) falls below \( V_{A,i} - \delta \cdot V_0 = V_{X,i-1}, \) then the current flowing while \( a_i - 1 \) is applied is not \( \sim 0 \) anymore, and always greater than \( q \) times the current flowing while \( c_i - 2 \) is applied. Then, \( a_i \) is implied by \( c_i \) and \( c_i - 1 \) but not by \( c_i - 2; \) its address is \( c_i - 1 \) which means that \( a_i \) acts as \( a_i - 1; \) this case is called the flip-down defect. The probabilities of flip-ups and -downs are given by the following expressions, which are independent of \( i. \) Here, \( f_i \) is the probability density function of \( V_{T,i}; \)

\[
p_u = \int_{V_{X,i}}^{\infty} f_i(x) \, dx \quad p_d = \int_{-\infty}^{V_{X,i-1}} f_i(x) \, dx
\]

When \( V_{T,i} \) falls within the range between the threshold values for flip-up and flip-down defects, the digit is correctly interpreted. We notice that the flip-down error never happens at digits having the smallest value, 0, since the corresponding \( V_{T,i} \) is by definition smaller than the smallest \( V_{A,i} \) available. For the same reason, the flip-up error never happens at the digits having the biggest value, \( n - 1. \) In order to study the size of the addressable code space, we consider flip-up and -down errors in the code space instead of flip-up and -down defects at the nanowires, since both considerations are equivalent.

### 3.3.4.2 Overall Impact of Variability

If \( V_T \) varies within a small range close to its mean value, then the pattern does not change, since the nanowire still conducts under the same conditions. Then, a 1-to-1 mapping between the code and the pattern space holds, which is shown in Fig. 3.7 for a ternary hot code with \( M = 3. \) On the contrary, if the \( V_T \) variation is large, then some digits may be shifted up or down, as explained above. When a pattern has a sequence of errors, it can be either covered by one or more codes or it can be uncovered. When we consider the codes, some of them cover one or more patterns and some cover no pattern under the error assumptions. The following example explains this conjecture:

**Example 3** Figure 3.7b illustrates the digit shift at some patterns. We notice that the first pattern 022 (which underwent a defect) is not covered by any code anymore. Thus its nanowire cannot be addressed. All the other patterns are covered at least by one code. Two categories among these covered patterns can be

---

\(^1\) If we consider short channel transistors, then the saturation current is proportional to \((V_{A,i} - V_T)\) and \( \delta = (x + v)/q \)
distinguished. On the one hand, the fourth pattern 120 is covered by the fourth code, which in turn covers another pattern (the fifth). Thus, by activating the fourth nanowire, the required control voltages activate either the fourth and fifth nanowire. Consequently, the fourth nanowire cannot be addressed uniquely. This case represents the patterns covered only by codes covering more than a single pattern. On the other hand, the complementary case is illustrated by the fifth pattern 100, which is covered by many codes. However, one of these codes (201) covers no other pattern except the considered one. Thus, it is possible to uniquely activate the fifth nanowire by applying the voltage sequence corresponding to the code 201.

The examples shown in Fig. 3.7 demonstrate that a pattern undergoing defects can be either i) not covered by any valid code word, in which case the nanowire cannot be identified as addressable and the pattern is useless; or ii) covered by at least one valid code word. In the second case, if two patterns or more are covered by the same code word, then this code word cannot be used because more than one nanowire would have the same address. Thus, in the second case, the pattern is only useful if at least one code word covering it covers no other pattern, insuring that the covered pattern can be addressable.

Assuming that in average every code word covers \( v \) patterns when errors happen, let \( p_t \) be the probability that a pattern becomes uncovered, and \( p_U \) the probability that a code word covers a unique pattern \( (p_U = 1 - p_t) \). Let \( |\Omega| \) be the original size of the code space and \( |\Omega'| \) the size after errors happen. The set \( \Omega' \) contains the useful addresses under defect conditions, i.e., those that address unique nanowires even though the nanowires are undergoing defects. The size of \( \Omega' \) indicates the number of nanowires that remain useful under high variability conditions. Then:

\[
|\Omega'| = |\Omega| \cdot (1 - p_t)(1 - p_U^v) \tag{3.1}
\]

In order to assess \( |\Omega'| \), we model multi-digit errors in the following sections. Then, we analytically derive \( p_t \) and \( p_U \), and we estimate \( v \) as a fit parameter from Monte Carlo simulations.
3.3.5 Errors in the k-Hot Code Space

3.3.5.1 Error Types

\( \Omega \) refers in the following to the code space of the maximal-sized multi-valued \((k, M)\) hot encoding in the base \(B = \{0, \ldots, n - 1\} \) with \( k = (k, \ldots, k) \) and \( M = k \cdot n \). We consider a code word \( c \) in \( \Omega \) undergoing a series of single digit errors. The multi-digit error is described by the following vector \( d = (d_0, \ldots, d_{n-1}) \), where each \( d_i \) represents a pair of integers \((d^u_i, d^d_i)\) expressing the number of flip-ups and flip-downs occurring in each digit group having the value \( i \). Since no flip-down occurs at digits with value 0 and no flip-up occurs at digits with value \((n - 1)\), then we impose \( d^d_0 = 0 \) and \( d^u_{n-1} = 0 \). For instance, we consider the ternary hot code 010221 undergoing a flip-up defect at the third digit, turning it into 011221. One digit ‘0’ flipped up, while no flip-error happened at the digits ‘1’ and ‘2’. This error is represented as \( d = \left( \begin{array}{ccc}
1 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0
\end{array} \right) \). Because the number of digits having the same value is by definition \( k \), it must hold for each \( d_i : 0 \leq d^u_i + d^d_i \leq k \).

We distinguish two types of multi-digit error \( d \) corresponding to uncovered (type I) and covered codes (type II) that we can formally describe the following way:

- **Type I**: \( \exists i \in \{0, \ldots, i - 2\} : d^u_i > d^d_{i+1} \).
- **Type II**: \( \forall i \in \{0, \ldots, i - 2\} : d^u_i \leq d^d_{i+1} \).

This conjecture can be illustrated by the following two examples:

**Example 4** We consider the same hot code 010221 turning into the error code 010220 with the defect \( d = \left( \begin{array}{ccc}
1 & 0 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0
\end{array} \right) \). By comparing the codes before and after the defect happened, we notice that the number of digits ‘0’ has increased, while the number of digits ‘1’ has decreased and the number of digits ‘2’ has remained the same. Any code coinciding with the error-free code at the digits ‘2’ and ‘1’ assigns ‘1’ and ‘0’ to the positions holding ‘0’ in the error code, for instance: 010221 and 110220. All these codes cover the error code because both ‘1’ and ‘0’ cover ‘0’. This remark can be generalized as follows: whenever the error induces a decrease of the higher-valued digits and an increase in the lower-valued digits, it is always possible to find codes covering the error code. \( \Box \)

**Example 5** The opposite case can be illustrated by considering the same code 010221 turning into the error code 010222 with the defect \( d = \left( \begin{array}{ccc}
0 & 1 & 0 \\
0 & 0 & 0 \\
0 & 0 & 0
\end{array} \right) \).

Here the number of digits ‘2’ has increased, while the number of digits ‘1’ has decreased and the number of digits ‘0’ has remained the same. Any code that
would cover the error code would have at least 3 digits with the value ‘2’. Such codes do not exist in the considered hot code space. Consequently the error code cannot be covered by any code in the considered space.

\[ \square \]

### 3.3.5.2 Error Type I

The proof for the multi-digit error of type I is given in the following. Consider the case that a code word in \( \Omega \) is transformed into a code word \( c^* \) by a multi-digit error \( d \) of type I. We denote by \( i \) the smallest position in \( d \) at which holds \( d_i^u > d_{i+1}^d \). The number of digits in \( c^* \) whose value is \( \geq (i + 1) \) becomes larger than the permitted number in any code word \( c^b \) in \( \Omega \), namely \( k \cdot (n - i - 1) \). No code word would imply each digit of \( c^* \) with a value \( \geq (i + 1) \). Consequently, the nanowire with the pattern equal to the code word \( c^* \) is not addressed by any permitted code word. The probability of defect type I is given by \( p_I \) and uses the recursive Algorithm 1:

\[
p_I = \sum_{u=0}^{k} \text{ProI} \left( u, 0, 0, \frac{k!}{u! \cdot (k-u)!} \cdot (p_u)^u \cdot (1-p_u)^{k-u} \right)
\]

Here, \( u = 0 \ldots k \) is a variable going through all possible numbers of flip-up errors that can happen at digit ‘0’ (flip-downs cannot occur at digit ‘0’); \( p_u \) is the probability of a flip-up error as explained in Sect. 3.3.4; and \( \text{ProI}() \) gives the probability that a type I error occurs at any higher-valued digit and that \( u \) flip-ups occur at digit ‘0’ (see Algorithm 1).

Algorithm 1 takes as input a defect described by \( \left( x^u_d \right) \) at the digit level \( i \) that can happen with a probability \( p \) (describing the history of defects happening at digit levels \( \leq i \)). It delivers the probability that the assumed defect description can lead to a type I defect. In order to calculate this probability, the algorithm

---

**Fig. 3.8** Partial representation of error subtree for error \( \left( \begin{array}{c} 1 \\ 0 \end{array} \right) \) at \( i = 0 \)

- \( p_{0,1}^{(0)} \)
- \( p_{0,2}^{(1)} \)
- \( p_{0,3}^{(2)} \)

- \( p_{1,1} \)
- \( p_{1,2} \)
- \( p_{1,3} \)
- \( p_{1,4} \)
- \( p_{1,5} \)
- \( p_{1,6} \)

- \( D(i=1) \)

- \( i=0 \)
- \( i=1 \)
- \( i=2 \)
considers all possible error sequences that can happen at digit levels \( > i \) and lead to a type I error. The algorithm is explained with the example of Fig. 3.8.

In Fig. 3.8 we assumed a ternary hot code with \( M = 6 \) and \( k = 2 \) (e.g. 001122, 012012, \ldots). Three types of errors can happen at digit ‘0’: \( \left( \begin{array}{c} 0 \\ 0 \end{array} \right), \left( \begin{array}{c} 1 \\ 0 \end{array} \right) \), and \( \left( \begin{array}{c} 2 \\ 0 \end{array} \right) \), because no flip-downs can happen at this digit. When \( u \) goes from 0 to \( k = 2 \) in Eq. 4.5 it describes these three errors. Each error has the probability \( \frac{k!}{(u! \cdot (k-u)!)} \cdot (p_u)^u \cdot (1-p_u)^{k-u} (u = 0, 1, 2) \), which are denoted by \( p_{0,1}, p_{0,2} \) and \( p_{0,3} \) respectively in Fig. 3.8. We assume now that Algorithm 4 is called for the error \( \left( \begin{array}{c} 1 \\ 0 \end{array} \right) \) at the digit ‘0’ (i.e., \( u = 1 \) in Eq. 4.5). The probability that any possible sequence of errors at higher level digits leads to a type I error is initialized at \( \pi_{\text{tmp}} = 0 \) (line 1 in Algorithm 4). We consider the digit level ‘1’, at which 6 different errors can happen: \( D(i = 1) = \left\{ \left( \begin{array}{c} 0 \\ 0 \end{array} \right), \left( \begin{array}{c} 1 \\ 0 \end{array} \right), \left( \begin{array}{c} 2 \\ 0 \end{array} \right), \left( \begin{array}{c} 0 \\ 1 \end{array} \right), \left( \begin{array}{c} 1 \\ 1 \end{array} \right), \left( \begin{array}{c} 0 \\ 2 \end{array} \right) \right\} \). Generally, for a given level \( i \) \( \notin \{0, n-1\} \) the elements of \( D \) are generated as follows: \( D = \left\{ \left( \begin{array}{c} x^u \\ x^d \end{array} \right), \text{s.t.} x^d = 0, \ldots, k \text{ and } x^u = 0, \ldots, (k-x^d) \right\} \). For \( i = n - 1 \), since no flip-ups can happen, \( D = \left\{ \left( \begin{array}{c} 0 \\ x^d \end{array} \right), \text{s.t.} x^d = 0, \ldots, k \right\} \). Every error in \( D(i = 1) \) has a probability designated by \( p_{1,1} \ldots p_{1,6} \) in Fig. 3.8 and calculated at line 5. We consider the element \( \left( \begin{array}{c} 0 \\ 0 \end{array} \right) \) in \( D(i = 1) \) for which holds: the number of flip-ups at level ‘0’ is higher than the number of flip-downs at level ‘1’. This error sequence induces a type I error. The algorithm saves the probability of this event (line 9). On the other hand, if we consider the element \( \left( \begin{array}{c} 1 \\ 0 \end{array} \right) \) in \( D(i = 1) \), then no type I error is detected and the algorithm is called iteratively for the next digit level ‘2’ (line 7). The algorithm constructs the elements of \( D(i = 2) = \left\{ \left( \begin{array}{c} 0 \\ 0 \end{array} \right), \left( \begin{array}{c} 0 \\ 1 \end{array} \right), \left( \begin{array}{c} 0 \\ 2 \end{array} \right) \right\} \) as explained above and goes through them. Only \( \left( \begin{array}{c} 0 \\ 0 \end{array} \right) \) fulfills the condition of a type I error; for which the error probability is saved (line 16). For the two others 0 is returned (line 14). Then, the updated type I error probability \( \pi \) is returned (line 19 and 21).
3.3.5.3 Error Type II

Now, let the code word \( c_a \) in \( X \) be transformed into \( c_a/C_3 \) by a multi-digit error \( d \) of type II. The number of digits in \( c_a/C_3 \) having the value \( i \) is always larger than the number of digits having the value \( (i + 1) \): It is possible to construct one or more code words \( c_b \) in \( X \) implying \( c_a/C_3 \): An intuitive way consists in starting with the smallest digit value 0, and filling the digits of \( c_b \) by 0 with respect to the positions hold by the value 0 in the digits of \( c_a/C_3 \): The procedure is repeated iteratively on next digit values until all digits of \( c_b \) are allocated (Algorithm 2). In Algorithm 2 we use the following notations: the number of digits having the value \( i \) in \( c_a/C_3 \) is \( l_i \): Their respective positions are \( p_{i_0}, \ldots, p_{i_{l_i-1}} \): The definition of the defect pattern \( d \) yields:

\[
l_i = k - d_i^a + d_{i+1}^i + d_{i-1}^a \forall i, \text{ with the convention } d_{i-1}^a = 0 \text{ for } i = 0 \text{ and } d_{i+1}^i = 0 \text{ for } i = n - 1.
\]

From the definition of the type II errors, there is at least a digit value \( i \) for which \( v_i < l_i \) holds. Each choice of \( v_i \) elements among \( l_i \) possible values (line 2) gives many possible choices for \( c_b \). This proves that it is possible to find more than one code word covering the considered error code type II. The following example explains Algorithm 2 by means of the error code \( c^* = 010220 \):

---

**Algorithm 1** \( p_1 = \text{ProI}(x^u, x^d, i, p) \)

1: \( \pi_{\text{sum}} \leftarrow 0 \)
2: Construct \( D = \text{all possible defects in subtree} \)
3: for all \( y = (y^u, y^d) \in D \) do
4: \hspace{1em} if \( i < n - 1 \) then
5: \hspace{1em} \hspace{1em} \( p \leftarrow p \cdot k!/y^a! \cdot (k - y^d)! \cdot (p_a)^y^a \cdot (p_d)^y^d \cdot (1 - p_a - p_d)^{k - y^a - y^d} \)
6: \hspace{1em} \hspace{1em} if \( y^d \geq x^a \) then
7: \hspace{1em} \hspace{1em} \hspace{1em} \( \pi \leftarrow \text{ProI}(y^u, y^d, i + 1, p) \)
8: \hspace{1em} \hspace{1em} else
9: \hspace{1em} \hspace{1em} \hspace{1em} \( \pi \leftarrow p \)
10: \hspace{1em} \hspace{1em} end if
11: \hspace{1em} else
12: \hspace{1em} \hspace{1em} \( p \leftarrow p \cdot k!/y^a! \cdot (k - y^d)! \cdot (p_d)^y^d \cdot (1 - p_a - p_d)^{k - y^d} \)
13: \hspace{1em} \hspace{1em} if \( y^d \geq x^a \) then
14: \hspace{1em} \hspace{1em} \hspace{1em} \( \pi \leftarrow 0 \)
15: \hspace{1em} \hspace{1em} else
16: \hspace{1em} \hspace{1em} \hspace{1em} \( \pi \leftarrow p \)
17: \hspace{1em} \hspace{1em} end if
18: \hspace{1em} end if
19: \( \pi_{\text{sum}} \leftarrow \pi_{\text{sum}} + \pi \)
20: end for
21: return \( \pi_{\text{sum}} \)

---
**Example 6** In order to find all code words which cover a given error code \(c^*\), Algorithm 2 is called with \(i = 0, v = k, \Delta\) a single space-holder code with \(M \times \text{**}^*\), where \('	ext{**}'\) means that the digits is not allocated yet and can take any possible value.

For \(c^* = 010220\), Algorithm 2 is called with \(i = 0, v = 2\) and \(\Delta = \{\text{**} \text{**} \text{**} \text{**} \text{**} \text{**} \text{**} \}\). We start with the digit level ‘0’, whose positions in \(c^*\) are given by \(S_1 = \{0, 2, 5\}\) (line 1). Then, \(\sigma = \{\{0, 2\}, \{0, 5\}, \{2, 5\}\}\) (line 2). Lines 4–12 set \(\Delta_{\text{tmp}}\) to \(\{0 \text{**1}, 0 \text{**1}, 0 \text{**1}, 1 \text{**0}, 1 \text{**0}, \text{**0}\}\). Line 16 calls the algorithm with \(i = 1, v = 1\) and \(\Delta\) set to \(\Delta_{\text{tmp}}\). During this call of Algorithm 2, \(S_1 = \{1\}\) (line 1) and \(S_2\) can take the only value \(\{1\}\). Then, lines 4–12 update \(\Delta_{\text{tmp}}\) to \(\{010 \text{**1}, 011 \text{**0}, 110 \text{**0}\}\). Subsequently, the algorithm is called again with \(i = 2, v = 2\) and \(\Delta\) set to \(\Delta_{\text{tmp}}\) (line 16). During this call, the lines 17–20 are executed and just fill in the remaining space-holders with the digit ‘2’. Finally, \(\Delta\) is set to \(\{010221, 011220, 110220\}\) and returned recursively to the top level. To conclude, the error code \(c^*\) can be covered by 3 possible code words.

**Algorithm 2** \(\Sigma = \text{CoveredSet}(i, \nu, \Delta, c^*)\)

1. Construct \(S_1 = \text{set of all positions of digit } i \text{ in } c^*\)
2. Construct \(\sigma = \text{set of subsets of } S_1 \text{ with } \nu \text{ elements}\)
3. if \(i < n - 1\) then
4. \(\Delta_{\text{tmp}} = \emptyset\)
5. for all \(S_2 \in \sigma\) do
6. \(\Delta_{\text{rep}} = \Delta\)
7. for all \(c^b \in \Delta_{\text{rep}}\) do
8. Allocate \(i\) to digits of \(c^b\) at positions \(S_2\)
9. Allocate \(i + 1\) to digits of \(c^b\) at positions \(S_1 \setminus S_2\)
10. end for
11. \(\Delta_{\text{tmp}} \leftarrow \Delta_{\text{tmp}} \cup \Delta_{\text{rep}}\)
12. end for
13. \(\nu \leftarrow \nu + k - |S_1|\)
14. \(i \leftarrow i + 1\)
15. \(\Delta \leftarrow \Delta_{\text{tmp}}\)
16. return \(\text{CoveredSet}(i, \nu, \Delta, c^*)\)
17. else
18. for all \(c^b \in \Delta\) do
19. Allocate \(n - 1\) to digits of \(c^b\) at remaining positions
20. end for
21. return \(\Delta\)
22. end if
3.3 Multi-Valued Logic Encoding

3.3.5.4 Immune Code Space

For simulation purposes that will be explained in Sect. 3.3.8, it is useful to know the size of the code space that did not undergo any defects, i.e., the probability that a code word is immune: $p_{\text{im}}$. In the $k$-hot code space with the length $M$ and $k = (k, \ldots, k)$, there are $k$ digits having the value 0, which can undergo only flip-up defects; thus, the probability of each one of them of being error-free is $1 - p_u$. On the other hand, there are $k$ digits having the value $n - 1$, which can undergo only flip-down defects; thus, the probability of each one of them of being error-free is $1 - p_d$. The remaining $(n - 2)k$ digits can undergo both flip-up and flip-down defects and the probability of each one of them of being error-free is: $1 - p_u - p_d$. Consequently, the probability that the whole hot code word is error-free is:

$$p_{\text{im}} = (1 - p_u - p_d)^{(n-2)k} \cdot (1 - p_u)^k \cdot (1 - p_d)^k$$

3.3.5.5 Unique Covering

In order to assess the probability that a code word $c^a$ uniquely covers an error code, we first need to enumerate the code words $c^b$ which can undergo a sequence of defects to become covered by $c^a$ and the probability of each one of these events. Then, we can derive the probability that exactly one of these events happens; which is equivalent to saying that $c^a$ covers a unique error code.

Enumerating these events is performed in two steps: We first define the set $S$ of code words $c^b$ that can be transformed into $c^a$ by a sequence of flip-ups and -downs. Then, the considered events consist in making each element $c^b$ undergo a sequence of defects that turn it not necessarily into $c^a$ but just make it covered by $c^a$. We describe the elements $c^b$ of $S$ in an abstract way consisting in a transformation matrix $T$ which describes the flip-ups and -downs that $c^b$ needs to undergo in order to turn to $c^a$. Then, each element is assigned the probability of covering under defects.

A code word $c^b$ is transformed into another code word $c^a$ by undergoing at each digit level $i = 0 \ldots n - 1$: $t^i_j$ ($j - i$)-order flip-ups (for $j = i + 1 \ldots n - 1$) and $t^i_j$ ($i - j$)-order flip-downs (for $j = 0 \ldots i - 1$). We use the following convention: $t^i_i$ designates the number of correct digits at the level $i$. A transformation $T$ affecting a whole code word is a set of the transformations $t^i [t^0_0, \ldots, t^{n-1}_n]^T$ affecting each digit level $i = 0 \ldots n - 1$. Thus, $T$ is the matrix $[t^0 \ldots t^{n-1}]$. The $(i + 1)$-th column of $T$ describes the transformation happening at digits level $i = 0 \ldots n - 1$. Since there are $k$ occurrences of the digit level $i$ in each code word, $T$ must verify: $\forall i \sum_j t^i_j = k$. The $(i + 1)$-th row indicates the number of digits with the value $i$ in the code obtained after the transformation (i.e., $c^a$). This number has to be set to $k$. Then, we derive the second condition on $T$: $\forall j \sum_i t^i_j = k$. 


Example 7 We consider the ternary hot code with $M = 6$ and $k = 2$. The digit level ‘0’ undergoes only one flip-up: $t^0 = [1, 1, 0]^\top$. The digit level ‘1’ undergoes one flip-up and one flip-down: $t^1 = [1, 0, 1]^\top$. The digit level ‘2’ undergoes no errors: $t^2 = [0, 0, 2]^\top$. Thus the code word 001122 undergoing $T = [t^0, t^1, t^2]$, can be transformed, for instance into 010222; which is not an element of the considered code space. This is due to the fact that the second condition is not fulfilled.

We suggest the transformation $\tilde{T}$ meeting both conditions:

$$
\begin{bmatrix}
1 & 1 & 0 \\
1 & 0 & 0 \\
0 & 1 & 2 
\end{bmatrix}
$$

$\tilde{T}$ would transform 001122 into 010212 (for instance), which is in the same code space.

Now, we inject defects that make $c^b$ controlled by $c^a$. At each digit value $i$, the $t^i_j$ digits, for which $j > i$ holds, must undergo a $(j - i)$-order (or higher) flip-down defect each, in order to decrease their value down to that of the corresponding digit in $c^a$. On the contrary, the $t^i_j$ digits, for which $j < i$ holds, already have a value which is smaller than that of the corresponding digit in $c^a$, so they can undergo any kind of first-order defect or be correct. The remaining $t^i_i$ digits in $c^b$ have the same values as their counterparts in $c^a$, so they can undergo a flip-down error or no error. Only first order flip errors are considered, because the likelihood of higher order defects is negligible. Thus, we set $t^i_j$ to 0 for $j > i + 1$ in order to reduce the computation time. This represents the third condition on $T$.

These conditions define the eligible set of transformations $T$. For each transformation, the probability that a sequence of defects transforms the digits of $c^b$ with the value $i$ into an error code covered by $c^a$ at the positions corresponding to these digits is: $\tilde{p}_i = (p_d)^{u_i} \cdot (1 - p_d)^{k - u_i - d_i}$ with $u_i = t^i_{i+1}$ and $d_i = \sum_{j < i} t^i_j \forall i$. The probability that the code word $c^b$ is transformed into an error code covered by $c^a$ is identical to the probability that the previous event holds for each digit value $i$. This gives the event probability $p = \prod_i \tilde{p}_i$.

Given the transformation $t^i$ affecting the $k$ occurrences of the digit value $i$, there are $\mu_i$ distributions of the $t^i_j$ transformations on the $k$ elements, given by: $\tilde{\mu}_i = k! / \prod_j t^i_j!$. Considering the $n$ possible values of $i$, there are $\mu$ code words $c^b$ transformed into an error code covered by $c^a$ with the same probability $p : \mu = \prod_i \tilde{\mu}_i$.

Example 8 The code $c^a = 001122$ in the previous example was transformed by $\tilde{T}$ to $c^b = 010212$. A series of defects transforms $c^b$ into $c^c$ that is covered by $c^a$. Thus the 1st digit in $c^b$ (corresponding to the 1st occurrence of ‘0’ in $c^a$) has to remain unchanged in order to be covered by $c^a$. The 2nd digit in $c^b$ (corresponding to the 2nd occurrence of ‘0’ in $c^a$) has to flip down to be covered by $c^a$. This gives
\[ \tilde{p}_0 = p_d (1 - p_u) \]. In a similar way, we find \( \tilde{p}_1 = p_d \) and \( \tilde{p}_2 = 1 \); Then \( p = (p_d)^2 (1 - p_u) \). The are many images of \( e^a \) through \( \tilde{T} \); for instance 102012, 102021 etc. Their number is obtained by permuting the flipping digits. For instance \( \tilde{T} \) requires that exactly one digit ‘0’ flips up: there are \( \tilde{\mu}_0 = 2! / (1! \cdot 1! \cdot 0!) = 2 \) permutations. In a similar way, we find \( \tilde{\mu}_1 = 2 \) and \( \tilde{\mu}_2 = 2 \) for the digits ‘1’ and ‘2’ respectively. Finally, the number of possible code words \( e^b \) is \( \mu = 8 \). □

Algorithm 3 represents a formulation of the method explained above. It assumes that all eligible transformations \( T \) were calculated, for instance, by selecting in an exhaustive way those meeting the 3 conditions mentioned above among all matrices in \( 0, \ldots, k^{n \times n} \). Then, it defines the set \( S \) of all \( (p, \mu) \); where \( p \) is the probabilities of the code words \( e^b \) represented by the valid transformation to undergo the right error sequence that makes it covered by \( e^a \), and \( \mu \) is the number of its equivalent occurrences.

Let \( S \) be the set of all eligible events with their respective occurrences; then, the probability that exactly one event happens (i.e., \( e^a \) covers exactly one error code) can be calculated as follows:

\[
p_U = \sum_{i=1,\ldots,|S|} \mu_i \cdot p_i / (1 - p_i) \times \prod_{i=1,\ldots,|S|} (1 - p_i)^{n_i} \tag{3.3}
\]

**Algorithm 3**

\[ S = \text{UniqueSet}(T) \]

1. Construct \( E = \{ \text{all eligible transformations } T \text{ meeting the 3 conditions above} \} \)
2. \( S \leftarrow \emptyset \)
3. for all \( T \in E \) do
   4. \( u_i \leftarrow t_{i+1} \forall i \)
   5. \( d_i \leftarrow \sum_{j<i} t_{j} \forall i \)
   6. \( \tilde{p}_i \leftarrow p_d^{n_i} \cdot (1 - p_u)^{k - u_i - d_i} \)
   7. \( p \leftarrow \prod_i \tilde{p}_i \)
   8. \( \tilde{\mu}_i \leftarrow k! / \prod_j t_{j}! \)
   9. \( \mu \leftarrow \prod_i \tilde{\mu}_i \)
10. \( S \leftarrow S \cup \{ (p, \mu) \} \)
11. end for
12. return \( S \)

In summary, this section suggested mathematical methods to estimate the probability of uncovered codes \( (p_i) \) and the probability of unique covering \( (p_U) \). The probability \( p_1 \) was calculated by constructing the whole set of type I errors and estimating the probability of each one of these errors. On the other hand, \( p_U \) was estimated from the set of code words that can undergo a sequence of errors and become covered by a given reference code word. Once \( p_1 \) and \( p_U \) are known, they
can be inserted in Eq. 4.4 in order to estimate the size of the addressable code space under defects, i.e., the number of nanowires that can be addressed in the array under high variability conditions.

3.3.6 Errors in the NRC Space

3.3.6.1 Error Types

In the following $\Omega$ refers to an arbitrary NRC space with length $M$ ($M$ is even) and base size $n$. We define a flip-up defect at digit $c_i$ in the code word $c$ to be canceled when a flip-down defect occurs at digit $c_{M/2+i}$. A canceled flip-down defect is defined in a complementary way.

As for the hot codes, we distinguish two types of multi-digit errors for reflexive codes, corresponding to the uncovered (type I) and covered codes (type II) that we can describe in the following way:

- Multi-digit errors of type I: The code word undergoes at least one uncanceled flip-up.
- Multi-digit errors of type II: The code word only experiments flip-downs and/or canceled flip-ups.

In order to illustrate this assumption, we suggest the following two examples:

**Example 9** We consider the code word $c^a = 00012221$ in the ternary ($n = 3$) reflexive code space with the length $M = 8$. $c^a$ undergoes an uncanceled flip-up error at the fourth digit, and turns to the error code $c^* = 00022221$. Any hypothetical code word that would cover $c^*$ would have at the forth digit the value 2 and at the last digit the value 0 (then the last digit would not covered); or it would have at the last digit the value 1 or 2 and at the fourth digit the value 1 or 0 respectively (then, in both cases, the fourth digit would not be covered). The uncanceled flip-up defect causes any hypothetical covering code to cover either the first half of the code or its reflected half, but never both of them at the same time. Consequently, the error code $c^*$ cannot be covered by any code in the considered NRC space.

**Example 10** On the other hand, if we consider the same code word $c^a$ undergoing the canceled flip-up defect at the fourth digit, then it turns to $c^* = 00022220$ which is a code word from the same space and it is consequently covered by itself. If, in addition to the canceled flip-up defect at the fourth digit a flip-down defect occurs, say at the fifth digit, then the code turns to the error code $c^* = 00021220$ which is in turn covered by the code words 00022220 and 10021220.
3.3.6.2 Error Type I

Here, we explain formally how codes undergoing type I errors are not covered by any code in the considered code space. Let the code word $c^*$ in $\Omega$ be transformed into $c^*$ by a multi-digit error of type I. We denote by $i$ one of the positions in the first half of the code word, at which an uncanceled flip-up error occurs. Then, $c_i^* = c_i^t + 1$ and $c_{M/2+i}^t \geq c_{M/2+i}^a$. Any code word $c^0$ which would imply the pattern corresponding to $c^*$ would verify $c_i^0 \geq c_i^* = c_i^t + 1$. Then, $c_{M/2+i}^0 = \text{NOT}(c_i^0) = n - 1 - c_i^0 \leq n - 1 - c_i^t - 1 = c_{M/2+i} - 1 < c_{M/2+i}^a$ and $\|\sigma(c^* - c^0)\| \neq 0$. Thus, there exists no code word in $\Omega$ that would cover $c^*$. The size of the addressable space is reduced by the number of code words undergoing the multi-digit error of type I.

Unlike hot codes, we give no recursive form for the probability of type I errors affecting reflexive codes, we rather derive the explicit analytical expression. However, it is much easier to consider the complementary case of the type I error, which is the type II error (probability $p_{II}$) or no errors (immune codes with the probability $p_{im}$): $p_I = 1 - (p_{II} + p_{im})$. The exact expression of $p_I + p_{im}$ and consequently the one of $p_I$ are derived in the following subsection.

3.3.6.3 Error Type II

Let the code word $c^a$ in $\Omega$ be transformed into $c^*$ by a multi-digit error $d$ of type II. It is possible to find more than one code word $c^0$ which covers the pattern corresponding to $c^*$. We construct first, say, the left half of $c^0$, then the right half is obtained by complementing the left half. The construction rule is the following ($i = 0, \ldots, M/2 - 1$):

- If $c_i^a$ undergoes a non-canceled flip-down error, then $c_i^0$ can be set to either $c_i^a - 1$ or $c_i^t$.
- If $c_i^a$ undergoes a canceled flip-down error, then $c_i^0$ is set to $c_i^a - 1$.
- If $c_i^a$ undergoes a canceled flip-up error, then $c_i^0$ is set to $c_i^a + 1$.
- If $c_i^a$ has no error, then $c_i^0$ is set to $c_i^a$.
- $c_{M/2+i}^0$ is set to $n - 1 - c_i^0$.

It is easy to verify that all patterns corresponding to $c^0$ constructed this way cover $c^*$. In order to calculate the probability $p_I$ of the multi-digit error type I, we observe the complementary event (multi-digit error type II or no error in code word $c^a$):

- If $c_i^a$ undergoes no error, then $c_{M/2+i}^a$ must undergo a flip-down or no error: $(1 - p_u - p_d)(1 - p_a)$.
- If $c_i^a$ undergoes a flip-down error, then $c_{M/2+i}^a$ can have any value: $p_d$.
- If $c_i^a$ undergoes a flip-up error, then $c_{M/2+i}^a$ must undergo a flip-down error: $p_u p_d$. 

3.3 Multi-Valued Logic Encoding
This scheme assumes that the digit $c_i^a$ have the value $1 \ldots n - 2$, i.e., it is not at the range borders. In this case, the probability that the digit $c_i^a$ and its complement in the reflected code half undergo no type I error is:

$$
\bar{p}_{1,1} = (1 - p_u - p_d)(1 - p_u) + p_d + p_u p_d
$$

If the considered digit $c_i^a$ has the value $n - 1$, then its complement $c_{M/2+i}^a$ has the value 0, and the conditions above become:

- If $c_i^a$ undergoes no error, then $c_{M/2+i}^a$ cannot undergo a flip-down error, it must be correct: $(1 - p_d)(1 - p_u)$.
- If $c_i^a$ undergoes a flip-down error, then $c_{M/2+i}^a$ can have any value: $p_d$.
- $c_i^a$ cannot undergo a flip-up error, then this case has the probability 0.

Thus, the probability that the digit $n - 1$ and its complement in the reflected code half undergo no type I error is given by:

$$
\bar{p}_{1,2} = (1 - p_d)(1 - p_u) + p_d
$$

In a similar way, we find that the probability that the digit with the value 0 and its complement in the reflected code half undergo no type I error is:

$$
\bar{p}_{1,3} = 1 - p_u + p_u p_d
$$

We notice that $\bar{p}_{1,2} = \bar{p}_{1,3}$, which is due to the reflexive principle. By averaging these probability (application of the binomial form), we can estimate the probability that any digit and its complement in the reflected code half undergo no type I error:

$$
\bar{p}_{1,\text{avg}} = \bar{p}_{1,1} \cdot (n - 2)/n + \bar{p}_{1,2}/n + \bar{p}_{1,3}/n
$$

Since a code word has $M/2$ couples of complementary digits, the probability that no type I error happens, i.e., that no error or only type II errors happen is given by:

$$
p_{II} + p_{im} = \bar{p}_{1,\text{avg}}^{M/2}
$$

Then, the value of $p_I$ can be given as $1 - (p_{II} + p_{im})$:

$$
p_I = 1 - \left(2 - p_u + p_u p_d, \frac{n - 2}{n} (p_u^2 - 1) \right)^{M/2}
$$

3.3.6.4 Immune Code Space

Now, we consider the part of the code space that did not undergo any defects in order to calculate the probability of an immune code word $p_{im}$. The number of
pairs of digits at the range borders can be any natural number between 0 and $M/2$. Because of the reflection principle, every pair among them has one digit with the value 0 and one digit with the value $n - 1$. Then, the probability of this digit pair to be error-free is $(1 - p_u)(1 - p_d)$. The error-free probability of each one of the remaining $M/2 - i$ digit pairs, which can undergo either flip-up or flip-down errors, is given by $(1 - p_u - p_d)^2$. Finally the probability that the whole code word is error-free is given by:

$$p_{im} = \sum_{i=0}^{M/2} \binom{M/2}{i} \left( \frac{2}{n} \right)^i \left( \frac{n - 2}{n} \right)^{M/2-i} \times \left( (1 - p_u - p_d)^2 \right)^{M/2-i} \left( (1 - p_u)(1 - p_d) \right)^i$$

By using the binomial form, this expression can be simplified to the weighted sum of the case that the pair is at the range border and the case that it is not:

$$p_{im} = \left( \frac{2}{n} (1 - p_u)(1 - p_d) + \frac{n - 2}{n} (1 - p_u - p_d)^2 \right)^{M/2}$$

### 3.3.6.5 Unique Covering

Now, we would like to calculate $p_U$. In principle, the same reasoning for hot codes can be applied on reflexive codes too. In order to assess the probability that a code word $c^a$ uniquely covers an error code, we first enumerate all the code words $c^b$ that can be transformed into $c^a$ by a sequence of flip-ups and -downs. Then, we calculate the probability that these code words $c^b$ undergo errors and become covered by $c^a$.

For hot codes, we considered all the transformations $T = (t^0, \ldots, t^{n-1})$ that transform $c^b$ into $c^a$ and we represented them by matrices. The fact that we considered only first-order defects fixed the limit $t_{ij} = 0$ for $j > i + 1$. Because of the reflection principle, the reflected half of the code word fixes the limit $t_{ij} = 0$ for $j < i - 1$. Consequently the transformation becomes much simpler than in the case of hot codes: for each digit $i$, $(c^a_i - c^b_i) \in \{-1, 0, 1\}$ must hold. We do not need the matrix representation in order to define the set of code words $c^b$ verifying this condition: indeed, the set of $c^b$ represents the neighborhood of $c^a$ that can be graphically represented in the space $\mathbb{B}^{M/2}$ by a hypercube with the edge length 2 and centered around $c^a$ (see Fig. 3.9). If $c^a$ has a certain number $u$ of digits at the range border (i.e., 0 or $n - 1$), then only $(c^a_i - c^b_i) \in \{-1, 0\}$ or $\{0, 1\}$ holds respectively; and the volume of hypercube is halved for each one of these digits.

Given this definition of the set of code words $c^b$, we can now enumerate them, first, by assuming that no digit is at the range borders. We consider only the first half of the code words because it completely defines the whole code word by
applying the reflection principle. Let $a$ be the number of digits in a half-word such that $c_i^b - c_i^a = 0 (x = 0, \ldots, M/2)$. There are $\mu_x = \left(\frac{M/2}{x}\right) \cdot 2^{M/2-x}$ possible code words fulfilling this condition. Each one of them can be transformed into an error code $c^b$ covered by $c^a$ if the $x$ digits in each half undergo no flip-up error, and each one of the $M/2 - x$ digits for which holds $c_i^b = c_i^a = 1$ (in the whole code word) undergoes a flip-down error. The other $M/2 - x$ digits for which holds $c_i^b - c_i^a = -1$ (in the whole code word) can undergo any kind of defect or they can be correct. The likelihood of each event is $p_x = (1 - p_u)^{2x} \cdot p_d^{M/2-x} \cdot 1^{M/2-x}$.

We can now consider the cases in which the first half of $c^b$ has $u$ digits at the range border such that $c_i^b - c_i^a \neq 0$ and $x$ is still the number of digits in the half-word such that $c_i^b - c_i^a = 0 (x = 0, \ldots, M/2$ and $u = 0, \ldots, M/2 - x)$. The number of code words having $u$ digits among $M/2 - x$ such that $c_i^b - c_i^a \neq 0$ is $2^{(M/2-x)-u} \cdot 1^u$. The average number of occurrences of each one of these words is $\left(\frac{M/2-x}{u}\right) (2/n)^u ((n-2)/n)^{M/2-x-u}$. Then, the average number of code words having $x$ digits such that $c_i^b - c_i^a = 0$ is:

$$\mu_x = \left(\frac{M/2}{x}\right) \sum_{u=0}^{M/2-x} 2^{M/2-x-u} \cdot \left(\frac{M/2-x}{u}\right) \left(\frac{2}{n}\right)^u \left(\frac{n-2}{n}\right)^{M/2-x-u}$$

This expression can be explicitly calculated by using the binomial formula, yielding the following:

$$\mu_x = \left(\frac{M/2}{x}\right) \left(\frac{2 \cdot n - 1}{n}\right)^{M/2-x}$$

We notice that the impact of the digits at the range border on the number of code words corresponding to a given $x$ is the additional factor $(n-1)/n$.

We consider now the probability of occurrence of the transformation affecting each one of the $\mu_x$ code words without discarding the digits at the range borders. The indexes $x$ and $u$ keep the same definitions as before ($x$ being again the number of digits in the half of a code word such that $c_i^b - c_i^a = 0$, and $u$ the number of
digits in the rest of the code half having their values at the range borders: \( z = 0, \ldots, M/2 \) and \( u = 0, \ldots, M/2 - z \). In addition we call \( v \) the number of digits in the set of \( \mu \) digits having their value at the range borders \( (v = 0, \ldots, z) \). First, we fix the value of \( z \). In order for \( c^b \) to undergo errors making it covered by \( c^a \), each one of the \( z \) digits must undergo no flip-up error. If the digit is among the ones at the range border (with the probability \( 2^n/z \)), then having no flip-up error happens with the probability \( 1^v \cdot (1 - p_u)^v \). If the digit is not at the range border (with the probability \( (2 - n)/z \)), then having no flip-up error happens with the probability \( (1 - p_u)^{2(z-v)} \). The remaining \( M/2 - z \) digits are divided into those for which holds \( c_i^b - c_i^a = 1 \), and which have to undergo a flip-down error, and those for which holds \( c_i^b - c_i^a = -1 \), and which can undergo any kind of error or be correct. Because of the reflection principle, the probability that the remaining \( M/2 - z \) digits undergo the right error sequence is independent of \( u \) and it is equal to \( p_d^{M/2-z} \). Consequently, the probability that each one of the \( \mu_z \) code words \( c^b \) undergoes the right defect sequence and becomes covered by \( c^a \) is given by:

\[
p_a = \sum_{v=0}^{z} \left( \frac{z}{v} \right) \left( \frac{2}{n} \right)^v \left( \frac{n-2}{n} \right)^{z-v} \times (1 - p_u)^v (1 - p_u)^{2(z-v)} p_d^{M/2-z}
\]

This expression can be explicitly calculated by using the binomial form and gives:

\[
p_a = \left( \frac{n-2}{n} (1 - p_u)^2 + \frac{2}{n} (1 - p_u) \right)^z \cdot p_d^{M/2-z}
\]

We notice that the value of \( p_a \) is just the weighted average of the cases whether the digit is at the range borders or not.

Finally, having the set \( S \) of all possible events that \( c^b \) becomes covered by \( c^a \) and their respective occurrences, the likelihood that only one event happens, i.e., only one code word in \( S \) is transformed into another word covered by \( c^0 \) is given by:

\[
p_U = \sum_{i=0}^{M/2} \mu_i \cdot p_i / (1 - p_i) \times \prod_{i=0}^{M/2} (1 - p_i)^{p_i}
\]

In summary, this section suggested the analytical expressions of the probability of uncovered codes \( (p_I) \) and the probability of unique covering \( (p_U) \). For each probability, we considered the simplified case, in which the digits are not at the range border, i.e., \( 0 \) and \( n - 1 \), then we derived the general case, where the digits can have any value. The NRC space is easier to describe analytically; thus, the given probabilities are in explicit forms unlike the recursive forms given in the k-hot code space. Once \( p_I \) and \( p_U \) are known, they can be used in Eq. 4.4 in order to analytically estimate the size of the addressable code space under defects conditions.
3.3.7 Assumptions of the Simulations

The defect models defined in Sect. 3.3.4 enable the assessment of the number of nanowires that the decoder can address uniquely even with variable threshold voltages. Based on this estimate, we can evaluate the effective capacity of the crossbar circuit, i.e., the number of addressable crosspoints. In order to keep the circuit used for simulations simple, we assumed that the crossbar operates as a memory. The memory effective capacity, i.e., the number of addressable crosspoints, is used as a metrics to evaluate the decoder and its impact on yield. Consequently, defects affecting the molecular switches were not included in the model. Then, we can estimate the effective capacity \( C_{\text{eff}} \) in a similar way to the memory yield in [1]:

\[
C_{\text{eff}} = \left( \eta P_{\text{contact}}^2 \cdot |\Omega'| \right)^2,
\]

with \( P_{\text{contact}} \) the probability that the nanowire ohmic contact is good (\( P_{\text{contact}} = 0.95 \) from [1]). In this expression, \( \eta \) is a statistic parameter that depends on the nanowire fabrication process. The area can be estimated from the geometry of the layout. Furthermore, we used the following simulation parameters: \( q = 100, V_0 = V_{\text{DD}} = 1 \text{ V}, \alpha \sim 1/n \) since the difference between the highest and lowest \( V_{T,i} \)’s is limited by the constant \( V_{\text{DD}} \). The following sections explain how the area and the value of \( \eta \) are derived.

3.3.7.1 Assumptions on the Circuits Geometry

The circuit geometry is assumed to be the same as the one explained in Sect. 3.1. The mesowires are defined according to the technology node 45 nm (DRAM half-pitch as estimated for 2010 in the ITRS review of 2007 [29]). We would highlight the assumptions resulting from the fact that we only consider feasible designs from the point of view of the physical designer (see Fig. 3.10):

- The polycrystalline silicon half pitch is designated by \( f \).
- The contact width \( (W_{\text{contact}}) \) is equal to the overlap area between the poly-Si and metal 1 layers. Although it is difficult to predict the contact width for highly scaled technologies depending on \( f \) in a generic way without any reference to a specific fabrication plant, we estimated it to be \( 2 \times f \).
- The mesowire pitch is equal to the sum of the contact width and the poly-Si half-pitch, i.e., \( 3 \times f \).
- The width of a contact group is the sum of the contact width and \( 2 \times \) the poly-silicon half-pitch; i.e., \( 4 \times f \).

The nanowire thickness and pitch are taken from the references in which the considered decoders were presented and we used the best proven values:

- Grown nanowires: the nanowire core has a thickness of 5 nm, An additional insulator shell adds 4 nm to the nanowire thickness, making the pitch = 9 nm [2].
3.3.7.2 Statistical Assumptions

Nanowires grown in a bottom-up process have several sources of defects and uncertainty due to their possible sublithographic pitch. These defects were analyzed in [1]. The impact of all these additional defects is modeled as a factor $\eta$ and the effective memory capacity for bottom-up approaches becomes:

$$C_{\text{BU}}^{\text{eff}} = (\eta_{\text{BU}}^2 \cdot P_{\text{contact}}^2 \cdot |\Omega'|)^2$$

We used the same sources of defects as those modeled in [1] and [2]:

- Probability of non-broken nanowires. $P_{\text{nbr}}$ is dependant on the nanowire length, and for 10 $\mu$m long nanowires, $P_{\text{nbr}} = 0.90$.
- Probability of unique nanowires. $P_{\text{unq}}$ is the probability that each nanowire in a contact group is unique. It depends on the code space size and the way it can be deduced is explained in [1].
- Probability of a good control of nanowires $P_{\text{cnt}}$. The nanowires may be displaced with respect to each other after the fluidic deposition process. The average value 0.80 was used as suggested in [2].
- Probability of no nanowire loss at the interfaces between contact groups. $P_{\text{int}}$ was estimated to be $(1 - 1/N)$, where $N$ is the number of nanowires in each contact group.
This factor $\eta^{BU}$ represents the impact of all these defects, thus it was set to the product of their probabilities:

$$\eta^{BU} = P_{nbr} \cdot P_{unq} \cdot P_{cnt} \cdot P_{int}$$

On the contrary, nanowires produced in a top-down approach rarely suffer from breakage ($P_{nbr} = 1$). Their codes are not assigned randomly, but by using lithography ($P_{unq} = 1$). Their ohmic contact was not reported to cause electrical loss since the nanowires are not dispersed randomly on the substrate to be functionalized ($P_{cnt} = 1$). For nanowires defined on the lithographic pitch, i.e., those fabricated with standard lithography, there is no interface loss ($P_{int} = 1$). However, those defined below the photolithography limit, for instance by using nanomolds, may have an interface loss quantized as suggested previously: $P_{int} = 1 - 1/N$, with $N$ the number of nanowires to be addressed in the contact region. Then, for top-down nanowires we have:

$$C_{eff}^{TD} = \left(\eta^{TD} \cdot P_{contact}^2 \cdot |\Omega'|\right)^2$$

with $\eta^{TD} = 1$ for top-down nanowires defined on the lithography pitch, and $\eta^{TD} = 1 - 1/N$ for top-down nanowires defined on a sub-lithographic pitch.

### 3.3.8 Simulations of the Addressable Code Space

In order to assess the variation of the addressable code space under variable $V_T$, we plotted separately the uncovered part $|\Omega|_{un} = p_{un} \cdot |\Omega|$, the addressable part $|\Omega'|$ and the immune part in which no defects occur $|\Omega|_{im} = p_{im} \cdot |\Omega|$. The fit parameter $\nu$ was estimated with Monte-Carlo simulations. Figure 3.11 shows the sizes of these subspaces for a ternary (3,14)-reflexive code depending on the $V_T$ variability.

![Fig. 3.11](image-url)Dependency of different code space subsets on $V_T$ variability
3σ-value of $V_T$. The monte-Carlo simulation confirms in the same figure the analytical results and gives the value 2.8 for the fit parameter $v$. The size of the addressable space $|Ω'|$ drops quickly when $3σ$ reaches 0.4 V. At the same time, more patterns become uncovered. Interestingly, there are more addressable than immune patterns, because some defective patterns can be randomly addressed. This tendency increases for unreliable technologies, and around 10% of the original code space size can be randomly addressed under extreme conditions. The simulation of hot codes was not shown, because the result is similar, except for large defect probabilities: under these conditions the size of the addressable space goes faster towards 0 because the construction of hot codes imposes more constraints than the NRC.

The fact that the hot code space is under more constraints than the reflexive code space could be illustrated by investigating the value of $v$ for different codes. It is important to emphasize the fact that $v$ is a statistical quantity that gives the average number of elements in the code spaces mapped onto the pattern space under variability conditions through the relation “code covers pattern”. Its value was estimated by matching the analytical results to the Monte-Carlo simulations. For hot codes, $v$ has a value between 1.3 and 1.4 for all bases and lengths used in the Monte-Carlo simulations. On the other hand, $v$ increases with the code length and the basis value for reflexive codes, which is illustrated in Table 3.2. Keeping in mind that $v$ represents the average number of codes covering a given pattern under fluctuations, we conclude that reflexive codes are under less constraints than hot codes. This is confirmed by the fact that hot codes are a subset of tree codes, which are used as a seed for reflexive codes.

The sizing of the memory blocks (i.e., the size of contact groups in Fig. 3.1) and the number of $V_T$ are dependent. As a matter of fact, Fig. 3.12 shows that increasing the number of $V_T$ has two opposite effects: on one hand, it enables the addressing of more wires with the same code length; on the other hand, it makes the transistors more vulnerable to defects and increases the number of lost codes. A typical trade-off situation is illustrated in Fig. 3.12 with the ternary (3,9) and binary (2,12) hot codes (with $(n,k) = (3,3)$ and (2,6) respectively) yielding almost the same number of addressable nanowires for $3σ$ around 0.4 V. The first one saves area because it has shorter codes, whereas the second one is technologically easier to realize (only 2 different $V_T$). The use of the ternary decoder is recommended for reliable technologies (insuring less area and more codes), but when the

<table>
<thead>
<tr>
<th>$M$</th>
<th>6</th>
<th>8</th>
<th>10</th>
<th>12</th>
<th>14</th>
<th>16</th>
<th>18</th>
<th>20</th>
<th>22</th>
</tr>
</thead>
<tbody>
<tr>
<td>$n = 2$</td>
<td>1.8</td>
<td>1.8</td>
<td>1.8</td>
<td>1.8</td>
<td>1.8</td>
<td>2.2</td>
<td>2.3</td>
<td>2.1</td>
<td>2.5</td>
</tr>
<tr>
<td>$n = 3$</td>
<td>1.9</td>
<td>2.2</td>
<td>2.2</td>
<td>2.5</td>
<td>2.8</td>
<td>2.8</td>
<td>3.0</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>$n = 4$</td>
<td>1.8</td>
<td>2.1</td>
<td>2.2</td>
<td>2.8</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>$n = 5$</td>
<td>2.1</td>
<td>2.2</td>
<td>2.7</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

Code space sizes $\geq 1$ Mbit identified by –
technology becomes more unreliable, there is a trade-off between the area-saving and the easier fabrication process.

We also investigated the impact of the placement of $V_A$ between two successive $V_{T,i}$. We postulated in an intuitive way that $V_A$ has to be the median of $V_{T,i}$ and $V_{T,i+1}$. While modeling the defects, we allowed $V_A$ to translate by a small value $\nu$. When $\nu$ increases (i.e., $V_A$ moves towards $V_{T,i}$), then the probability of a flip-up increases and that of a flip-down decreases. The opposite happens when $V_A$ moves towards $V_{T,i+1}$. The normalized number of addressable nanowires has been plotted in Fig. 3.13. For unreliable devices with $3\sigma > 0.1 \text{ V}$, the optimal position of $V_A$ is slightly shifted from the middle of $V_{T,i}$ and $V_{T,i+1}$ towards $V_{T,i+1}$ by a few tens of mV. While reliable devices show a plateau around the optimal value of $\nu$ and necessitate no accurate calibration of $V_A$, the circuit designer has to calibrate the applied

**Fig. 3.12** Number of addressable nanowires for different hot codes

**Fig. 3.13** Impact of the value of $V_A$ on the number of addressable nanowires
voltages $V_A$ in a precise way when the transistors are not reliable; otherwise a certain loss in the number of addressable nanowires has to be taken into account. This circuit level issue has to be considered for either hot or $n$-ary reflected codes (both codes showed a similar behavior). In technologies with a $3\sigma$-value below 100 mV, no calibration will be needed.

### 3.3.9 Simulations of the Effective Memory Area

#### 3.3.9.1 Top-Down Approach: GAA Decoder Based Memories

We considered the gate-all-around (GAA) SiNW technology, explained in [30], since it represents a good candidate for crossbar decoders, then we explored the memory effective capacity/area design space and we performed a simulation of the design space (Fig. 3.14). The processes based on 2 and 4 $V_T$ were considered for both hot and $n$-ary reflected codes. As Fig. 3.14 shows, the hot code generally reduces the decoder area, and consequently the memory area, because with the same code length, it is possible to address a larger code space. For instance, by using the same technology with 4 $V_T$ to fabricate a 4 kb-memory, the NRC has an area overhead of $\sim 10\%$ compared to the hot code. The use of a simpler technology with 2 $V_T$ implies an area overhead of $\sim 24\%$ for the same memory size. The area saving can be performed at either the technology or system level. It is worth to notice that the area saving is more significant for small memory sizes (typically less than 0.1 Mb), because the memory area for large memories is dominated by the area of the programmable array. The programmable array can be split into smaller blocks defined by the size of the contact groups (Fig. 3.1a) in order to reach the optimal size.

![Fig. 3.14 Memory area/capacity design space for different decoders](image)
Since the placement of $V_A$ between the successive $V_T$’s is a critical design aspect, we investigated its impact on the overall memory area/capacity design space. In Fig. 3.15 we assumed a less reliable technology with $3\sigma$-value around 1.2 V. If $V_A$ is simply placed at the median of each two successive $V_{T,i}$ and no care is taken to place it at the optimal position (i.e., $\nu = -20$ mV), then the memory has an area overhead of around 11%. On the other hand, even at the suboptimal position ($\nu = 0$), the area saving compared to the position corresponding to $\nu = 20$ mV is 17% for a hot decoder and 27% for a $n$-ary reflected decoder. These remarks confirm the importance of accurately controlling $V_A$ in crossbar memories.

### 3.3.9.2 Bottom-Up Approach: Axial Decoder Based Memories

Multi-valued logic decoding can be used also within other decoder technologies. As an example of bottom-up approaches, we chose the axial decoders. The yield of axial decoders was investigated under different scenarios, by changing the process variability $\sigma$ and the code type. The parameters used to estimated the memory area and density were given in [1] and summarized in Sect. 3.3.7. The particularity of decoders in this bottom-up approach is that they assign addresses randomly. The size of the contact group is set to the smallest value allowed in the considered technology in order to maximize the efficiency of the random address assignment [1].

We considered memory blocks with 32 kB raw density, where the raw density designates the memory density in the error-free case. We first set $\sigma$ to a very low value (below 10 mV), so that the defects have a negligible impact on the bit area. The results are plot in Fig. 3.16, using a reflexive code with $n = 2$ and 3.
The results for hot codes have a similar qualitative behavior. For \( n = 2 \), the effective bit area is large for short codes because small code spaces do not insure enough unique codes. The randomness in the bottom-up decoding schemes necessitates a code space that is large enough to insure unique addressing. On the other hand, long codes increase the size of the decoder and the cost in terms of area cancels the gain in terms of unique codes. Between these two regions, an optimal code length exists for which the effective bit area is minimal. When we use a larger number of logic levels, the code space for short codes is already large enough to insure the unique addressing with random decoders. Thus, the optimal code length becomes smaller for larger \( n \).

The same simulation described above was performed with a higher variability. The normalized results with the binary hot code are plotted in Fig. 3.17 for low (\( 3\sigma = 30 \text{ mV} \)) and larger (\( 3\sigma = 300 \text{ mV} \)) levels of variability. Each one of the two

\[ \text{Fig. 3.16 Effective bit area versus code length for binary and ternary reflexive codes and } C_{\text{raw}} = 32 \text{ kB} \]

\[ \text{Fig. 3.17 Effective bit area normalized to the minimal value versus binary reflexive code length for low and high variability and } C_{\text{raw}} = 32 \text{ kB} \]
considered cases was normalized to its own minimal value, in order to illustrate how the optimal code length varies with variability. The decrease in effective bit area does not happen in a uniform way for all codes. Consequently, the position of the optimal code is shifted depending on $\sigma$. For instance, the optimal binary reflexive code has $M = 18$ for low $\sigma$, whereas the optimum is shifted towards $M = 16$ for the same code when $\sigma$ becomes large. Designing the circuit with an overestimated reliability would result in about 10% larger effective bit area. This result shows that optimizing the choice of the decoder strongly depends on the estimated reliability level.

### 3.3.9.3 Summary

The benefits of using multi-valued logic to design bottom-up decoders is summarized in Table 3.3. Among the decoders presented in Sect. 3.2.1, the radial decoder would need several oxide shell thicknesses and the random contact decoder would need more than one level of conduction in order to be extended to $n$-ary logic. These features are not inherent to the decoders as presented in [2] and [4]; thus they cannot be extended to multi-level logic. On the contrary, it is possible to assume more than 2 levels of doping for the axial decoder and more than 1 oxide thickness for the mask-based decoder in order to perform a multi-valued logic addressing without altering the underlying decoding paradigm. Consequently, only these two decoders were extended to multi-valued logic addressing. The bottom-up approaches promise a high effective density under technological assumptions that are still to be validated. The use of ternary logic in 32 kB raw area memories saves area up to 20.2% for memories with axial decoder, and up to 11.8% for memories with masked-based decoder.

On the other hand memories using GAA decoders have a lithography-dependent pitch; and they consequently compete with the lithography-based memories. The crossbar architecture is the main reason for the smaller bit area when compared to DRAM or Flash memories. The larger the raw memory density, the better is the yield of the crossbar memory with GAA decoder (see Table 3.4). For high raw memory density, the ultimate limit is defined by the square of the minimal features (poly pitch) which is almost the same limit for Flash memories.

The area saving shown in the previous simulations is due to the decoder part, since no defects were assumed in the programmable part, where the information is

<table>
<thead>
<tr>
<th>Raw size (kB)</th>
<th>Base</th>
<th>Axial decoder</th>
<th>Mask-based</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>2</td>
<td>1576</td>
<td>622</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>1196</td>
<td>550</td>
</tr>
<tr>
<td>8</td>
<td>$\Delta$</td>
<td>24.1%</td>
<td>11.5%</td>
</tr>
<tr>
<td>32</td>
<td>2</td>
<td>846</td>
<td>423</td>
</tr>
<tr>
<td>32</td>
<td>3</td>
<td>676</td>
<td>373</td>
</tr>
<tr>
<td>32</td>
<td>$\Delta$</td>
<td>20.2%</td>
<td>11.8%</td>
</tr>
</tbody>
</table>

Table 3.3 Yield of different bottom-up decoders in terms of area per working bit ($\text{nm}^2$) at the technology node 45 nm
stored. The efficiency of using a higher logic level is related to the percentage of the decoder area compared to the whole memory; which is given in Table 3.5. For 32 kB raw memories, it ranges between 20 and 50% for bottom-up approaches and it is less than 10% for memories with GAA decoders because the programmable area is dominating. For smaller memories (8 kB raw density), the decoder part is larger in percentage because the decoder area scales logarithmically with the memory size.

### Table 3.5 Decoder area in percentage of circuit area at the technology node 45 nm for different decoder types

<table>
<thead>
<tr>
<th>Raw size (kB)</th>
<th>Base</th>
<th>Axial (%)</th>
<th>Mask-based (%)</th>
<th>Radial (%)</th>
<th>Random cont. (%)</th>
<th>GAA (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>2</td>
<td>49.8</td>
<td>45.0</td>
<td>41.7%</td>
<td>45.1%</td>
<td>14.4</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>41.2</td>
<td>37.3</td>
<td>N/A</td>
<td>N/A</td>
<td>9.8</td>
</tr>
<tr>
<td>32</td>
<td>2</td>
<td>45.8</td>
<td>36.4</td>
<td>34.4%</td>
<td>38.3%</td>
<td>9.0</td>
</tr>
<tr>
<td>32</td>
<td>3</td>
<td>36.5</td>
<td>23.9</td>
<td>N/A</td>
<td>N/A</td>
<td>4.4</td>
</tr>
</tbody>
</table>

3.4 The MSPT Decoder

In Chap. 2, the multi-spacer technique was demonstrated as a possible future technology for highly dense nanowire crossbars with a sub-photolithographic pitch. By enhancing the fabrication technique, it is possible to account for the decoder during the nanowire fabrication phase, as explained in Sect. 2.7.4. In this section, the design aspects related to the MSPT-based decoder concept are investigated. The focus will be on the optimization of the code type in order to simplify the fabrication complexity on the one hand, and to reduce the circuit variability on the other hand. The multi-valued logic codes defined in the previous section are used and optimized for the MSPT decoder.

#### 3.4.1 Design of the Decoder

The MSPT decoder fabrication technique introduced in Sect. 2.7.4 yields a decoder operation identical to the description in Sect. 3.1. However, the layout
differs because the nanowires lie within parallel caves having a symmetry axis going through their central axis as depicted in Fig. 2.27. The unique addressing of every nanowire in a half cave insures the unique addressing of every nanowire in the whole array. We will therefore consider only half caves in this study.

Every half cave contains \(N\) nanowires having \(M\) doping regions each. Recall that the pattern and doping profiles are the distribution of the threshold voltages and dopant concentrations among the \(M\) doping regions respectively. Let \(P_i = [P_i^0 \ldots P_i^{M-1}]\) and \(D_i = [D_i^0 \ldots D_i^{M-1}]\) be the pattern and doping profile of the nanowire \(i\) respectively. For the considered technique, whenever a nanowire \(i\) is patterned by receiving a doping dose, all nanowires \(k = 0, \ldots, i - 1\) receive the same doping dose simultaneously. Consequently, the doping profile of a nanowire \(i\) depends not only on its own doping dose but also on all doping doses received by the nanowires \(k = i + 1, \ldots, M - 1\). We therefore need to determine the analytical multivariable application that links \(P_j^i\) and \(D_j^i\) for \(i = 0, \ldots, N - 1\) and \(j = 0, \ldots, M - 1\), in order to specify whether we can find a set of doping profiles that results in a given set of patterns.

Assuming that a set of doping profiles exists for any set of patterns, it is possible to optimize the choice of patterns according to different cost functions. We consider first the impact of this decoding technique on the fabrication cost. The nanowire profile implies a certain number of lithography/doping steps per nanowire, \(\phi_i\) for \(i = 0, \ldots, N - 1\). From the fabrication point of view, it is of the highest importance to reduce the total number of lithography/doping steps, i.e., \(\sum \phi_i\). We therefore need to establish the link between \(P_j^i\) and \(\phi_i\) in order to minimize \(\sum \phi_i\).

Then, we consider the impact of this decoding technique on the circuit yield by analyzing the variability of the decoder. Every doping region \(j\) of the nanowire \(i\), referred to as region \((i,j)\), receives successive doping doses bit by bit. With every additional doping dose, the variability of region \((i,j)\), quantified as the standard deviation of the threshold voltage of this region \(\Sigma_j^i\), accordingly increases. It is therefore desirable to establish the link between \(P_j^i\) and \(\Sigma_j^i\) and to optimize the choice of the patterns in order to minimize the variability.

In the following section, we will derive the analytical mapping between patterns and doping profiles. Then, we will define the cost functions related to the fabrication complexity and to the threshold voltage variability. These cost functions will be minimized by choosing the best code type for the decoder.

### 3.4.2 Problem Formulation of MSPT-Based Nanowire Decoder

In this section we provide an abstract description of the decoder part of the nanowire array in a single half cave. The matrices defined below describe the most relevant design and fabrication aspects. Using these definitions, we will derive the
cost functions of the fabrication complexity and the circuit variability. Further, we assume a multi-valued logic addressing with $n$ values.

**Definition 8** The pattern matrix $P$ is an $N \times M$ matrix in $\{0, \ldots, n-1\}^{N \times M}$ representing the patterns of $N$ nanowires within a half cave, where every nanowire has $M$ doping regions.

We assume that the $N$ nanowires within every half cave are patterned and have $M$ doping regions each. The pattern corresponds to a set of $V_T$’s having any one of the $n$ possible values $V_T(0), \ldots, V_T(n-1)$. The patterns are represented by the discrete values $0, \ldots, n-1$, which correspond to the ordered discretization of the threshold voltages $V_T(0), \ldots, V_T(n-1)$. Consequently, the set of patterns on the $N$ nanowires forming one half cave can be represented by an $N \times M$-matrix in $\{0, \ldots, n-1\}^{N \times M}$.

**Definition 9** The final doping matrix $D$ is an $N \times M$ matrix in $\mathbb{R}^{N \times M}$ representing the doping level distribution along the $N$ nanowires within a half cave after the definition of the whole array.

Every $V_T$ needs a unique doping level $N_D$ fixed by the device physics and geometry [31]. Consequently, the pattern matrix, that is uniquely mapped onto a set of $V_T$’s, defines a unique final doping matrix.

**Example 11** For $n = 3, N = 3$ and $M = 4$, we assume that $V_T$ can have the values 0.1 V, 0.3 V and 0.5 V corresponding to the digits 0, 1 and 2 and to the doping levels $2, 4$ and $9 \times 10^{18}$ cm$^{-3}$. The patterns are represented with the first $N$ code words of the $n$-ary tree code. With $V$ the matrix covering all $V_T$’s, we obtain:

$$
P = \begin{bmatrix} 0 & 1 & 2 & 1 \\ 0 & 2 & 2 & 0 \\ 1 & 0 & 1 & 2 \end{bmatrix} \quad V = \begin{bmatrix} 1 & 3 & 5 & 3 \\ 1 & 5 & 5 & 1 \\ 3 & 1 & 3 & 5 \end{bmatrix} \cdot 0.1 \text{V} \quad D = \begin{bmatrix} 2 & 4 & 9 & 4 \\ 2 & 9 & 9 & 2 \\ 4 & 2 & 4 & 9 \end{bmatrix} \cdot 10^{18} \text{cm}^{-3}
$$

**Proposition 5** A non-linear bijective application $h$ maps $P$ onto $D$ as follows: $D'_i = h(P'_i) \forall i,j$

**Proof** The mapping between digits of the patterns and $V_T$ is a discrete ordering, which is a bijective application $g$. The mapping between $V_T$’s and $N_D$’s is a monotonic non-linear function $f$, which is also a a bijection. The interested reader is invited to look into [31] to obtain the exact expression of $f$. Since $h$ is a composition of $f$ and $g$, it is bijective as well.

**Definition 10** The step doping matrix $S$ is an $N \times M$ matrix in $\mathbb{R}^{N \times M}$ representing the additional doping levels after every lithography/doping step.

There is a lithography/doping procedure that follows the definition of every one of the $N$ nanowires. Every procedure $i = 0, \ldots, n-1$ is characterized by $M$ doping
levels \( [N_{D,i}^0, \ldots, N_{D,i}^{M-1}] \) along the \( M \) doping regions of the nanowires. The set of \( M \) doping levels in the \( N \) steps can be represented by the matrix \( S \) in \( \mathbb{R}^{N \times M} \).

**Proposition 6** A multi-linear application maps the elements of \( S \) onto those of \( D \) as follows: 
\[
D'_i = \sum_{k=i}^{N-1} S'_k
\]

*Proof* Every nanowire \( j \) that is defined, is subsequently patterned by means of doping doses \( [S_j^0, \ldots, S_j^{M-1}] \). Any nanowire \( i \) defined before the nanowire \( j(i < j) \) receives the same dose simultaneously. Thus, the doping level of the nanowire \( i \) is the sum of all the levels defined in the steps \( i, \ldots, N - 1 \) following the definition of the nanowire \( i \).

**Example 12** The following step and final doping matrices verify the property stated in Proposition 6. Negative and positive doping levels correspond to the doses with n- and p-type dopants respectively:

\[
D = \begin{bmatrix}
2 & 4 & 9 & 4 \\
2 & 9 & 9 & 2 \\
4 & 2 & 4 & 9
\end{bmatrix} \cdot 10^{18} \text{ cm}^{-3} \\
S = \begin{bmatrix}
0 & -5 & 0 & 2 \\
-2 & 7 & 5 & -7 \\
4 & 2 & 4 & 9
\end{bmatrix} \cdot 10^{18} \text{ cm}^{-3}
\]

**Definition 11** The technology complexity is quantified by \( \Phi \) representing the total number of additional lithography/doping steps needed to pattern the nanowires.

Every row in \( S \) (\( S_i = [S_i^0, \ldots, S_i^{M-1}] \), \( i = 0, \ldots, N - 1 \)) represents the doping doses used in a single step doping procedure. The number of unequal non-zero elements in \( S_i \) represents the number of different doses used at this doping step. The more doping doses, the more lithography steps are needed and the more complex is the fabrication. Let \( \phi_i (i = 0, \ldots, N - 1) \) be the number of unequal non-zero elements in \( S_i \), then the total number of lithography/doping steps is \( \Phi = \sum_i \phi_i \).

**Example 13** For \( S \) given in Example 12, we have: \( \phi_1 = 2, \phi_2 = 4 \) and \( \phi_3 = 3 \). Then, \( \Phi = 9 \) holds.

**Definition 12** The decoder variability is quantified by a \( N \times M \) matrix \( \Sigma \), describing the standard deviation of the threshold voltages in every doping region in the decoder of a half cave.

Every doping operation yields a \( V_T \) with a given variability \( \sigma_T \), measured as the standard deviation. In the proposed technique every doping region is doped at most \( N \) times (Proposition 6). We expect the variability to increase with increasing number of doping operations. The number of times a doping regions \( (i,j) \) receives a doping dose decreases with increasing \( i \) and increasing number of zero-elements
in the column $j$ of $S$. Let $v_i^j$ be this number, then $v_i^j = \sum_{k=i\ldots N-1} (1 - \delta(S_k^j))$, where 
$\delta(x)$ is the Kronecker delta function: $\delta(x) = 1 \iff x = 0$, otherwise $\delta(x) = 0$. 
Doping operations are assumed to be stochastically independent. The addition of two independent stochastic variables with standard deviations $\sigma_1$ and $\sigma_2$ respectively yields a stochastic variable with the standard deviation $\sigma_0 = \sqrt{\sigma_1^2 + \sigma_2^2}$. Therefore, if we define $\Sigma$ as the $N \times M$-matrix describing the variability of the decoder by setting $\Sigma_{ij}$ to the square of the standard deviation of the doping region $(i,j)$, we obtain: $\Sigma_{ij} = \sigma_0^2 \cdot v_i^j$.

**Example 14** For $S$ given in Example 12, we have:

$$S = \begin{bmatrix} 0 & -5 & 0 & 2 \\ -2 & 7 & 5 & -7 \\ 4 & 2 & 4 & 9 \end{bmatrix} \cdot \frac{10^{18}}{\text{cm}^3} \quad \Sigma = \begin{bmatrix} 2 & 3 & 2 & 3 \\ 2 & 2 & 2 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix} \cdot \sigma_T^2 \cdot v_i^j$$

**Proposition 7** Optimizing the decoder fabrication complexity consists in finding the best pattern $P$ that minimizes $\Phi$. Optimizing the decoder reliability consists in finding the best pattern $P$ that minimizes $\|\Sigma\|_1$, where $\|\Sigma\|_1$ is the sum of all elements of $\Sigma$, known as its entrywise 1-norm.

**Proof** This follows directly from Definitions 11 and 12.

### 3.4.3 Optimizing Nanowire Codes

In Sect. 3.2.2 we reviewed two main types of codes that have been used to uniquely address the nanowires in any logic with $n$ values: hot codes and tree codes. We also reviewed the properties of the Gray code, which is an arrangement of the tree code that sets the number of transitions between two successive code words to 1.

In order to insure a unique addressing, tree codes have been used in a reflexive form, i.e., by appending to every code word its $n$-complement. We will use the reflexive form of the tree code and Gray code without any explicit reference. The length of the whole code word—including the reflected part—is $M$. The first two columns in Table 3.6 represent the binary tree and Gray codes (in their reflected form) with $M = 4$. Notice that the reflection multiplies the number of transitions between successive code words by 2.

In the following, we will prove that the Gray code is the optimal arrangement of the tree code with respect to the defined cost functions. Then, we will investigate the opportunity of arranging the hot codes in a similar fashion to Gray codes in order to optimize the costs of the decoders designed with hot codes.
3.4.3.1 The Gray Code

**Proposition 8** Among all arrangements of tree codes, the Gray code minimizes the decoder cost in terms of variability $|\Sigma|_1$.

Proof For $i = N - 1, S^j_{N-1} = D^j_{N-1} = h^{-1}(P^j_{N-1})$ is fixed by the pattern of the last nanowire, i.e., by $P^j_{N-1}$. Thus, $v^i_{N-1} = 1 - \delta(S^j_{N-1}) = 1$, because $S^j_{N-1} \neq 0 \forall j$, since every region receives a doping dose in order to define the pattern of $V_T$’s of the last nanowire $P^j_{N-1}$. For $i \neq N - 1, v^i_j - v^j_{i+1} = 1 - \delta(S^j_i) = 1 - \delta(D^j_i - D^j_{i+1})$. This difference is 1 if $D^j_i \neq D^j_{i+1}$, i.e., $P^j_i \neq P^j_{i+1}$, and 0 if $P^j_i = P^j_{i+1}$. Then, $v^i_j$ can only increase by steps of 1 or remain constant with decreasing $i$ for a fixed $j$. It remains unchanged if and only if the pattern $P^j_i$ remains unchanged.

Consequently, $|\Sigma|_1$ monotonically increases with increasing transitions in the pattern matrix $P$ between every two successive rows. Given that the rows of $P$ are the code words in the chosen code space, it is desirable to use the code that minimizes the number of transitions between successive code words in order to minimize $|\Sigma|_1$. This condition is fulfilled by the Gray code. \qed

**Example 15** Instead of $P$ given in Example 11, which includes a tree code sequence with the cost $|\Sigma|_1 = 22 \cdot \sigma_T^2$ (from Example 14), we use a sequence from the Gray code that avoids the forbidden transition in $P$ 0220 $\Rightarrow$ 1012. Then we obtain $|\Sigma|_1 = 18 \cdot \sigma_T^2$:

$$P = \begin{bmatrix} 0 & 1 & 2 & 1 \\ 0 & 2 & 2 & 0 \\ 1 & 2 & 1 & 0 \end{bmatrix} \quad S = \begin{bmatrix} 0 & -5 & 0 & 2 \\ -2 & 0 & 5 & 0 \\ 4 & 9 & 4 & 2 \end{bmatrix} \cdot 10^{18} \quad \Sigma = \begin{bmatrix} 2 & 2 & 2 & 2 \\ 2 & 1 & 2 & 1 \\ 1 & 1 & 1 & 1 \end{bmatrix} \cdot \sigma_T^2.$$

**Proposition 9** Among all arrangements of tree codes, the Gray code minimizes the fabrication cost $\Phi$. \qed
Proof In a similar way to the proof of Proposition 8, we notice that the value of $S_j^i$ is unequal to zero if there is a transition $P_j^i \Rightarrow P_{j+1}^i$ between two successive code words in $P$ at the digit $j$. Then, any $\phi_i$, and consequently $\Phi$, increase with the number of transitions in $P$. Since the Gray code minimizes the number of transitions, then it is optimal with respect to $\Phi$. \hfill \Box

Example 16 The Gray code in Example 11 has a fabrication cost $\Phi = 9$ (Example 13). By using the Gray code in Example 15, the fabrication cost was reduced to $\Phi = 7$ ($\phi_1 = 2, \phi_2 = 2$ and $\phi_3 = 3$). \hfill \Box

3.4.3.2 Arranged Hot Codes

The previous section demonstrated that the arrangement of the tree code words into a sequence that minimizes the number of transitions between every successive code words, defined a new code (the Gray code) that minimizes the decoder variability and the fabrication cost. We therefore considered the question whether the code words of hot codes can be arranged in a similar way to the Gray code, such that the number of transitions is minimized, and to assess the possible benefits of such codes, that we called arranged hot code (AHC).

Since the number of digits with a given value in every hot code $(M, k)$ is fixed, then the minimum number of transitions is 2. We used an exhaustive algorithm for most of the hot codes with a reasonable code space size ($\leq 100$) for nanowire arrays, and we found that the arrangement in a Gray-code-fashion always exists. An example is given in the last two columns of Table 3.6.

It is possible to show in a very similar way to Proposition 8 and 9 that, when an arrangement of a given hot code exists, in such a way that the number of transitions between every successive code words is minimized, then this arrangement is the optimal hot code with respect to $||\Sigma||_1$ and $\Phi$ compared to all possible arrangements of the same hot code. In the next section we will therefore assess the performance of the optimized versions of both tree and hot codes in terms of fabrication complexity and circuit costs.

3.4.4 Simulations of the Decoder

3.4.4.1 Simulation Platform

In order to assess the impact of the decoder design (meaning the choice of the code type) on the fabrication complexity and the circuit features, we performed a statistical analysis of a crossbar circuit (Fig. 3.1a). The function of the crossbar circuit was assumed to be a memory as in the simulations done in the previous section. The defects affecting the molecular switches or the phase change layer
were not simulated, given the fact that only defects affecting the decoder are addressed. Then, only the defects happening at the decoder part due to the variability of the $V_T$’s in the doping regions were considered.

We assumed the same number of caves in both layers forming the crossbar. Their number and the one of the nanowires in every half cave $N$ was fixed according to the raw crosspoint density set to $D_{RAW} = 16 \text{kB}$. The number of contact groups per half cave was minimized with respect to the code type (code space size $\Omega$ and code length $M$) and geometry (lithography pitch $P_l$ and nanowire pitch $P_n$). While $\Omega$ and $M$ were used as simulation parameters, $P_l$ was set to 32 nm and $P_n$ to 10 nm. According to standard layout rules, the minimum width of every contact group had to be set to $1.5 \times P_l$. The maximum width of every contact group was limited by the width of $\Omega$ nanowires at most that can fit in every contact group.

The threshold voltages $V_T$’s were assumed to be distributed within the range 0 to 1 V, in order to account for a maximum supply voltage of 1 V. The doping levels were estimated from $V_T$’s by using the assumptions in [31] for the most common materials used in standard CMOS processes. The variability $\sigma_T$ of $V_T$ was set to 50 mV. A nanowire was assumed to be addressable if $V_T$ at every doping region varies within a small range. In this way, the probability that a nanowire is addressed was calculated from the Gaussian distributions of $V_T$’s with the standard deviations given by $\Sigma$. We accounted for nanowires that may be addressed by two adjacent contact groups, as explained in [1], and we removed them from the set of addressable nanowires. This gives the estimate for the yield of every cave $Y$. Consequently, the effective array density that denotes the number of working crosspoints can be estimated as: $D_{EFF} = D_{RAW} \cdot Y^2$.

The simulation platform is presented in a schematic way in Fig. 3.18.

![Fig. 3.18 Statistical analysis platform](image)
3.4.4.2 Simulation Results

We calculated the technology complexity $U$ for different code and logic types. The results, plotted in Fig. 3.19 for $N = 10$ show that $U$ is constant for all binary codes and equal to the double of the number of nanowires in a half cave. Higher logic level was suggested as a way to reduce the area overhead of the decoder as explained in Sect. 3.3.9. However, Fig. 3.19 shows that the higher logic level comes with some fabrication cost: 20% more steps for the tree code. For ternary and quaternary logic, the Gray code performs better than the tree code (17%) by completely canceling the fabrication complexity overhead.

The variability matrix was calculated for various types of binary codes. $N$ was set to 20 and the plots in Fig. 3.20 show the variability level at every digit in the $N \times M$-matrix $\Sigma$, as square roots of elements of $\Sigma$ normalized to $\sigma_T$. By comparing Fig. 3.20a, c and e, we see that the Gray code and its balanced version reduce the
variability level at every digit in comparison to the tree code. The balanced Gray code distributes the variability more evenly than the other codes. In this way, the average variability $\|\Sigma\|_1/(N \cdot M)$ could be reduced by 18%. Similar results were obtained for these codes with a higher logic level, as well as for hot codes and their arranged version. Next, we compared the distribution of the elements of $\Sigma$ for a fixed code type and different code lengths (Fig. 3.20a, c, e vs. b, d, f). We noticed that longer codes have less digit transitions and help reduce the average variability.

The elements of $\Sigma$ provide the inputs to estimate the crossbar yield, that we quantified as the effective crossbar density normalized by the raw crossbar density of 16 kB. The crossbar yield is plotted in Fig. 3.21 for various binary code types and lengths. The yield generally increases with increasing code length, until it reaches the maximum ($M \sim 10$ for TC/BGC and $M \sim 6$ for HC/AHC). The yield improvement of the tree code and the arranged hot codes, by increasing the code length from 6 to 10 and 4 to 8 respectively, is $\sim 40\%$. For a fixed code length, the optimized codes (i.e., BGC and AHC) perform better than their non-optimized versions (i.e., TC and HC respectively). For instance, the balanced Gray code yields 42% more than the tree code, and the arranged hot code 19% better than the hot code with the same length $M = 8$.

The dependency of the yield on the code length is explained by two factors: $i)$ the variation with $M$ of the percentage of nanowires at adjacent contact groups, which have to be removed from the set of addressable nanowires; and $ii)$ the dependency of the variability on $M$. On one hand, by increasing $M$ and keeping the code type fixed, the code space size increases and less contact groups are needed, so less nanowires are removed at adjacent contact groups and the yield increases. This effect saturates when the code space size is large enough to neglect the number of nanowires at adjacent contact groups. On the other hand, the average variability $\|\Sigma\|_1/(N \cdot M)$ decreases with increasing $M$, because longer codes have less digit transitions, as we showed previously. So, for a fixed code type, the yield, first, increases with increasing $M$, then it starts decreasing for larger $M$ because the increasing number of digits cancels the benefits of decreasing variability of each digit taken separately. This decrease is just slightly seen for the hot code when $M$ increases beyond 6; and for other code types, it starts appearing from $M \sim 10$. 

![Fig. 3.21 Crossbar yield in terms of percentage of addressable crosspoints for different binary code types and lengths](image)
From the geometrical data, we estimated the crossbar area; then, by considering the effective density, we estimated the bit area for different code types and lengths (Fig. 3.22). For the tree code and its optimized versions (Gray and balanced Gray codes), the bit area decreases with increasing code length mainly due to the vanishing effect of adjacent contact regions, as explained previously. An area saving by 51% can be achieved by setting $M$ to 10 instead of 6 for the tree code. The balanced Gray code yields a denser crossbar than the Gray code, which in turns yields better than the tree code; for instance crossbars with the balanced Gray code are 30% denser than those with the tree code (for $M = 8$). The hot and arranged hot codes yield the most dense crossbars with $M = 6$. For larger $M$, the bit area slightly start to increase, as the yield starts to decrease because of higher variability of longer codes, as explained previously. The arranged hot code performs better than the hot code with 13% less bit area for $M = 6$. Among all these codes, the smallest bit area is 169 nm$^2$ for the balanced Gray code, followed by the arranged hot code with 175 nm$^2$.

3.5 Discussions

This chapter represents an abstract formalism of nanowire encoding for different fabrication technologies. The idea of the chapter is to consider the nanowire part included in the decoder in an abstract way, as a sequence of digits in a code word. The impact of the physical variation of the underlying nanowire was modeled in an abstract way also, as a variation of the code word. The physical effect inducing the code word variation is assumed to be the variability of the threshold voltage. However, the operation of the crossbar is current-driven, i.e., the level of current indicates whether a crosspoint is addressed or not. It is therefore useful to extend the physical defect model to other sources of variation of the drive current, such as the nanowire cross-section dimension and the oxide thickness in the decoder part of the nanowires.

The purpose of the simulations presented in this chapter is to assess the impact of the decoder design (i.e., code type and decoder size) on yield and area. If a more
accurate estimation of the memory yield is needed, then the defect model of the molecular switches has to be included in the model as well.

The predicted results are more reliable for nanowires with reasonably small width and pitch. It is difficult to accurately determine how small reasonably small dimensions are; but the results are expected to hold beyond 10-nm. If the nanowire thickness is much less than 10 nm, then the number of dopants is expected to be too small to generate an accurate statistical distribution of the threshold voltage, unless a high $\sigma_T$ and an approximative statistical description are accepted. If the nanowire pitch is too small, then every two adjacent nanowires are expected to have an electrostatical coupling and to mutually affect their threshold voltages, in a similar way to SOI double gate transistors [32].

A possible question that may arise with respect to the proposed approach is: how much is the benefit of using a computational model compared to Monte-Carlo simulations? Looking at the proposed models, it is clear that the most of them are explicit analytical expressions that can be computed instantaneously; and only some of those models describing hot codes are based on an iterative approach that is exhaustive in some cases necessitating all code words to be listed. Only these special cases are expensive in computation time, but they are still faster than Monte-Carlo simulation with many thousands of runs, given the fact that only reasonable code spaces with no more than a few hundreds of code words are needed for crossbars. In general, Monte-Carlo simulations are possible, but the utilization of the given computational models gives a better insight into the impact of every circuit or process parameter on the final result.

The use of MVL to encode nanowires showed non-negligible benefits in terms of area and yield, namely in dense crossbars with a small pitch between nanowires. The decoder, which represents the link between the crossbar and the CMOS part of the circuit, has to be designed in MVL as well, and needs to convert all signals from the binary level that is used in the CMOS part. It is consequently important to consider the costs of the proposed solution in terms of area overhead, power consumption and delay caused by the 2-to-$n$ conversion.

The MSPT decoder is the first proposed digital decoder for MSPT-based nanowires. Its yield and area can be improved by optimizing the choice of the code type and length. However, it comes with a high cost in terms of fabrication complexity, i.e., number of photolithography/doping steps. The impact of this problem can be reduced by considering the advantage of the MSPT process to parallelize the processing within one single batch (by increasing the number of crossbars processed in parallel) and among different batches (by processing different wafers at the same time) as explained in Sect. 2.8 and Fig. 2.28. This decoder technology is not the only one, and it represents another solution besides the existing ones. The choice among different technologies will be made depending on the most relevant aspect for the final product: CMOS-compatibility, time-to-market, yield, etc.
3.6 Chapter Contributions and Summary

In this chapter, a novel approach to model nanowires and their defects in the decoder part of the crossbar is presented. The abstraction level gives a better insight into the impact of the decoder design parameters on the whole circuit. In this approach, the nanowires are modeled as a set of code words in a code space, and the variability affecting their physical parameters is modeled as errors affecting the code space.

This chapter proposes a method to construct a new set of codes that can be used to encode nanowires. The new codes are a generalization of hot and tree codes. Besides their ability to uniquely address the nanowires in a compact way, they offer interesting trade-off opportunities between code size and error probability if the variability is high. Another set of codes is obtained by arranging hot codes in an optimized way (like Gray codes), and shown to be compact and fault-tolerant. The existence of this set of codes has not been known and its use has not been considered in the past.

The decoder design has been given a novel direction in this chapter. Optimizing the decoder design is not only a matter of physical dimensions of the decoder, but highly depend on the encoding scheme as well. Then, the decoder design can be driven by yield: there are different trade-off situations between the circuit area and yield depending on the code size and type used in the decoder.

This chapter explores for the first time the design aspects of the MSPT-decoder. It presents a novel approach to look at the problem across the fabrication and circuit design. The wide choice of codes spans the design space between the fabrication complexity and the overall circuit variability, and offers novel opportunities to simplify the fabrication, while keeping the variability of the circuit low enough.

The variability caused by the small nanowire dimensions is driving this chapter. The decoder design methodology presented here takes the variability into account, and tries to minimize the number of nanowires which are badly addressed; i.e., either not addressed by the decoder, or sharing the same address with other nanowires. The existence of such codes is demonstrated in this chapter, and the way to minimize their occurrence is the core of the proposed design methodology. The system needs to be provided with a method to detect their existence and to discard them. Such a system provides a decoder test procedure, which is the topic of the following chapter.

References

12. Gardner M (1972) The curious properties of the Gray code and how it can be used to solve puzzles. Sci Am 227:106–109