1 Optimal encoding inside a just-barely 3D thin rectangle

In this section, we give an optimal encoding construction that self-assembles the bits of an input bit string along the top of a thin rectangle.

Let $x = x_0 x_1 \ldots x_{n-1}$ be a string with $x_i \in \{0, 1\}$. We define a just-barely 3D shape $L(h, x, w, s)$ that is essentially a very thin rectangle (a thick line, actually). More precisely, this shape is an approximate (thin) rectangle with height $h \in \mathbb{Z}^+$ and width proportional to the number n of bits in x. Furthermore, this shape geometrically encodes the bits in x using one-tile-high *bit bumps* that protrude from the north edge of the rectangle. Each bit is encoded by a line of tiles (the "bump") of width w in the plane $z = 0$ (resp., $z = 1$) if the corresponding bit value is 0 (resp., 1), with an additional spacing of s empty locations on each side of the bump. Therefore, each bit occupies $w + 2s$ positions in a horizontal line. Formally, this shape is defined by $(R(h, x, w, s) \times \{0\}) \cup B(h, x, w, s) \subseteq L(h, x, w, s) \subseteq (R(h, x, w, s) \times \{0, 1\}) \cup B(h, x, w, s)$, where $R(h, x, w, s) = \{0, 1, 2, \ldots, n(w+2s)-1\} \times \{0, 1, 2, \ldots, h-1\}$, $B = \bigcup_{i=0}^{n-1} B_i$, and $B_i = \{i(w+2s)+s, i(w+2s)+s+1, \ldots, i(w+2s)+s+w-1\} \times \{h\} \times \{x_i\}$.

We will give a construction that proves the following.

Theorem 1. *For $x \in \{0,1\}^n$, $K^1_{USA}(L(5, x, 2, 2)) = O\left(\frac{n}{\log n}\right)$.*

Going forward, let k be the smallest integer satisfying $2^k \geq \frac{n}{\log n}$, $m = \left\lceil \frac{n}{k} \right\rceil$ and write $x = w_0 w_1 \ldots w_{m-2} w_{m-1}$, where each w_i is a k-bit substring. Note that w_0 is padded to the left with leading 0's, if necessary. The basic idea of our construction is to have a modified binary counter count from 0 to $m-1$ and extract each of the m k-bit substrings of x in order. We will use the notation $width(m)$ to denote the width of (number of bits in) the modified binary counter. In the counter for the construction, the bits are configured horizontally, right underneath the bits of x. Figure 1 shows a high-level overview of a valid optimal encoding instance, drawn to scale.

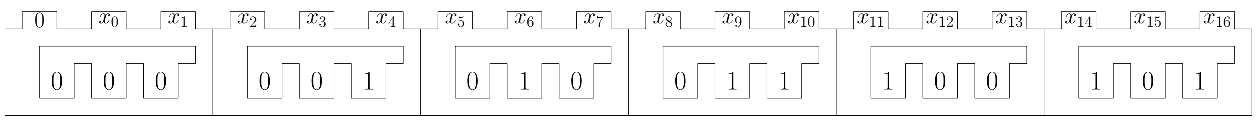


Figure 1: A high-level overview of the construction for $n = 17$, drawn to scale, showing the horizontal configuration of the bits of the counter. In this example, $k = 3$ is the smallest integer greater than or equal to $\frac{17}{\log 17}$, $width(m) = 3$ and w_0 is padded to the left with one leading 0. This is technically a valid optimal encoding instance.

It is worthy of note that, in the valid optimal encoding instance depicted in Figure 1, we have $width(m) = 3$ and $k = 3$. It turns out that, in general, k and $width(m)$ are very closely related.

Lemma 1. *$k \leq width(m) \leq k + 1$.*

Proof. Observe that $2^k < \frac{2n}{\log n}$. If this were not the case, then we would have $2^k \geq \frac{2n}{\log n}$. Dividing both sides by 2, we get $2^{k-1} \geq \frac{n}{\log n}$. Thus, $k' = k - 1$ would satisfy $2^{k'} \geq \frac{n}{\log n}$, contradicting the fact that k is the smallest integer satisfying $2^k \geq \frac{n}{\log n}$. Thus, we have

$$\frac{1}{2}\log n < \log n - \log\log n \leq k < \log n - \log\log n + 1 < \log n.$$

We will first prove that $\frac{n}{k} \leq 2^{k+1}$. Assume that $\frac{n}{k} > 2^{k+1}$. Then, we have

$$2^k < \frac{n}{2k} < \frac{n}{2\frac{1}{2}\log n} = \frac{n}{\log n}.$$

This is a contradiction because we know that $2^k \geq \frac{n}{\log n}$. We will now prove that $\frac{n}{k} > 2^{k-1}$. Assume that $\frac{n}{k} \leq 2^{k-1}$. Then, we have

$$2^k \geq \frac{2n}{k} > \frac{2n}{\log n}.$$

This is a contradiction because we know that $2^k < \frac{2n}{\log n}$. Thus, we have

$$2^{k-1} < \frac{n}{k} \leq 2^{k+1}.$$

The width of a counter that counts from 0 to $m-1$ is $width(m) = \lceil \log m \rceil$ and we have:

$$width(m) = \lceil \log m \rceil = \left\lceil \log \left\lceil \frac{n}{k} \right\rceil \right\rceil = \left\lceil \log \frac{n}{k} \right\rceil,$$

See [?] for a proof of the final equality. The proof concludes with the following two cases:

1. First, suppose that $2^{k-1} < \frac{n}{k} \leq 2^k$. In this case, $\left\lceil \log \frac{n}{k} \right\rceil = k$.
2. Second, suppose that $2^k < \frac{n}{k} \leq 2^{k+1}$. In this case, $\left\lceil \log \frac{n}{k} \right\rceil = k+1$.

□

In $L(5, x, 2, 2)$, the effective width of each bit is 4 tiles (the actual width of each bit is 2 tiles and there is 1 additional space on either side of each bit). This means each block has a total width of $4k$ tiles. Each bit of the counter in the construction will be represented by a series of two gadgets of total width 3. Thus, by Lemma 1, all the bits of the counter can be represented in a horizontal configuration entirely underneath each block and with room to spare. Figure 2 depicts a complete example of the construction that does not correspond to a valid optimal encoding instance but does show the connectivity of all the tiles. Within a given block, the construction works as follows.

First, the bits of the counter self-assemble from left-to-right using a series of `Write_counter_bit` gadgets (see Figure 5), where each `Write_counter_bit` gadget is followed by a corresponding `Spacer` gadget (see Figure 4). The first `Write_counter_bit` gadget binds to and takes as input the value of the counter from either the `Seed` gadget (see Figure 3) or a previous `Start_next_block` gadget (see Figure 13). By construction, each `Write_counter_bit` gadget has a height of 3 tiles because this allows each bit to be read twice without being re-written in between reads.

After the bits of the counter have been written, the `Start_read_counter_bits_rl` gadget (see Figure 6) initiates the process of reading the bits of the counter from right-to-left using a series of `Read_counter_bit_rl` gadgets (see Figure 7). The `Read_counter_bit_rl` gadget that reads bit i of the counter, for $i = 0, \ldots, width(m)$, inputs a bit string of length i and outputs a bit string of length $i+1$, with the most significant bit equal to the bit that it read. Each `Read_counter_bit_rl` gadget guesses the value of the next bit by attempting to self-assemble a path in both the $z = 0$ and $z = 1$ planes. However, by construction, exactly one of these paths is prevented from proceeding and the correct bit is read.

The `Start_extract_bits` gadget (see Figure 8) binds to the final `Read_counter_bit_rl` gadget and maps the m-bit value of the counter to the current k-bit block. That is, the input to the `Start_extract_bits` gadget is the value of the counter as a $width(m)$-bit string and it outputs to the first `Extract_bit` gadget (see Figure 9) the k-bit string w_j, where j is the value of the counter in decimal.

Then, a series of `Extract_bit` gadgets self-assembles from left-to-right, where each bit in the current block is extracted. Each `Extract_bit` gadget inputs a bit string of length i, for $i = 0, \ldots, k-1$, self-assembles the geometric pattern of tiles that corresponds to its most significant bit, removes the most significant bit and then outputs the resulting bit string of length $i-1$ to the next `Extract_bit` gadget via two consecutive `Spacer` gadgets.

After the final `Extract_bit` gadget, the `Start_read_counter_bits_lr` gadget (see Figure 10) initiates the process of re-reading the bits of the counter. The width of the `Start_read_counter_bits_lr` gadget is configured to be $4k$ (the width of a block in the construction). The bits of the counter need to be re-read because the value of the counter is – and must be – forgotten after the `Start_extract_bits` gadget maps the value of the counter to the current block.

The bits of the counter are re-read for the second (and final time) from left-to-right via a series of `Read_counter_bit_lr` gadgets (see Figure 7). The `Read_counter_bit_lr` gadgets read the bits of the counter similar to but in reverse order of the previously-described `Read_counter_bit_rl` gadgets.

To the final `Read_counter_bit_lr` gadget, a series of `Find_opening` (see Figure 12) gadgets self-assemble until they are prevented from proceeding to the right by a previous portion of the `Start_read_counter_bits_lr` gadget. Since the width of a block in the construction is $4k$ tiles and each bit of the counter has an effective width of 3, the series of `Find_opening` gadgets will be arbitrarily long. Once the series of `Find_opening` gadgets is prevented from proceeding to the right, depending on the value of the counter, either a `Start_next_block` (see Figure 13) or `Last_block` (see Figure 14) gadget self-assembles.

On the one hand, f the value of the counter is less than $m-1$, then the value of the counter is incremented and propagated via the `Start_next_block` gadget to the first `Write_counter_bit` gadget in next block. On the other hand, if the value of the counter is equal to $m-1$, then that was the last block and the construction terminates with the self-assembly of the `Last_block` gadget.

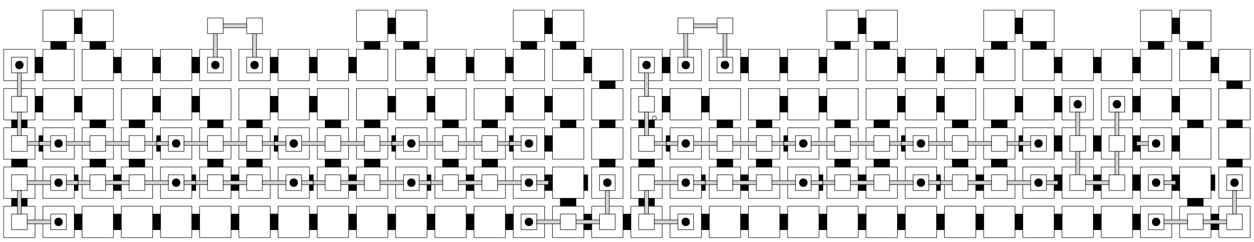
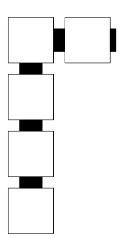


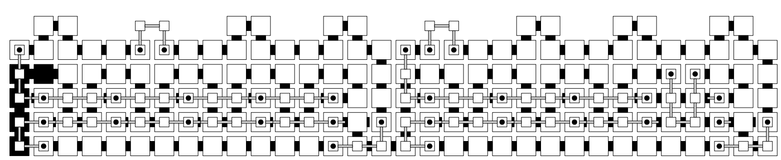
Figure 2: An example of the construction that does not correspond to a valid optimal encoding instance. In this example, $n = 8$ with $x = 01001000$ and $k = 4$. The seed tile type is the tile in the leftmost column and second from the top. The last tile to attach is the bottommost tile in the rightmost column.

In the following subsections, we give the full details of all the tile types that uniquely self-assemble into $L(5, x, 2, 2)$.

1.1 The `Seed` gadget



(a) The tile types for the general `Seed` gadget. The bottommost tile is the actual seed tile type for the construction.



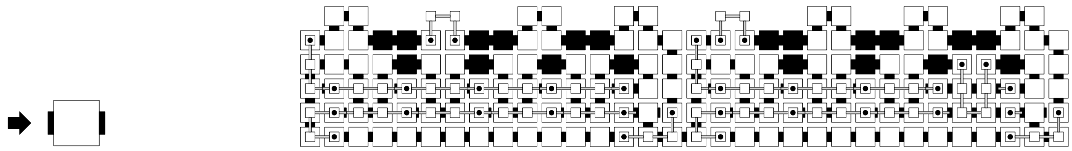
(b) The black tiles show the location of the `Seed` gadget in an example construction.

Figure 3: The `Seed` gadget.

Create `Seed`($\langle$`write_counter_bit`, $\underbrace{0\cdots0}_{width(m)}$ $\rangle$) from the general gadget shown in Figure 3a. Here, the only parameter to the general `Seed` gadget represents its output glue.

1.2 The `Spacer` gadget

The `Spacer` gadget shown in Figure 4 is used throughout the construction.



(a) The general `Spacer` gadget.

(b) The black tiles show the location of the `Spacer` gadgets in an example construction.

Figure 4: The `Spacer` gadget.

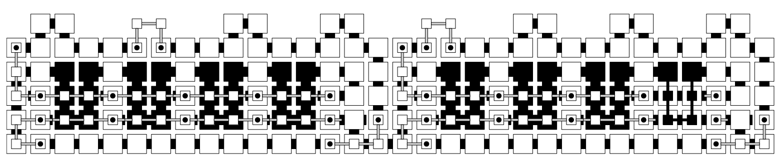
The purpose of the `Spacer` gadget is to provide spacing in between other gadgets. By construction, it always propagates information from left-to-right.

1.3 The `Write_counter_bit` gadgets

The gadgets shown in Figure 5 are used to write the bits of the counter.

(a) The tile types for the general `Write_counter_bit_0` gadget.

(b) The tile types for the general `Write_counter_bit_1` gadget.



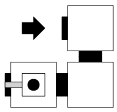
(c) The black tiles show the location of the `Write_counter_bit` gadgets in an example construction.

Figure 5: The `Write_counter_bit` gadgets.

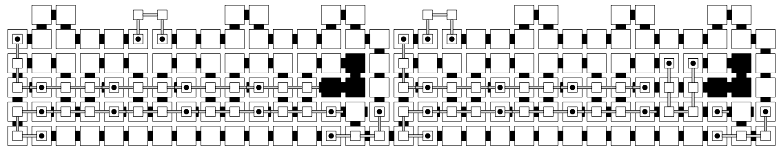
- For each $i = 0, \ldots, k-1$ and each $u \in \{0,1\}^i$, create `Write_counter_bit_0` $(\langle \texttt{write\_counter\_bits}, 0u \rangle, \langle \texttt{write\_counter\_bits\_space}, u \rangle)$ from the general gadget shown in Figure 5a.
- For each $i = 0, \ldots, k-1$ and each $u \in \{0,1\}^i$, create `Write_counter_bit_1` $(\langle \texttt{write\_counter\_bits}, 1u \rangle, \langle \texttt{write\_counter\_bits\_space}, u \rangle)$ from the general gadget shown in Figure 5b.
- For each $i = 1, \ldots, k-1$ and each $u \in \{0,1\}^i$, create `Spacer` $(\langle \texttt{write\_counter\_bits\_space}, u \rangle, \langle \texttt{write\_counter\_bits}, u \rangle)$ from the general gadget shown in Figure 4a.
- Create `Spacer` $(\langle \texttt{write\_counter\_bits\_space}, \lambda \rangle, \langle \texttt{read\_counter\_bits\_rl} \rangle)$ from the general gadget shown in Figure 4a.

1.4 The `Start_read_counter_rl` and `Read_counter_rl` gadgets

The `Start_read_counter_bits_rl` gadget shown in Figure 6a initiates the process of reading the bits of the counter from right-to-left.



(a) The tile types for the general `Start_read_counter_bits_rl` gadget. By construction, exactly one of its output glues will be blocked.

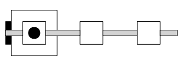


(b) The black tiles show the locations of the `Start_read_counter_bits_rl` gadget in an example construction.

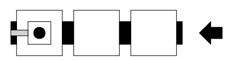
Figure 6: The `Start_read_counter_bits_rl` gadget.

Create
`Start_read_counter_bits_rl` $(\langle$`read_counter_bits_rl`$\rangle, \langle$`read_counter_bits_rl`$, 1\rangle, \langle$`read_counter_bits_rl`$, 0\rangle)$ from the general gadget shown in Figure 6a.

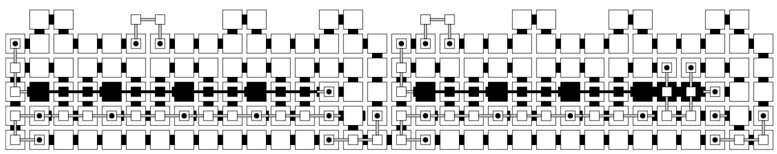
The gadgets shown in Figure 7 read the bits of the counter, starting at the least significant bit on the right and going to the most significant bit on the left.



(a) The tile types for the general `Read_counter_bit_rl_0` gadget.



(b) The tile types for the general `Read_counter_bit_rl_1` gadget.



(c) The black tiles show the locations of the `Read_counter_bit_rl` gadgets in an example construction.

Figure 7: The `Read_counter_bit_rl` gadgets.

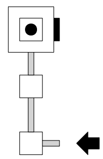
- For each $i = 0, \ldots, k-2$ and each $u \in \{0,1\}^i$, create `Read_counter_bit_rl_0` $(\langle$`read_counter_bits_rl`$, u0\rangle, \langle$`read_counter_bits_rl`$, u10\rangle, \langle$`read_counter_bits_rl`$, u00\rangle)$ from the general gadget shown in Figure 7a.
- For each $i = 0, \ldots, k-2$ and each $u \in \{0,1\}^i$, create `Read_counter_bit_rl_1` $(\langle$`read_counter_bit_rl`$, u1\rangle, \langle$`read_counter_bits_rl`$, u11\rangle, \langle$`read_counter_bits_rl`$, u01\rangle)$ from the general gadget shown in Figure 7b.

1.5 The `Start_extract_bits` and `Extract_bit` gadgets

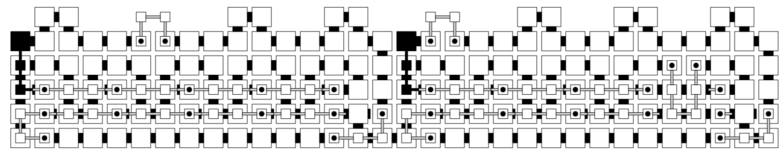
The `Start_extract_bits` gadget shown in Figure 8 starts the process of extracting the bits of a k-bit block.

For each $u \in \{0,1\}^k$, create `Start_extract_bits` $\left(\langle\right.$`read_counter_bits_rl`$, u\rangle, \langle$`extract_bits`$, w_{val(u)}\left.\rangle\right)$ from the general gadget shown in Figure 8, where $val(u)$ denotes the decimal value of u.

The gadgets shown in Figure **??** extract the bits of a k-bit block.

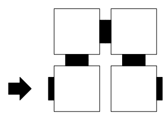


(a) The tile types for the general `Start_extract_bits` gadget.

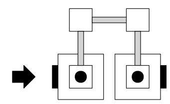


(b) The black tiles show the locations of the `Start_extract_bit` gadget in an example construction.

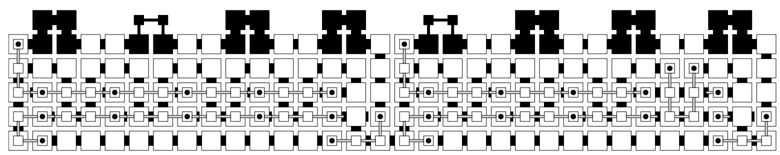
Figure 8: The `Start_extract_bits` gadget.



(a) The tile types for the general `Extract_bit_0` gadget.



(b) The tile types for the general `Extract_bit_1` gadget.



(c) The black tiles show the locations of the `Extract_bit` gadgets in an example construction.

Figure 9: The `Extract_bit` gadgets.

- For each $i = 0, \ldots, k-1$ and each $u \in \{0,1\}^i$, create $\texttt{Extract\_bit\_0}\left(\langle \texttt{extract\_bits}, 0u\rangle, \langle \texttt{extract\_space\_1}, u\rangle\right)$ from the general gadget shown in Figure 9a.
- For each $i = 0, \ldots, k-1$ and each $u \in \{0,1\}^i$, create $\texttt{Extract\_bit\_1}\left(\langle \texttt{extract\_bits}, 1u\rangle, \langle \texttt{extract\_space\_1}, u\rangle\right)$ from the general gadget shown in Figure 9b.
- For each $i = 1, \ldots, k-1$ and each $u \in \{0,1\}^i$, create $\texttt{Spacer}\left(\langle \texttt{extract\_space\_1}, u\rangle, \langle \texttt{extract\_space\_2}, u\rangle\right)$ from the general gadget shown in Figure 4a.
- For each $i = 1, \ldots, k-1$ and each $u \in \{0,1\}^i$, create $\texttt{Spacer}\left(\langle \texttt{extract\_space\_2}, u\rangle, \langle \texttt{extract\_bits}, u\rangle\right)$ from the general gadget shown in Figure 4a.

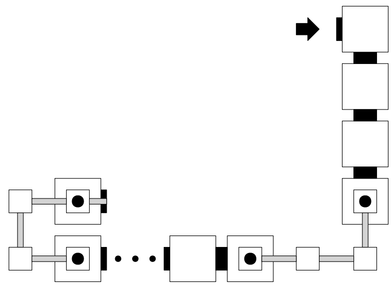
1.6 The `Start_read_counter_bits_lr` and `Read_counter_bit_lr` gadgets

The `Start_read_counter_bits_lr` gadget starts the process of reading the bits of the counter, starting at the most significant bit on the left and going to the least significant bit on the right.

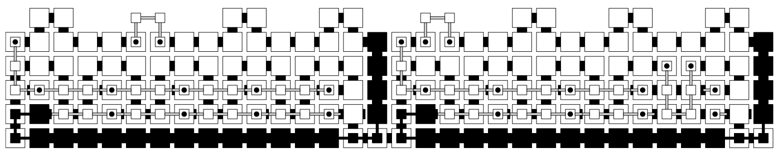
Create $\texttt{Start\_read\_counter\_lr}\left(\langle \texttt{extract\_space\_1}, \lambda\rangle, \langle \texttt{read\_counter\_bits\_lr}, 1\rangle, \langle \texttt{read\_counter\_bits\_lr}, 0\rangle\right)$ from the general gadget shown in Figure 10a.

The gadgets in Figure 11 read the bits of the counter, starting at the most significant bit on the left and going to the least significant bit on the right.

- For each $i = 0, \ldots, k-2$ and each $u \in \{0,1\}^i$, create $\texttt{Read\_counter\_bit\_lr\_0}\left(\langle \texttt{read\_counter\_bits\_lr}, 0u\rangle, \langle \texttt{read\_counter\_bits\_lr}, 10u\rangle, \langle \texttt{read\_counter\_bits\_lr}, 00u\rangle\right)$ from the general gadget shown in Figure 11a.

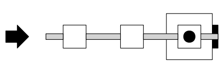


(a) The tile types for the general `Start_read_counter_bits_lr` gadget. The width of the gadget is hard-coded, depending on k.

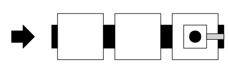


(b) The black tiles show the locations of the `Start_read_counter_bits_lr` gadget in an example construction.

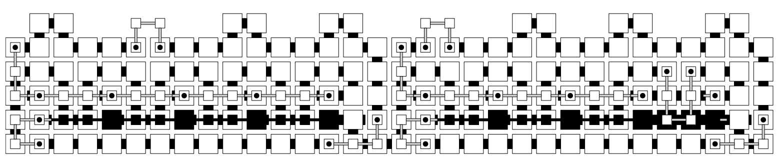
Figure 10: The `Start_read_counter_bits_lr` gadget.



(a) The tile types for the general `Read_counter_bit_lr_0` gadget.



(b) The tile types for the general `Read_counter_bit_lr_1` gadget.



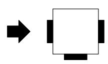
(c) The black tiles show the locations of the `Read_counter_lr` gadgets in an example construction.

Figure 11: The `Read_counter_bit_lr` gadgets.

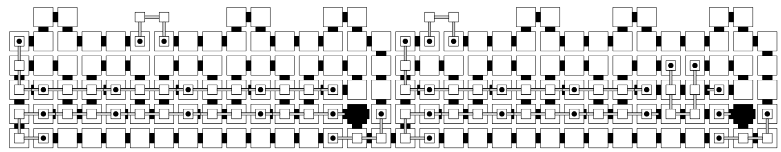
- For each $i = 0, \ldots, k-2$ and each $u \in \{0,1\}^i$, create
  `Read_counter_bit_lr_1` $(\langle \texttt{read\_counter\_bits\_lr}, 1u \rangle, \langle \texttt{read\_counter\_bits\_lr}, 11u \rangle, \langle \texttt{read\_counter\_bits\_lr}, 01u \rangle)$ from the general gadget shown in Figure 11b.

1.7 The `Find_opening`, `Start_next_block` and `Last_block` gadgets

The `Find_opening` gadget shown in Figure 12 finds the "opening" in the $z = 0$ plane from the `Start_read_counter_bits_lr` gadget.



(a) The general `Find_opening` gadget.

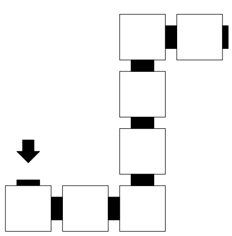


(b) The black tiles show the locations of the `Find_opening` gadget in an example construction. In general, the sequence of `Find_opening` gadgets can be arbitrarily long, depending on k.

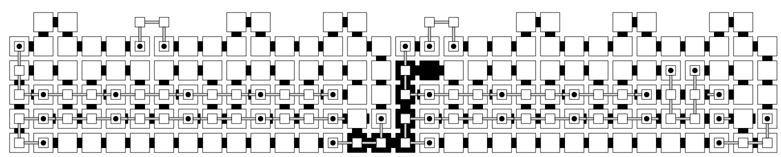
Figure 12: The `Find_opening` gadget.

- For each $u \in \{0,1\}^k$, create `Find_opening` $(\langle \texttt{read\_counter\_bits\_lr}, u \rangle, \langle \texttt{find\_opening}, u \rangle, \langle \texttt{find\_opening}, u \rangle)$ from the general gadget shown in Figure 12a.
- For each $u \in \{0,1\}^k$, create `Find_opening` $(\langle \texttt{find\_opening}, u \rangle, \langle \texttt{find\_opening}, u \rangle, \langle \texttt{find\_opening}, u \rangle)$ from the general gadget shown in Figure 12a.

The `Start_next_block` gadget shown in Figure 13 increments the value of the counter and propagates it to the next block.



(a) The tile types for the general `Start_next_block` gadget.



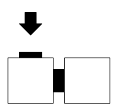
(b) The black tiles show the location of the `Start_next_block` gadget in an example construction.

Figure 13: The `Start_next_block` gadget.

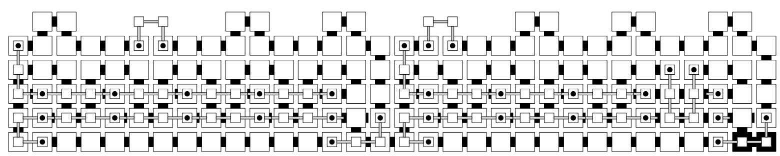
- For each $u \in \{0,1\}^k$, if $val(u) \neq m-1$, then create
  `Start_next_block` $(\langle \texttt{find\_opening}, u \rangle, \langle \texttt{write\_counter\_bits}, inc(u) \rangle)$ from the general gadget shown in Figure 13a. Here, we use $inc(u)$ to denote the k-bit binary representation of $u+1$.
- For each $u \in \{0,1\}^k$, if $val(u) = m-1$, then create
  `Start_next_block` $(\langle \texttt{find\_opening}, u \rangle, \langle \texttt{last\_block} \rangle)$ from the general gadget shown in Figure 13a.

The `Last_block` gadget shown in Figure 14 terminates the construction.
Finally, create `Last_block` $(\langle \texttt{last\_block} \rangle)$ from the general gadget shown in Figure 14a.

1.8 Correctness



(a) The tile types for the general `Last_block` gadget.



(b) The black tiles show the location of the `Last_block` gadget in an example construction.

Figure 14: The `Last_block` gadget.