Digital Mechanical Metamaterials

Alexandra Ion, Ludwig Wall, Robert Kovacs, and Patrick Baudisch
Hasso Plattner Institute, Potsdam, Germany
{firstname.lastname}@hpi.de

ABSTRACT
In this paper, we explore how to embody *mechanical* computation into 3D printed objects, i.e., without electronic sensors, actuators, or controllers typically used for this purpose. A key benefit of our approach is that the resulting objects can be 3D printed in one piece and thus do not require assembly. We are building on 3D printed cell structures, also known as *metamaterials*. We introduce a new type of cell that propagates a digital mechanical signal using an embedded bistable spring. When triggered, the embedded spring discharges and the resulting impulse triggers one or more neighboring cells, resulting in signal propagation. We extend this basic mechanism to implement simple logic functions. We demonstrate interactive objects based on this concept, such as a combination lock. We present a custom editor that allows users to model 3D objects, route signals, simulate signal flow, and synthesize cell patterns.

Author Keywords
Metamaterials; Fabrication; Programmable Matter

ACM Classification Keywords
H.5.m. [Information interfaces and presentation] Misc.

INTRODUCTION

Personal fabrication machines, such as 3D printers, allow users to make custom objects. While early work on 3D printing revolved around designing the outside of such objects [24, 32], recently researchers started exploring 3D printing as a means to design the *inside* of objects. Applications include moving objects' centers of gravity so as to make them stand [20] or spin [1].

Pushing this further, researchers created objects that consist internally of a large number of 3D cells arranged on a regular grid [15]. Since each cell is designed to perform a specific deformation, objects that entirely consist of such cells literally offer thousands of degrees of freedom. Such structures are also known as *metamaterials* [18].

Authors' copy.

Published in Proceedings of *CHI 2017,*
May 06 – 11, 2017, Denver, CO, USA

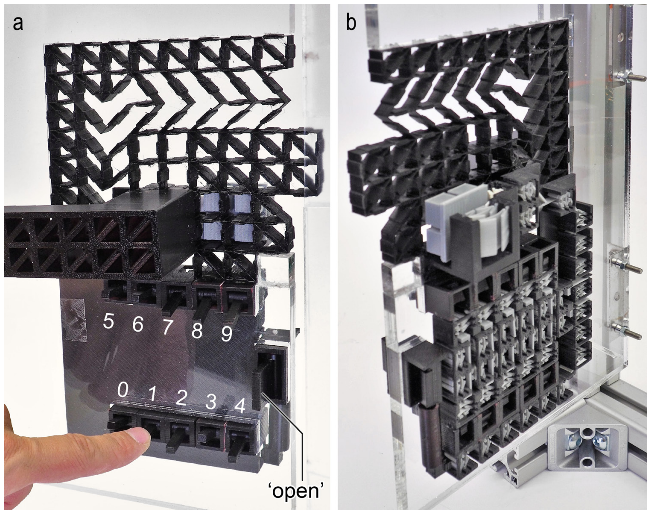


Figure 1: (a) This combination door lock is implemented as a *digital mechanical metamaterial*, i.e., a single block of material based on a regular grid of cells. It allows users to input a numeric code, it processes the code, checks its correctness, and unlocks the latch. (b) Under the hood, the lock consists of an array of cells that transmit and process a mechanical signal.

While metamaterials were initially understood as materials, we recently proposed to think of them as *machines;* such metamaterial mechanisms [10] consist of a single block of material, the cells of which play together in a well-defined way in order to achieve macroscopic movement. We used this principle previously to implement simple mechanical objects, such as a door latch (Figure 7).

Such *analog* machines, however, are limited in terms of complexity. As forces are passed on from one cell to the next, they are damped and the activation energy dissipates, causing the mechanical "signal" to decay exponentially. This limits the number of mechanisms that can be concatenated and therefore the complexity of the machine.

In this paper, we explore how to extend this concept towards *digital* mechanisms. Combining metamaterial mechanisms [10] with concepts from mechanical computing and mechanical signal propagation [16, 23], we introduce a new type of cell that propagates a digital mechanical signal, i.e., it counteracts signal decay and thus allows signals to pass through an *arbitrary* number of cells. We extend this basic mechanism to implement simple logic functions.

To illustrate this concept, Figure 1 shows a combination lock implemented using digital metamaterials. The device offers ten digit buttons on the front. Users tap these buttons to enter their code, then press the 'open' button to unlock the door.

BASICS OF DIGITAL MECHANICAL METAMATERIALS

Digital metamaterials are based on a new type of cell that propagates a mechanical signal reinforced by an embedded bistable spring.

The *bit cell* is the main underlying mechanism

Figure 2 shows the key element behind digital metamaterials, which we call *bit cell*. Bit cells contain a bistable spring, which allows them to take on two discrete states. Figure 2a shows the bit cell in its *tense* state. (b) When triggered, the spring discharges, causing the cell to switch from its *tense* state to its *relaxed* state.

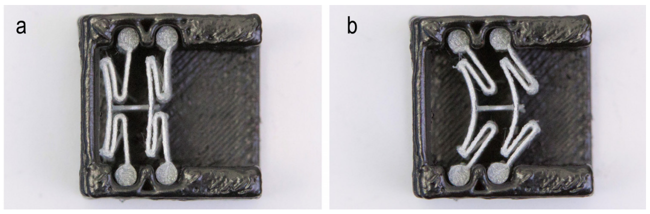


Figure 2: (a) When triggered, this *bit cell* changes its state from tense to (b) relaxed.

As shown in Figure 3, bit cells feature an input port and output port. A mechanical impulse that reaches the input port triggers the cell, which creates an impulse at the output port. Because discharging the spring releases mechanical energy, the impulse at the output port is larger than the required trigger impulse at the input port.

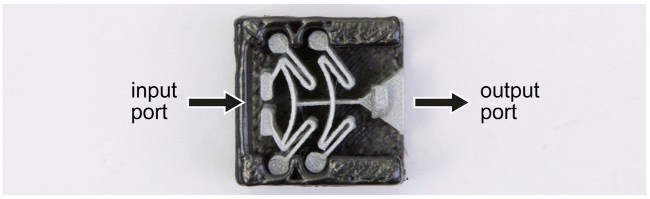


Figure 3: Bit cells offer an input and an output port.

As illustrated by Figure 4, this allows us to concatenate bit cells in a way that allows cells to trigger their immediate neighbors, resulting in a simple signal propagation mechanism similar to [23].

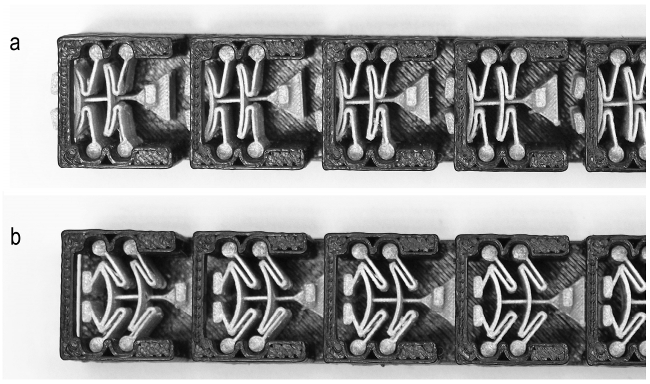


Figure 4: Concatenating bit cells creates a signal transmission. (a) Initially all bit cells are in their tense position. (b) Triggering the leftmost cell causes the signal to propagate through all cells from left to right.

The combination lock example

Bit cells and the resulting concept of signal propagation allow us to implement a hierarchy of digital mechanisms of increasing complexity. We discuss these logic functions and mechanisms in full detail later in this paper, as well as a simple manual recharge mechanism to set discharged springs back into their tense state. However, Figure 5 provides a rough overview of the different elements that implement the combination lock.

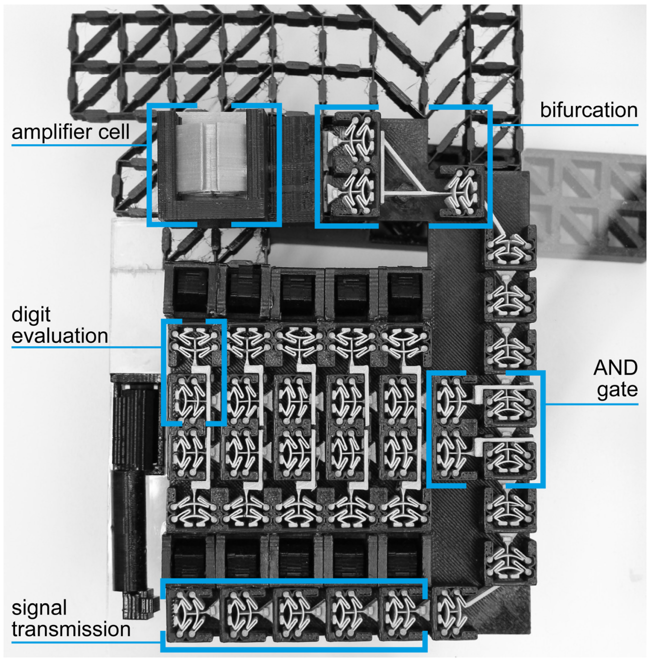


Figure 5: Our door lock consists of 82 cells, which implement the signal transmisison, the evaluation of each digit input by the user, an AND gate, and one amplifier cell with a pre-amplification step to move the blocking bolts sufficiently.

(1) To input the code, users tap one of the digit buttons on the front, which changes the state of the *digit evaluation* cells. The device contains 10 of these—one for each possible digit. (2) When the user pushes the 'open' button, three signal transmission lines are set off simultaneously; two of which run through the digit evaluation units and (3) set the state of the *AND gate*. The AND gate evaluates the correctness of the code by computing a logical AND the two rows of digits input by the user. The third signal transmission line runs from the bottom left towards the right, around the corner, and upwards where (4) the signal is *bifurcated*. This allows triggering (5) a double-sized *amplifier cell* that actuates the bolts to unblock the door.

Figure 6 shows a close-up of these bolts. (a) As long as the bolts are in place, they prevent the shearing cells in the middle from shearing, thereby blocking the door. (b) When the bolts are retracted, the shearing cells can shear and pushing down the handle retracts the latch—as discussed in [10]. This is where our digital metamaterials connect to the analog metamaterial door latch mechanism.

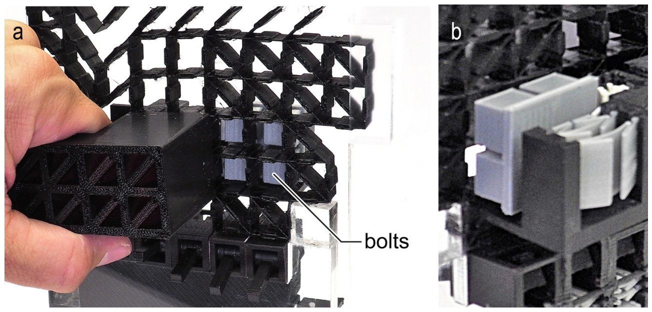


Figure 6: We effectively lock the latch mechanism by stiffening the shearing area that enables it. We do so by inserting bolts. Once users entered the key code correctly, our lock signal retracts the bolt and enables the latch mechanism.

CONTRIBUTION, BENEFITS & LIMITATIONS

Our main contribution is the concept of digital mechanical metamaterials. They allow integrating computational abilities into the structure of 3D printed objects. We provide a modular system consisting of digital cells (hardware) and an editor (software) that provides a toolkit to users, enabling them to create new digital mechanisms.

While analog metamaterial mechanisms are subject to damping, which causes the mechanical “signal” to decay exponentially and limits the number of ‘steps’ that can be performed, *digital* mechanical metamaterials enable transmitting signals through an *arbitrary* number of cells.

When we contrast digital mechanical metamaterials to the traditional approach of augmenting objects with electronic microcontrollers, sensors, and actuators [26, 31], our approach results in an entirely mechanical solution and can be produced entirely using a 3D printer. However, since our approach lacks loops, clocks, and memory, our approach is limited to much simpler devices.

RELATED WORK

We build on previous work in interactive personal fabrication, mechanical metamaterials, and analog computers.

Personal fabrication

3D printers and other personal fabrication machines simplify the process for users to make custom objects. Besides printing decorative object, users often create functional objects the functionality of which is determined by their external shape [32, 14, 7]. Integrating heating elements during the print process allows to change the shape after the fabrication process to adapt, e.g. to users’ bodies [9].

To fabricate mechanical assemblies, users can print the structural parts from rigid plastic (e.g., links, axles, bearings, gears, etc.) and assemble them to construct machines [2]. Such assemblies can also be printed in one process [6]. The usability of existing mechanical objects can be enhanced by fabricating additional handles or contraption using personal fabrication machines [7].

To allow users to go beyond mechanical systems made on their personal fabrication machines, researchers proposed techniques to integrate sensors and microcontrollers into objects. They range from (capacitive) position sensing [11, 29, 26] to sensing light beams for detecting user interaction [33], to complex systems like integrating a camera to track markers [25]. Adding sensors to 3D printed parts after they were fabricated even enables users to make adaptations to their everyday objects and to effectively customize the way they, for example, interact with their toaster [22]. Peng et al. recently demonstrated the idea of 3D printing motors, which allows fabricating arbitrary objects with integrated motors in the same process, enabling complex interactive objects with actuation capabilities [19].

While complex tasks, such as image processing are not possible in purely mechanical systems, we argue that we can integrate mechanical *and* simple information processing capabilities within a 3D printed object to alter its properties. To do so, we design the internal structure of 3D printed objects.

Designing the internal structure

Researchers in HCI and computer graphics started to alter the internal structure of 3D printed objects to, e.g., optimize their strength-to-weight ratio [12], to move their center of gravity in order to balance objects [20], or to allow arbitrary shapes to spin [3]. Vidimce et al. recently proposed a system that allows users to create and edit the internal structure and material distribution of 3D printed objects [30].

Mechanical metamaterials

Metamaterials are “artificial structures with mechanical properties that are defined by their usually repetitive cell patterns, rather than the material they are made of [18]. Metamaterials consist of large numbers of 3D cells organized on a regular grid. Since each cell can be designed to deform in a specific way [17, 27], they literally offer thousands of degrees of freedom.

Based on this concept, researchers have created objects with unusual behavior, such as objects that collapse abruptly when compressed [15], that shrink in two dimensions upon one-dimensional compression [8], or objects that combine layers of different degrees of stiffness (i.e., soft and hard cells) in order to emulate different materials, such as leather or felt [4].

However, metamaterials are usually seen as materials. In [10], we proposed thinking of metamaterials as *machines* instead. Figure 7 shows one of the objects we demonstrated—a door latch implemented as a single part.

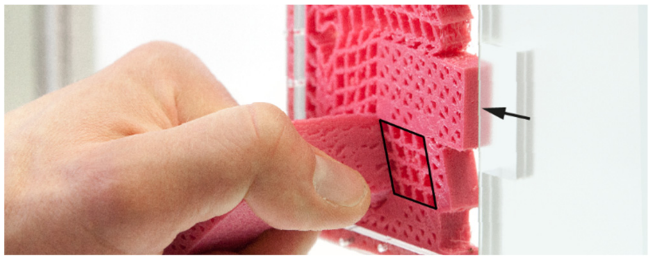


Figure 7: Metamaterial mechanisms [10].

In this paper, we extend on the concept of metamaterial mechanisms by introducing the notion digital signal processing, which allows us to produce more complex mechanisms, such as the combination lock shown in Figure 1. Our hardware design is inspired by mechanical wave propagation using bistable springs [16, 23].

Mechanical logic systems

Our work combines metamaterials with concepts from mechanical computing. Since we use bistable springs for storing energy, we build our logic based on three-state logic [28], which assumes the additional output state 'high impedance'. This allows us to distinguish a cell being in its tense state from when it is in its relaxed state.

One approach to implement mechanical logic systems is dual rail logic [5]. It duplicates the signal path to provide a distinct 0-signal. However, this comes at the cost of space. Rod logic presents an interesting system that can scale to nanotechnology [13]. It is based on rods that can let signals pass, or block them. We build our signal transmission on this work.

ROUTING SIGNALS BASED ON CELLS

In this and the following section, we now show the individual cells that implement the combination door lock we showed in Figure 5. We begin with the cell types that allow us to route signals through 3D objects. We already looked at signal propagation along a straight line (Figure 4); in this section, we demonstrate how to route signals around corners, across other signal lines, and how to bifurcate signals.

Routing signals is important because 3D printed objects can have arbitrary shape and routing allows transmitting a signal from where it emerges to where the information is needed. For the door lock, for example, we route users input from the digit inputs to the door latch mechanism—which is located elsewhere in our object.

The more specialized routing cells are all based on *bit cells*. However, we position their output ports to be oriented towards the neighbor cell we want to trigger. So while the bit cells in Figure 4 feature an output port on the side opposite to the input port, the cell shown in Figure 8 redirects the signal by 90° by adding a beam to the arm of our bistable spring. This beam rotates with the arm of the spring, allowing it to tap the input port of the rotated cell on the top right.

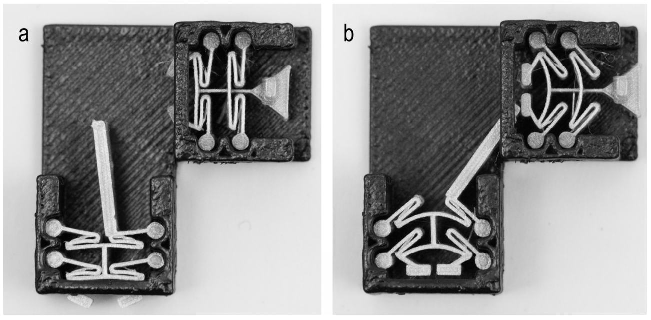


Figure 8: We use a new type of output port to redirect the signal by 90°. We exploit the rotational movement of the spring and attach a beam that taps its neighboring cell.

As illustrated by Figure 9, we can route signals in 3 dimensions by concatenating multiple such mechanisms. Here we route the signal from the x/y plane to the x/z plane to the y/z plane.

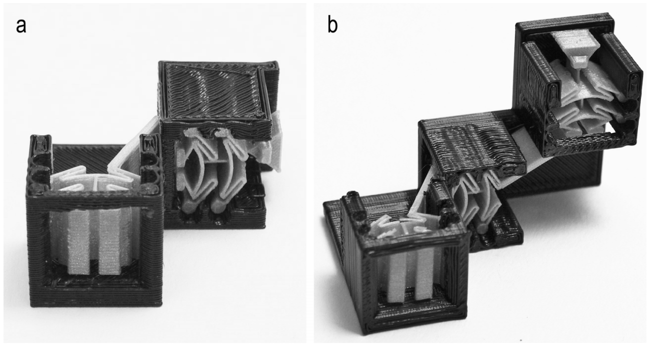


Figure 9: (a) Rotating the receiving cell allows us to redirect signals from one plane to another. (b) Concatenating three assemblies allows us to route signals in 3D.

Figure 10 shows a specialized three-cell mechanism that allows two signals to pass each other in minimal space. We use a crossbar that reaches from the output port of the left cell to the input port of the right cell that spans across the middle cell.

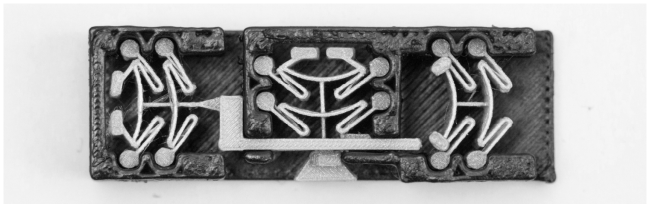


Figure 10: We cross signals by running a crossbar across another cell.

Figure 11 shows two mechanisms that bifurcate signals. The design shown in (a) triggers two parallel signal lines. The design shown in (b) triggers two signal lines oriented in opposite directions. Both designs exploit the fact that our bistable springs require less energy to be triggered than they output, which allows triggering two cells from one.

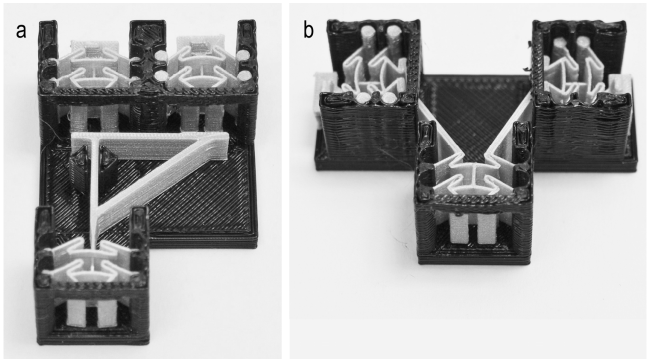


Figure 11: We can bifurcate signals (a) in a parallel manner or (b) let the two signals run in opposite directions.

Figure 12 shows how we merge two signals. This is an interesting construct, because it implements an OR gate.

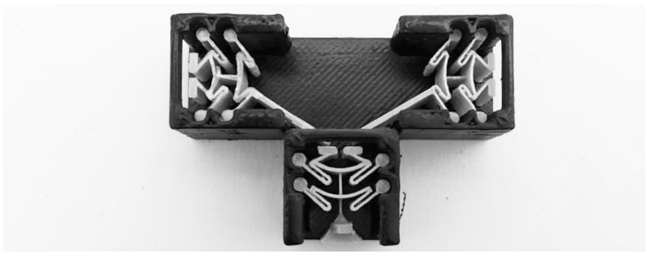


Figure 12: We use the opposite assembly to merge signal as we did to bifurcate them. This implements an OR gate.

LOGIC FUNCTIONS

To implement logic functions, we need to go beyond merely transmitting signals to also evaluating signals, which we achieve by blocking them. In the combination lock from Figure 5, we block signals for wrong digit inputs so that the door stays blocked. Later in this section, we present cell arrangements that implement basic logic, such as AND or NAND.

Blocking signals using gate cells

To allow for asynchronous input, we have designed cells capable of storing the first input that reaches them and do not act until the last signal has been received. We call these cells *gate cells*. Our approach is based on rod logic [13].

As illustrated by Figure 13a these cells work by placing a "blocker" across their neighboring cell. When the cell on the right is triggered before the cell on the left, the blocker is aligned with the output port of the right cell so that it cannot pass and signal is blocked. However, triggering the left cell, as shown in Figure 13b, moves the blocker out of the way and the signal can pass through. The position of the blocker can also be defined to initially let signals through and only after actuation to block signals, as shown in Figure 13c–d.

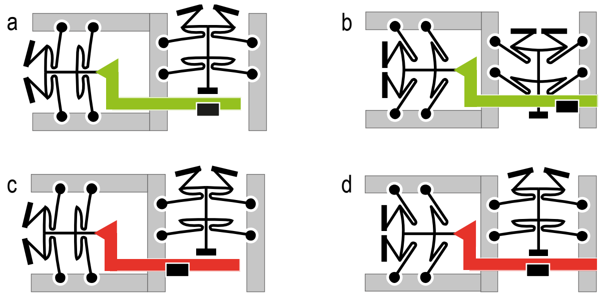


Figure 13: *Gate cells* validate signals and can be configures to block signals (a-b) or let signals pass (c-d) in their tense state.

Figure 14 shows the design of the two cells that form the gate cell. Each crossbar has a blocker attached on its underside.

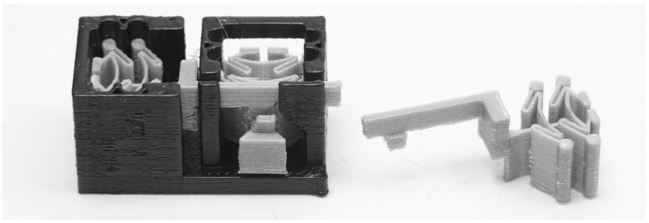


Figure 14: We position a blocking element that is intended to either block the signal output or let it pass.

Logic functions based on gate cells

We can concatenate multiple gate cells to create combinational logic functions. Figure 15 uses simplified symbols to illustrate how the positions of the blockers are configured to implement the function $A \wedge \neg B \wedge C \wedge D \wedge \neg E$. The positive input cells A, C, and D need to be triggered to move the blocker out of the way and let the signal pass. The negated inputs B and E are implemented by positioning the blocker so that they let the signal pass when they are *not* triggered and block the signal otherwise.

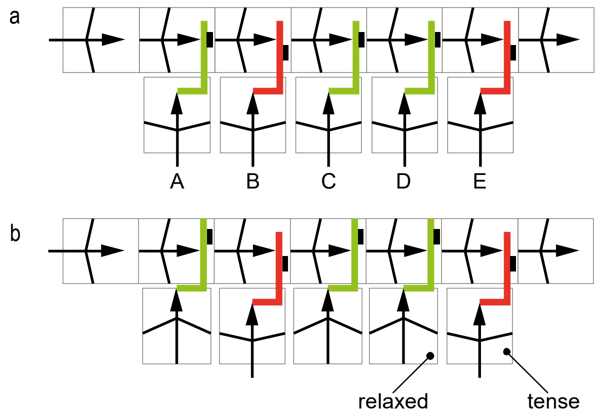


Figure 15: (a) When all inputs are tense, the signal cannot pass. (b) Triggering the correct inputs, here A, C, and D, moves the blockers so that the signal can pass.

If we rename the inputs of the logic function that is shown in Figure 15 from A–E to 0–4, it implements a 5-digit code evaluation. To implement the combination lock with 10 digits, we add a second row of inputs. Now we have two logic functions (one in each row), which both need to be correct, thus we add an AND gate. Figure 16 illustrates that the key code is '0 2 3 8'.

Again, we use our *gate cells* to employ the AND gate. Each code evaluation row has a gate cell at the end. Only if all the inputs of the corresponding row were correct, the blocker is moved out of the way for a third signal to pass, the *evaluation signal*.

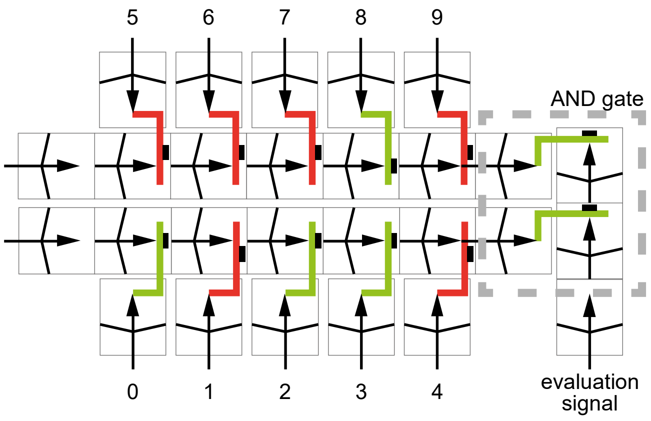


Figure 16: We add an AND gate to validate the two rows that yield the 10-digit.

Combinational logic using an evaluation signal

Our implementation of the AND gate has three inputs, namely two values and one additional *evaluation signal*. We add this additional signal because our mechanical computation is fundamentally different from electronic circuits, yet adding only one single signal allows us to implement any Boolean predicate without any electronics.

A 'signal' within our system is not an applied voltage, but an *impulse*, i.e., a mechanical force within the object. This impulse *changes the system state* by changing physical properties of the material, such as the position of the blockers. Since we block invalid signals, the output of gate cells is no signal instead of a 0-signal (logical low). However, not receiving a signal is indistinguishable from a dormant system. This means that we cannot provide an 0-signal that can serve as an input to the next gate, as in classical electronic circuits.

Despite this, we are still able to employ combinational logic within our materials. The most space efficient way is to integrate the inversion into logic functions, e.g., by using a NOR instead of an OR. Figure 17 illustrates a selection of logic gates implemented with our digital cells. Note that we show a different OR gate compared to the one shown in Figure 12. The one shown here uses the general-purpose assembly that is also used in the NOT and NAND gate.

We add the additional evaluation signal as a second computation step. After the inputs are provided to the system, we send a signal off that evaluates the inputs in order to produce output. Independent of the complexity of the logic, only one single evaluation signal is necessary, since it can be furcated, merged, and routed through the material. Race conditions within operations can be resolved by adapting the length of signal paths.

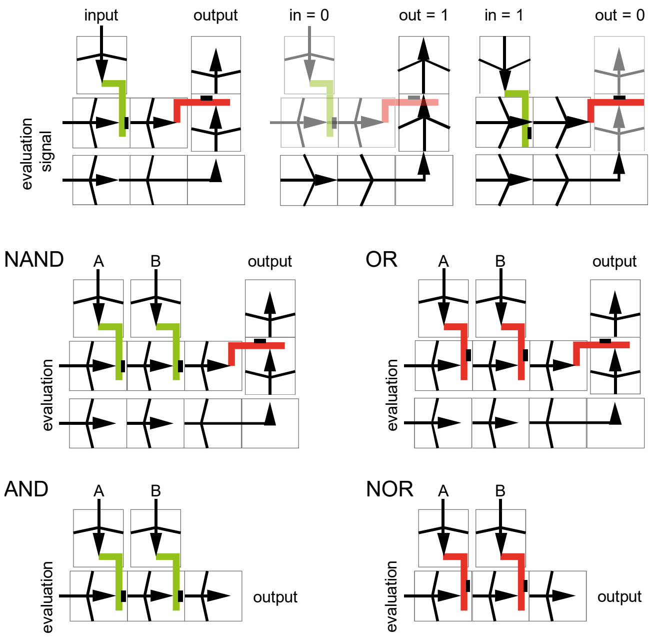


Figure 17: Only one additional signal allows us to implement combinational logic, despite not having a traditional 0-signal.

AMPLIFYING THE OUTPUT

While the cells that implement the signal transmission can be arbitrarily small, the *output* cells that move material to change the material properties might need to produce a certain amount of movement or force.

In the example of the door lock, we need to move the bolts sufficiently far into the door latch's structure to stiffen it. We use what we call an *amplifier cell*, which is inspired by the metaphor of operational amplifiers in the electronics domain. Such an *amplifier cell*, as shown in Figure 18, is a cell that is doubled in size. This allows us to add a larger spring to produce more stroke length.

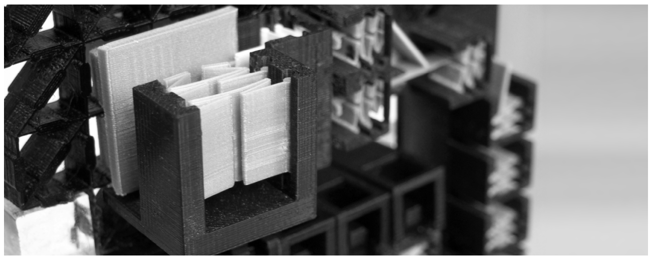


Figure 18: We amplify the stroke length of our output by going from small cells to a double-sized cell.

To transition from small cells to bigger cells, we bifurcate the signal. This gives us the energy of two cells, which together trigger the spring within the *amplifier cell*. In our door lock example, our 30 mm *amplifier cell* moves the bolts by 6 mm as compared to the stroke length of 3 mm of the 15 mm bit cells.

RECHARGING

After the springs were triggered and they are in their relaxed state, they need to be reset to their tense state before the computation can be run again. To do so, we designed a small lid on top of each cell, which uses the cell's third dimension to recharge the spring. Figure 19 shows how as the lid is pushed down, the attached wedges move the spring backward to its tense position. We use an additional plate to push multiple recharge lids at the same time. This design enables one recharge action for every plane of computation. We added a small bump on the underside of the lid, which causes the lid to spring back upward in order to not hinder the signal transmission between the cells.

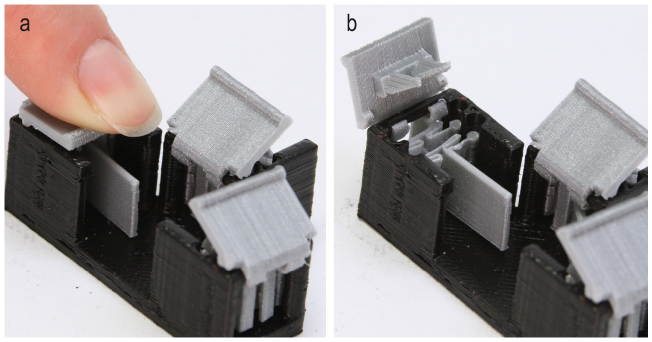


Figure 19: (a) Each cell features a lid with wedges, pushing it (b) recharges the spring underneath.

ADDITIONAL APPLICATION EXAMPLE

We see digital mechanical metamaterials being particularly useful for objects that have (1) many mechanical inputs (e.g., the code lock), and/or (2) many mechanical outputs (e.g., the following example of a plant pot), and (3) which are not frequently reconfigured. For example, the density plant pot might be reconfigured when seasons change, or the door might be locked once a day. In contrast, for objects that require frequent updates (e.g., displays) or more complex programming involving loops, etc., we recommend traditional electronics.

Example: plant pot

Figure 20 shows an example of a plant pot, where (a) users input the plant's size (small–large) and its water demands (little–much) using sliders. This triggers the computation in the bottom layer of the pot, which determines (b) how many *density cells* will be closed. After users configured the pot's density, (c) they place it into a cachepot with water. The density of the plant pot now determines how fast water can pass through to the plant.

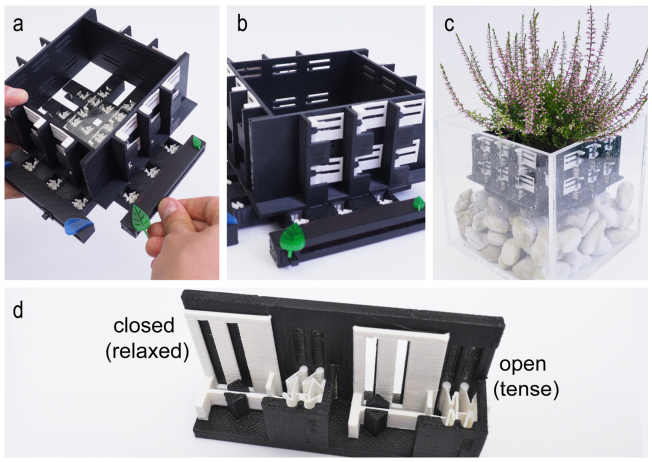


Figure 20: We implement a plant pot that changes its density based on user input of the plant's size and water demands.

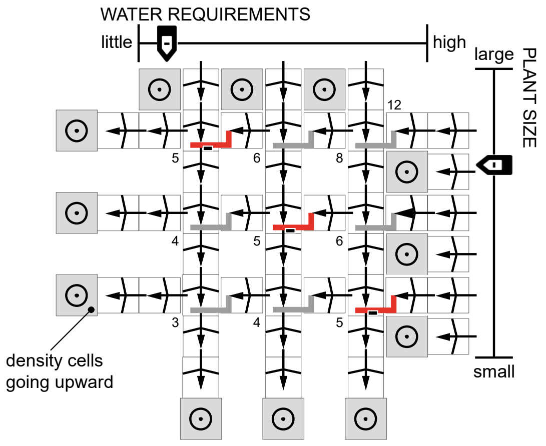


Figure 21: The weighted computation of the plant pot's density ensures that small plants get enough water by preventing some density cells from closing. The numbers indicate how many density cells are open for each parameter combination.

We use *gate cells* to change the weight of the parameters of the plant pot example. Figure 21 shows that by simply placing gate cells along the diagonal, we give more weight to the low values of the parameters. Since the gate cells prevent the signal from passing through, they prevent all density cells from closing, so that even for a small plant with little water demand the plant pot's permeability is 25% (3 out of 12 density cell rows remain open).

FABRICATION OF CELLS

We print our prototypes from the commonly available filaments ABS and PLA. While our cells are designed to be printed in an assembled state, we tend to print the parts of our prototypes separately. This allows us to print all elements without support material, which tends to be faster than the single-part design that requires dissolving the support material. We printed the springs from PLA using the *Ultimaker 2+* 3D printer, and the frames that hold the springs from ABS on our *Dimension SST 1200es*. Our cell size is 15 mm for all our prototypes with a printed spring thickness of 0.4 mm.

The cell shown in Figure 22 is printed in an assembled state. We had it made at *shapeways* using their "frosted detail plastic" material, which is a UV cured acrylic polymer that is printed using the MultiJet Modeling process.

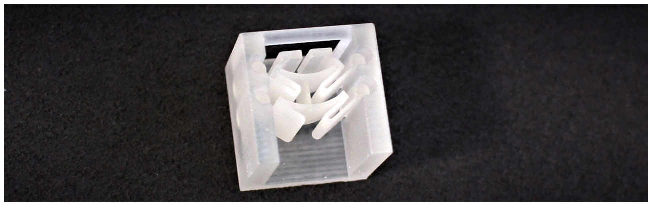


Figure 22: A cell printed fully assembled using shapeways' "frosted detail plastic" material.

We empirically tested how the cells miniaturize while retaining the same stress values using Autodesk Fusion's simulation. The results showed that reducing the spring thickness to ½ allows it to be shortened to ¼ of its length, i.e., to $^{1}/_{64}$ of the cell volume. For example, a 0.2 mm thick spring allows for a cell size of 3.75 mm, which is a matter of printer resolution.

BISTABLE SPRING DESIGN

The bistable spring in our cells differs from a typical bistable spring that is shown in Figure 23b, which is a simple prebent beam that is fixed within rigid walls [21]. However, such designs have very high width-to-length ratios, which do not utilize the space within a rotation-invariant cubic cell well.

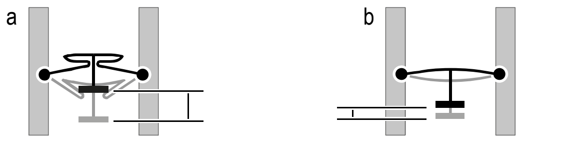


Figure 23: (a) The spring we use in our bit cells is longer and thus weaker than (b) conventional bistable springs. Or, for the same force, our cells produce more stroke length.

Figure 23 illustrates that our spring design includes an additional 'loop', which prolongs the beam and therefore makes it weaker, i.e., it requires less force to be triggered and charged. We measured 45% less force required to charge our type of spring compared to the conventional spring. Another way to view it is that our longer springs produce more output length (by 23% according our measurements) while requiring similar force.

Note that our cells incorporate two connected springs. This is a common technique [21] for increasing the stability of bistable springs during the so-called 'snap-through', i.e., the point where the spring is compressed the most as it is forced to its other second position.

TECHNICAL EVALUATION

The geometry of our spring allows us to make limited changes in stroke length and force by varying the spring parameters. For example, we used slightly stronger springs in the plant pot example to compensate for the higher density of water. While the output of bit cells usually needs to be only strong and far enough to trigger the neighbor cell, the output cells may have to meet specific requirements in terms of amount of force or stroke length.

Our evaluation informs the geometrical spring parameters for achieving bistability and the maximum possible fan-out of a cell, i.e., how many cells can be triggered by one single cell.

Independent variables: We compared a total of 75 springs of our design where we varied three parameters independently: (1) the arm angle, (2) the length of the bent bridge in the middle, and (3) the strength of the bridge, varied trough changing its buckling magnitude and its thickness concurrently. We varied the values for the bridge strength from 1.05 times the normal spring thickness to 1.85 times, and a buckle distance from 8% of the bridge length to 40%. The spring thickness is limited by the 3D printer's resolution; we use 0.18 mm. Bridge length values ranged from 45% to 85% the total distance between the walls. We tested these values for 20°, 30° and 40° arm angles. This yields 25 springs for each arm angle.

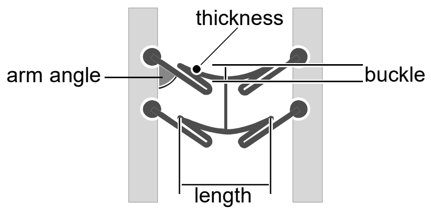


Figure 24: We vary the parameters of bridge length and bridge strength for three different arm angles each. We measure the stroke length and the forces for charging and triggering the springs, as well as their output energy.

Dependent variables: We measured (1) the force it takes to push a spring to its tense position, (2) the force necessary to trigger the spring, (3) its stroke length, and (4) the force it outputs when triggered.

Test setup: Figure 25 shows our test setup. We placed a ruler (error 0.5 mm) under the spring to measure the stroke length. We used a force gauge with an error of 0.05 N, which was constrained to linear movement centered to the spring and precisely moved by a threaded rod. We pushed the force gauge against the spring to measure the charge energy, we released the pressure while slowly moving the force gauge backward to measure the output energy, and we measured the trigger energy by pushing the rotated spring.

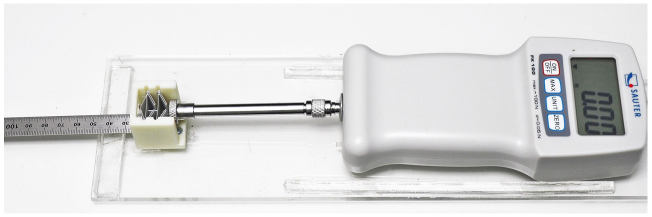
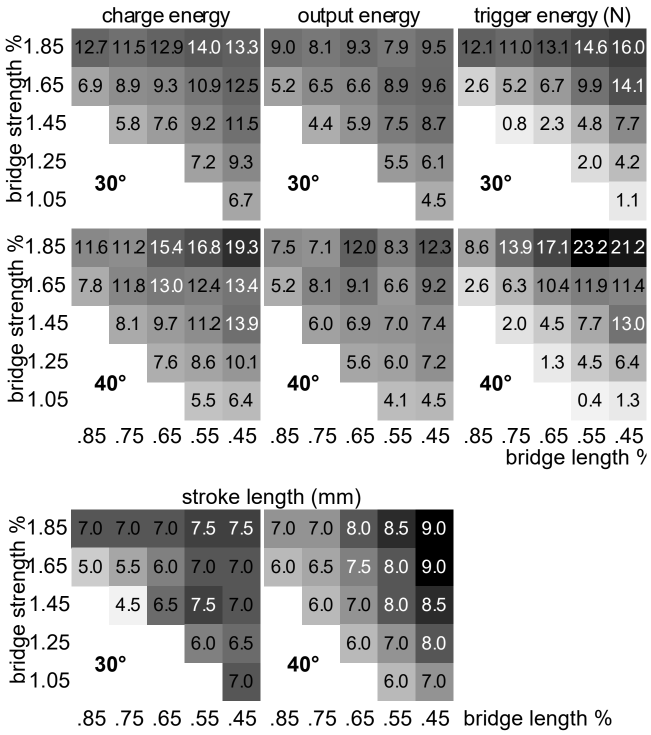


Figure 25: We measure the forces using a force gauge (error 0.05 N), and the stroke length using a ruler (error 0.5 mm).

Results

Figure 26 shows charge, output, and trigger energy and stroke lengths for 50 springs. Empty fields denote springs that were not bistable. The results for springs with a 20° arm angle were omitted since only 2 of them were bistable.



		charge energy					output energy					trigger energy (N)				
	bridge strength %	.85	.75	.65	.55	.45	.85	.75	.65	.55	.45	.85	.75	.65	.55	.45
30°	1.85	12.7	11.5	12.9	14.0	13.3	9.0	8.1	9.3	7.9	9.5	12.1	11.0	13.1	14.6	16.0
30°	1.65	6.9	8.9	9.3	10.9	12.5	5.2	6.5	6.6	8.9	9.6	2.6	5.2	6.7	9.9	14.1
30°	1.45		5.8	7.6	9.2	11.5		4.4	5.9	7.5	8.7		0.8	2.3	4.8	7.7
30°	1.25				7.2	9.3				5.5	6.1				2.0	4.2
30°	1.05					6.7					4.5					1.1
40°	1.85	11.6	11.2	15.4	16.8	19.3	7.5	7.1	12.0	8.3	12.3	8.6	13.9	17.1	23.2	21.2
40°	1.65	7.8	11.8	13.0	12.4	13.4	5.2	8.1	9.1	6.6	9.2	2.6	6.3	10.4	11.9	11.4
40°	1.45		8.1	9.7	11.2	13.9		6.0	6.9	7.0	7.4		2.0	4.5	7.7	13.0
40°	1.25			7.6	8.6	10.1			5.6	6.0	7.2			1.3	4.5	6.4
40°	1.05				5.5	6.4				4.1	4.5				0.4	1.3

bridge length %

	stroke length (mm)									
	30°					40°				
bridge strength %	.85	.75	.65	.55	.45	.85	.75	.65	.55	.45
1.85	7.0	7.0	7.0	7.5	7.5	7.0	7.0	8.0	8.5	9.0
1.65	5.0	5.5	6.0	7.0	7.0	6.0	6.5	7.5	8.0	9.0
1.45		4.5	6.5	7.5	7.0		6.0	7.0	8.0	8.5
1.25				6.0	6.5			6.0	7.0	8.0
1.05					7.0				6.0	7.0

bridge length %

Figure 26: Raw results of our technical evaluation for charge, output and trigger energy in N and stroke length in mm. Missing values indicate non-bistable springs.

All four measured values increase when increasing the arm angle or the bridge strength, or when decreasing the length of the bridge. The *output energy* was on average 73% of the charge energy for 30° springs and 66% for 40° springs.

The difference between output energy and *trigger energy* is greatest right when the springs start becoming bistable. The ratio between the two decides the maximum possible fan-out of the springs, thus a 2:1 ratio is necessary for bifurcation. Choosing a higher trigger energy however increases the fault-tolerance of the system with regards to unwanted activation, e.g., by dropping the object.

Stroke length is affected most by the arm angle, i.e., the stroke length increases with the arm angle. Stroke is least affected by the strength of the bridge.

In contrast, *charge energy* of the spring is affected most by the strength of the bridge and least by the arm angle, which can also be seen from Figure 26 in the rapid changes along the y-axis.

Choosing appropriate values for each can tune the spring toward a longer stroke or a higher output energy without changes to its bistability. Note that these values apply to the springs we tested with and that due to differences in manufacturing they might vary slightly.

EDITOR

To allow expert users to create and fabricate objects from digital metamaterials, we implemented a specialized 3D voxel-style editor, which is based on the editor for metamaterial mechanisms [10]. The main intent is to allow users to draw signal paths and verify them within the editor (Figure 27). We support users by allowing them to enter simple logic functions, which our editor converts to cell arrangements that implement that function.

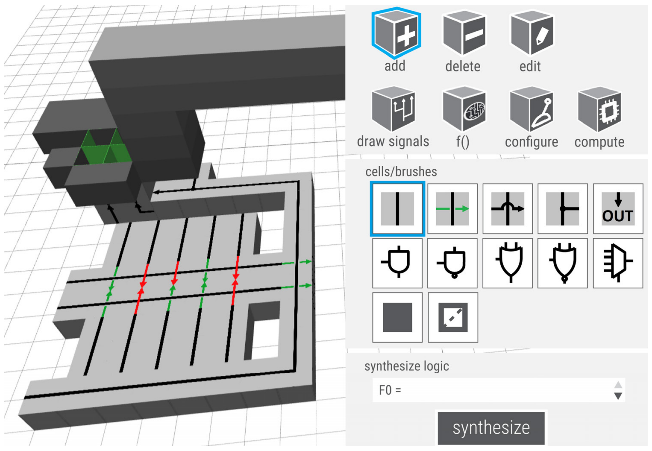


Figure 27: Our editor helps users create digital metamaterials.

While the editor is built to help users design digital metamaterials efficiently, knowledge about signals and logic remains necessary, i.e., this editor is for expert users.

Walkthrough

Figure 28 illustrates how users create the door lock example from Figure 1. (a) They first draw the signal line that evaluates the upper 5 digits by dragging over the ground plane using our "draw signals" tool. (b) Then, using the same tool, they draw signals perpendicular to the first signal line. (c) When the two signals cross, the editor automatically draws a gate cell. (d) They do the same for the lower row of digits. (e) In this example, users manually configure the gate cells using the "configure", i.e., they change the initial state of 5 gate cells from initially 'pass' to 'block' by clicking on the respective gate cell. (f) The configured gate cells implement the key code for the lock.

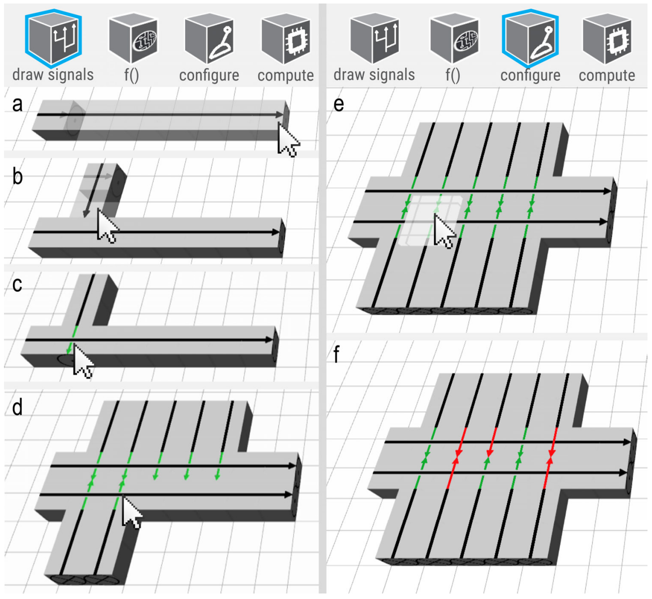


Figure 28: (a) Users draw the signal routing using the "draw signals" tool. (b) Once they cross an existing signal route, (c) the editor automatically draws a gate cell. (d) After creating all cells for the digit evaluation, (e) users set the initial states of the gate cells using the "configure" tool (f) to define the key code.

Users continue by adding the evaluation line, the AND gate and the output cells, which will move the bolts. Finally, they model the analog door latch mechanism on top of the digital metamaterial.

Figure 29 shows how users verify the signal transmission in our custom editor. They first charge the cells by selecting the "compute" tool. The editor visualizes charged cells by turning the signal lines blue. Clicking on a cell, as shown in Figure 29a, sets a signal off. The impulse runs through the cells, being visualized in yellow at the currently active cell. After the impulse has passed a cell, the signal path is shown in black again, because the cell is back in its relaxed state (Figure 29b). To verify the whole computational assembly, users trigger the inputs first and then the evaluation signal, as they do on the 3D printed object. They subsequently watch if the signal runs all the way through to the door. If not, they see where the signal stopped and can correct the error.

To help users create logic functions efficiently, e.g., by avoiding the need for manual configuration of cells, we allow users to input logic functions. Figure 30 shows an example, where users enter the function 'A & ~B & C & D & ~E' and click 'synthesize'. Then, they indicate where the synthesized cell arrangement shall be positioned by simply clicking on the grid. Our editor automatically synthesizes the cells that implement the entered logic functions.

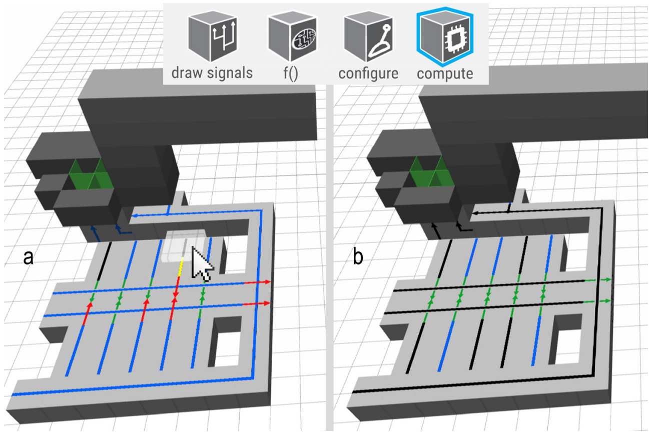


Figure 29: Users can verify their logic and signal routing. They first charge all springs, then (a) they click the inputs to trigger the signal there, and lastly (b) they trigger the evaluation line and find that the signal passes all the way through to the latch.

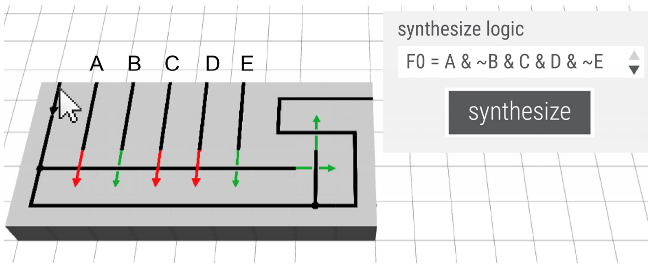


Figure 30: Users enter the function 'A & ~B & C & D & ~E', and indicates the location by clicking on the grid. Our editor responds by automatically inserting the corresponding cells.

Implementation

We build on the metamaterial mechanisms voxel-style editor [10] and extend it to allow users to draw signal routes and to input logic functions. Our extension of the editor is based on a node.js javascript framework, using the three.js graphics framework and WebGL for rendering the basic geometries.

Rendering of 3D-printable .stl files was done in modular hierarchical OpenSCAD script files. The editor simply exports the cell geometries as OpenSCAD script commands to render a cell with the specific parameters.

Drawing signal paths

Drawing with signal paths is realized through a freeform line tool that places appropriately connected cells following the cursor. We use a pathfinding library for 3D[1] that provides the A* pathfinding algorithm to simplify connecting cells via a signal line by finding the shortest available path while crossing existing signals where necessary.

Synthesizing cell arrangements

To generate cell arrangements of minimal size that implement a user-defined logic function, we use a version of the Espresso heuristic logic minimizer[2].

The logic minimizer parses the user text input, minimizes the described input function and returns it in its disjunctive normal form. This minimized DNF is parsed a second time by our editor to identify its terms and literals, which are used to create a very compact cell arrangement that forms a disjunction of minterm conjunctions. A minterm is a minimal conjunction of the input literals that returns true. This means that an array of signals representing the disjunctions of the function is run in parallel. All input variables of the function intersect and potentially block these lines, forming conjunctions along each of the parallel lines. The combined arrangement implements the function as a whole. To choose the most succinct cell representation of the function, we also minimize the negated input function and negate its result again directly on the cell level. The cell arrangement variant that requires the least cells to implement the input function is constructed and placed in the editor at the last user-selected cell location.

The logic minimizer runs in a separate python virtual environment using the PyEDA[3] library for electronic design automation. This python server is queried via HTTP requests to a REST architecture and replies with minimized functions to logic functions encoded in the request-URL.

CONCLUSIONS

We presented digital mechanical metamaterials. While (analog) metamaterial mechanisms suffer from signal decay, digital metamaterials are not subject to such decay, allowing us to create larger/more complex objects. Unlike solutions based on sensors, actuators, and microcontrollers, our approach still is entirely mechanical.

We showed the design of bit cells, which are the key element for the signal transmission. They contain bistable springs, which have two states, the relaxed and tense state. We also presented cells that allow for routing signals in 2D and 3D. To employ simple computation, we showed gate cells. We demonstrated our approach at the example of two prototypes: a digital door lock without electronics and a configurable plant pot. To help expert users create such digital metamaterials, we contribute an editor that allows for routing signals, verifying them, and synthesizing cell arrangements from user-defined logic functions.

For future work, we plan to explore cells that have the ability to shear *and* transmit signals, which would allow us to change the properties of every single cell within an object.

Acknowledgements

We want to thank David Lindlbauer, Pedro Lopes, Stefanie Müller, and Willi Müller for their help and feedback.

[1] https://github.com/schteppe/PathFinding3D.js

[2] https://embedded.eecs.berkeley.edu/pubs/downloads/espresso/index.htm

[3] http://pyeda.readthedocs.io/en/latest/index.html

REFERENCES

1. Jeffrey K. Anderson, Larry L. Howell, Jonathan W. Wittwer, and Timothy W. McLain. 2006. Piezoresistive sensing of bistable micro mechanism state. *Journal of Micromechanics and Microengineering* 16.5 (2006): 943. http://dx.doi.org/10.1088/0960-1317/16/5/010
2. Moritz Bächer, Stelian Coros, and Bernhard Thomaszewski. 2015. LinkEdit: interactive linkage editing using symbolic kinematics *ACM Transactions on Graphics* 34, 4, Article 99 (July 2015), 8 pages. http://dx.doi.org/10.1145/2766985
3. Moritz Bächer, Emily Whiting, Bernd Bickel, and Olga Sorkine-Hornung. 2014. Spin-it: optimizing moment of inertia for spinnable objects. *ACM Transactions on Graphics* 33, 4. http://dx.doi.org/10.1145/2601097.2601157
4. Bernd Bickel, Moritz Bächer, Miguel A. Otaduy, Hyunho Richard Lee, Hanspeter Pfister, Markus Gross, and Wojciech Matusik. 2010. Design and fabrication of materials with desired deformation behavior. *ACM Transactions on Graphics* 29, 4, Article 63 (July 2010), 10 pages. http://dx.doi.org/10.1145/1778765.1778800
5. Marco Bucci, Luca Giancane, Raimondo Luzzi, and Alessandro Trifiletti. 2006. Three-phase dual-rail precharge logic. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pp. 232-241. Springer Berlin Heidelberg, 2006.
6. Jacques Cali, Dan A. Calian, Cristina Amati, Rebecca Kleinberger, Anthony Steed, Jan Kautz, and Tim Weyrich. 2012. 3D-printing of non-assembly, articulated models. *ACM Transactions on Graphics* 31, 6, Article 130 (November 2012), 8 pages. http://dx.doi.org/10.1145/2366145.2366149
7. Xiang 'Anthony' Chen, Jeeeun Kim, Jennifer Mankoff, Tovi Grossman, Stelian Coros, and Scott E. Hudson. 2016. Reprise: A Design Tool for Specifying, Generating, and Customizing 3D Printable Adaptations on Everyday Objects. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology* (UIST '16). ACM, New York, NY, USA, 29-39. DOI: https://doi.org/10.1145/2984511.2984512
8. Juan Carlos Álvarez Elipe, and Andrés Díaz Lantada. 2012. Comparative study of auxetic geometries by means of computer-aided design and engineering. *Smart Materials and Structures* 21.10 (2012): 105004.
9. Daniel Groeger, Elena Chong Loo, and Jürgen Steimle. 2016. HotFlex: Post-print Customization of 3D Prints Using Embedded State Change. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '16). ACM, New York, NY, USA, 420-432. DOI: https://doi.org/10.1145/2858036.2858191
10. Alexandra Ion, Johannes Frohnhofen, Ludwig Wall, Robert Kovacs, Mirela Alistar, Jack Lindsay, Pedro Lopes, Hsiang-Ting Chen, and Patrick Baudisch. 2016. Metamaterial Mechanisms. In *Proceedings of the annual ACM symposium on User interface software and technology* (UIST '16). ACM, New York, NY, USA, 529-539. DOI: https://doi.org/10.1145/2984511.2984540
11. Yuichiro Katsumoto, Satoru Tokuhisa, and Masa Inakage. 2013. Ninja track: design of electronic toy variable in shape and flexibility. In *Proceedings of the 7th International Conference on Tangible, Embedded and Embodied Interaction* (TEI '13). ACM, New York, NY, USA, 17-24. http://dx.doi.org/10.1145/2460625.2460628
12. Lin Lu, Andrei Sharf, Haisen Zhao, Yuan Wei, Qingnan Fan, Xuelin Chen, Yann Savoye, Changhe Tu, Daniel Cohen-Or, and Baoquan Chen. 2014. Build-to-last: strength to weight 3D printed objects. *ACM Transactions on Graphics* 33, 4. http://dx.doi.org/10.1145/2601097.2601168
13. Merkle, Ralph C. 1993. Two types of mechanical reversible logic. *Nanotechnology* 4.2 (1993): 114.
14. Stefanie Mueller, Tobias Mohr, Kerstin Guenther, Johannes Frohnhofen, and Patrick Baudisch. 2014. faBrickation: fast 3D printing of functional objects by integrating construction kit building blocks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '14). ACM, New York, NY, USA, 3827-3834. http://dx.doi.org/10.1145/2556288.2557005
15. Tom Mullin, S. Deschanel, Katia Bertoldi, and Mary C. Boyce. 2007. Pattern transformation triggered by deformation. *Physical review letters* 99, no. 8 (2007): 084301. http://dx.doi.org/10.1103/PhysRevLett.99.084301
16. Nadkarni, Neel, Chiara Daraio, and Dennis M. Kochmann. 2014. Dynamics of periodic mechanical structures containing bistable elastic elements: From elastic to solitary wave propagation. *Physical Review E* 90.2 (2014): 023204.
17. Julian Panetta, Qingnan Zhou, Luigi Malomo, Nico Pietroni, Paolo Cignoni, and Denis Zorin. 2015. Elastic textures for additive fabrication. *ACM Transactions on Graphics* 34, 4. http://dx.doi.org/10.1145/2766937
18. Jayson Paulose, Anne S. Meeussen, and Vincenzo Vitelli. 2015. Selective buckling via states of self-stress in topological metamaterials. In *Proceedings of the National Academy of Sciences*, 112(25), 7639-7644.
19. Huaishu Peng, François Guimbretière, James McCann, and Scott Hudson. 2016. A 3D Printer for Interactive Electromagnetic Devices. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology* (UIST '16). ACM, New York, NY, USA, 553-562. DOI: https://doi.org/10.1145/2984511.2984523

20. Romain Prévost, Emily Whiting, Sylvain Lefebvre, and Olga Sorkine-Hornung. 2013. Make it stand: balancing shapes for 3D fabrication. *ACM Transactions on Graphics* 32, 4. http://dx.doi.org/10.1145/2461912.2461957

21. Jin Qiu, Jeffrey H. Lang, and Alexander H. Slocum. 2004. A curved-beam bistable mechanism. *Journal of microelectromechanical systems* 13.2 (2004): 137-146.

22. Raf Ramakers, Fraser Anderson, Tovi Grossman, and George Fitzmaurice. 2016. RetroFab: A Design Tool for Retrofitting Physical Interfaces using Actuators, Sensors and 3D Printing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '16). ACM, New York, NY, USA, 409-419. DOI: https://doi.org/10.1145/2858036.2858485

23. Jordan R. Raney, Neel Nadkarni, Chiara Daraio, Dennis M. Kochmann, Jennifer A. Lewis, and Katia Bertoldi. 2016. Stable propagation of mechanical signals in soft media using stored elastic energy. In *Proceedings of the National Academy of Sciences* (2016): 201604838.

24. Greg Saul, Manfred Lau, Jun Mitani, and Takeo Igarashi. 2010. SketchChair: an all-in-one chair design system for end users. In *Proceedings of the fifth international conference on Tangible, embedded, and embodied interaction* (TEI '11). ACM, New York, NY, USA, 73-80. DOI: http://dx.doi.org/10.1145/1935701.1935717

25. Valkyrie Savage, Colin Chang, and Björn Hartmann. 2013. Sauron: embedded single-camera sensing of printed physical user interfaces. In *Proceedings of the 26th annual ACM symposium on User interface software and technology* (UIST '13). ACM, New York, NY, USA, 447-456. http://dx.doi.org/10.1145/2501988.2501992

26. Valkyrie Savage, Ryan Schmidt, Tovi Grossman, George Fitzmaurice, and Björn Hartmann. 2014. A series of tubes: adding interactivity to 3D prints using internal pipes. In *Proceedings of the 27th annual ACM symposium on User interface software and technology* (UIST '14). ACM, New York, NY, USA, 3-12. http://dx.doi.org/10.1145/2642918.2647374

27. Christian Schumacher, Bernd Bickel, Jan Rys, Steve Marschner, Chiara Daraio, and Markus Gross. 2015. Microstructures to control elasticity in 3D printing. *ACM Transactions on Graphics* 34, 4. http://dx.doi.org/10.1145/2766926

28. Three-state logic. https://en.wikipedia.org/wiki/Three-state_logic#Tri-state_Buffer Retrieved on September 21st, 2016

29. Tatyana Vasilevitsky and Amit Zoran. 2016. Steel-Sense: Integrating Machine Elements with Sensors by Additive Manufacturing. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '16). ACM, New York, NY, USA, 5731-5742. http://dx.doi.org/10.1145/2858036.2858309

30. Kiril Vidimce, Alexandre Kaspar, Ye Wang, and Wojciech Matusik. 2016. Foundry: Hierarchical Material Design for Multi-Material Fabrication. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology* (UIST '16). ACM, New York, NY, USA, 563-574. DOI: https://doi.org/10.1145/2984511.2984516

31. Voxel8. http://www.voxel8.co/ Retrieved on September 21st, 2016

32. Christian Weichel, Manfred Lau, David Kim, Nicolas Villar, and Hans W. Gellersen. 2014. MixFab: a mixed-reality environment for personal fabrication. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '14). ACM, New York, NY, USA, 3855-3864. http://dx.doi.org/10.1145/2556288.2557090

33. Karl Willis, Eric Brockmeyer, Scott Hudson, and Ivan Poupyrev. 2012. Printed optics: 3D printing of embedded optical elements for interactive devices. In *Proceedings of the 25th annual ACM symposium on User interface software and technology* (UIST '12). ACM, New York, NY, USA, 589-598. http://dx.doi.org/10.1145/2380116.2380190