MD5 design and implementation

We designed and implemented a hashing algorithm based on the MD5 algorithm for a 2-bit computer. The full MD5 hashing algorithm is designed to be run efficiently on a 32-bit computer and takes as input a 512-bit message, outputting a 128-bit hash. To design a variant that can be feasibly implemented using living cells, we re-designed the algorithm to perform all the same calculations but run on a 2-bit computer, thus reducing the total number of gates required. The 2-bit version takes as input a 32-bit string and generates an 8-bit hash. To generate a hash, the user begins with a message of arbitrary length. An initial data processing step would pad the message to be 32-bits long (if shorter) or break the message into 32-bit chunks (if longer), enabling any length message to be hashed. At this stage, the “core” hashing algorithm is then applied. The 32-bit message is broken into 16 2-bit chunks, M1…M16. The algorithm proceeds through 64 “iterations” where each iteration is further classified into 4 “rounds” where each round applies a different non-linear transformation to the message. The entire message, broken into 2-bit chunks, is processed through each round in 16 iterations. The exact operations applied in each iteration are identical to the full MD5 algorithm (supp figure XX). At initialization, 4 additional 2-bit variables --- A, B, C, D --- are instantiated with user defined values. At each iteration, the values for A, B, C, D are updated. After the final iteration, the current values of A, B, C, D would be concatenated together to produce an 8-bit hash. If the initial message was longer than 32-bits, the entire process is repeated with the next portion of the message except the initial values for A, B, C, D are set to final values of from the 1st 32-bit message. There are 2 additional parameters that are pre-computed constants, S and T. S and T are tables containing 64 2-bit values, where a particular value is used at each given iteration. Each iteration involves a left-shift by the value determined by S and an addition step using the value from T. In the original implementation, the values for S were found to empirically create a fast avalanche effect. The values for T were generated from an arbitrary equation. We make no claims as to what the ideal values for these parameters are in the 2-bit case and test our design using all possible input states to confirm the circuit works regardless of the chosen parameters. Our circuit runs 1 iteration of the MD5 algorithm. The user supplies a 2-bit message chunk (mi), values for A (ai), B (bi), C (ci), D (di), S (si), T (ti), and a 2-bit value R (ri) that indicates which round should be used for the calculation (i here denotes a single bit of the 2-bit value). The circuit generates a 2-bit output, which is inputted into the next iteration as the updated B parameter.



**Figure 4:**

1. The schematic shows the complete, fully minimized circuit diagram for the MD5 circuit implemented. The 16 inputs are shown on the left and the 2 outputs are shown on the right. Shaded boxes represent circuits in individual cells (colors have no meaning and are just to aide in visualization). Any gates not within a shaded box were implemented as a single cell containing that gate. Black lines represent inputs to the circuit as a whole, implemented using non-quorum chemical inducers. The 4 colored lines represent the cell-cell signaling inputs/outputs for the circuit. Blue, purple, green, and orange lines represent OC6 (Lux), OHC12 (Cin), pC-HSL (Rpa), and DAPG respectively. The subscript i denotes a single bit of each 2-bit value.
2. Characterization of the quorum output as measured via the receiver cells for each cell in the MD5 circuit is shown. The bars are colored by quorum signal with output measured in RNAP/s (legend in top right shaded box). The plots are spatially organized to approximate the correct position in the circuit diagram. As before, lines represent either inputs to the whole circuit or cell-cell communication signals. Letters in the top right of each plot show cells that are identical to each other.



**Supplementary Figure XX: Detailed example of 2-bit MD5 calculations:** The MD5 algorithm requires 4 rounds of 16 iterations each to hash a 32-bit input message, outputting an 8-bit hash. Our implementation performs 1 such iteration. The initial message is divided into 16 2-bit chunks, each of which is fed into the algorithm in a particular sequence for a given round[cite original algorithm]. There are 8 2-bit inputs to each iteration. Mj represents one 2-bit chunk of the message. Ai, Bi, Ci, and Di are variables that contain the final hash values and are updated at each step of the iteration (i indicates iteration number). T and S are constants that are pre-determined by the user. At each iteration, 6 different operations are performed. The first operation in the process (giving Y1) is a non-linear equation that changes each round: F() in round 1, G() in round 2, H() in round 3, and I() in round 4. All other calculations are equivalent. The schematic on the right visually depicts the operations and input/outputs (red box denotes modulo), as well as how the variables A, B, C, D are updated. After 64 iterations, the final values of A, B, C, and D are concatenated to produce the final 8-bit hash. Note the initial values for A, B, C, D, S, and T are supplied by the user and must be held constant to produce the same hash for the same input. In the original 32-bit MD5, the values were empirically determined for producing optimal hashing. In addition, the initial input message is padded to make the message 32-bit longs (padded bits denoted as “b”). This step, and the final concatenation step, are not performed by our circuit.