F

Beyond CMOS: Ternary and mixed radix CNTFET circuit design, simulation and verification

Not available online due to publisher restrictions

G

Additional material

G.1 Continuous-time and discrete-time signals

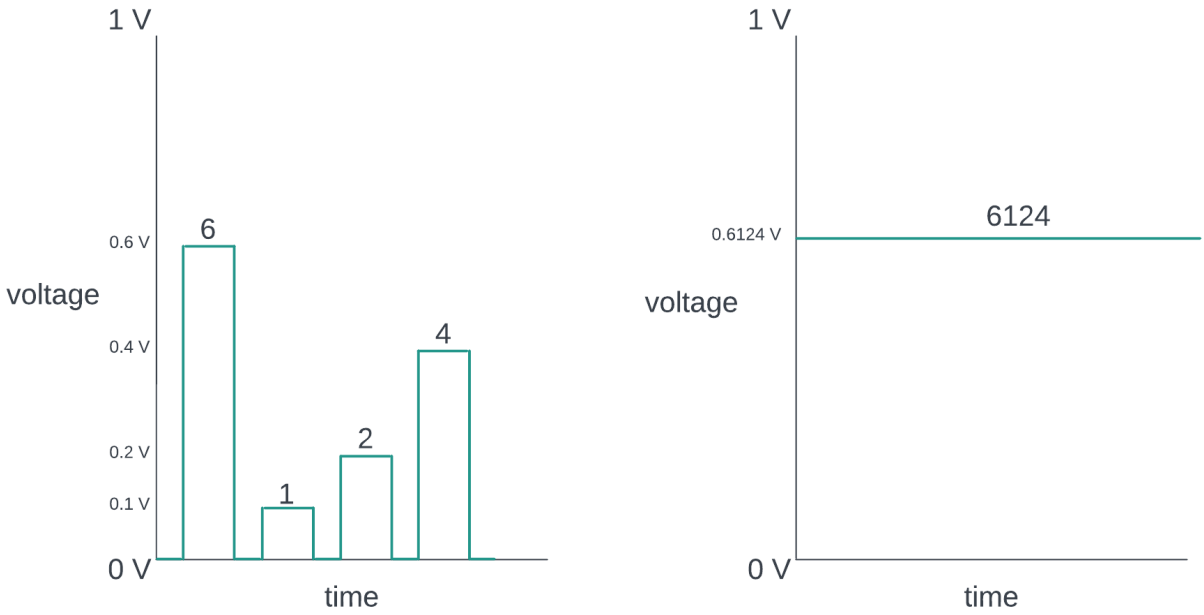


Figure G.1: Example comparison of a decimal (radix-10) digital signal to an analog signal for the number 6124. Four clock pulses are needed to convey the encoded number to a single output (serial transmission). Analog computers encode information in a continuous range of states.

McClellan et al. [326] defines signals to be "*patterns of variations that represent or encode information*" and that virtually all signals originate as continuous-time signals. These signals are *analogous* to another quantity such as voltage, temperature, pressure etc. It is often desired to transform continuous-time signals to discrete-time signals for the reasons below. The difference can be seen in Fig. G.1. Digital signals are processed sequentially and in discrete time and amplitude steps. Discretization of the time-domain is done by sampling at the Nyquist rate which allow perfect reconstruction of the analog signal. Discretization of the amplitude-domain is done by quantization, the mapping of a continuous value to a discrete value. Binary quantization (1-bit quantization) results in 2 discrete values and has the highest possible separation (noise immunity). The combination of sampling and quantization is known as analog-to-digital conversion (ADC). The inverse is digital-to-analog conversion (DAC). Discretization gave huge benefits for digital computers over analog computers: scalable precision at the cost of additional positions (such as transistors, memory cells), reliability through noise immunity and determinacy using a clock [20], [38], [304]. Analog computers at the time were large and their precision was determined by how precise the subdivision of the voltage signal could be made. With every subdivision the reliability also suffered as any noise in the signal affected the interpretation of the number. They had one major advantage though: computations were fast as they are done in parallel and in continuous-time.

G.2 Rebuttal of Buchholz' 9 arguments

Buchholz's 9 arguments on why radix-2 trumps radix-10 are shown in bold [12].

1. **Information Content**. The plot of the equation $E = \frac{log_2(N)}{\#positions}$ assumes that radix-10 is implemented with bi-stable devices and encodings like binary coded decimals (BCD). The number of positions are clearly much less with higher radixes when implemented with multi-stable devices or memory cells.
2. **Arithmetic Speed**. The same BCD implementation assumption is mentioned. Higher radixes have fewer carry signals than binary as less positions need switching thus resulting in shorter delays.
3. **Numerical Data**. The argument that general purpose computers should not include multiple ALU's for different radixes depends on the technology and application. Artificial intelligence loads have become general purpose computing and require less data movement and thus conversion. Higher radix arithmetic voids the need for conversion.
4. **Non-numeric Data**. Non-numerical information is radix independent. Higher radixes allow more efficient encoding.
5. **Addresses**. A BCD implementation is assumed for memory addresses making it far less compact. It is clearly more compact with higher radixes when implemented with multi-stable devices or memory cells
6. **Transformation**. A BCD implementation is assumed for performing data-to-address transformation (ALU operations) thus requiring handling of illegal states per BCD (the numbers 10 to 15). The balanced ternary adders and multipliers in Chapter 4 show that ALU operations can actually become less complex as they can handle negative numbers.
7. **Partitioning of Memory**. Binary has a clear advantage in terms of resolution when representing data. If data is inherently binary, then bi-stable devices encode this best. However if data is ternary then it would be inefficient to encode with bi-stable devices (25% loss per trit). Chapter 4 shows that a ternary ISA can be extremely compact as many useful signals are inherently ternary.
8. **Program Interpretation**. Direct ASCII encoding would be possible with a higher radix (such as radix-128) voiding the need for conversion. A alphanumeric subset (such as radix-36) is possible with a smaller radix.
9. **Other Number Bases**. Buchholz considers radix-10 as the only alternative to radix-2 as all other radixes require conversion to radix-10 for human readable numerical data. He ignores the radix economy, the limits of radix-2 and reduced need for human readable data in domains like Internet-Of-Things (IoT).

G.3 A radix compatible form of the CMOS power equation

The formula for power consumption in CMOS at the transistor level is well known [8], [63], [64], [336] and shown in Equations G.1-G.5. These equations are extremely simplified as for example not all short-channel effects (SCE) are modelled. To control power each term is minimized. Most emphasis is put on dynamic power consumption as this is typically the dominant term. Dynamic power consumption estimation is difficult as switching activity (α) is data or use case dependent [337, p. 14].

$$P_{device} = P_{dynamic} + P_{static} \tag{G.1}$$

$$P_{dynamic} = P_{switching} + P_{short-circuit} \tag{G.2}$$

$$P_{switching} = \alpha * C * V_{dd} * V_{swing} * F_{clk} \tag{G.3}$$

$$P_{short-circuit} = \tau * \alpha * V_{dd} * I_{short} * F_{clk} \tag{G.4}$$

$$P_{static} = V * I_{leakage} = V * (I_{subthreshold} + I_{gateoxide}) \tag{G.5}$$

Where α is switching activity, C is capacitive load, F_{clk} is the clock frequency and τ is a short period of short circuit $\frac{t_{rise}+t_{fall}}{2}$ when doing a CMOS state transition. This short circuit happens due to transitions between the pull-up networks and pull-down networks and this a feature of CMOS. V_{swing} is often the same as V_{dd}, a full rail to rail swing, hence the common V^2 term seen in literature. For higher radix signals this is not the case, as the output can also be charged to for example $\frac{V_{dd}}{2}$, a half swing. A more generalized form of the CMOS power equation is given in [73]. A derivation of this generalized form can be found in [338, p. 24] and [335]. Capacitive load is a function of gate and interconnect capacitances and fanout [2, p. 40] and is roughly proportional to chip area [337, p. 23].

Equation G.3 and G.4 can be improved to better reflect the important role of switching activity α. A single switch cycle or pulse has two stages, charging (transition from $a->b$, where $a>b$) and discharging (transition from $b->a$, where $b>a$) of the load capacitances. Each stage dissipates half the energy. Secondly, switching activity α is a probability of state transitions ($0 \leq \alpha \leq 1$) since a transistor does not need to switch every clock cycle. It depends on the logic function [337, p. 12]. For example, a binary NOR gate with inputs $P[A=1]=0.5$ and $P[B=1]=0.5$ has 16 possible states [337, p. 14]. The

probability of an actual state transition from 0 to 1, $P_{0->1}$, given the input probabilities is $P_{0->1} = P[Out = 0] * P[Out = 1] = 3/16$. Note that the reverse, $P_{1->0}$ is also $3/16$ and that the other states ($0-> 0$ and $1-> 1$) are not state transitions. This also explains why a binary clock tree is expensive (such as 30% of CPU power budget [8]) since it transitions twice every clock cycle. The amount of possible transitions depend on the radix r and can be calculates with $r * r - 1$.

$$P_{switching_{a->b}} = (\sum_{i=0, j=1, i<j}^{\frac{r*(r-1)}{2}} P_{a_i->b_j}) * 0.5 * C * V_{dd} * V_{swing} * F_{clk} \tag{G.6}$$

Ternary with r=3 results in 6 state transitions. The three positive transitions are shown in Eq. G.6. The summation term should be read as $P[0-> 1] + P[0-> 2] + P[1-2]$. It also interesting to note that Tau is dependent on the transition (see Eq. G.7). In the case of binary transistors for a ternary transition, the transistor gate might already be at $\frac{V_{dd}}{2}$, thus requiring less time to reach V_{dd} or ground. The same effect can be seen with SRAM and DRAM that reduce latency with a $\frac{V_{dd}}{2}$ pre-charge circuit.

$$P_{short-circuit_{0->1}} = \tau_{0->1} * P_{0->1} * V_{dd} * I_{short} * F_{clk} \tag{G.7}$$

At the chip level, equation G.1 depends on the amount of transistors n. Depending on the implementation, logic gates with a higher radix can be made with less transistors. The complete power equation shown in Eq. G.8 thus depends on the amount of transistors and their usage.

$$P_{chip} = \sum_{i=1}^{n} P_{device_i} \tag{G.8}$$

G.4 Using the radix economy argument

The analogy to see a truth table as uncompressed data with possible redundancies and dependence on practical device implementation will be important to show that the information limit of 58.5% (see Appendix G.7) and the radix economy does not apply for logic in practice. In CMOS two devices are used to encode binary (NMOS for 0, PMOS for 1) which is a radix-1 implementation. In practice optimal electrical circuits are not guaranteed even when the form is optimal [132, p. 522].

In Chapter 4 logic synthesis is discussed. Logic synthesis has two phases; synthesis and technology mapping. In the synthesis step a truth table is transformed to another representation that allows better compression using for example algebraic rules. In the technology mapping step the compressed representation is mapped to a circuit using available devices. Examples are bi-stable devices, bi-stable devices with different thresholds, tri-stable devices or a combination. Surprisingly, two similar phases are found in data compression theory [118, p. 6]; modelling and coding. In the modelling phase redundancy is extracted from the data and together with the data described compactly. In the coding phase this description is encoded in a radix, normally radix-2 as computers are binary. Extracting redundancy falls in the art of pattern finding which is "*an experimental science at best*" [118, p. 6]. Optimal methods, like the Quine-McCluskey method explode in runtime with large circuits such that non-optimal approximate methods are preferred. A compression result is evaluated in different ways such as how fast it was made, the compression ratio, the size, etc. For logic, the final size (area), the power consumption and delay are often the prime requirement, although synthesize speed becomes more relevant as the amount of transistors increases.

The more capabilities a synthesis algorithm has, such as knowledge about the technology mapping possibilities with various devices, the better logic functions can be modelled and exploited. Some patterns are better modelled in radix-3 than radix-2. A ternary signal like {forward, stop, backwards} can be modelled with 1 tri-stable transistor while in binary 2 bi-stable transistors are needed. In terms of device count 100% more transistors are needed which is more than the baseline of 58.5%. The overhead for binary is 25% as one state is not used (see Eq. G.10 in **Appendix G.5**). A similar ternary signal {forward, backwards, backwards} can be modelled with 1 bi-stable transistor with a shifted threshold. It is still a ternary signal at the gate as the middle value is $\frac{V_{DD}}{2}$ but the transistor response is binary. Implementing this with only binary signals would cost more since now the powerful binary input - binary output relation (Boolean logic/bi-stable transistor) is broken. The middle value can only be made by using another pin and encode this state with inefficiency (not using 1 state). This makes the radix-2 implementation cost more than the 58.5% baseline.

The radix economy does not model uncompressed data such as a logic gate well. The **r*w** cost function models information storage with r-stable state devices and not state

transitions mappings with r-stable state devices. This is not surprising considering the two phases in compression theory. Coding is a different step working under different assumptions than modelling. It might therefore be inconclusive to use the radix economy to argue which radix is optimal for logic, use the information limit (such as 58.5%) to judge design efficiency or use the information limit to predict how it scales. There are too many practical variables. As a rough approximation it might still be useful though. For example, a binary (3,2) ripple carry adder encodes it inputs and outputs without redundancy. A balanced ternary (4,2) parallel adder [202] does the same. For both scaling to higher number resolution means chaining them, which means that there is a direct correlation between pin encoding and transistor count of these two implementations. Pin encoding is number encoding, not number transformation (logic function) encoding. In this case a direct comparison between the binary and ternary solution is possible and the static information limit 1.585 can be used as a decision threshold. It might be that there exist a better binary design or a better ternary design with fewer transistors, so the comparison does not conclude anything on how close each one is to the optimal circuit given certain devices and design constraints.

G.5 Overhead calculation

The change of base (change of radix) equation $\log_b(s) = \frac{\log_a(s)}{\log_a(b)}$ shows mathematically how to express any number in some radix, such as the number of states s in radix-a, in another radix, radix-b. The relation between binary and ternary was described in Eq. 1.8 in Chapter 1.4.3. The number n and m indicate how many positions are needed in binary, respectively ternary to represent s. These positions are typically implemented as discrete devices such as memory cells or transistors whereby each device can read/write the number of symbols in the radix. The number of states s often represents a huge number like the bus architecture (such as 64-bit). As digital computers use discrete devices, rounding to integers is needed. The rounding operation is a floor operation such that the comparison is done against the maximum capacity of the rounded number. This is done such that the overhead is always positive. The rounding difference is the overhead and is calculated with one radix being precise and the other one being approximated. The overhead equations for binary and ternary are Eq. G.9 and Eq. G.10:

$$Overhead_{ternary}(m) = \frac{3^m - \lfloor 2^{log_2 3^m} \rfloor}{3^m} * 100\%, m \in N. \tag{G.9}$$

$$Overhead_{binary}(n) = \frac{2^n - \lfloor 3^{log_3 2^n} \rfloor}{2^n} * 100\%, n \in N. \tag{G.10}$$

Overhead are states that are never used but can be made with the given radix and positions. With specific pairs the overhead can be increasingly smaller, such as {5,8} (5.1%), {12,19} (1.3%), {53,84} (0.2%) and {306,485} (0.1%). For more pairs see [318], [319]. A nice visualisation of the overhead can be found in [282, p. 9]. The resulting pairs are "weird" device numbers and not very practical so one radix is typically chosen as the reference. If radix-2 is chosen as the reference, radix-3 will always be inefficient, especially for common architectures like 16, 32 or 64-bit (see [282, p. 9]. The smallest pair with reasonable small overhead (in this case disadvantage for binary) is the mentioned {5,8}. However since 5 is not a multiple of 3, logic design is more complex when scaling to more inputs. Ternary architectures should therefore ideally use radix-3 to be efficient.

Approximate overhead calculations are also found in literature [160] which use the ratio of discrete numbers in different bases to show its approximation to a baseline like $log2(3)$. For example, the {5,8} pair results in an approximate overhead $overhead - approximation = |\frac{8}{5} - log2(3)| \approx 0.015$ or 1.5%. This is significantly less than the 5.1 % found in the exact calculation and should thus be used with care.

G.6 Comparing baselines to 58.5% or to 63.1%

Various paper use different reference points with radix-3 or radix-2 as the baseline. For radix-3 the baseline is $\frac{log(3)}{log(2)} = log2(3) \approx 1.585$. This can be interpreted as "if ternary needs 100 tri-stable transistors to encode number s, binary would need 158.5 bi-stable transistors to encode the same information according to information entropy". Note that transistors can be substituted for pins or interconnects. For memory this interpretation can easily be found in practice: one tri-stable memory cell like a memristor or capacitor can differentiate (read/write) 3 stable states compared to two states in a bi-stable memory cell. Encoding 9 states (independent of balanced or unbalanced encoding) requires 2 tri-stable cells and 2 *1.585 = 3.17 bi-stable cells.

Instead of the baseline being radix-3 with $log2(3) \approx 1.585$, the baseline can also be radix-2 with $\frac{log(2)}{log(3)} = log3(2) \approx 0.631$. A nice table with this baseline can be found in [126]. This can be interpreted as "if binary needs 100 bi-stable transistors to encode number s, ternary would need 63.1 tri-stable transistors to encode the same information according to information entropy". Encoding 8 states needs 3 bi-stable devices and 3 * 0.631 = 1.893 tri-stable devices. Since ternary architectures should use radix-3 for their architecture, 1.585 should be the baseline.

G.7 58.5% is an information limit, not a system limit

The information theoretical density difference of ternary compared to binary is $log2(3) \approx 1.585$. As discussed in Chapter 1.5.3 and **Appendix G.4** memory is fairly straightforward to compare but logic is difficult. The bigger picture is that memory and logic devices are integrated into a functional system. The system limit is a practical and often economical limit and is obviously more important than the information limit. If ternary memory devices can encode information at or above the information limit (due implementation with discrete components, see **Appendix G.4**) but cost twice as much then special circumstances are needed to justify it such as the potential to be cheaper. The same holds for other metrics like performance, area and power. Individually a tri-stable transistor might be superior, but it is at the system level what counts [138].

This makes comparison between radixes even more difficult because the design and fabrication challenges are different for trivial circuits compared to dense, high performance circuits. Interconnect density becomes a problem when cramming transistors in a specific area with some target performance requirements. The realisation that transistor count is not the only factor of importance was already noted in 1955 [12, p. 6]. Many of the benefits of using a richer computer alphabet as discussed in Chapter 2 and paper D happen at the system level.

G.8 Ternary computers architectures from 2004-2022

- Alexander Shabarshin (2004) [306]. 3-nity alpha: 9-trit ternary computer architecture. The website also links to a java emulator and to the largest ternary computing forum (to my knowledge). The architecture has not been implemented but Shabashin has made ternary logic IC with CMOS in 2015 [307].
- Connelly et al. (2008) [308]. 3-Trit ternary computer architecture. The PDF report covers many ternary computing concepts and features a PCB implementation with off-the-shelf components.
- Viktor Lofgren (2008) [309]. Tunguska: 6-trit ternary computer architecture. The CPU is based on the (binary) MOS Technology 6502 processor. The source code includes an emulator, compiler and assembler.
- Thomas Leathers (2016) [310]. The Simple Balanced Ternary Computer Virtual Machine (SBTCVM) project written in python. A complete balanced ternary computer virtual machine with operating system, graphics capabilities, 18-trit CPU, ternary assembly and several ternary applications/games. The Github repository contains detailed documentation.
- Douglas W. Jones (2017) [311]. Trillium: 9-trit ternary computer architecture. The architecture is specification only. The website is a treasure trove for many ternary researcher.
- Dmitry V. Sokolov (2017) [313]. Triador: 3-trit balanced ternary computer architecture. This project features a custom PCB, emulator and off the shelf components to build this computer. Several parts were documented on Youtube.
- Wust et al. (2018) [320]. CMOS/Memristor MIPS ternary computer architecture. This paper simulated (with SystemC/Spectre) a MIPS based binary and ternary CPU and compared three filter algorithms. They report that ternary outperforms binary in PDP when the architecture is larger than 32-bit.
- Louis Duret-Robert (2019) [314]. SAP-1: 9-trit balanced ternary computer architecture. This ”Simple-As-Possible” architecture by Albert Paul Malvino is made in both binary and ternary. The ternary computer is made with ternary verilog, a custom made HDL extension and features a emulator. The blog[315] also contains a 2-trit balanced ternary ALU implementation with off-the-shelf components. The ALU is compared to a binary equivalent in cost, energy consumption, performance and complexity.
- Cesare Di Mauro (2019) [316]. 3ISA: 20-trit ternary computer architecture. The ALU is part of the 24-trit computer project 5500FP by Claudio La Rosa [317]. This project is made with a custom PCB and off-the-shelf components.

- Reichenbach et al. (2021) [188]. RISC-V3: 64-bit CPU architecture with ternary datapath. Simulation shows that ternary arithmethic (ALU) can improve performance of a binary CPU with 130nm and 150nm open and commercial PDKs. A slow down is seen in logic and branch-intensive applications. They note that with larger architectures ternary has an increasing benefit [188, p. 43698]: "processors made with a ternary data path allow a much wider word width without having any impact on the clock frequency".
- Kam et al. (2022) [187]. ART-9: 9-trit RISC-based balanced ternary computer architecture. This paper described a pipeline converting regular RISC-V assembly to ternary assembly. The ternary assembly can be executed on a 150 MHz Intel Stratix-V FPGA with binary encoded ternary or emulated with multi-threshold CNTFET devices. The work uses the widely used Dhrystone benchmark and several others benchmarks to assess power, memory, ISA complexity and performance. Reported is that their 24 instruction ternary ISA outperforms modern ARMv6M and RV32i instruction sets.
- Gadgil et al. (2023) [287]. A 3-trit unbalanced ternary CNTFET CPU architecture. The 14 instruction ternary ISA is not based on RISC. No CPU benchmark was performed to validate performance.

G.9 Experimental 2-trit memristor results using uMemristorToolbox

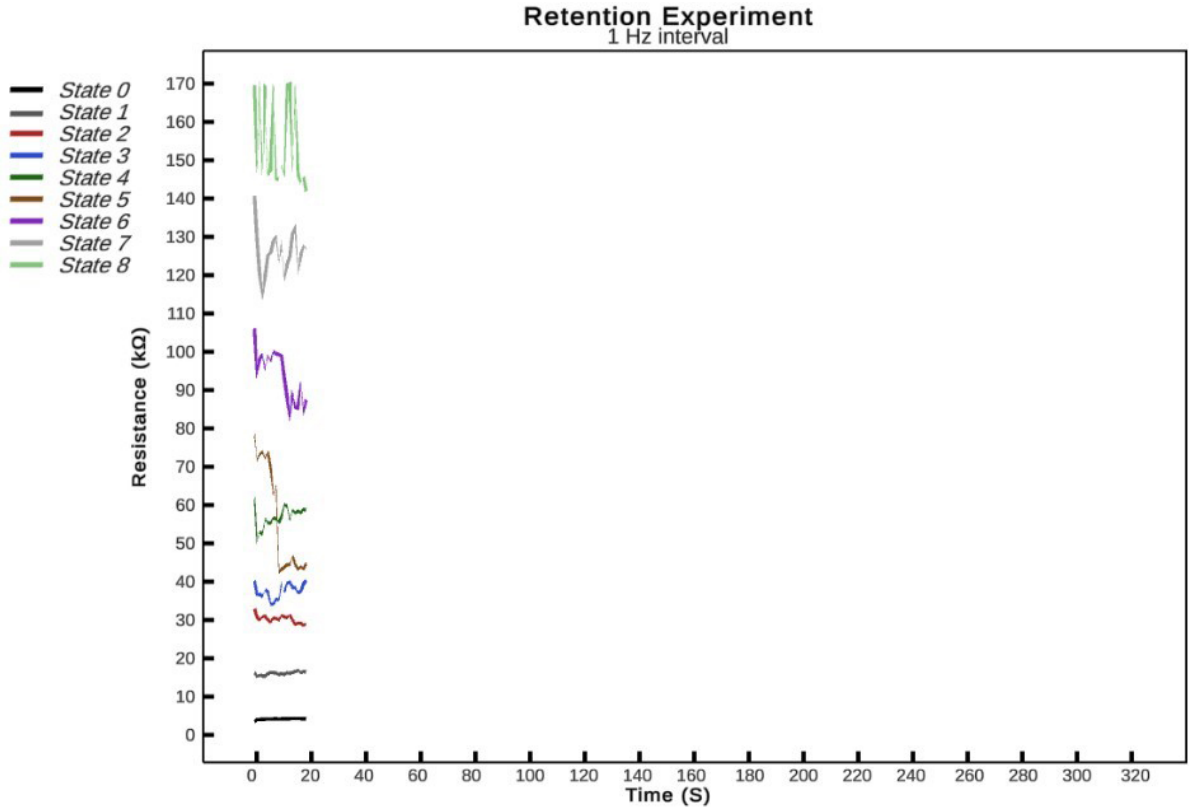


Figure G.2: Measurement of nine memristance levels using the Knowm PCB and uMemristortoolbox. Shown is an erroneous switching event after 5 seconds. A similar event is reported in **Paper A**. Image source: MSc thesis by Virk [241].

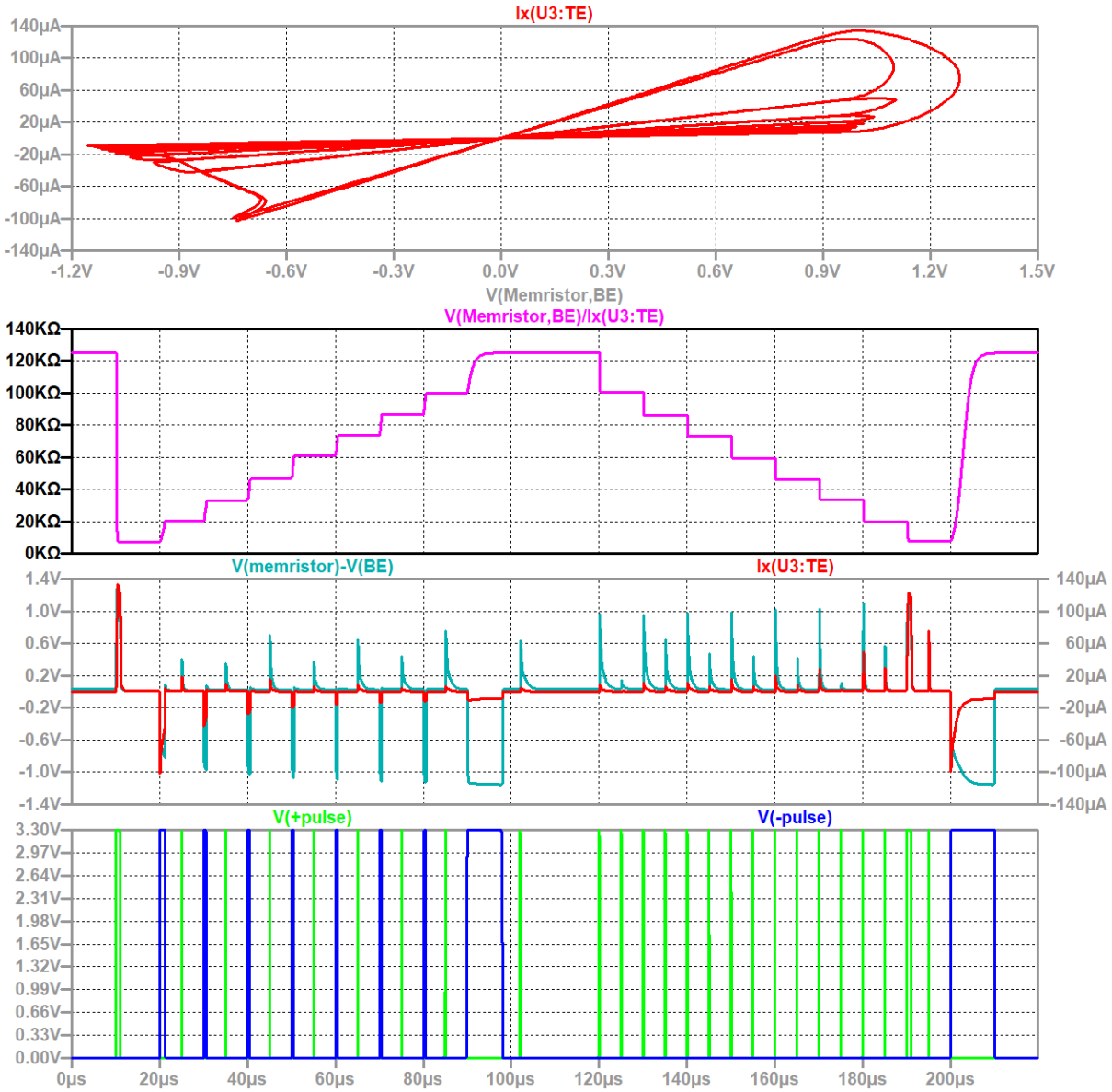


Figure G.3: Simulation of nine memristance levels using the multi-state RRAM development platform described in Chapter 3.3, LTspice and Knowm's Mean Metastable Switch Memristor Model [256]. The delta programming scheme and other details can be found in [241]. Image source: MSc thesis by Virk [241].

G.10 Multi-state RRAM development platform prototype

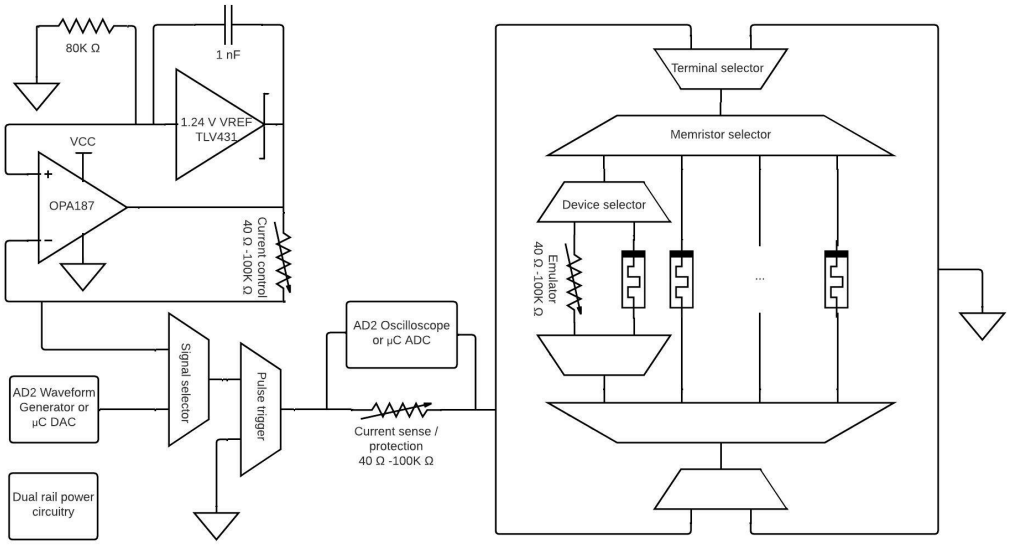


Figure G.4: A novel multi-state RRAM development platform with programmable voltage and current pulses and programmable resistor for memristor emulation.

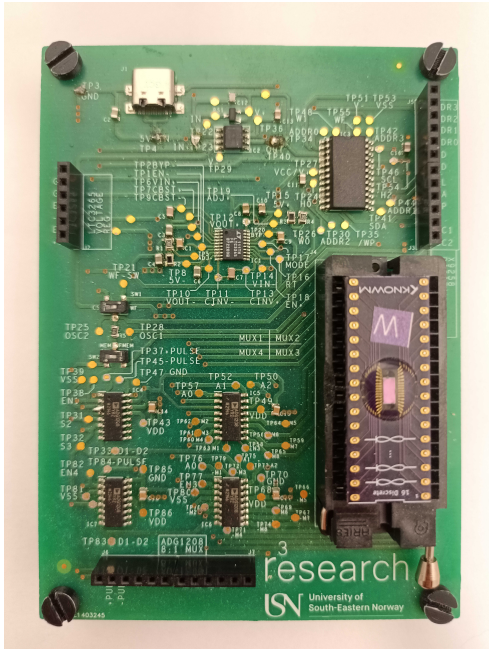


Figure G.5: First PCB implementation of the multi-state RRAM development platform. More details, see [241]

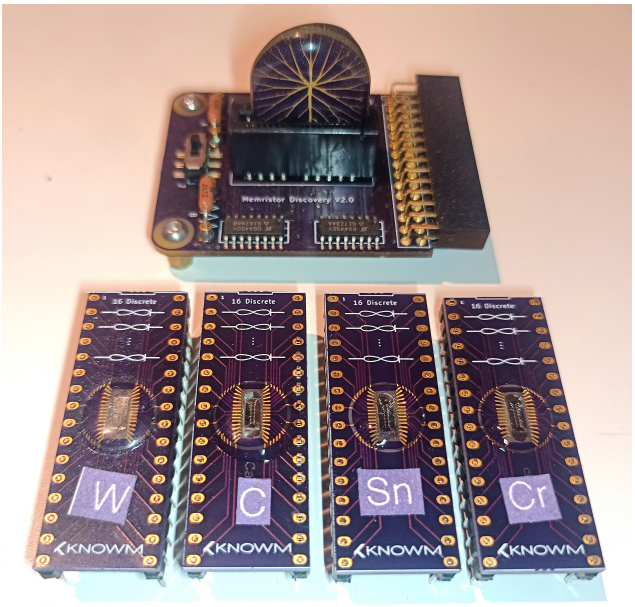


Figure G.6: Overview of used memristors. Shown are four different SDC models (W is Tungsten, C is carbon, Sn is Tin, Cr is Chromium with different characteristics in two different packages. See Knowm datasheet [238].

G.11 Getting started with MRCS

Installation

MRCS runs on three platforms; the web, Windows and the Unity editor. The online version of MRCS does not require installation and can be accessed via ternaryresearch.com [262] using a modern browser. The Windows version can be downloaded from the Github repository [261]. The executable does not require installation. The Unity editor version is designed for software developers. It requires the installation of Unity version 2023.1 or later with WebGL plugin and Visual Studio. After cloning the Github repository [261], the project can be opened via the Unity Hub and selecting *Open > Add project from disk*. The added project now points internally to the file main.unity which is the entry point containing the UI. After opening the project the project should look like Fig. 4.2 from Chapter 4 which includes a hierarchy and inspector view of each component.

User interface and user experience (UIUX)

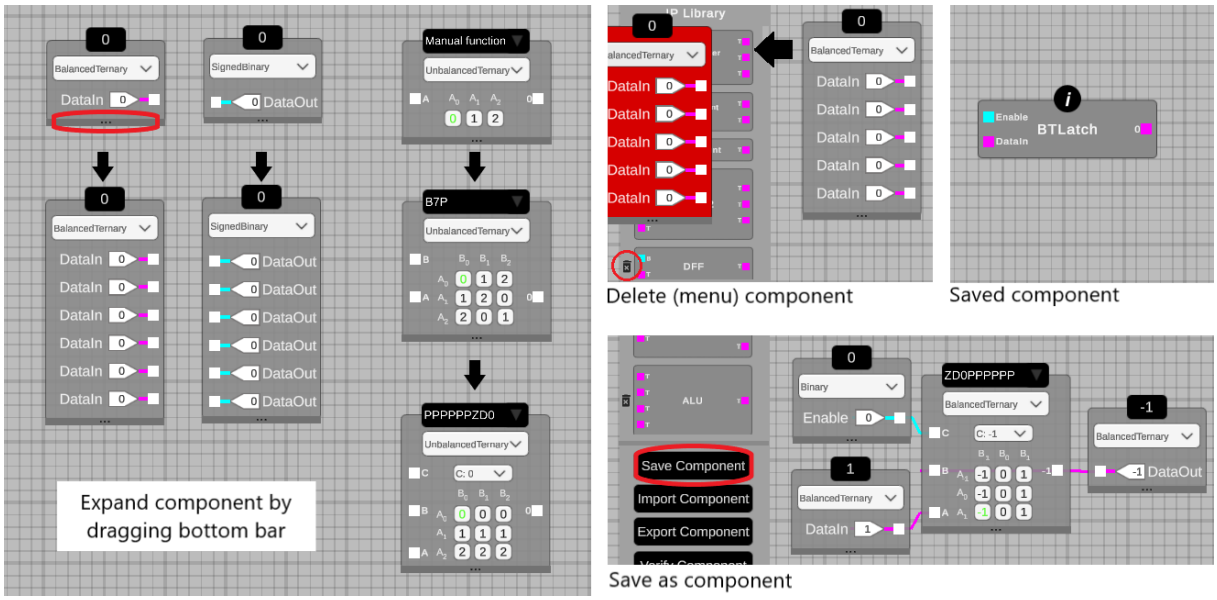


Figure G.7: **Left.** A standard component can be dragged from the drag&drop panel. Change radix with the dropdown box. Depending on the type of component, the arity can be changed by dragging the bottom bar. **Bottom-Right.** Components can be saved as a new component after attaching input and output to it and clicking "Save component". **Top-Right.** A saved component has an i button which reveal the statistics (such as the amount of transistors) and the schematic. **Middle.** Delete saved component by clicking the trash icon next to it. Canvas components can be deleted by dragging them left off-screen.

In **Paper F** the menu options and general interface of MRCS is shown. Figure 4 of **Paper F** contains a step-by-step guide to create a simple binary latch circuit in MRCS.

An animated video is shown in [340]. In Fig. G.7 the most important user interface interactions are shown which were not visualized in the paper. The interface of MRCS is still in flux and is likely to change in future versions. For example, a large IP library requires better organisation than the currently implemented scrollbar. The workspace has no zoom functionality and space for about 20 pins which limits creating large designs.

Persistent storage and file management
A saved component in MRCS has a set of files associated to it with a fixed file/folder structure. This structure is persistent for all three versions but has different locations. All data is stored locally, nothing is stored on remote servers. For the WebGL version, all saved components and settings are stored in cache [339]. It is important that the user exports the IP library regularly as a backup since the files are stored in browser cache. For Windows and the Unity editor version, the filepath is `C:\Users\[username]\AppData\LocalLow\USNKongsberg\MRCS`. The name of a saved component will be added to the settings file named *library.csv* such that it can appear in the IP library panel. The generated files associated to the saved component can be found in the folder `filepath\Generated\[componentname]`. This folder contains two folders; HSPICE and verilog. The HSPICE folder contains a set of .sp files and the Stanford CNTFET .lib file. It also contains a main.sp which is the top level interface. This file contains the CNTFET settings MRCS uses for ternary signals and includes a standard test pattern for simulation with Synopsys PrimeSim HSPICE version 2020+. The verilog folder contains a set of .v files and the file [filename]_singlefile.v which is the flattened version of the set of .v files in a single file.

G.12 MRCS Limitations

The tool is still experimental and has the following limitations:

Simulator
The event-based gate-level simulator uses discrete-time simulation steps (unit delays). Each gate that is triggered by an input change (event) can only evaluate all of its inputs once per simulation step. The new output is propagated to the next connected gate or output component after 1 unit delay. Wires have zero delays associated with them and are always driven to a known logic state. High-z or tri-state logic is not supported yet. Glitches that happen faster than a unit delay are not caught which might affect proper simulation of some designs such as edge detectors. Components can have unlimited fan-out and only one input which is both unrealistic and impractical for some circuits.

Sequential circuits with Verilog
Circuits with feedback loops such as latches and flip-flops have strict timing requirements (set-up and hold times) to be functionally correct in practice. Mixed signal simulation and post-layout verification of FPGA and ASIC tooling might show different behavior than the gate-level simulator of MRCS. Clock signals are not annotated in MRCS and the generated verilog code does not contain for example *always @(posedge clk* blocks. The workaround used in tapeout-4 [266] was to design a reusable d-flip-flip (0tZD0PPPPPP) as presented in Fig. G.12 in **Appendix G.15** using two identical d-latches. The generated verilog of that latch was replaced with the code block shown in Fig. G.8. A future version of MRCS might introduce a setting where these blocks are automatically generated when feedback loops are used in digital designs.

```
module f_ZD0PPPPPP_bet (
     input wire[1:0] portC,
     input wire[1:0] portB,
     input wire[1:0] portA,
     output reg[1:0] out
     );

     always @(posedge portC[1])
     out <=
     (portA == 2'b01) ? 2'b01 :
     (portA == 2'b10) ? 2'b10 :
      2'b11 ;
endmodule
```

Figure G.8: Verilog workaround for BCT with ternary d-latch. The generated *assign* block was replaced with the shown *always* block. The interface was unchanged.

Synthesis engine
The synthesis algorithm is not optimal. More efficient manually designed circuits are certainly possible as discussed in paper B and [282]. An example is the balanced ternary full adder discussed in section 4.4. The synthesized implementation reported in [155] requires 118T but can be reduced to 110T by wiring the output of the SUM gate to the CARRY gate. Extracting reusable components such as input inverters or whole logic functions and the usage of clever wiring requires a powerful ternary algebra. For Boolean functions no universal synthesis method has been discovered that produces optimal result in little runtime and for all types of logic [322]. Trivial situations where an inverter at the output cost less than inverting the inputs are not yet considered by the algorithm nor reusing shareable inverters of multiple sub-circuits. This optimization needs to be done manually. As can be seen in Fig. 4.6, optimizations are done locally in one of the four maps (networks), not across. This is a huge optimization opportunity.

The original C++ version of the algorithm allows higher arities than 3, but these are not yet supported in MRCS. Arity-4 circuits as basic building blocks make a lot of sense for ternary circuits. For example T-gates are arity-4. Arity-4 balanced ternary counters (4:2 compressors) are also optimal as 4 1-trit inputs have a total range from -4 to +4, which is exactly a 2-trit output.

The synthesis engine adds metadata as comments to the .sp files when saving logic gates as a component. MRCS expects this format when loading components and restore the location and settings of the design. This makes editing the subcircuit .sp files manually error prone. The top-level module .sp file can be modified without worry.

Heptavintimal implementation
The largest quirk of MRCS is a coordinate system misalignment between the synthesis engine (bottom-right) and the user interface (top-left) as shown in Fig. G.9. In hindsight a top-left notation for both would enable reading the heptavintimal index row by row from a truth table starting from the top. Alignment is a breaking change which unfortunately requires new heptavintimal indexes for all truth tables discussed this thesis.

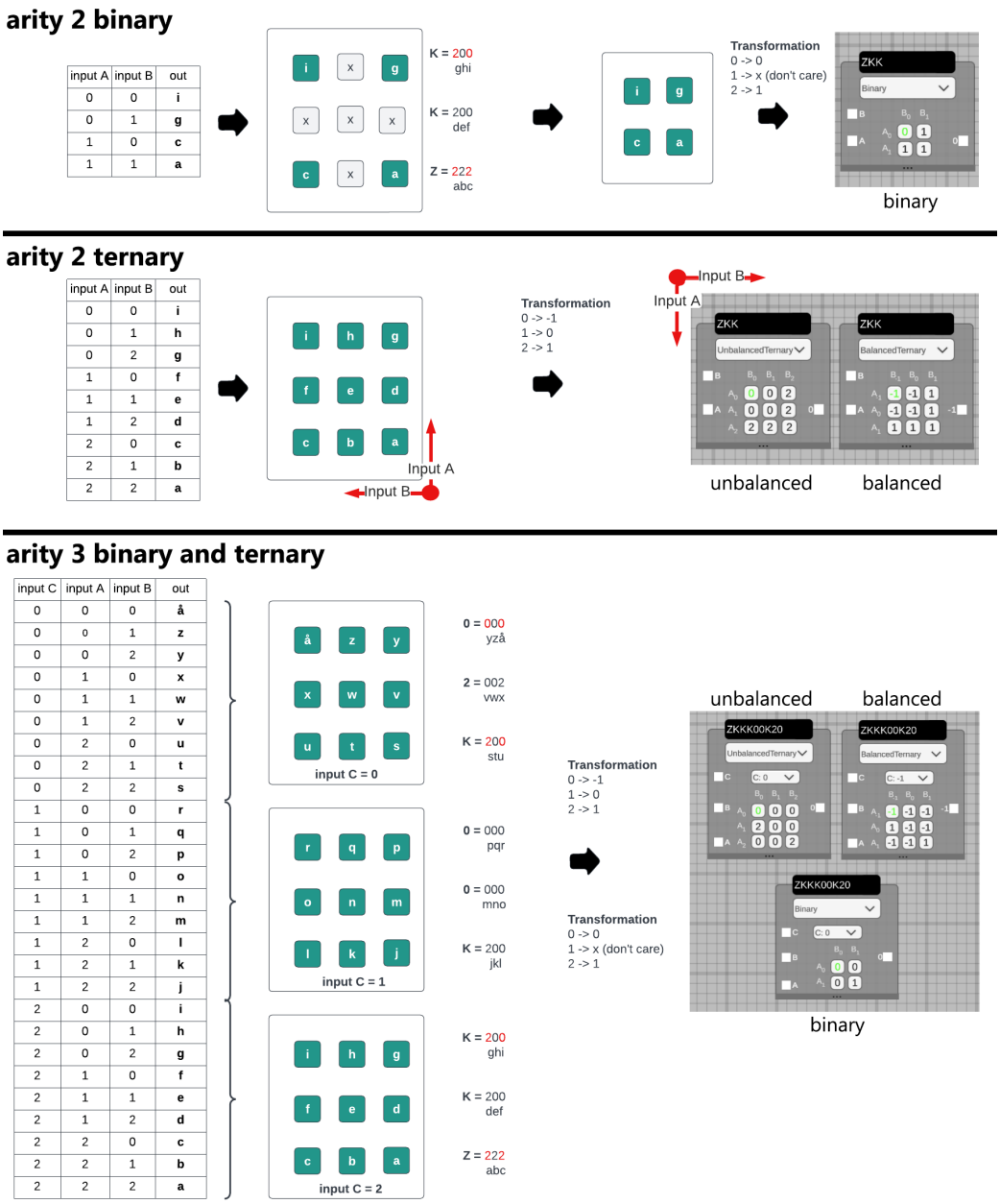


Figure G.9: Heptavintimal implementation in MRCS. Digits in red are binary truth table cells.

G.13 Ternary algebra

Early literature in the field of ternary algebra and switching theory researched mathematical optimizations without caring too much for circuit implementation [139], [166], [168], [170], [267]. Logic representation, algebra and circuit implementation are intimately related. Picking the right combination is needed to make ternary competitive to binary [141, p. 32]: *"In considering MVL algebra, we are most interested in those where the algebraic operations represent functions which have straightforward circuit implementations and which have sufficient representative power that effective circuit implementations can be constructed for general p-valued functions"*. In other words, switching devices require ternary logic representations [325] that can model them efficiently and ternary algebra that can use these models to construct optimal circuits according to an VLSI objective.

Literature on ternary computers from 1964 show that ternary algebra's that are currently heavily used such as chain-based Post algebra's [141, p. 38] are not optimal. For example, one of the most essential logic gates, the adder, is not optimal with this type of algebra without efficient implementation of the cycling gate [171, p. 38]: *"The major disadvantage of the post algebra is that it requires a large number of 'cycling' and 'or' units when functional circuits, say for addition or subtraction of numbers, are attempted."*

Chain-based Post algebra's for ternary logic is a functionally complete ternary input to binary output algebra [99]. This means that the algebra can map all possible ternary (M=3) input combinations of arity n to a binary valued output:

$$f : M^n \rightarrow \{0,1\} \tag{G.11}$$

Eq. G.11 is fundamentally sub-optimal for ternary due to the ternary-to-binary encoding (see theorem and proof in [141, p. 31]). The three values of M form a single totally ordered chain $0 < 1 < 2$, hence the name chain-based Post algebra. In binary (M=2) a functionally complete algebra can be achieved with three operations: MIN, MAX and the literal operation NOT. For ternary input with binary output, the binary MIN and MAX operations can also be used. However an extension is needed for the NOT operation in the form of several literal operations (see proof in [99]). The short notation of Post algebra for ternary is given in Eq. G.12:

$$\langle M; +, \cdot, L; \{0,1\} \rangle \tag{G.12}$$

Where M is the input set {0,1,2}, + the binary MAX operation, $\cdot$ the binary MIN operation, L the literal operations and {0,1} the output set.

Table G.1: Bi-stable subset of ternary unary functions. Green functions miss efficient implementations.

F(a) / IN	LOW	NTI	MTI	PTI	$\overline{PTI}$	$\overline{MTI}$	$\overline{NTI}$	HIGH
	0t0	**0t2**	**0t6**	**0t8**	**0tK**	**0tN**	**0tV**	**0tZ**
0	OFF	ON	OFF	ON	OFF	ON	OFF	ON
1	OFF	OFF	ON	ON	OFF	OFF	ON	ON
2	OFF	OFF	OFF	OFF	ON	ON	ON	ON

For chain-based Post algebra, a literal operation is defined as a unary (1 ternary input to 1 binary output) function of the type $0,1,2 \rightarrow 0,1$ [99], [168]. Physically a binary signal operates at the extremes, so the mapping is better represented as $\{0,1,2\} \rightarrow \{0,2\}$ [168]. In the case of binary logic circuits, the logical state $\{0,2\}$ can be implemented with a bi-stable single threshold transistor with the operating regions $\{TransistorOFF, TransistorON\}$. In CMOS logic $\{0,2\}$ is implemented with $\{NMOS, PMOS\}$. For ternary logic with bi-stable multi-threshold transistors more options exist. Multi-threshold transistors can be ON or OFF for the middle value. From the 27 unary functions in Table 2.2 from Chapter 2, only eight possibilities can be extracted that feature the set $\{0,2\} or \{OFF, ON\}$. This subset of literal operations are ideally suited for bi-stable transistors and is shown in Table G.1:

Two literals in Table G.1, LOW and HIGH are trivial and require only proper wiring to GND or VDD. From the 6 remaining only 4 (NTI , $\overline{NTI}$, PTI and $\overline{PTI}$) are commonly found in literature [225], [278], [280]. With multi-threshold CNTFET the V_{th} can be shifted to create OFF-ON-ON or OFF-OFF-ON behavior (and their inverses). The remaining two toggling gates are only found in literature as compositions of other gates and transistors [323], [155]. No names were found for the missing two literals in literature and are abtly named MTI ("middle toggling inverter") inline with the PTI and NTI naming. With complementary logic, the MTI should be made with 2 devices to handle ON and OFF just like the other literals. It should be noted that the hypothetical MTI still produces a binary signal. To create a ternary signal either two literals in voltage division mode are needed or a dual power source solution. In the latter case Table G.1 is duplicated with ON meaning ON_{VDD} or $ON_{\frac{VDD}{2}}$ and thus significantly increases the literal count in Post-algebras.

Much is uncertain how to fabricate the missing MTI's without composition and is not the focus of this thesis. Literature on ternary device technologies are provided in Chapter 2.1. For example, one possible direction could be to create a more sophisticated device gate that only responds to the middle value such as resonating tunneling devices [324, p.358]: "*Externally, resonant-tunneling compressed-function transistor circuits are binary, while internally certain circuit nodes are multivalued, to achieve the increased function per device.*". Another possible direction could be based on NEMS (nano-electro-mechanical) such as rotating discs with two states (toggling) or three states (cycling).

G.14 Towards a ternary standard cell library

The amount of logic functions one can make is based on the radix r and arity n. The arity of a function is the number of inputs, argument or operands (synonymous terms). Assuming input and output in the same radix the amount of logic functions can be formulated as

$$\sum F_{r,n} = r^{r^n} \tag{G.13}$$

In binary there are $2^{2^1} = 4$ monadic (1-ary) logic functions of the type $y = F(a)$ and $2^{2^2} = 16$ diadic (2-ary) logic functions of the type $y = F(a,b)$. Each function is named (such as AND, OR, BUF) but not all of them are equally useful. By substituting logic functions in each other higher arity functions are possible. For example triadic (3 input, 1 output) function can be made with 2 diadic ones $y = F(c,F(a,b))$. The proof for this can be found in [167]. Perhaps surprisingly, only a small set of functions is needed to construct all functions. This is called a functionally complete set or a set of universal gates. This similar to the concept of *basis* in linear algebra which spans a vector space with just a few linear independent vectors. This also implies that more than one basis or functionally complete set can be found which is proven in [167]. The usefulness of this set depends on the complexity (costs) of the implementation. For binary the absolute minimum set is just one gate: the NAND or NOR gate. By chaining ("compounding") and wiring the inputs together great flexibility with a single cell is gained at the cost of (much) more gates. In modern chip design standard cell libraries are often constructed for a certain technology node. These libraries often include the functionally complete set INVERT,AND,OR as individual gates and as a single 4-ary AndOrInvert (AIO) gate as well as commonly used 2-ary XOR gates. A standard cell library contains transistor layout variations of this set such as high density, high speed or high drive strength. No standard cell library for ternary exist yet. This section proposes a list of useful monadic (1 input), diadic (2 inputs) and triadic (3 inputs) logic functions that could evolve in a ternary standard cell library when ternary device technology becomes mature.

In ternary there are $3^{3^1} = 27$ monadic logic functions and a stellar $3^{3^2} = 19683$ diadic ones. Some of these been named and identified in literature, but most of them have not. Since binary is a subset of ternary, all binary functions can be represented as ternary functions. Unlike binary, ternary also has sets of universal gates which can be made with 3 unary functions STI, PTI, NTI. This set is comprised of three inverters: the standard ternary inverter (STI), positive ternary inverter (PTI) and negative ternary inverter (NTI). Like binary, many other functionally complete sets can be made but no consensus is found in literature which set is the most economical in terms of implementation as inherently ternary devices are not available. Other functionally complete sets with proof can be found in [168], [169]. An often cited universal set is the *ternary gate* or T-gate [167]. This

single gate is implemented as a multiplexer of 3 ternary functions with 1 ternary select signal. The term *ternary gate* should be reserved exclusively to refer to generic ternary logic gates. The term T-gate should be used when referring to them.

Tables G.2, G.3 and G.4 are a compilation of common single-gate logic functions to design mixed-radix circuits. Since the full list from arity 1 to arity 3 contains 27 + 19683 + 7625597484987 = 7625597504697 logic functions, much is still to be discovered. The large majority of designs found in this thesis can be made with this set of building blocks.

Table G.2: Overview of useful arity-1 building blocks

Hepta Index	Name/Alias	Radix	Comment
2	INVERT	2	
K	BUFFER	2	

0	CONST_LOW	$\overline{3}$	
2	NTI	$\overline{3}$	DETECT_LOW
5	STI	$\overline{3}$	
6	MTI	$\overline{3}$	DETECT_MIDDLE
7	INCREMENT	$\overline{3}$	NEXT, SUCCESSOR
8	PTI	$\overline{3}$	
B	DECREMENT	$\overline{3}$	PREV, PREDECESSOR
C	CLAMP_DOWN	$\overline{3}$	
D	CONST_MIDDLE	$\overline{3}$	
K	$\overline{PTI}$	$\overline{3}$	DETECT_HIGH
N	$\overline{MTI}$	$\overline{3}$	$\overline{DETECT\_MIDDLE}$
P	BUFFER	$\overline{3}$	
R	CLAMP_UP	$\overline{3}$	
V	$\overline{NTI}$	$\overline{3}$	
Z	CONST_HIGH	$\overline{3}$	

Table G.3: Overview of useful arity-2 building blocks

Hepta Index	Name/Alias	Radix	Comment
20K	SUM	2	XOR
K02	NXOR	2	
K00	MIN	2	AND
RDC	MAX	2	OR
22Z	NMIN	2	NAND
002	NMAX	2	NOR
B7P	SUM	3	
C90	CONS	3	
EHZ	NCONS	3	
R99	ANY	3	
4HH	NANY	3	
7PB	SUM	$\overline{3}$	TRISHIFT (DEC,BUF,INC)
RDC	CONS	$\overline{3}$	CONSENSUS
4DE	NCONS	$\overline{3}$	
PC0	MIN	$\overline{3}$	Ternary AND
ZRP	MAX	$\overline{3}$	Ternary OR
045	NMIN	$\overline{3}$	
5EZ	NMAX	$\overline{3}$	
5DP	XOR	$\overline{3}$	
PD5	MULTIPLY	$\overline{3}$	DIVIDE (div/0 is 0)
PRZ	IMPLICATION	$\overline{3}$	Kleene Logic version
XP9	ANY	$\overline{3}$	
15H	NANY	$\overline{3}$	
H51	COMPARE	$\overline{3}$	MORE,LESS,EQUAL (MLE)
RD4	ENABLE	$\overline{3}$	ENABLE w/ binary
VP0	DESELECT	$\overline{3}$	A or B, with one being $\overline{\textit{ENABLE}}$

Table G.4: Overview of useful arity-3 building blocks

Hepta Index	Name/Alias	Radix	Comment
KKKK00Z00	2:1 MUX	2	FE D-LATCH (feedback to B)
Z00K00KKK	2:1 MUX	2	RE D-LATCH (feedback to B)
K0200020K	SUM	2	XOR
ZKKK00K00	CARRY	2	
ZZZZKKZKK	MAX	2	OR
K00000000	MIN	2	AND

PPPPPPZD0	2:1 MUX	$\overline{3}$	FE D-LATCH (feedback to B)
ZD0PPPPPP	2:1 MUX	$\overline{3}$	RE D-LATCH (feedback to B)
ZD0DDDPPP	2:1 TRIMUX	$\overline{3}$	Tristate with zero, not HZ
B7P7PBPB7	SUM	$\overline{3}$	
XRDRDCDC9	CARRY	$\overline{3}$	
ZZZZRRZRP	MAX	$\overline{3}$	
PC0CC0000	MIN	$\overline{3}$	

G.15 Combinatorial and sequential building blocks

In this section ternary and mixed-radix combinatorial and sequential building blocks are discussed that were developed in this thesis work. These HSPICE and FPGA verified building blocks are essential for building a ternary computer.

Ternary data-latch

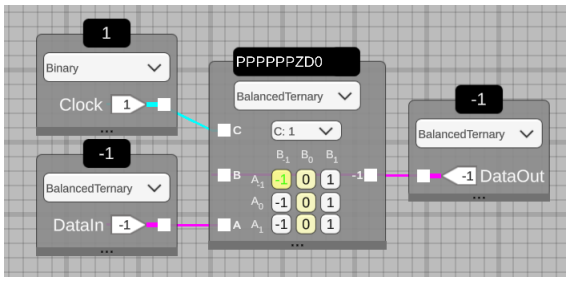


Figure G.10: 28T gated balanced ternary d-latch based on 2:1 MUX

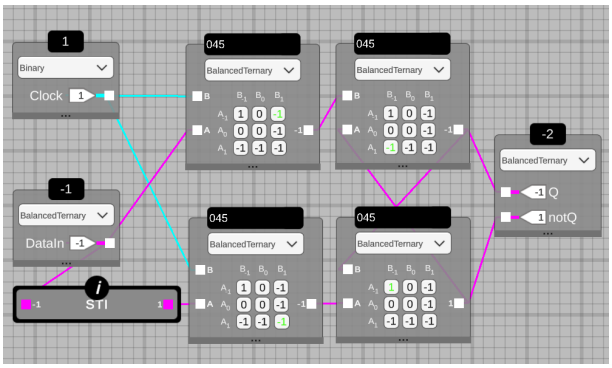


Figure G.11: 46T gated balanced ternary d-latch based on NMIN

The latch is one of the most important memory elements. In Fig. G.10 a 28T level-controlled balanced ternary data-latch (d-latch) implementation is shown that was presented in paper F. The d-latch is effectively a 2:1 MUX with feedback from the output to input B. Compared to a naive binary CMOS implementation of a 2:1 MUX with 2 AND, 1 OR and 1 INV (=20T) the MRCS synthesized design is within the 58.5% margin. There exist many other types of d-latch designs such as cross-coupled inverters with transmission gates or gated Set-Reset latches using 4 NAND gates. These are often found in high density binary SRAM cells. The same topologies can be used for ternary SRAM [321], [332]. A NAND (NMIN) ternary d-latch shown in Fig. G.11 costs 46T when made with MRCS and is not efficient. Both the NMIN and MUX d-latch design cost many more transistors/area than a 8T d-latch design based on two 2T cross-coupled STI's and two 2T transmission gates [149]. Transistor count is not always the most important metric. The 20T ternary SRAM made with CNTFET reported in [321] has 85% better performance and PDP compared to ternary 6 transistor, 2 capacitor (6T2C) DRAM also made with CNTFET.

Ternary data-flip-flop

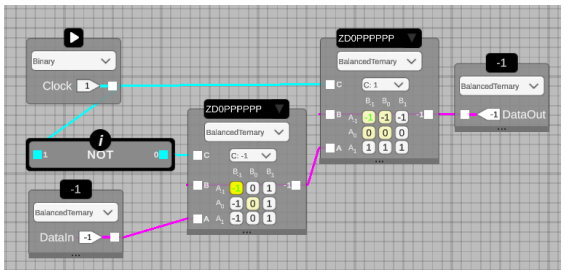


Figure G.12: 54T rising-edge master-slave configuration balanced ternary d-flip-flop

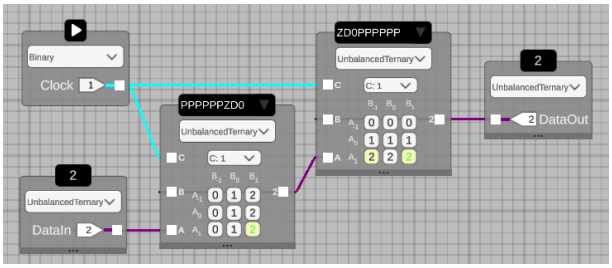


Figure G.13: 52T rising-edge master-slave configuration unbalanced ternary d-flip-flop

A pletoria of binary flip-flop designs exist[331]. While the latch is a level controlled memory element, the flip-flop is designed to be edge-controlled. Contrary to binary's bi-stable flip-flops ternary's flip-flops are tri-stable. Sometimes tri-stable flip-flops are called flip-flap-flops [332]. This term should be avoided as it is confusing and the name becomes rather grotesque with higher radixes. Ternary flip-flops can be either binary clocked or ternary clocked. This is important as in modern (synchronous) computers the clock tree network (CTN) is the always-on backbone that can consumes 30% of the CPU power budget [8]. The rising-edge master-slave configuration of a MUX based ternary flip-flop is discussed in paper F and shown in Fig. G.12. This configuration consists of two ternary d-latches with a binary inverter. The inverter can be integrated (reducing 2T) by flipping the heptavintimal index of the second latch from 0tPPPPPP**ZD0** to 0t**ZD0**PPPPPP. By reversing the latches the flip-flop becomes either a rising-edge or falling-edge flip-flop. Note that in Fig. G.12 a balanced ternary version of the D-flip-flop is shown while in Fig. G.13 an unbalanced ternary version of the d-flip-flop is shown. Both have identical implementation which is not always the case (such as the carry function).

Ternary data-flip-flop with double/quad data rate

The d-flip-flop design in Fig. G.12 is a single data rate (SDR) or single edge d-flip-flop. It only transitions with a rising-edge. If tighter timing is permissible, then double data rate (DDR) d-flip-flops or both rising and falling edge triggered d-flip-flops make sense. A novel 76T DDR ternary d-flip-flop design is shown in Fig. G.14 which uses both edges of the binary clock. The design is based on the latches 0pPPPZD0PPP and 0pZD0PPPZD0. With a ternary clock the transitions can be increased to 4 edges. Quad data rate or quad edge triggered d-flip-flops require even tighter timing. The power benefits of QDD memory versus SDD is substantial as the voltage swings are smaller in both the CDN and d-flip-flops. Kim et al. [330] show that QDD ternary flip-flops are 31% more efficient

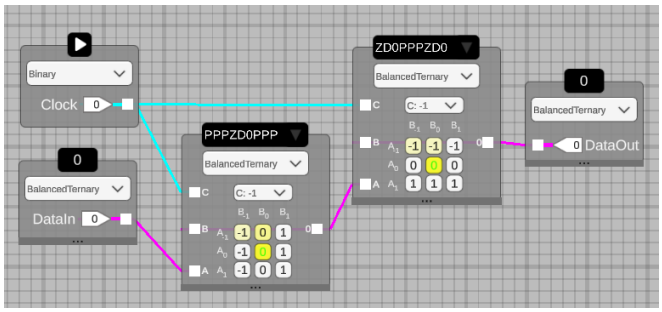


Figure G.14: 76T DDR master-slave configuration balanced ternary d-flip-flop

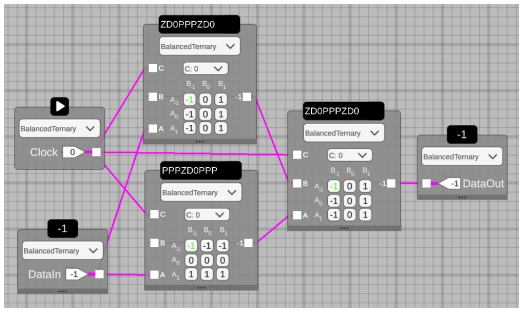


Figure G.15: 110T QDR master-slave configuration balanced ternary d-flip-flop

than SDD ternary flip-flop (only slightly higher delay) while the clock tree consumes 75% less power. A novel MUX-based 110T QDR ternary flip-flop design is shown in Fig. G.15. This design is based on the latches 0tPPPZD0PPP and 0tZD0PPPZD0 and one ternary 2:1 MUX. This MUX has the same heptavintimal index but has no feedback making it a combinatorial block. The top latch is responsible (active) for the edges -1 to 0 and 1 to 0 while the bottom latch is responsible for the edges 0 to 1 and 0 to -1. The wiring is important as the MUX responds to the active latch only during an edge. During level it "listens" to the inactive latch thus ignoring any changes.

Ternary register

The ternary register can be constructed with a mixture of binary and ternary gates. The *Write-enable* flag is a binary signal and can be made with a binary 6T AND (0tK00) or 4T NAND gate (0t22Z) when a binary clock is used. The memory element can be made with MUX-based ternary d-flip-flops shown in Fig.G.12. The *Read* flag is a binary signal while the output is ternary, thus requiring the usage of a ternary logic gate. Enabling the read flag allows the output to reflect the truth table output, else a constant zero is output. This is useful in combination with a DESELECT component (0tVP0) allowing multiple registers to be connected without using a transmission gate and bus architecture. The register design is thus a pure logic gate based design. Transmission gates however are far more efficient, costing just 2T instead of 16T for the *Read* flag. Transmission gates seem a better fit for this functionality in combination with higher radix circuits as transmission gates signals are analog in nature. It should be noted that Kim et al. [280] report that transmission gates "have worse power, speed, and noise margin than static gates for ternary". The ternary register is an example of a mixed-radix design, combining various binary and ternary signals which reduces the transistor count compared to a pure ternary implementation.

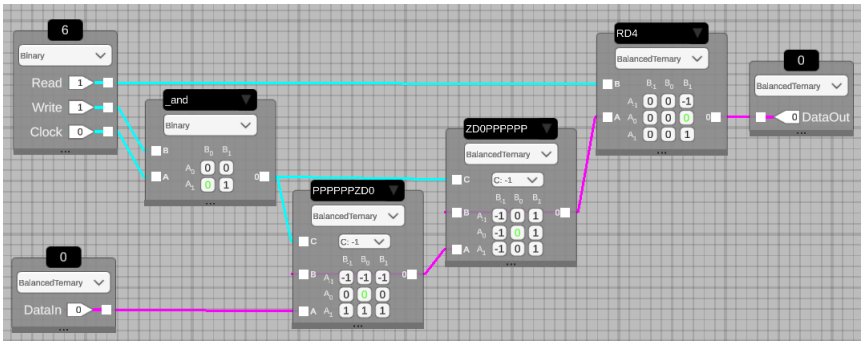


Figure G.16: 80T balanced ternary register

Ternary ROM/RAM

Registers give a blueprint to construct larger memory elements such as blocks of RAM or ROM. Typically application code and immutable data (constants) is stored in ROM while processed data is put in RAM. ROM is often a non-volatile memory array such as FLASH while processed data is stored in SRAM or DRAM and is lost after a power cycle. RAM/ROM have the same interface as registers but feature a DEMUX/MUX to select individual (blocks of) memory elements. The building blocks of both ROM and RAM are clusters of balanced ternary d-flip-flops and is shown in Fig. G.17 .

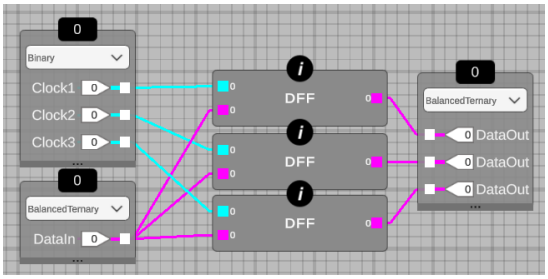


Figure G.17: RAM-3 implementation with three d-flip-flops

A 2x1 block of RAM is shown in Fig. G.18. The 2x1 RAM has 2-trit addresses meaning thus can refer to 9 unique addresses for both reading and writing. Each address refers to a single ternary d-flip-flops. By adding another 2x1 block of RAM in parallel a 2x2 RAM block can be made, thus expanding the content while keeping the address width the same. Parallel read actions are made possible by adding another MUX with two 2-trit register-source addresses (RsAddr and RsAddr2). Parallel write actions are slightly more complex as two write actions might want to update the same register with different data. In MRCS the RAM/ROM content is *uninitialized* at first and needs to be reset

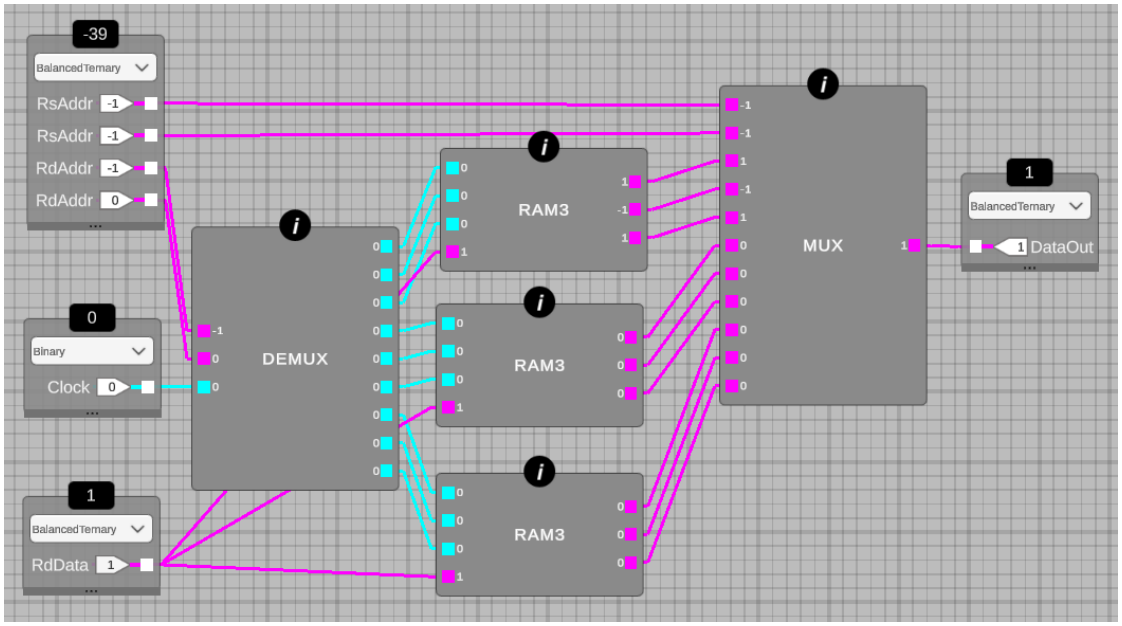


Figure G.18: Ternary ROM/RAM. Implementation of (DE)MUX are shown in Subcomponents

(for instance to $0_{\overline{3}}$, 1_3 or 0_2). This is similar to actual behavior of physical memory elements which exhibit unknown states at initialization. Initializing or programming the RAM/ROM can be accomplished manually or automated by reusing the *verify component* functionality.

Ternary full adder

The addition instruction is the cornerstone of most CPU architectures [2], [328]. It is one of the basic arithmetic operations next to subtraction, multiplication and division. These four operations are not uniformly used as they are often reduced to addition and shift operations. Subtraction is addition with one input inverted (2's complement), multiplication is repeated addition and division is repeated subtraction (complete algorithms for all three operations are slightly more complex, see [2]. Even the program counter uses an adder with a constant (such as a hardwired 1) to compute the next instruction address from the present instruction address. Statistics vary, but in [2] the most common instruction of RISCV CPU's is the ADD instruction. Optimizing the adder result in massive, system-wide performance boost. For this reason the binary adder is historically well researched and is still being improved [327], [328].

The recent large scale survey on ternary full adders (TFA) by Nemati et al. [143] show that a multitude of adder designs exist including many based on multi-threshold CNTFET. Surpisingly, 90% of the reported ternary adders in literature used unbalanced ternary

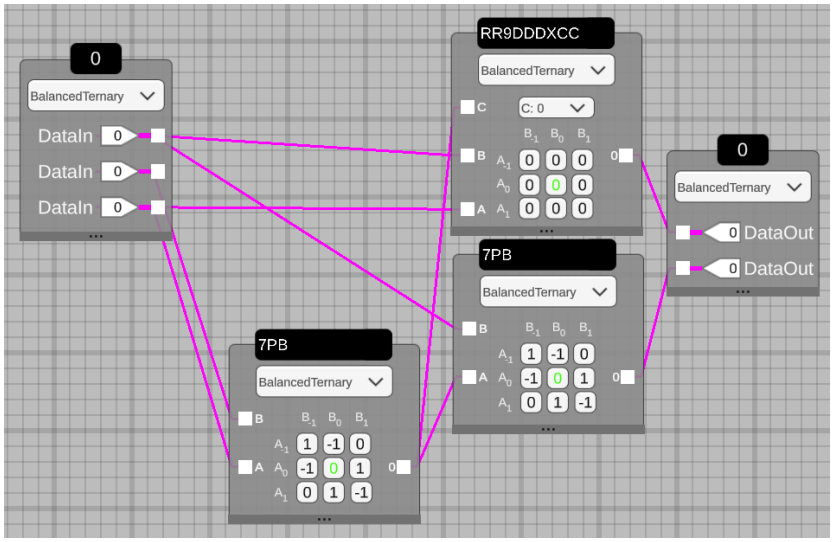


Figure G.19: 110T BTA design with SUM-based CARRY

encoding. Consider for example the ternary half adders (THA) consisting of a SUM and CONS (carry) gate. The 2-ary unbalanced SUM circuit (0tB7P) is 31T and CONS (0tC90) is 10T plus 8T for shared NTI/PTI inverters, totalling 49T. For balanced ternary SUM (0t7PB) is 32T, CONS (0tRDC) is 10T plus 8T for shared inverters, totalling 50T. With nearly identical transistor count, lower switching activity due to balanced ternary arithmetic is preferred. This advantage is discussed in more detail in Chapter 2.

The survey papers by Nemati et al. [143] concludes: "A TFA with faster operation, lower power consumption, and fewer transistors is needed to be considered a potential rival for the binary counterparts". A similar conclusion is found in a survey by Etiemble [225]. Although full adder designs certainly exist with competitive transistor count [202], they have not been demonstrated to be competitive in direct comparison according to PPAC metrics. A survey on binary full adders usings CNTFET [327] show that 14T is possible with transmission gate logic (TG) compared to the well known 28T design using static logic. As mentioned in Chapter 2, for fair comparison to ternary similar logic styles, functionality, resolution, input pattern, etc should be used. Only then will PPAC comparison makes sense. This is unfortunately rarely done in survey papers. For example, the balanced ternary SUM gate made with MRCS is 32T which is 4x larger than the 8T SUM (CMOS XOR) gate in binary. However, the difference in transistor count becomes small when compensating for identical features. The binary 2-bit signed addition circuit requires 2x 2-bit input to cover the range of 2x1 trit input, additional circuitry, wiring, and a add/sub functionality pin. A straightforward implementation would require 3 SUM gates and a AND (carry) for a total of 30T. Even when compensating for theoretical identical resolution as the 2 bit signed binary adder can compute 3 more states (-3,-4 and +2), the difference in transistor count is competitive.

Kim et al. [155] shows a 118T balanced ternary full adder which is replicated in [282]. Just like in a logical level optimized 28T binary full adder design, a TFA with carry that depend on SUM output can be constructed (see Fig. G.19). This design cost 110T and is thus 8T smaller. The SUM components are made with 0t7PB and carry with 0tRR99DDDXCC. Unclear is if this design has been reported earlier.

Ternary asynchronous counter

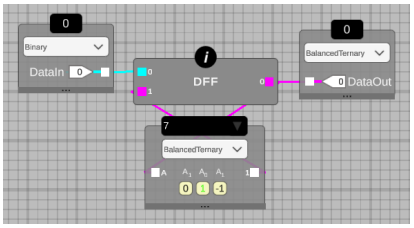


Figure G.20: Balanced ternary ripple counter

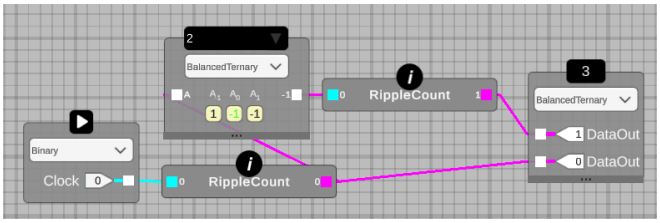


Figure G.21: 2-trit balanced ternary ripple counter

With a memory and adder block another common building block can be constructed: the counter. The counter is used for many types of functionality in digital electronics such as counting clock pulses or as part of a finite state machine(FSM). When counting program instructions it fulfills a role as program counter (PC). Two classes of counters exist, asynchronous or ripple counters and synchronous or parallel counters [329]. Asynchronous balanced ternary counters can be constructed with a d-flip-flop such as Fig. G.12 and INCREMENT (0t7). Contrary to binary counters using JK-flip-flops ternary counters don't toggle between two states such that the output is reusable as a binary clock signal. By adding a NTI (0t2) to detect the overflow, multiple ternary ripple counters can be stacked (see Fig. G.20 and Fig. G.21).

Ternary synchronous program counter

In [260] binary and balanced ternary synchronous counters are shown which were made with MRCS. The various designs are verified with HSPICE simulations. These counters are classified as up/down counters and can be loaded to a specific count value. This makes them usable as a program counter (PC). The PC normally increments linearly but some instructions jump to other instructions at a different address (count value), creating the data-driven flows needed for non-trivial computations. The design of the PC consists of 3 components: ADDER (0t7PB) to count up/down or not count, MUX to choose between the adder or load input and a d-flip-flop (see Fig. G.12) to store the input. In Fig. G.22 a single balanced ternary counter is shown. A CONSENSUS (0tRDC) gate is used between the counters to detect the positive or negative overflow, depending on the direction. This

design is another example of a mixed-radix design where some signals are binary and some are balanced ternary resulting in a smaller transistor count than binary encoded ternary or pure ternary.

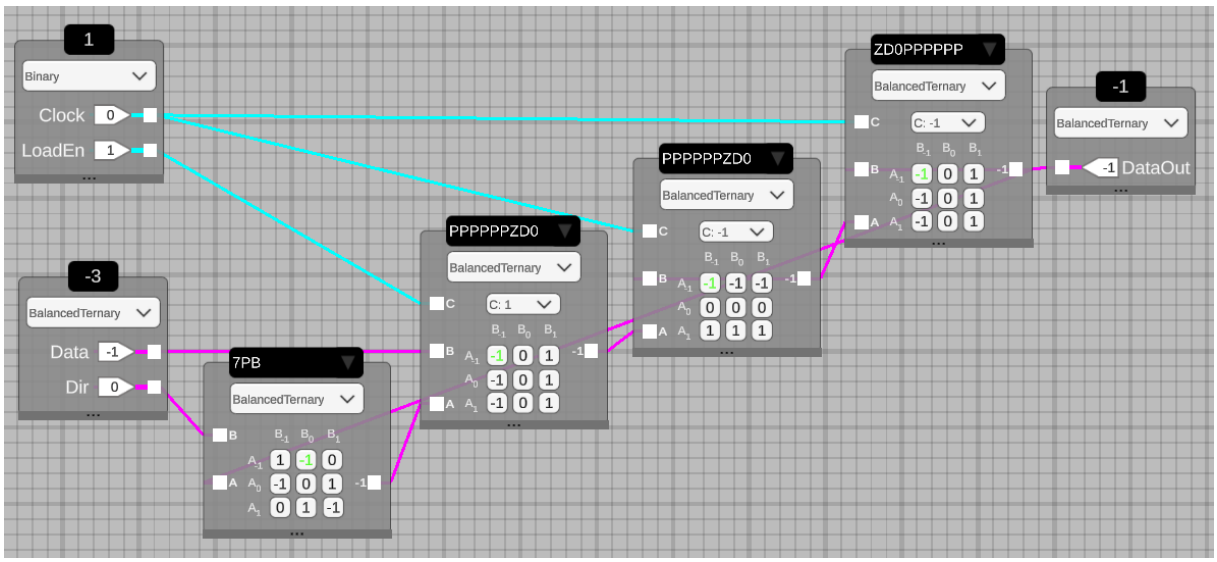


Figure G.22: 1-trit synchronous balanced ternary tri-directional loadable program counter

Both the 6-bit binary counter in [260] and 4-trit balanced ternary counter in Fig. G.23 have been submitted for tape-out [266], [334]. These counters are designed to have identical features. With CNTFETs the 6-bit binary counter needs 542 transistors while the 4-trit ternary needs 8 less, 534 transistors. Less transistors are needed for ternary while the resolution of 4-trits being 81 is higher than the resolution of 6-bits (= 64). Note that no radix economy compensation is applied as the resolution is more or less comparable. A slight compensation to have identical resolution would benefit ternary even more but is purely theoretical since it would requires non-discrete devices. The small transistor/die area advantage for ternary increases with higher trit comparisons since the designs are compoundable.

Although in this comparison ternary has a lower transistor count, the average power consumption and delay and thus the PDP is much worse compared to binary. The binary design used 18 μW for a basic testbench (see [260]) while the ternary design used 137 μW. This big difference is the result of the synthesis method discussed earlier. The $\frac{VDD}{2}$ state is made by voltage division and consumes a disproportional amount of current. In [155] Kim et al. show that this state consumes 266.36 nW while logical -1 and logical 1 consume 0.15 nW and 0.31 nW respectively. In the same work they propose to use the body effect to reduce the static power consumption of the middle voltage. They improved the power consumption from 266.36 to 3.57μW. New simulations are needed to ascertain if the PDP is below the binary design with this improvement.

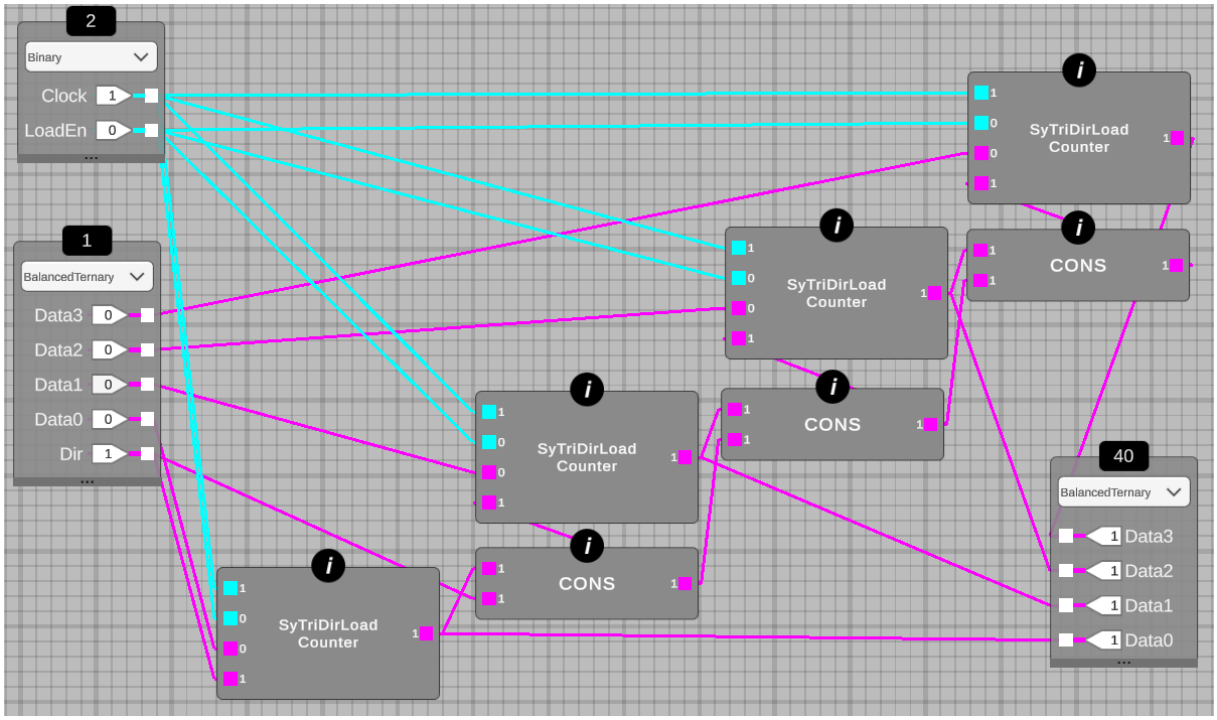


Figure G.23: 4-trit synchronous balanced ternary tri-directional loadable program counter

G.16 Subcomponents

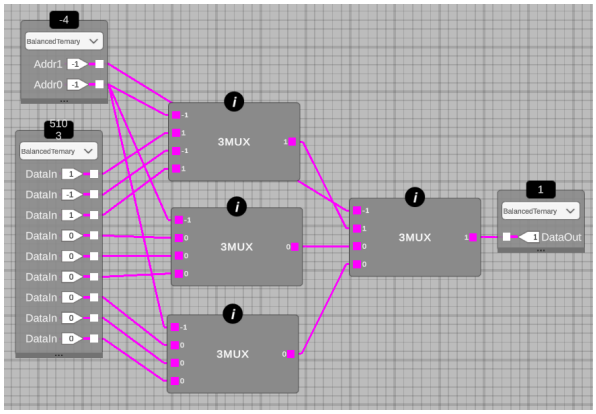


Figure G.24: MUX level 2 implementation, part of Fig. G.18

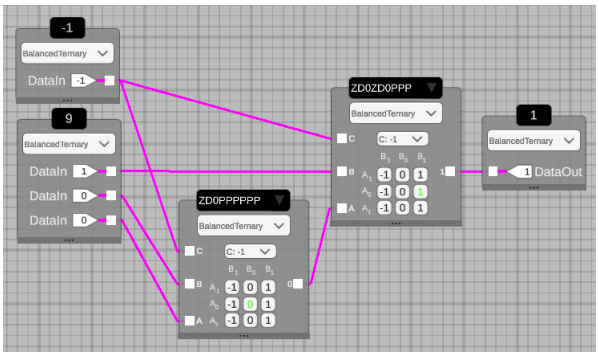


Figure G.25: MUX level 1 implementation, part of Fig. G.18

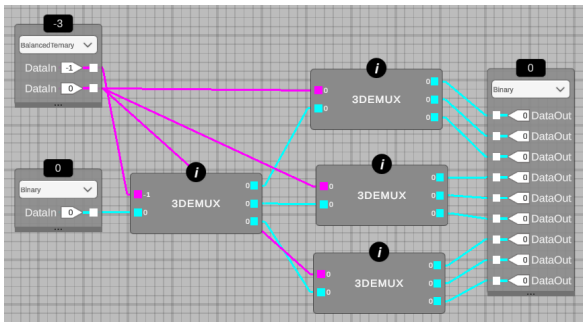


Figure G.26: DEMUX level 2 implementation, part of Fig. G.18

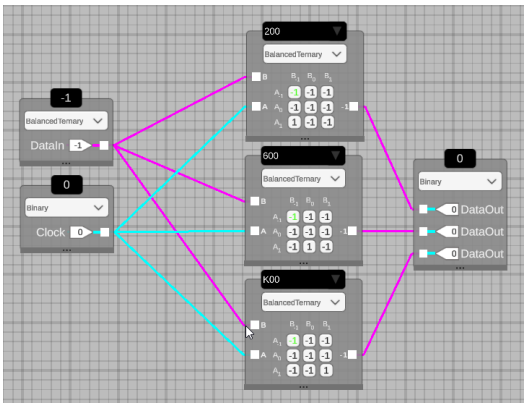


Figure G.27: DEMUX level 1 implementation, part of Fig. G.18

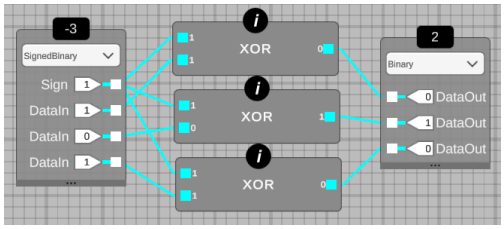


Figure G.28: XOR-3 implementation, part of Fig. 4.9. The XOR gate is made with binary $0t20K$.

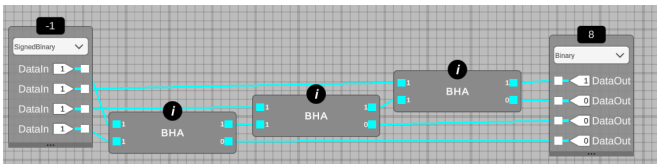


Figure G.29: BHA-3 implementation part of Fig. 4.9. The BHA gate is made with binary $0t20K$ for the sum and binary $0tK00$ for the carry.

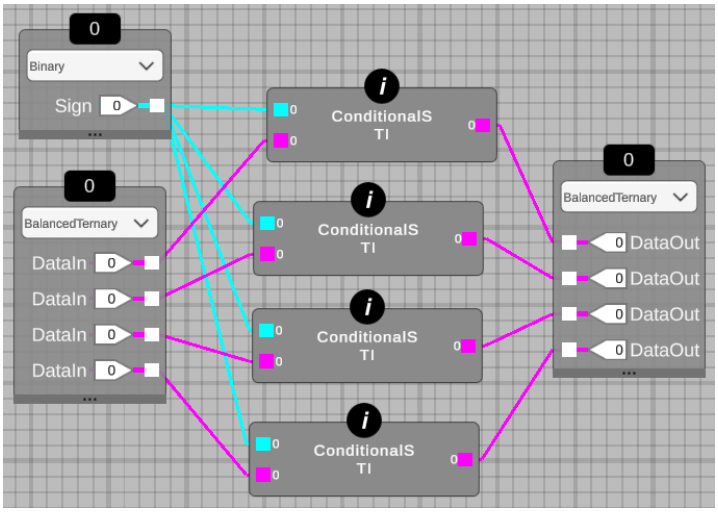


Figure G.30: Conditional-STI-3 implementation part of Fig. 4.9. The Conditional-STI gate is made with 0*t5DP*.

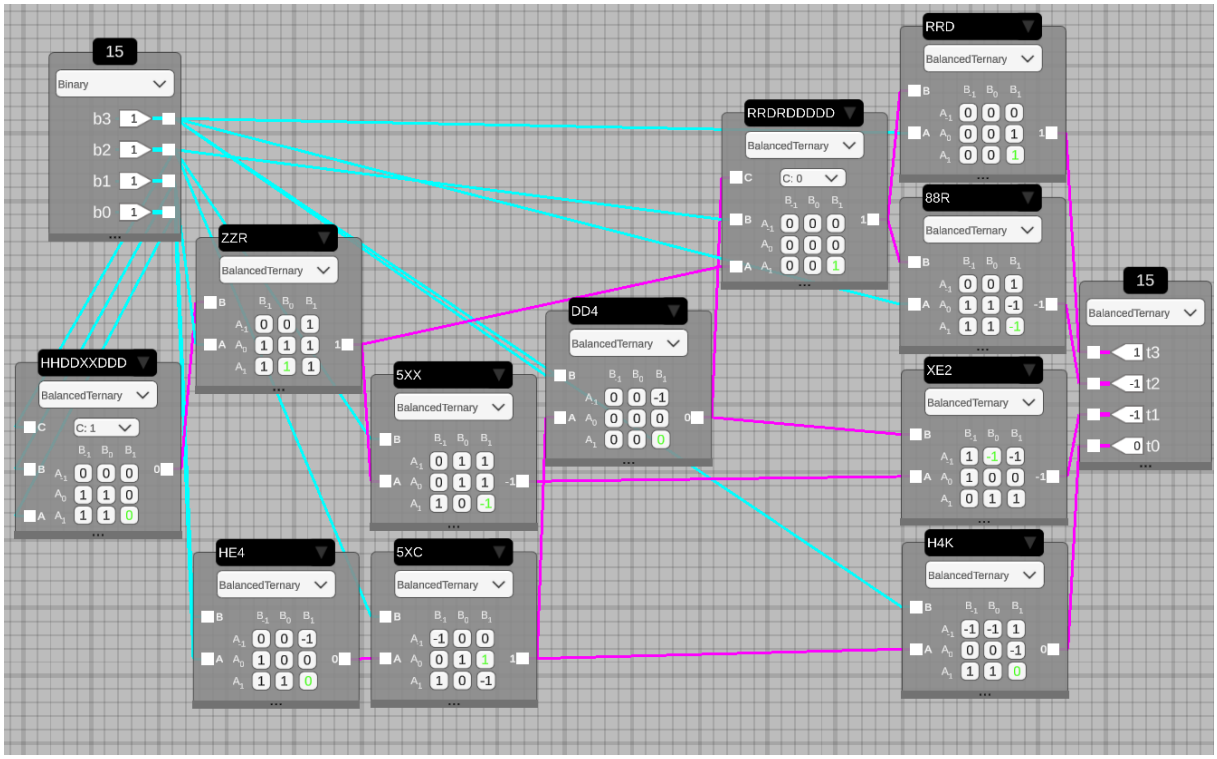


Figure G.31: The 170T 4-bit unsigned binary to 4-trit balanced ternary radix converter in paper E and part of Fig. 4.9

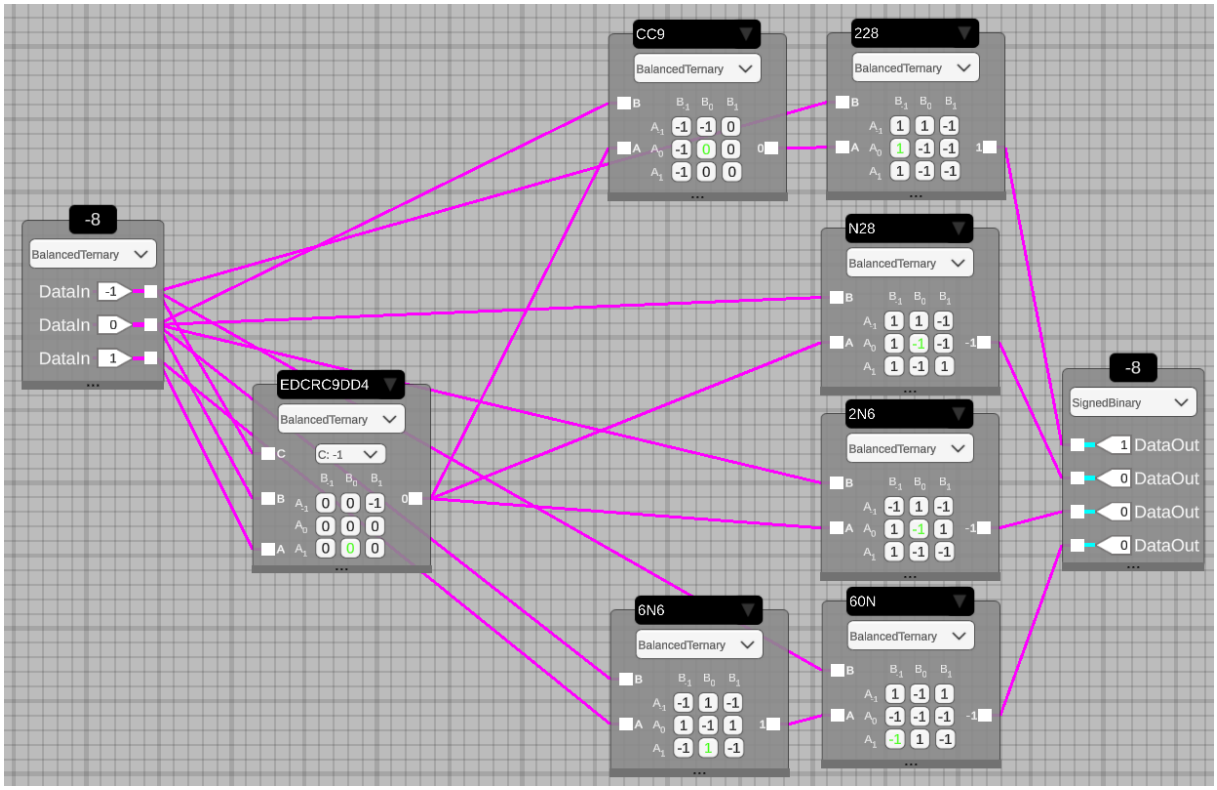


Figure G.32: A novel 139T 3-trit balanced ternary to 4-bit 2's complement signed binary radix converter

G.17 Online radix conversion tool

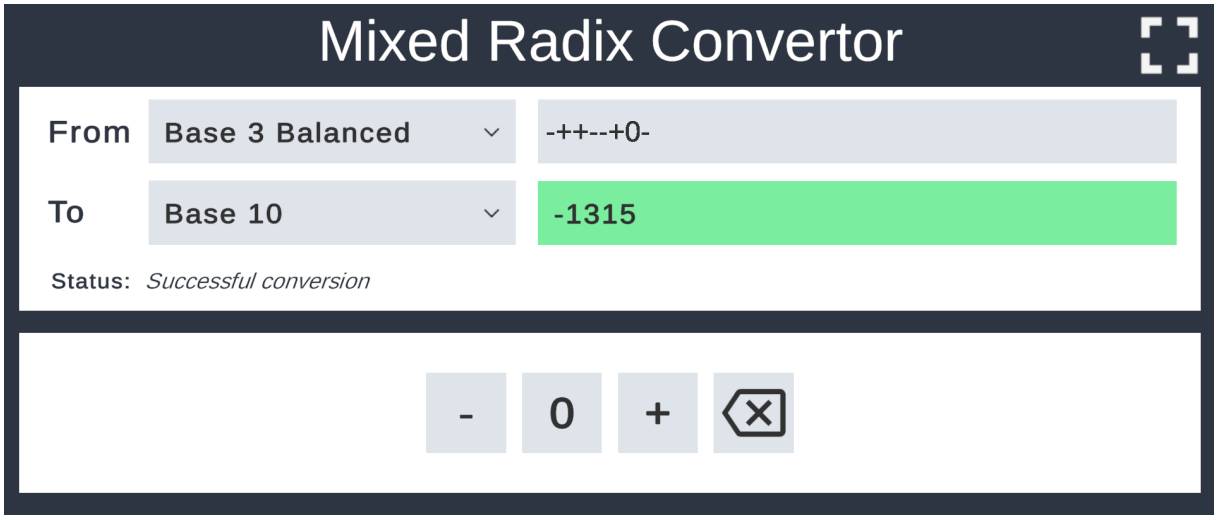


Figure G.33: User interface of the online radix converter tool.

During the development of MRCS and the radix conversion circuits many conversions were needed. For ternary to binary and the inverse a very fast software converter can be found [312]. A more general approach between all possible radixes is found in [295]. No all-in-one and browser-based radix converter could be found that was able to convert between radix-2, radix-3 and radix-10 in both signed and unsigned encoding. This lead to the development of the open source radix converter tool [333]. It was made with Unity WebGL and is hosted on `TernaryResearch.com/mixed-radix-converter/`. It is planned to add heptavintimal, octal, hexadecimal and base-64 conversion in a future version.

H

TNNN 2023: Ternary VLSI with CMOS

Ternary VLSI with CMOS using MRCS

Steven Bos*
Dept. of Science and Industry Systems
University of South-Eastern Norway
Kongsberg, Norway
steven.bos@usn.no

Henning Gundersen
Dept. of Science and Industry Systems
University of South-Eastern Norway
Kongsberg, Norway
henning.gundersen@usn.no

***Abstract*—Moore's exponential scaling law became unsustainable after Dennard scaling stopped in 2006. This has not stopped the industry to continue transistor scaling, leading to exponentially increasing inefficiency and complexity: the power wall, the memory wall and the EDA wall. Radix-3 or ternary is a higher radix than binary and allows information to be more compressed. A higher radix improves utilization of the inherently analog interconnect infrastructure. The design and verification of ternary silicon and especially ternary Very Large Scale Integration (VLSI) has been notoriously hard due to the lack of Electronic Design Automation (EDA) tools and flows. In this one page paper we discuss our latest version of Mixed Radix Circuit Synthesizer (MRCS) that can automatically translate ternary truth tables to binary encoded ternary (BET) high level register transfer level (RTL) code in verilog. The work enables rapid digital design and verification of both ternary ASIC using industry standard CMOS as well as ternary FPGA using off-the-shelf hardware such as Basys 3 with Vivado software. The Openlane flow with various open PDK's is used to convert RTL to GDS-II and has been used to tape-out designs at Skywater 130nm foundry using the affordable TinyTapeout service.**

***Index Terms*—ternary computing, mixed-radix chip design, ternary EDA**

I. Introduction

The Shannon limit in eq. 1 shows mathematically what the highest theoretical information rate (capacity C) of error-free communication through a noisy channel such as an interconnect is.

$$C = B \log_2 \left(1 + \frac{Signal}{Noise}\right) \tag{1}$$

The theorem also shows that more signal levels ie. a higher radix is feasible by adjusting bandwidth (B), power and/or noise. This might seem costly but the relentless focus on computation rather than communication results in 1300x more energy consumption and 100x more clock cycles to communicate data when it is not in cache compared to a typical 32-byte ALU operation [1]. The complexity of binary arithmetic is often taken for granted but a higher radix can make design and verification much more intuitive. For example balanced ternary notation using -1, 0, 1 symbols does not require 2's complement for negative numbers. Unfortunately, higher radix EDA tooling and flows for chip design are missing. A focus on efficiency is needed for future scaling.

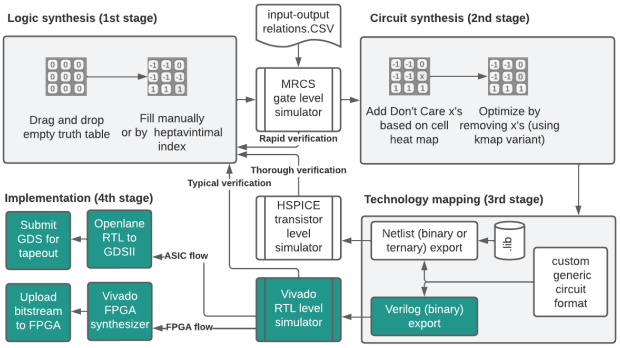


Fig. 1. **Mixed-radix chip design and verification flows in MRCS. New flows highlighted in green.**

II. Ternary VLSI with MRCS

We first introduced MRCS at ISCAS in 2022 [2] and gave a talk about it at the 1^{st} TNNN. In the previous version we only enabled SPICE simulation of ternary signals using multi-threshold CNTFET as efficient transistors for ternary signals are not yet available. This new version enables automatic conversion of ternary truth tables to binary encoded truth tables and generates the resulting hierarchical netlist in verilog (see Fig. 1. Ternary logic can thus be physically emulated using binary CMOS technology while native transistors are being developed. Emulation is not as efficient as the underlying CMOS is binary valued. We tested the new MRCS workflow with complex multi-trit designs such as ALU's in HSPICE. A 2-trit multiplier and adder/subtract ALU design has been submitted for tape-out using Openlane, the sky130 OpenPDK and Tinytapeout service. The open source design has also been successfully deployed on a Basys 3 (Artix-7) FPGA [3].

References

[1] P. Ruch, T. Brunschwiler, W. Escher, S. Paredes, and B. Michel, "Toward five-dimensional scaling: How density improves efficiency in future computers," *IBM Journal of Research and Development*, vol. 55, no. 5, pp. 15:1–15:13, 2011.

[2] S. Bos, H. N. Risto, and H. Gundersen, "Beyond cmos: Ternary and mixed radix cntfet circuit design, simulation and verification," in *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2022, pp. 80–85.

[3] S. Bos. (2023, May) Tinytapeout: Balanced ternary calculator. [Online]. Available: https://github.com/aiunderstand/tt03-balanced-ternary-calculator

Ternary VLSI with CMOS

The next chip design paradigm is 3?

Steven Bos and Henning Gundersen
steven.bos@usn.no *henning.gundersen@usn.no*

Takeaway: With our open source tool Mixed Radix Circuit Synthesizer (MRCS) you can design, verify and make ternary logic chips with industry standard CMOS

USN ^{3}research

2nd TNNN Conference **Vestfold 2023**

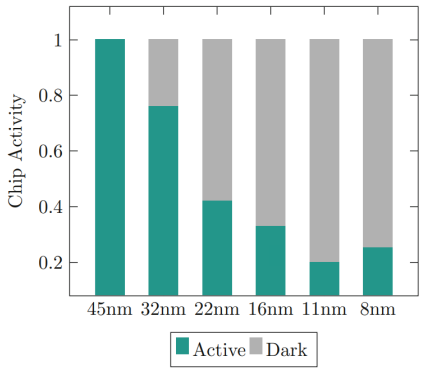


Fig 1. As chips become smaller -> less efficient. This effect is known as Dark Silicon and started after Dennard scaling stopped in 2005 [1].

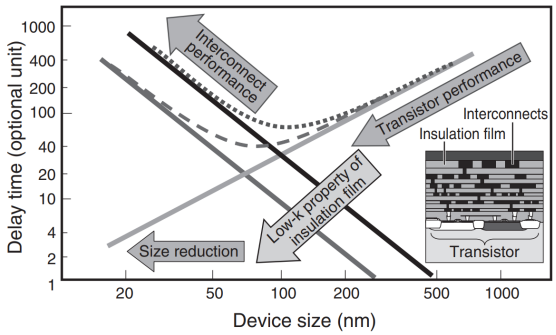


Fig 2. Chips are power constrained. Moore's Law pushes to double performance/watt every 2 year. Transistor scale well, but interconnects do not due to the RC effect [2].

"The energy required for off-chip communication was 260 times greater than that for arithmethic operations at 130 nm, whereas this ratio increased to 1300 for the 45 nm node" [3]

Interconnect bottleneck as a consequence of **device-centric scaling**

Shannon limit: increase effiency of interconnect

$$C = B \log_{\cancel{2}\ 3,4,..} \left(1 + \frac{Signal}{Noise}\right)$$

Fig 4. A ternary logic chip generated with MRCS and submitted for tapeout in May 2023. This ASIC contains a 4-trit tri-directional loadable program counter[5].

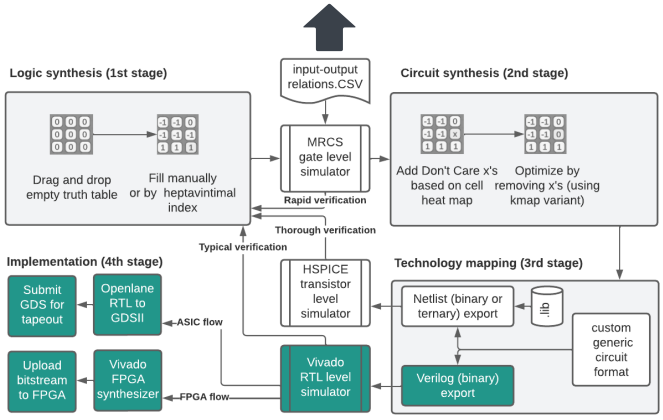


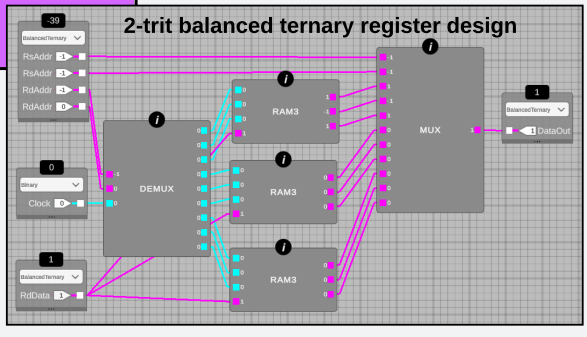
Fig 3. The three workflows of MRCS. Design a ternary state machine with sequential and combinatorial logic using binary and ternary truth tables. Verify correct gate-level behavior using the internal simulator. Automatically optimize the circuit for transistor count and generate a CNTFET netlist and a CMOS verilog file. **Flow 1**) Use netlist file for mixed-signal verification with HSPICE. **2)** Use verilog file to generate a GDS file with OpenLane. Tapeout with TinyTapeout for ~$100! **3)** Use verilog file to upload bitstream to FPGA. **Read more in [4]**

Strategies to improve efficiency

Strategy	Frequency	Signal/Noise Ratio (noise margin)	Effect	Action
1	F↓	S/N↑	Radix↑	Reduce frequency to improve SnR
2	F =	N↓	Radix↑	Reduce noise to improve SnR
3	F =	S↑	Radix↑	Increase Vdd to improve SnR
4	F =	S↓	Radix↑	Reduce Vdd and improve device noise tolerances
5	F =	S/N =	Radix↑	Improve device noise tolerances

Only binary digital Electronic Design Automation (EDA) tools exist

New! Mixed Radix Circuit Synthesizer. An open source EDA tool to design and verify binary, ternary and mixed radix circuits.



Literature (DOI links)
[1] 10.1007/978-3-319-31596-6_1
[2] 10.1016/j.mee.2014.10.019
[3] 10.1147/JRD.2011.2165677
[4] 10.1109/ISCAS48785.2022.9937259
[5] 10.5281/zenodo.3557410

Did you know?
State-of-the-art SSD's, videocards network cards and other peripheral cards increasingly switch to a higher radix for more bandwidth?

Benefits despite binary devices?
Higher order logic needs fewer carry switching thus saving power and increasing performance

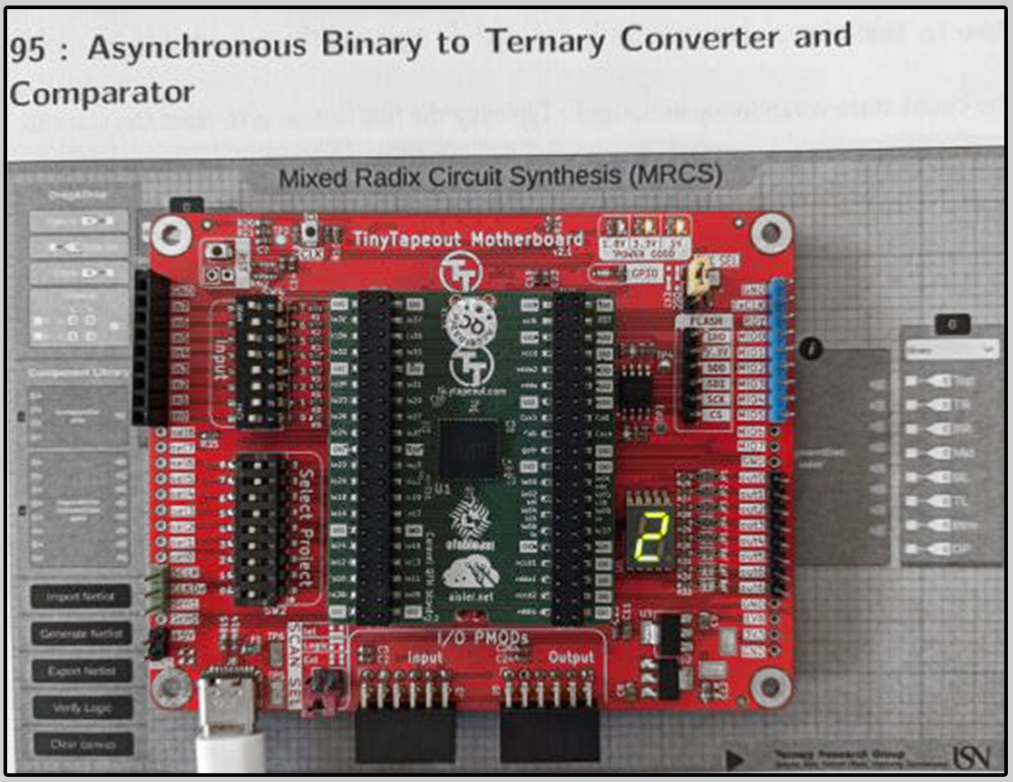
Why 3?
3 is the closest integer to the optimum *e*
(Radix economy theorem)

I

Tape-outs

Hardware verification

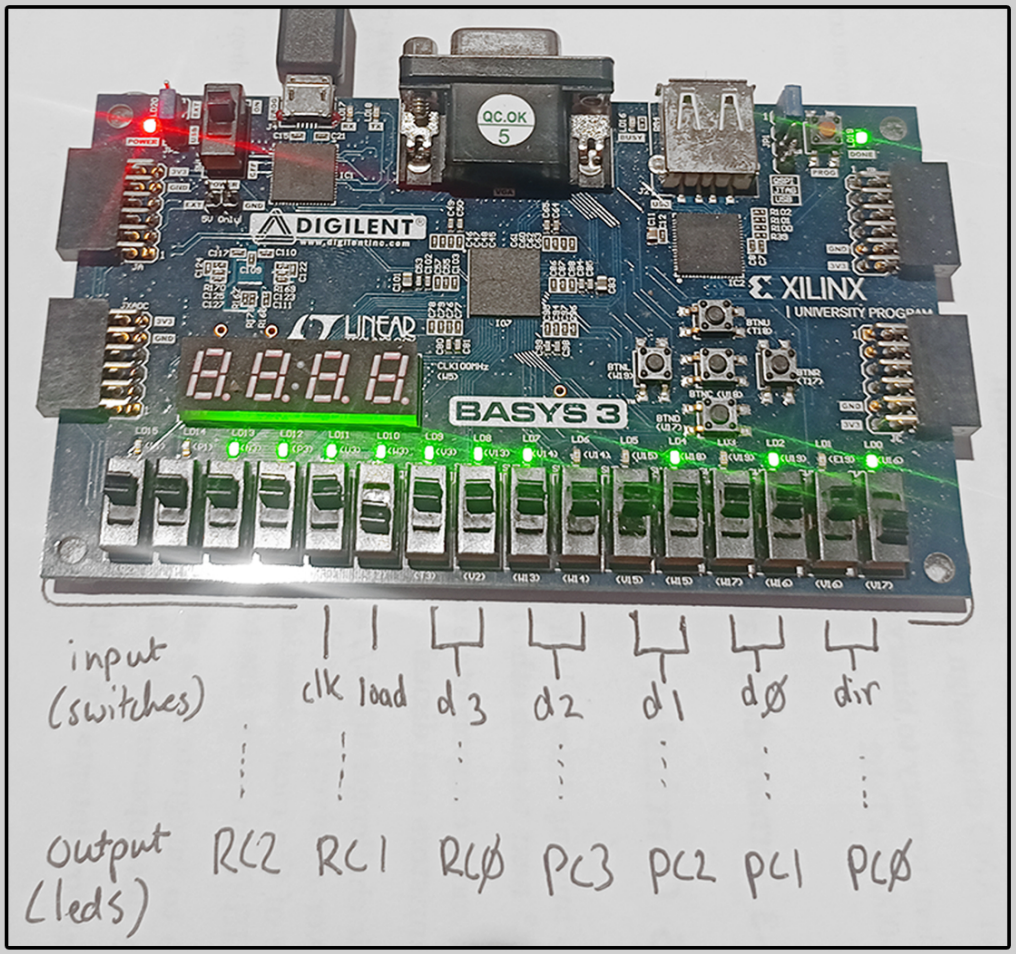
(Top) TinyTapeout 2 ASIC **(Bottom)** Digilent Basys-3 FPGA



binary to ternary converter & comparator

github.com/aiunderstand/tt02-async-binary-ternary-convert-compare

Correct unary coded ternary to decimal conversion on 7 segment display (Image: Matt Venn)

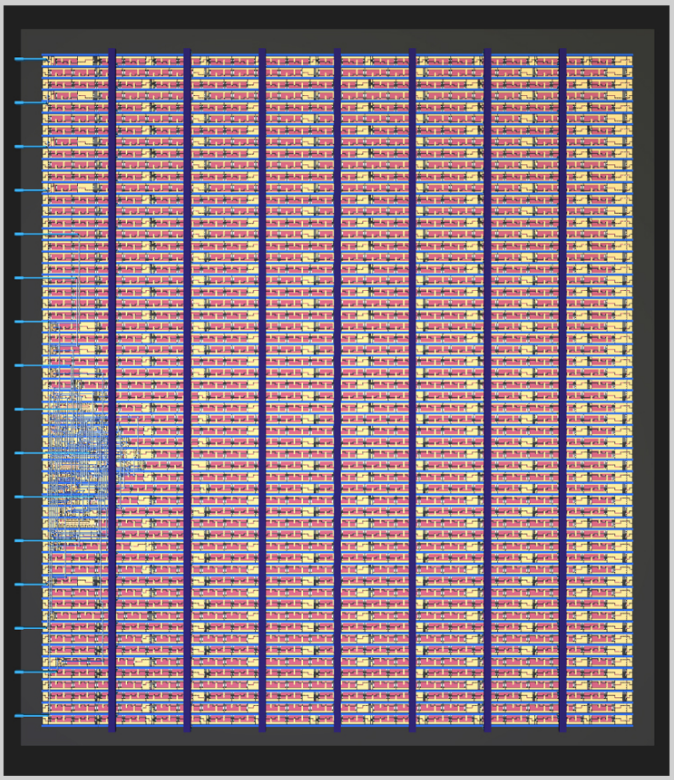


4-trit tri-directional loadable program counter & radix converter

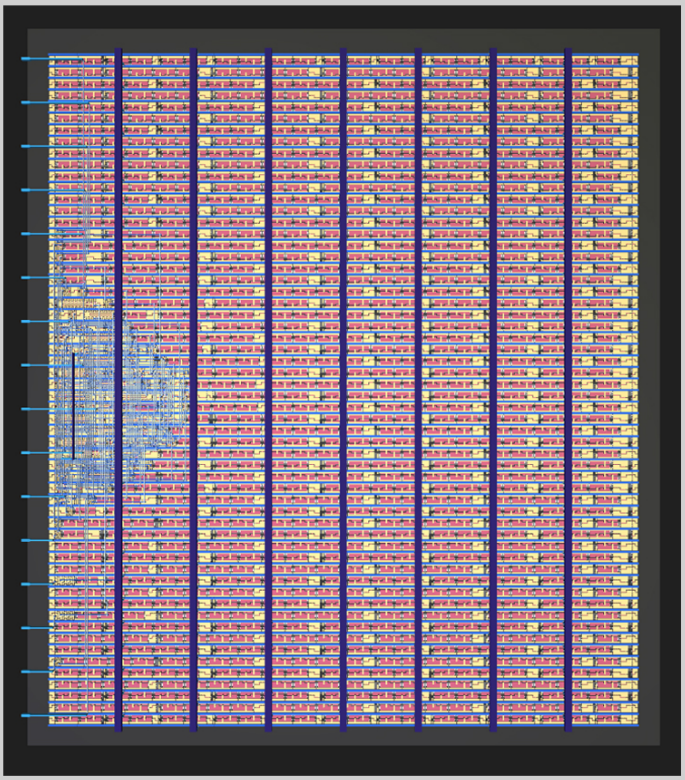
https://github.com/aiunderstand/tt03p5-4-trit-balanced-ternary-counter-bt_signb_bt-radix-convertor

Demonstrating balanced ternary count down with current value being 14 on LED's (+---)

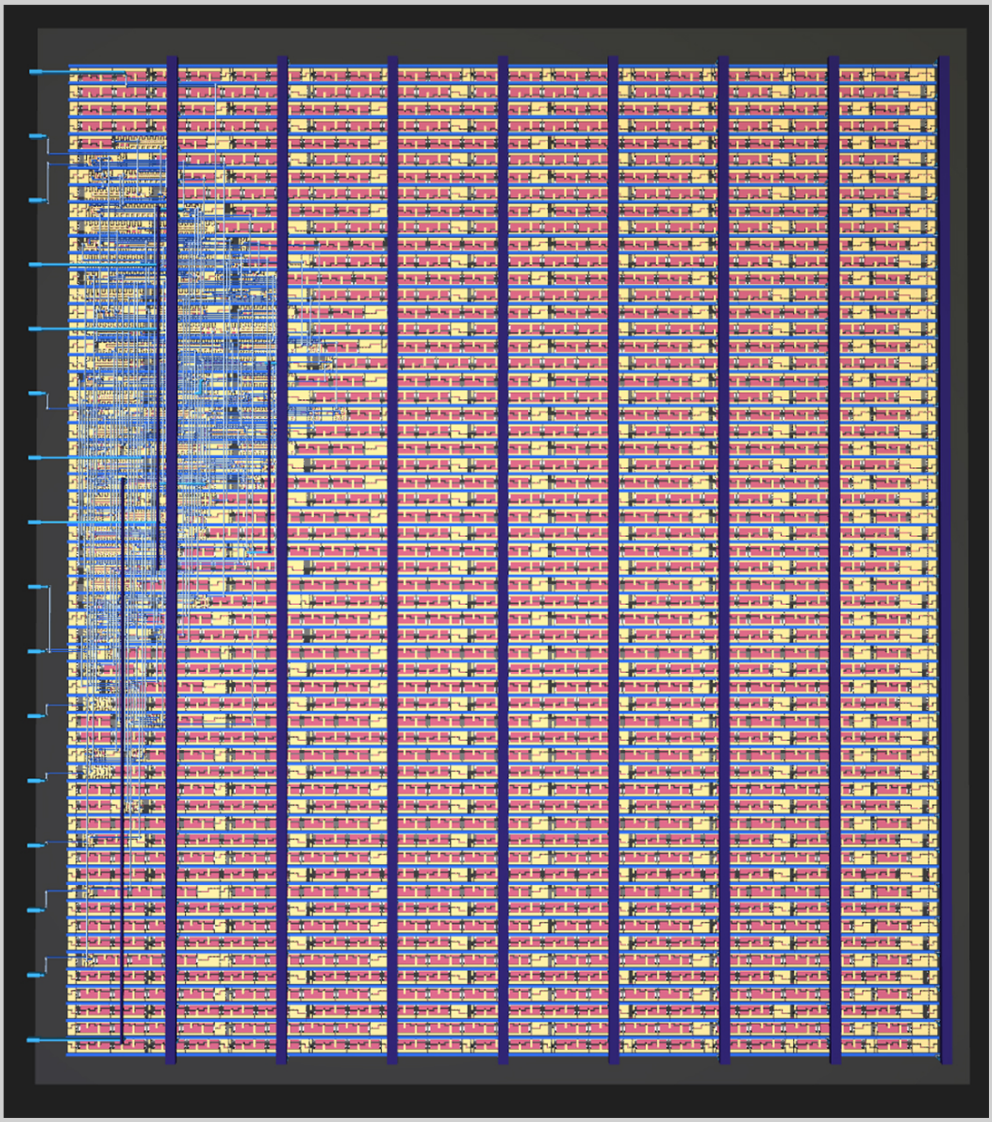
Skywater 130nm Tapeouts



4-bit tri-directional loadable program counter
github.com/aiunderstand/tt02-4bit-tristate-loadable-counter

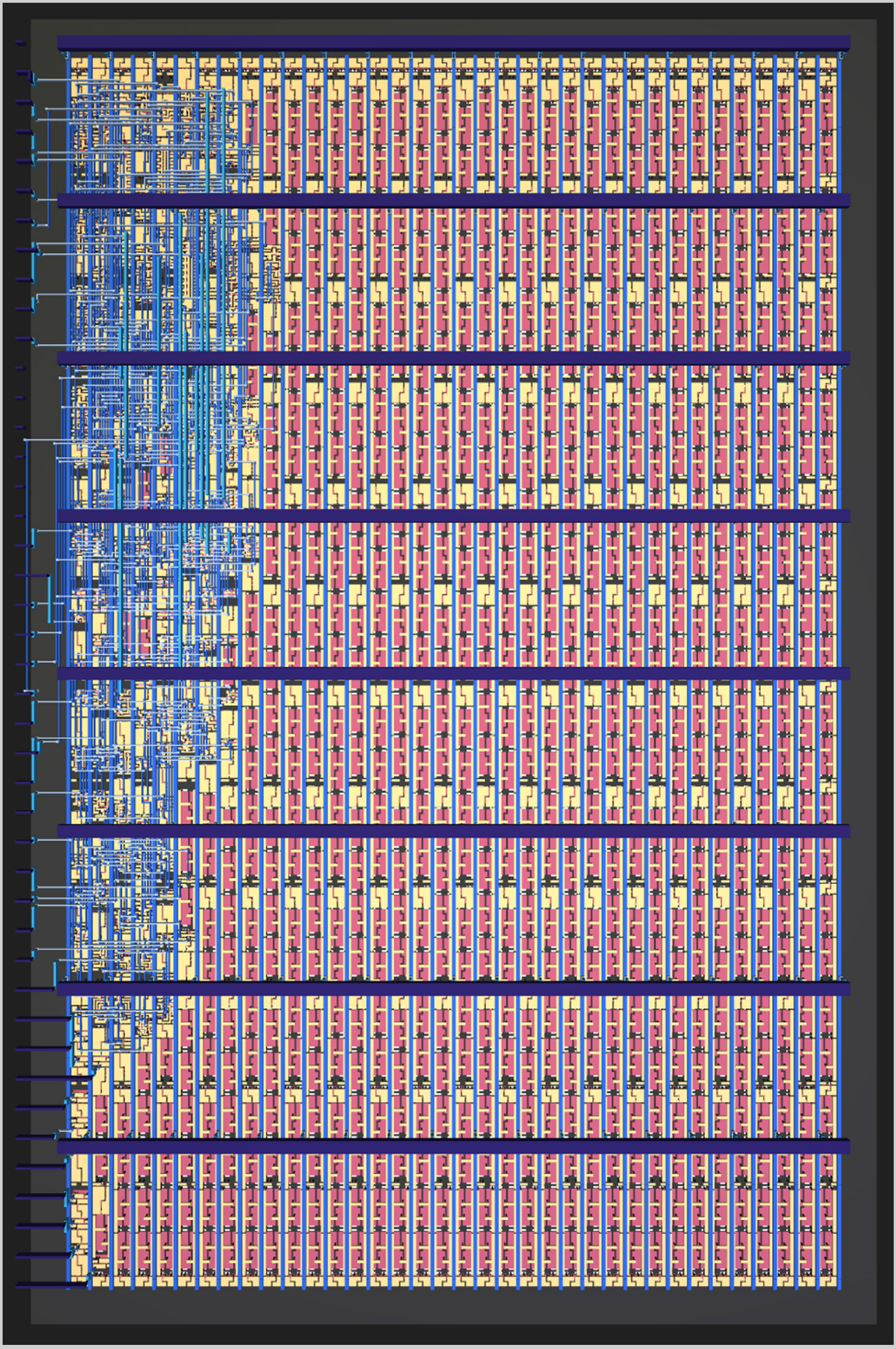


binary to ternary converter & comparator
github.com/aiunderstand/tt02-async-binary-ternary-convert-compare



binary encoded balanced ternary calculator
github.com/aiunderstand/tt03-balanced-ternary-calculator

Skywater 130nm Tapeouts



4-trit tri-directional loadable program counter & radix converter

github.com/aiunderstand/tt03p5-4-trit-balanced-ternary-counter-bt_signb_bt-radix-convertor

Beyond 0 and 1: A mixed radix design and verification workflow for modern ternary computers
Steven Bos

Doctoral dissertations at the University of South-Eastern Norway no. 189

ISBN 978-82-7206-854-6 (print)
ISBN 978-82-7206-855-3 (online)

usn.no