HARDWARE IMPLEMENTATION OF AUTONOMOUS PROBABILISTIC COMPUTERS

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Ahmed Zeeshan Pervaiz

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

December 2019

Purdue University

West Lafayette, Indiana

THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF DISSERTATION APPROVAL

Dr. Supriyo Datta, Chair

School of Electrical and Computer Engineering

Dr. Joerg Appenzeller

School of Electrical and Computer Engineering

Dr. Zhihong Chen

School of Electrical and Computer Engineering

Dr. Sunil A. Bhave

School of Electrical and Computer Engineering

Approved by:

Dr. Dimitrios Peroulis

Head of Electrical and Computer Engineering

To my parents

ACKNOWLEDGMENTS

The first time I met Professor Datta was in his class “Fundamental of nanoelectronics”. I still remember walking out amazed, as to how little I knew about Ohms law. It would be the first of great many times, to be amazed by his approach to science. Professor Datta taught me how to learn, how to do things others would find worth learning. Under his tutelage I have grown immensely both as a researcher and as a person. It has been a privilege to have had this opportunity.

This journey would not have been possible without the love and affection of my parents. My father, a sailor whose only voyage in life has been to help find happier shores for his children and my mother whose only happiness is to have her children in front of her. They have both been so incredibly patient with me. Whatever good in me comes from them and this life and all its joys would not have been possible without them.

I have been blessed with incredible friends and colleagues at Purdue. A very special thanks to Kerem Yunus Camsari for his kindness and patience. The countless hours of his time helped sculpt a great many ideas that have shaped this thesis and even greater many perspectives that have molded me as an individual. For that I am truly indebted to him. I would like to thank Brian Sutton for being both a friend and teacher and Lakshmi Anirudh Ghantasala for helping me with my work. The long evenings spent in the lab with Shehrin Sayed motivated me to do more. The wit of Orchi Hassan and Rafatul Faria brought me joy a great many times. I would also like to thank Shuvro Chowdhury, Jan Kaiser, Samiran Ganguly and Professor Ernesto E. Marinero for many thoughtful and engaging conversations. A significant part of this thesis is in collaboration with William A. Borders, Professor Shunsuke Fukami and Professor Hideo Ohno from Tohoku university. I would like to thank them for the opportunity to collaborate with them. It has been a pleasure working with such experts.

Lastly, I would like to thank my wife, my rock, Rehma. Her love and affection are the fuse that drive me forward, and her support has given me a new lease on life itself.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

ABSTRACT

Pervaiz, A. Z. Ph.D., Purdue University, December 2019. Hardware implementation of autonomous probabilistic computers. Major Professor: Supriyo Datta.

Conventional digital computers are built using stable deterministic units known as “bits”. These conventional computers have greatly evolved into sophisticated machines, however there are many classes of problems such as optimization, sampling and machine learning that still cannot be addressed efficiently with conventional computing. Quantum computing, which uses q-bits, that are in a delicate superposition of 0 and 1, is expected to perform some of these tasks efficiently. However, decoherence, requirements for cryogenic operation and limited many-body interactions pose significant challenges to scaled quantum computers. Probabilistic computing is another unconventional computing paradigm which introduces the concept of a probabilistic bit or “p-bit”; a robust classical entity fluctuating between 0 and 1 and can be interconnected electrically. The primary contribution of this thesis is the first experimental proof-of-concept demonstration of p-bits built by slight modifications to the magnetoresistive random-access memory (MRAM) operating at room temperature. These p-bits are connected to form a clock-less autonomous probabilistic computer. We first set the stage, by demonstrating a high-level emulation of p-bits which establishes important rules of operation for autonomous p-computers. The experimental demonstration is then followed by a low-level emulation of MRAM based p-bits which will allow further study of device characteristics and parameter variations for proper operation of p-computers. We lastly demonstrate an FPGA based scalable synchronous probabilistic computer which uses almost 450 digital p-bits to demonstrate large p-circuits.

1. INTRODUCTION

1.1 Probabilistic computing

Conventional digital computers are largely built from stable deterministic units know as "bits" using the standard MOS (metal-oxide-semiconductor) transistors. These computers have greatly evolved over the course of last few decades, but there are many classes of computational problems such as optimization, sampling and machine learning that cannot be addressed efficiently with conventional computers.

Richard Feynman in his seminal paper titled "Simulating physics with computers" [1] elaborated on computers which simulate the many quantum mechanical phenomenon in nature. This work is greatly credited for pioneering the field of quantum computing; a computing paradigm which uses quantum mechanical objects called "q-bits" which are in a delicate superposition of 0 and 1. While advances have been made in the field of quantum computing, decoherence, requirement for cryogenic operation and limited many-body interactions still pose significant challenges. As a prelude to quantum computers, Feynman hinted *" ⋯ but the other way to simulate a probabilistic nature, which I'll call **R** for the moment, might still be to simulate a probabilistic nature by a computer **C** which itself is probabilistic, ⋯ "*

Probabilistic computing is an unconventional computing paradigm which introduces probabilistic bits or "p-bits". These p-bits are classical three-terminal units whose telegraphic output m_i is continuously fluctuating between two states 0 and 1. This output can be tuned using an analog input I_i at the input terminal. In mathematical terms,

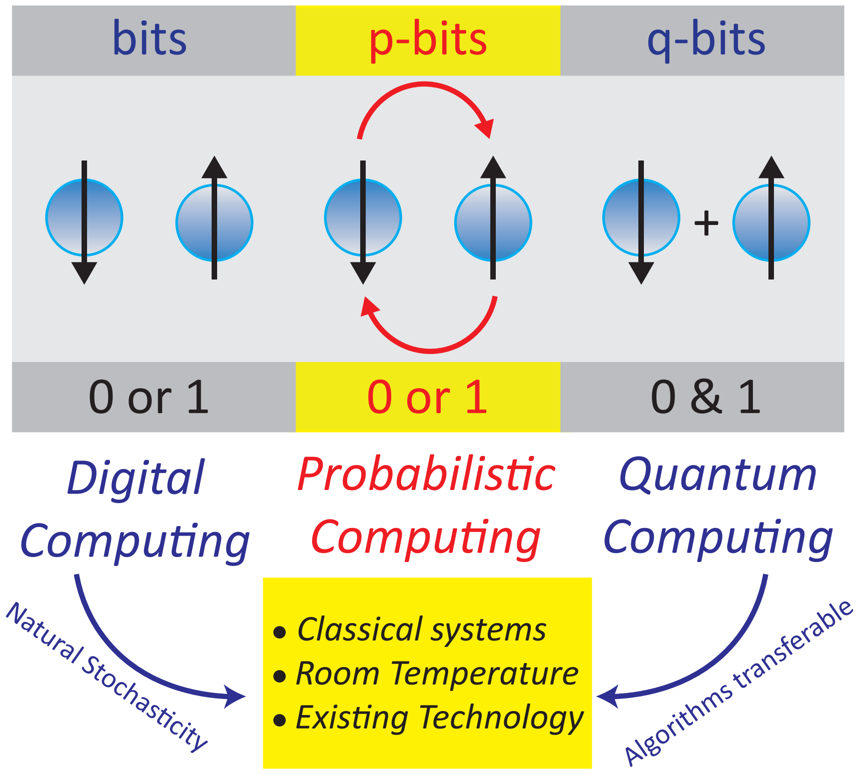


Fig. 1.1. **Probabilistic computing.** Digital computing relies on stable deterministic units called "bits" which are either '0' or '1' at any given point in time. On the other end of the spectrum lies q-bits; quantum entities which are in a delicate super-position state of '0' and '1'. In this thesis we experimentally demonstrate probabilistic bits or "p-bits" that are robust, classical entities fluctuating between '0' and '1'. These p-bits can be used as natural hardware for many classical algorithms that introduce stochasticity using aritifical means, while a subset of problem reserved for quantum computing can also be mapped to networks of p-bits.

$$m_i(t) = \vartheta\Big[\sigma(I_i) - r\Big] \tag{1.1}$$

where ϑ is the unit step function, σ is the sigmoidal activation function acting on the analog input signal I_i and r is a random number uniformly distributed between 0 and 1. In the absence of the input I_i, the p-bit is just a random number generator with the statistics of the fluctuating resistance r. The input I_i functions as a bias on top of the fluctuating resistance r; in the sense that a negative input I_i will bias the output m_i negatively, resulting in more negative values at the output m_i while a positive input will bias the p-bit vice versa. Thus at the heart of a p-bit lies a classical fluctuating resistance r moving back and forth between two states.

The ability to bias the p-bits allows p-bits to get correlated to one another is what we call p-circuits. In these correlations lies the physics of the problem to be solved. So for example, the i^{th} p-bit can be driven by a synaptic input I_i derived from a problem description which is a function of all other outputs $\{m_i, \cdots, m_N\}$. This derivation of the interconnect strengths (synapse) could be either done using first principles or also learned via machine learning principles.

An advantage of the classical nature of p-circuits lies in the various types of synapses that could be used to interconnect them. For example, linear synapses commonly used in neural networks

$$I_i = \Big[\sum_j J_{i,j} m_j + h_i\Big] \tag{1.2}$$

where $J_{i,j}$ is the interconnect strength between the i^{th} and j^{th} p-bit and h_i is the on-site bias for the i^{th} p-bit; could easily be used in p-circuits. Such synapses have been used to implement invertible boolean logic gates as demonstrated in this thesis through out.

Alternately the interconnect strengths I_i for each p-bit can also be obtained using principles of calculus from an energy function E,

$$I_i = -\partial E(m_1, \cdots, m_N)/\partial m_i \tag{1.3}$$

These networks will visit different energy configurations with probabilities given by the Boltzmann law $P\big(m_1, \cdots, m_N\big)$ that are proportional to $exp\big[-E(m_1, \cdots, m_N)\big]$, with the ground states of the energy landscape having the highest probability. This

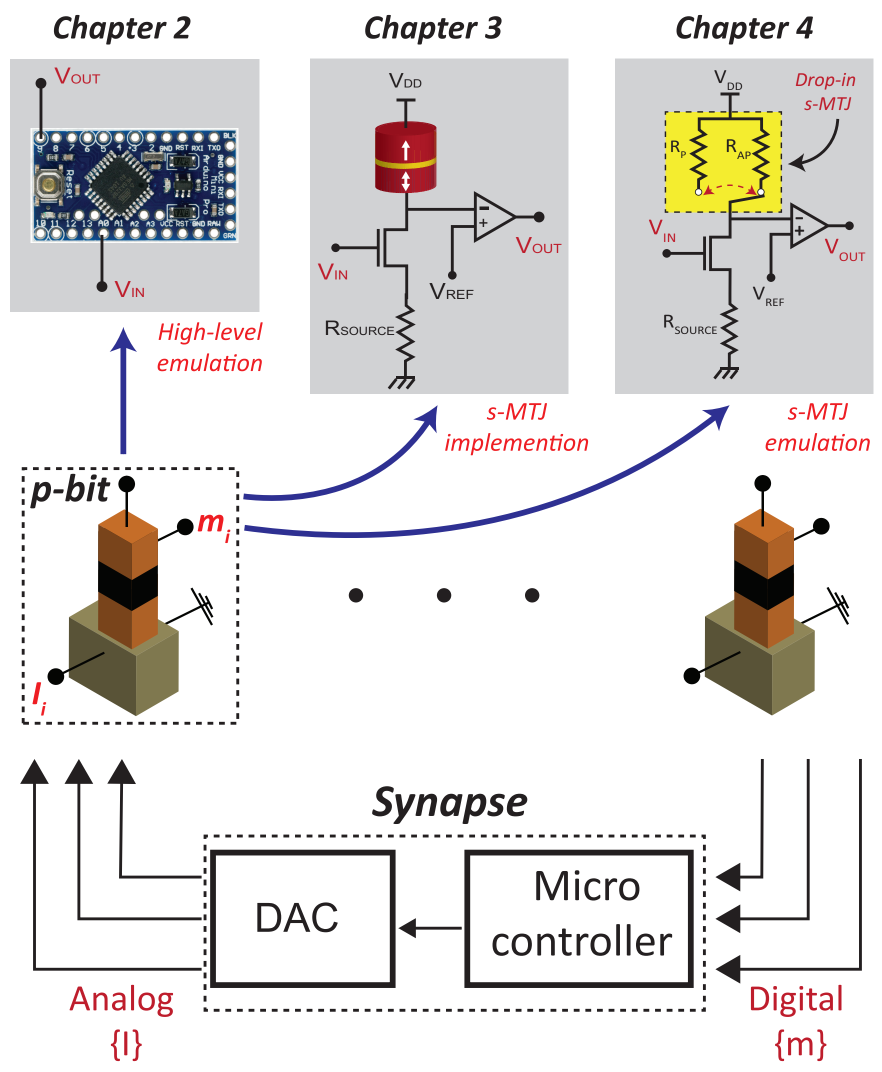


Fig. 1.2. **Thesis Overview.** In this thesis we present an autonomous probabilistic computer which uses clock-less p-bits interconnected using a synapse as shown in the figure above. To achieve this, we first used a high-level emulation of p-bits using micro-controllers which helped establish important rules of operations for such systems. This was then followed by the "very first" experimental demonstration of a thermally unstable stochastic Magnetic Tunnel Junction (s-MTJ) based autonomous probabilistic computer. While we work towards scaling the s-MTJ based p-computer, we have built a low-level emulation of p-computers which will assist in further studying the role of device variations and new application spaces for probabilistic computers.

property allows such networks to be used for solving optimization problems, similar to ising and adiabatic quantum computing, where the global minimum of the energy landscape is the configuration which minimizes the cost function identified by E.

We would like to point out that the term probabilistic computing has been used interchangeabley with stochastic computing historically. However, we would like to distinguish our “probabilistic computers” from the well known field of “stochastic computing”. The expression stochastic computing was coined in the 1960s following the pioneering work of von Neumann [2], Gaines [3] and Poppelbaum et al. [4]. Stochastic computing focused on the reliable implementation of boolean algebra and probabilistic arithmetic. This was done using stochastic components whose major attraction lied in the use of low complexity units and inherent error tolerance. Stochastic computing used streams of bits to represent numbers which could be processed by simple circuits boolean gates such as AND gate. The outputs of these gates were then tabulated into probabilities. Our approach to probabilistic computing is very different to the above mentioned paradigm of stochastic computing — in the sense that we use a bit which is probabilistic in nature. For certain algorithms the probabilistic nature of our bit is needed, which is otherwise introduced artificially using pseudo random number generators at the significant expense of area and energy consumption.

1.2 Organization of Thesis

In Chapter. 2 we first demonstrate a high-level emulation of p-bits using off the shelf microcontrollers. This emulation helped establish important rules of operation for autonomous p-computers such as the method of bypassing serial operation of p-bits; a mode of operation used in computer simulations.

We then present in Chapter. 3 the very first experimental proof-of-concept demonstration of probabilistic bits built using slight modifications of market ready magneto-resistive random-access memory devices. Eight of these devices are interconnected to form a clock-less autonomous probabilistic computer to which we map two distinct problems **a)** an Invertible boolean gate **b)** a hardware optimizer which maps integer factorization as an optimization problem.

We then demonstrate in Chapter. 4 a much lower-level emulation of s-MTJ based p-bits which will allow us to study the role of device parameter variations and device characteristics on proper system operation.

Finally we present in Chapter. 5, a CMOS based implementation of p-computers which demonstrates a 32-bit invertible Ripple Carry Adder (RCA) using 450 digital p-bits operating in a clocked synchronous manner.

1.3 Publications associated with Thesis

1. **A. Z. Pervaiz**, L. A. Ghantasala, K. Y. Camsari, & S. Datta (2017). Hardware emulation of stochastic p-bits for invertible logic. Scientific reports, 7(1), 10994.

2. **A. Z. Pervaiz**, B. M. Sutton, L. A. Ghantasala & K. Y. Camsari (2018). Weighted p-bits for FPGA implementation of probabilistic circuits. IEEE transactions on neural networks and learning systems. vol. 30, no. 6, pp. 1920-1926, June 2019.

3. **A. Z. Pervaiz**, K. Y. Camsari & S. Datta (2019). Probabilistic computing with Binary Stochastic Neurons. **In press: BCICTS 2019**

4. W. A. Borders, **A. Z. Pervaiz**, S. Fukami, K. Y. Camsari, H. Ohno & S. Datta (2019). Integer Factorization using stochastic Magnetic Tunnel Junctions. Nature 573, no. 7774 (2019): 390-393.

   Equal Contributing First author.

Other contributions

1. K. Y. Camsari, R. Faria, O. Hassan, **A. Z. Pervaiz**, B. M. Sutton & S. Datta (2017). p-transistors and p-circuits for Boolean and non-Boolean logic. In Spintronics X (Vol. 10357, p. 103572K). International Society for Optics and Photonics.

2. K. Y. Camsari, P. Debashis, V. Ostwal, **A. Z. Pervaiz**, T. Shen, Z. Chen, S. Datta and J. Appenzeller (2018). From charge to spin and spin to charge: Stochastic magnets for probabilistic switching. **Under review: Proceedings of the IEEE**

3. **A. Z. Pervaiz**, K. Y. Camsari & S. Datta (2019). Asynchronous computing with p-bits. *Invited talk.* ASYNC 2019, Hirosaki Japan.

4. K. Y. Camsari, **A. Z. Pervaiz**, R. Faria, E. E. Marinero & S. Datta (2016). Ultrafast spin-transfer-torque switching of synthetic ferrimagnets. IEEE Magnetics Letters, 7, 1-5.

2. HARDWARE EMULATION OF STOCHASTIC P-BITS FOR INVERTIBLE LOGIC

Materials in this chapter have been extracted verbatim from the paper: **A. Z. Pervaiz**, L. A. Ghantasala, K. Y. Camsari, & S. Datta (2017). Hardware emulation of stochastic p-bits for invertible logic. Scientific reports, 7(1), 10994.

2.1 Introduction

This chapter presents the blue print of our autonomous p-computers. To begin our development of nano-device based p-computers we first emulate the idealized behavior of p-bit using off the shelf microcontrollers. The striking properties of p-circuits such as invertible logic are quite intriguing, but these were previously demonstrated using pure software implementations of Eqs. (1.1,1.2). In simulations of these p-circuits, it was well known that each p-bit needs to be updated in a serial manner. This process is enabled using control loop statements, however this represents a significant challenge towards scaling p-circuits to incorporate for example several millions of p-bits. A serial operation where each p-bit is updated one at a time seems rather difficult for scaled systems, and still says nothing of the dedicated hardware needed to implement such a large scale serial operation. Other questions addressing the effect of parameter variations among p-bits also need to be addressed.

In this chapter we demonstrate an *autonomous* p-computer where the serial updating of p-bits comes naturally without any peripheral control circuity. This fortuitous result comes due to the asynchronous operation of p-bits which result from natural time delays between p-bits in this case, while in later systems due to the statistics of the p-bit retention times. This paper presents a first pass in discovering and answering important questions that would arise in the hardware development of p-circuits. Our approach is similar to ref. [5] in the sense that we use microcontrollers to emulate both Eq. (1.1) and the interconnections between p-bits described by Eq. (1.2.

The hardware emulation presented in this chapter shares many of the essential feature of s-MTJ based p-computers; The output $m_i(t)$ and the input $I_i(t)$ appearing in Eqs. (1.1) and (1.2) are both actual voltages and both sub-systems are completely independent of each other. This allows us to later simply drop-in the s-MTJ based p-bits (chapter. 3) while using the same synapse. We also study the variability in retention times τ_N of the p-bits and the role of interconnect delay τ_{inter} on system operation. To summarize, the hardware emulation presented here helped establish important rules of operation for the autonomous p-circuits that are to follow in this thesis.

Next we describe our approach to emulate Eqs. (1.1) and (1.2). Fig. 2.1 shows a microcontroller based emulation of a p-bit. We then present a 3 p-bit circuit which implements an invertible AND gate (Figs. 2.2,2.3). We use this simple circuit to study the effects of τ_N and τ_{inter} (Figs. 2.4,2.5,2.6). This is followed by demonstrations of larger p-circuits; the 4-bit adder and the 4-bit multiplier, both working in the inverted mode.

2.2 Methods

2.2.1 Arduino pro mini as a p-bit

A version of Eq. (1.1) suitable for emulation of a p-bit by a microcontroller is given as

$$V_{OUT}(t) = sgn\Big\{S\big(V_{IN}(t)\big) - rand\big(0,1\big)\Big\} \tag{2.1}$$

where V_{OUT} is the digital output and V_{IN} is the analog input voltage of the p-bit. S(x) is a sigmoidal activation function which acts on an input x. Mathematically,

$$S(x) = \frac{1}{1+e^{-2x}} \tag{2.2}$$

I/O characteristics: An Arduino pro-mini is a 24 pin microcontroller [6]. We program it to emulate the behavior of a p-bit using Eq. (2.1). A pseudo-code is given in Alg. 1. The microcontroller has 6 input pins which are connected to an internal 10-bit analog-to-digital converter. The analog input pins have very high input resistances ($\approx 100M\Omega$). The arduino pro-mini also has dedicated pins for PWM (Pulse-width modulation) outputs with low output resistances ($\approx 100\Omega$) and the ability to source 40 mA of current.

p-bit operation: Fig. 2.1(a) shows a time snapshot of the output voltage of a p-bit for a set of applied input voltages as captured on an oscilloscope (Tektronix DPO7104). For each applied input voltage V_{IN} the average output voltage is measured. At a $V_{IN} = 2.5V$ the p-bit is fluctuating randomly between low and high states. When the input voltage is lowered or increased, the p-bit output gets biased towards more 0's or 1's. DC average measurements of the output voltage taken over 100 seconds are shown in Fig. 2.1(b). The shape of the average response of a p-bit follows the sigmoidal function given by Eq. 2.2.

Retention time τ_N: Each p-bit is characterized by a retention time τ_N; a time period for which the output voltage maintains its state. Stochastic magnetic tunnel junctions (s-MTJs) have been proposed [7] as the building blocks for p-bits. These s-MTJs are two magnets, one stable (called a fixed layer) and the other (called a free layer), seperated by a thin insulating oxide. In s-MTJs the retention time is dependent on the energy barrier of the free layer whose energy barrier is lowered to make it stochastic. For example, MTJs meant to be used for digital memory are designed to have large energy barriers ($\gg 40k_BT$), such that a bit of information stored remains intact for many years to come. For magnets the retention time is given by [8] :

$$\tau_N = \tau_0 \exp\left(\frac{\Delta}{k_BT}\right) \tag{2.3}$$

where τ_0 is a material dependent quantity ranging from a few ps to a ns [9], Δ is the energy barrier of the free layer and k_BT is the Boltzmann energy. For stochastic MTJs with energy barriers of $10 - 20\ k_BT$, the retention time is in the ms regime typically [10]. A user defined delay τ_N shown in Alg. 1 is used to emulate the retention time of p-bits. This user defined delay is systematically changed to characterize its effects on system operation.

Algorithm 1 Pseudocode for p-bit

```
Parameters:
      Digital output $\mathrm{V_{OUT}}$;
      Analog input $\mathrm{V_{IN}}$;
Repeat:
      $x \leftarrow$analogRead($\mathrm{V_{IN}}$);          $\triangleright\ \mathrm{V_{IN}} \in (0, 5\ \mathrm{V}), \mathrm{x} \in (0.5)$
      $m \leftarrow 2x - 5$;                              $\triangleright\ \mathrm{m} \in (-5, 5)$
      Bias $\leftarrow$ S(m);                             $\triangleright$ Bias $\in (0, 1)$ from Eq.(2.2)
      W $\leftarrow$ rand(0, 1);                          $\triangleright\ \mathrm{W} \sim \mathrm{U}(0, 1)$
      If(Bias > W)
        $\mathrm{V_{OUT}} \leftarrow 1$;
      Else                                               $\triangleright\ \mathrm{V_{OUT}} \in \{0, 5\ \mathrm{V}\}$
        $\mathrm{V_{OUT}} \leftarrow 0$;
      EndIf
      Wait $\tau_{\mathrm{N}}$;
EndRepeat
```

2.2.2 Synapse using a microcontroller and DAC

Fig. 2.2(a,b) shows the schematic and the block diagram of a 3 p-bit circuit. The p-bits are interconnected via a synapse which calculates the input voltage V_{IN} of the i^{th} p-bit using

$$V_{IN}(t) = I_0 \Big\{ h_i + \sum_j J_{ij} V_{OUT}(t) \Big\} \tag{2.4}$$

A pseudo-code for implementing the synapse is described in Alg. 2 where another user defined delay τ_{inter} is added. This delay is meant to simulate the interconnect delay, which any real synapse would have in practice. Please note that Eq. (2.4) describes a liner synapse as described by Eq. (1.2), but the synapse can also implement non-linear interconnections as shown in later chapters.

Synapse using microcontroller: Our synapse is implemented using an Arduino Mega2560 microcontroller along with a MAXIM 5825 8 channel 12 bit digital-to-analog converter [11] (DAC). The Arduino Mega2560 reads the output voltage of each p-bit and calculates the inputs voltage for each p-bit accordingly. This calculation is turned into an analog voltage using DACs; a process which requires the Arduino Mega2560 to use one of its peripheral interfaces. We use a serial interconnect protocol known as I_2C, which limits the interconnect delay τ_{inter} to $\approx ms$. The synapse can set a maximum voltage of 5 V, which requires the synapse to threshold the input voltages V_{IN}. The Arduino Mega2560 also provides a serial port which allows communication over the USB port. This is particularly useful for observing the state of p-circuits on a computer. Another possible method of observing the p-circuits is to create an artificial node (an analog voltage). An example of this is shown in Fig. 2.2(c), where an analog voltage $4\times A+2\times B+C$ is set via the DAC and observed on the oscilloscope.

Algorithm 2 Pseudo code for weight logic

```
Parameters:
        Analog outputs $V_{IN}$;                        ▷ The input voltages of p-bits
        Digital inputs $V_{OUT}$;                       ▷ The output voltages of p-bits
        Parameters [J],{h} and $I_0$;
        $n \leftarrow$ Number of p-bits;
        $k \leftarrow$ DAC terminal for word;
Repeat:
        For i $\in \{1 \cdots, n\}$
            S $\leftarrow$ digitalRead($V_{OUT}$[i]);          ▷ $V_{OUT} \in \{0, 5V\}, S \in \{0, 1\}$
            m $\leftarrow$ 2S − 1 ;                            ▷ $m \in \{-1, 1\}$
        EndFor
        For j $\in \{1 \cdots, n\}$
            Evaluate $I_j^{'} \leftarrow I_0\big(h_j + \sum_j J_{ij} m_j\big)$     ▷ $I_j^{'} \in (-\infty, +\infty)$
            If($I_j^{'} > 5$)
                $I_j = 5$ ;
            ElseIf($I_j^{'} < -5$)
                $I_j = -5$ ;                                   ▷ $I_j \in (-5, +5)$
            EndIf
            $V_{IN}[j] \leftarrow 2I_j - 5$                    ▷ $V_{IN} \in (0, +5V)$
            Set DAC[j] $\leftarrow V_{IN}[j]$
        EndFor
        Set DAC[k] $\leftarrow 4 \times V_{OUTA} + 2 \times V_{OUTB} + V_{OUTC}$     ▷ Output word
        Set Serial() $\leftarrow V_{OUT}$ for all p-bits      ▷ Output through the USB port
        Wait $\tau_D$ ;
EndRepeat
```

2.3 Results

2.3.1 AND Gate as a Boltzmann Machine

Correlated network of p-bits: Fig. 2.2(c) shows a time snapshot of an artificial node $4 \times A + 2 \times B + C$. Using three p-bits an AND gate is realized with each p-bit corresponding to one terminal of the AND gate. The interconnect matrices $[J]$ and self-bias vector $\{h\}$ are taken from ref. [12]. The strength of correlations between p-bits is changed by varying I_0 (Eq. (1.2)). I_0 can be thought of as the inverse (pseudo) temperature, in the sense that as I_0 increases the *temperature* of the system decreases allowing the system to reach equilibrium. When the system is uncorrelated by using an $I_0 = 0$, the 3 p-bits are independent of each other which results in the system fluctuating back and forth between $2^3 = 8$ states with equal probability. This behavior is shown in Fig. 2.2(d). When the correlations are turned on (using $I_0 = 0.8$) the system now locks in to the states that are highlighted by the truth table of the AND gate as shown in Fig. 2.2(e). This mode of operation in which all p-bits are left floating is quite interesting, and has no equivalent in CMOS gates. Fig. 2.2(d,e) show the steady-state statistics for both the uncorrelated and correlated cases using approximately 5×10^4 samples.

Computing with p-bits: For Boolean computation, the p-bits need to be *clamped* to a given input. This is done by simply connecting the input voltage of a p-bit to either ground or 5 V. An intriguing feature of p-circuits is that all p-bits are treated equally, which allows both the inputs and outputs to be clamped to a user defined input.

Direct Operation: Fig. 2.3 shows two cases of using an AND gate for computation purposes. Fig. 2.3(a) shows a time snapshot when the inputs A and B have been clamped to 1. The output C as expected is consistent with the inputs A and B. This is highlighted using the statistics collected in Fig. 2.3(b).

Inverted Operation: A remarkable feature of p-circuits is the *inverted* operation. Fig. 2.3(c) shows the time evolution of the system, when C is clamped to 0. Remarkably the inputs A and B now fluctuate between the states that are allowed for C=0. This inverted mode of operation is highlighted using the steady state statistics collected in Fig. 2.3(d). The inverted mode of operation holds up even in large p-circuits and more importantly in interconnected p-circuits themselves, a feature used in this chapter to demonstrate an invertible multiplier which works as a factorizer.

2.3.2 Interconnect delay and retention time

For a p-circuit such as the one presented in Fig. 2.2, there are two major time constants:

- **Retention time τ_N:** Time interval for which the output state of the p-bit is held stable. This interval is programmed using a user defined delay given in Alg. 1. In this chapter we start off with a constant delay for each p-bit, while in chapter. 4, this delay is sampled from an exponential distribution at every iteration.

- **Interconnect time τ_{inter}:** The total time delay for the synapse to read the outputs of p-bits and apply back the inputs. The interconnect delay is the sum of the user defined delay τ_D of Alg. 2, and the time it takes to perform everything else in the Repeat block of Alg. 2.

Boltzmann Law: We now study the effect of both these time constants on the operation of the system using the AND gate. For p-circuits, an energy functional E for the state $\{m\} = \{m_i, \cdots m_N, \}$ can be defined as [7]:

$$E(\{m\}) = -I_0 \Bigg\{ \sum_{i,j} \frac{1}{2} J_{ij} m_i m_j + \sum_i h_i m_i \Bigg\} \tag{2.5}$$

The Boltzmann Law accurately captures the steady state probabilities of the system to be in different states $\{m\}$ according to,

$$P(\{m\}) = \frac{\exp\big(-E(\{m\})\big)}{\sum_{i,j} \exp\big(-E(\{m\})\big)} \tag{2.6}$$

Interconnect delay: Fig. 2.4 shows the steady state statistics for the AND gate with each of the three p-bits having $\tau_N = 200ms$, with there interconnect delay τ_{inter} varied from 1 ms to 400 ms. It can be seen from Fig. 2.4(a) that for extremely small

τ_{inter} the behavior of the system is captured well by the Boltzmann law. However as τ_{inter} is increased to 100 ms, two incorrect states [001] and [110] start getting highlighted. As τ_{inter} is increased to 200 ms and beyond, the system breaks down completely, with only the [001] and [110] states getting highlighted.

Fig.2.5(b) shows the euclidean distance between steady state distributions for various interconnect delay (τ_{inter} varied from 1 ms to 400 ms with $\tau_N = 200\ ms$ for all p-bits). We observe that a boundary ($\tau_{inter} \approx \tau_N$) exists for proper operation of the system. Beyond this boundary, p-bits are in essence *updating* in parallel instead of the required serial update. An important conclusion of this simple experiment is that *autonomous p-circuits can naturally perform serial updating as long as* $\tau_{inter} \ll \tau_N$.

An essential requirement for Hopfield networks and unrestricted Boltzmann Machines is the need for sequential updating, where each p-bit is updated serially but in any random order [13, 14], as opposed to parallel updating where each p-bit is updated at once. To enforce serial updating in simulation requires control flow statements which regulate the updating procedure of p-bits to one at a time or serial. Serial updating arises naturally in our setup since each p-bit is completely independent of each other and small phase differences that are present initially get greatly magnified as the system is run for longer periods of time, in the absence of a central clock signal. This type of updating is also known as the "asynchronous dynamic" in Hopfield networks [13]. This is shown for an AND gate with 3 p-bits in Fig.2.5(a), where each of the 3 p-bits are almost perfectly aligned to each other initially, however this alignment is broken as system continues to run with time. Asynchronous machines are known to converge slowly, while their synchronous counterparts allow for parallel updating, allowing much faster convergence. For hardware implementations, it is the synchronous machines that would require some master control to ensure parallel updating making the system grow in resources as the number of p-bits increase.

Retention time distribution: We now investigate the behavior of the AND gate when each of the three p-bits has a different retention time. This is expected to happen in any real system due to process variations of nanodevices used. Fig. 2.6(a) shows the histogram for three different retention time configurations for the AND gate. In the most trivial case, all three p-bits have the same retention time $\tau_N = 200\ ms$ while having an interconnect delay $\tau_{inter} = 1\ ms$. The steady state statistics for this case shows a good match with the Boltzmann law (Fig. 2.6(b)).

A more realistic scenario is that of the 3 p-bits having different retention times. Fig. 2.6(a) shows two cases where p-bits are distributed in two sets of {137, 200, 263} ms and {50, 200, 350} ms with a spread of ±33% and ±75% around the mean value of 200 ms respectively, while maintaining very low interconnect delay of $\tau_{inter} = 1\ ms$. Both cases show a good match with the Boltzmann Law (Fig. 2.6(b)). We conclude that if the interconnect delay τ_{inter} is much smaller than the smallest τ_N present in the system, the system operation is well described by the Boltzmann Law, which can be attributed to the much reduced probability of parallel updating.

2.3.3 Full Adder as a Boltzmann Machine

Fig. 2.7(a) shows a schematic of a Full Adder which is implemented using 14 p-bits. 5 of these 14 p-bits are the actual terminals of the Full Adder while the remaining 9 are auxiliary p-bits (see Ref. [7]). Both τ_N and τ_{inter} are 200 ms and 10 ms respectively. Since more than 8 p-bits are used, we now use two DACs for the 14 p-bit Full Adder.

The design of $[J]$ and $\{h\}$ matrices follows the prescription of ref. [7]. Direct computations can be performed by clamping p-bits as discussed earlier. Fig. 2.7(c,d) shows an example of 1-bit binary addition. The inputs A, B and C_{IN} have been clamped to 110 respectively, and the time evolution of S and C_{OUT} are shown in Fig. 2.7(c). This is highlighted by the steady state statistics shown in Fig. 2.7(d).

Similar to the AND gate, the Full Adder is also be operated in the inverted mode. A time snapshot of the inputs A, B and C_{IN} is shown in Fig. 2.6(e) when the outputs S and C_{OUT} are clamped to 0 and 1 respectively. The steady state statistics shown in Fig. 2.7(f) shows that the system follows the states prescribed by the truth table of the Full Adder.

2.3.4 Directed Networks of p-circuits

To build more complex systems, one possible approach is to design the entire system as a single p-circuit. A more practical alternative is to interconnect simpler p-circuits with *directed* connections to build up more complex systems such as a 4-bit Ripple Carry Adder (RCA) (Fig.2.8(a)) or a 4-bit multiplier/factorizer (Fig.2.9(a)).

Directed Connections: Separate p-circuits can be connected in a *directed* fashion such that the connections between the two are not reciprocal $J_{ij} \neq J_{ji}$. In hardware, this corresponds to disconnecting the input voltage of p-bit "i" from its native synapse and connecting to it the output voltage of p-bit "j" from a different p-circuit so that $J_{ij} = 1$ and $J_{ji} = 0$. Consider the case of a 4-bit adder that is built using a Half Adder and 3 Full Adders. In this case there are 3 directed connections as shown in Fig. 2.8(a). Each connection takes the output voltage of C_{OUT} of the $(n-1)^{th}$ adder and connects it to the input terminal of C_{IN} of the n^{th} adder. Due to this connection scheme, no information can flow from the n^{th} adder to the $(n-1)^{th}$ adder. However, as noted in ref. [7], bidirectional connections of adders hinders proper operation of a n-bit adder. Also note that since the connection from one p-circuit to another is an electrical connection, the strength of the correlation between the two machines is at most 1 ($J_{ji} = 1$).

4-bit Adder: We next demonstrate the correct operation of a 4-bit RCA comprised of 48 p-bits each having different τ_N as shown in the inset of Fig. 2.8(d). The values of τ_N are normally distributed around an average of 200 ms with a minimum of 137 ms to a maximum of 263 ms, with a interconnect delay of 10ms for all Full Adders. 4-bit binary addition is performed by clamping the input p-bits of each adder, as demonstrated by the time snapshot of the sum shown in Fig. 2.8(c) with A=10 and B=13 resulting in the sum being 23 when converted to decimal. We observed for AND gates that there exists a boundary for proper operation of p-circuit with all p-bits had the same retention time. Similarly with a distribution such as the one studied here there also exists a boundary for proper operation which is $\tau_{inter} \lessapprox min(\tau_N)$.

Inverted mode: A more remarkable case is that of the sum being clamped to S=23, with the inputs A and B left floating. In this case, A and B fluctuate among 8 possible integer combinations that satisfy $A + B = S = 23$. Note that since A and B are 4-digit binary numbers, not all integer combinations can be probed by the system, for example A=22 and B=1. This can be seen from the histogram presented in Fig. 2.8(f). Although there are 8 peaks in the histogram, the height of each peak is not the same since statistics presented in Fig. 2.8(f) are not exactly steady state. With 48 p-bits in the system, the number of samples needed for steady state statistics is prohibitively large.

4-bit multiplier/factorizer: In this final example, we show how a digital multiplier built out of AND gates and Full Adders can be operated in inverted mode to function as a factorizer shown in Fig. 2.9. Implementation of practically useful factorizers usually requires dedicated algorithms, here our purpose is simply to illustrate the remarkable invertibility of directed networks of p-bits.

The block diagram of a digital multiplier is shown in Fig. 2.9(b). The individual bits of A and B are first multiplied to produce A_1B_1, A_2B_1, A_1B_2 and A_2B_2 which are then added together to produce the product S. To convert this multiplier to a factorizer, we invert the directed connections from the AND gates to the adders, while keeping the original directed connections of the Full Adders from the LSB to the MSB.

The output voltages from the A_X and B_X (where X is the n^{th} Full Adder) are now sent as inputs to the output p-bits of the 4 AND gates. The 4 AND gates used here are part of one p-circuit instead of 4 separate p-circuit. This is because some inputs of the AND gates need to be the cloned to each other as they go to different gates. For example, in Fig. 2.9(b), A_1 is a common input for the two right most AND gates, while A_2 is a common input for the two left most AND gates. The retention time and interconnect delay are chosen as $\tau_N = 200\ ms$ for all the p-bits with a $\tau_{inter} = 100\ ms$.

Fig. 2.9(c) shows the time evolution of output voltages of A_1, B_1, A_2 and B_2 using an oscilloscope when the sum of the adder is clamped to 6. This results in the input p-bits of the AND gates producing the correct factors of 3×2 and 2×3. This can also be seen by the statistics of the input p-bits of the AND gate as shown in Fig. 2.9(e). As previously, the heights of both peaks are not the same due to the statistics not being exactly steady state. For comparison, we also show the statistics for an uncorrelated factorizer where 16 combinations are equally probable as shown in Fig. 2.9(d).

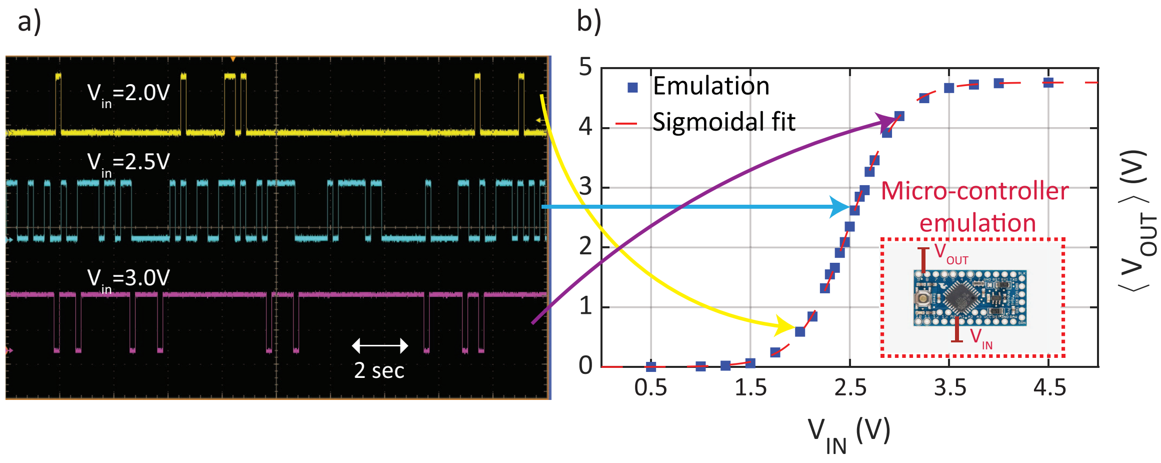


Fig. 2.1. **Emulating p-bits with microcontrollers.** An Arduino pro-mini microcontroller is used to emulate Eq. (1.1), using the details given in Alg. 1. The Arduino pro-mini shown in the inset of **(b)** has an internal 10-bit ADC which can be used to read analog inputs. It also has dedicated pins that can be used to provide PWM outputs which are used for the output. **(a)** shows the output m_i as voltage V_{OUT} as it changes with the applied input I_i as voltage V_{IN}. At an applied input voltage of $V_{IN} = 2.5\ V$ the p-bit behaves as a uniform random number generator. Increasing(decreasing) the inputs biases the output towards $V_{DD} = 5(0)\ V$. The average output voltage $\langle V_{OUT}\ (V)\rangle$ as a function follows the sigmoidal function shown in Eq. (2.2).

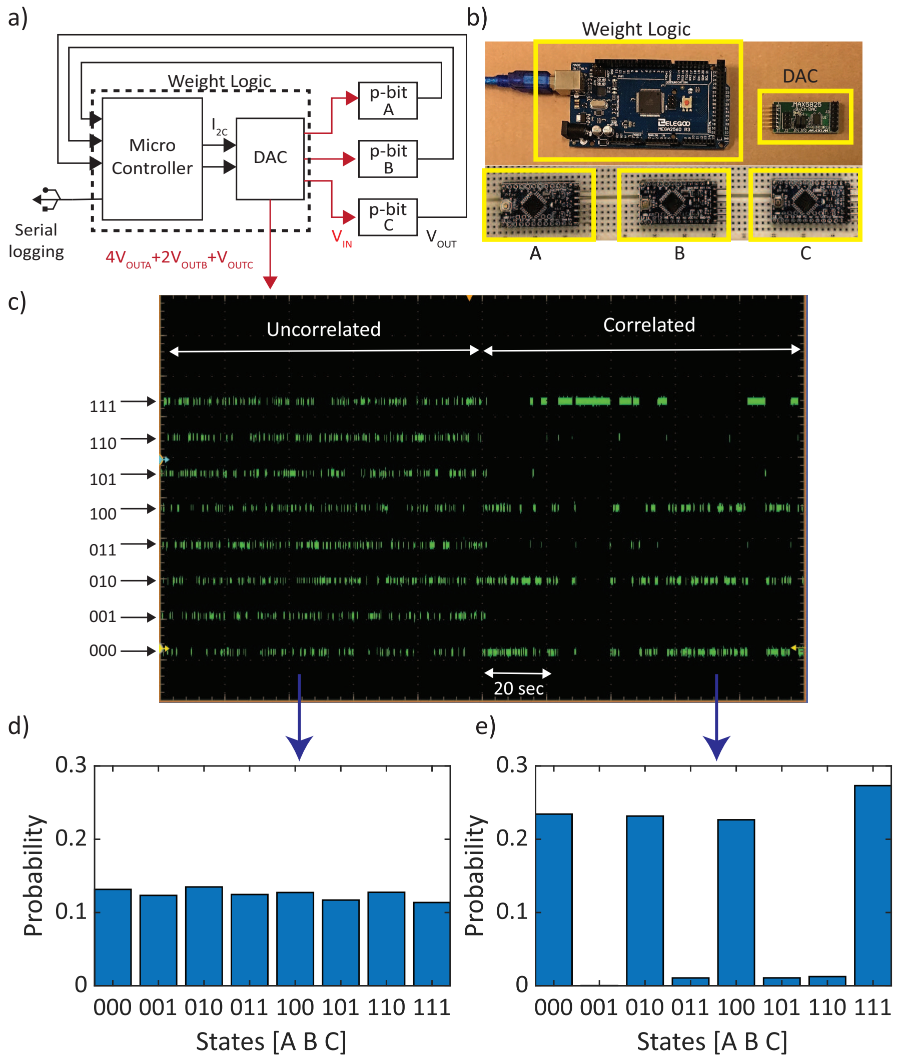


Fig. 2.2. **AND gate emulated using 3 p-bits.** **(a)** and **(b)** show the block diagram and the schematic of the emulated AND gate. Each terminal of the AND gate is one p-bit. These p-bits are interconnected using a weight logic block which comprises of an Arduino Mega microcontroller and a digital-to-analog converter. The microcontroller reads the digital output voltages $\{V_{OUT}\}$ of p-bits and provides the analog inputs $\{V_{IN}\}$. The weight logic is also used to observe the state of the AND gate by constructing an artifical signal 4×A+2×B+C shown in **(c)**. The AND gate is initially left uncorrelated which results in $2^3 = 8$ states uniformly distributed as shown with the statistics in **(d)**. When the correlations are turned on (using an $I_0 = 0.8$), the truth table of the AND gate is highlighted as shown in **(e)**.

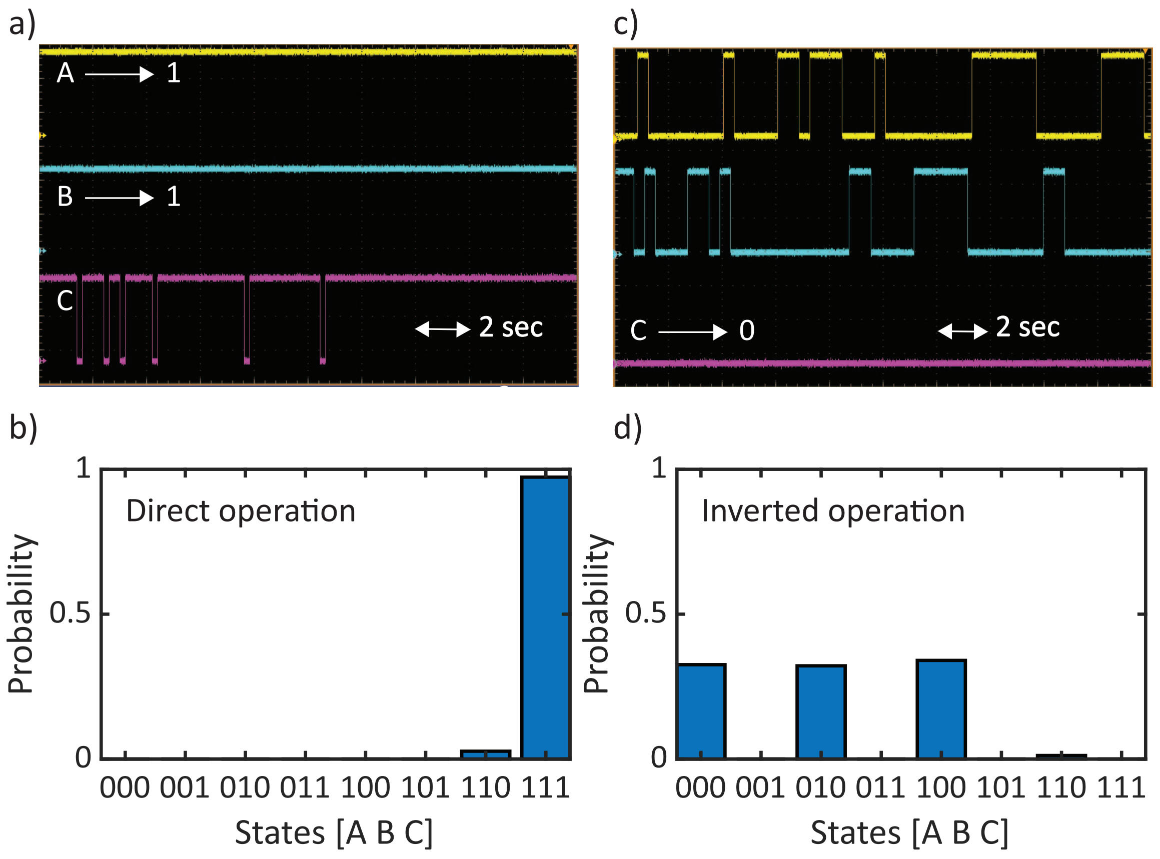


Fig. 2.3. **Computation using AND gates.** Invertible AND gates can be used for computation much like CMOS gates. **(a)** shows a time snapshot of the three p-bits when the inputs A and B have been clamped to 1. The statistics for this set of applied inputs is shown in **(b)** and this is called the "directed" mode of operation. A more interesting case, the "inverted" mode of operation is shown in **(c)** and **(d)**. When the output p-bit C is clamped to 0,the p-bits A and B to fluctuate between all the three states allowed to an AND gate for an output of 0.

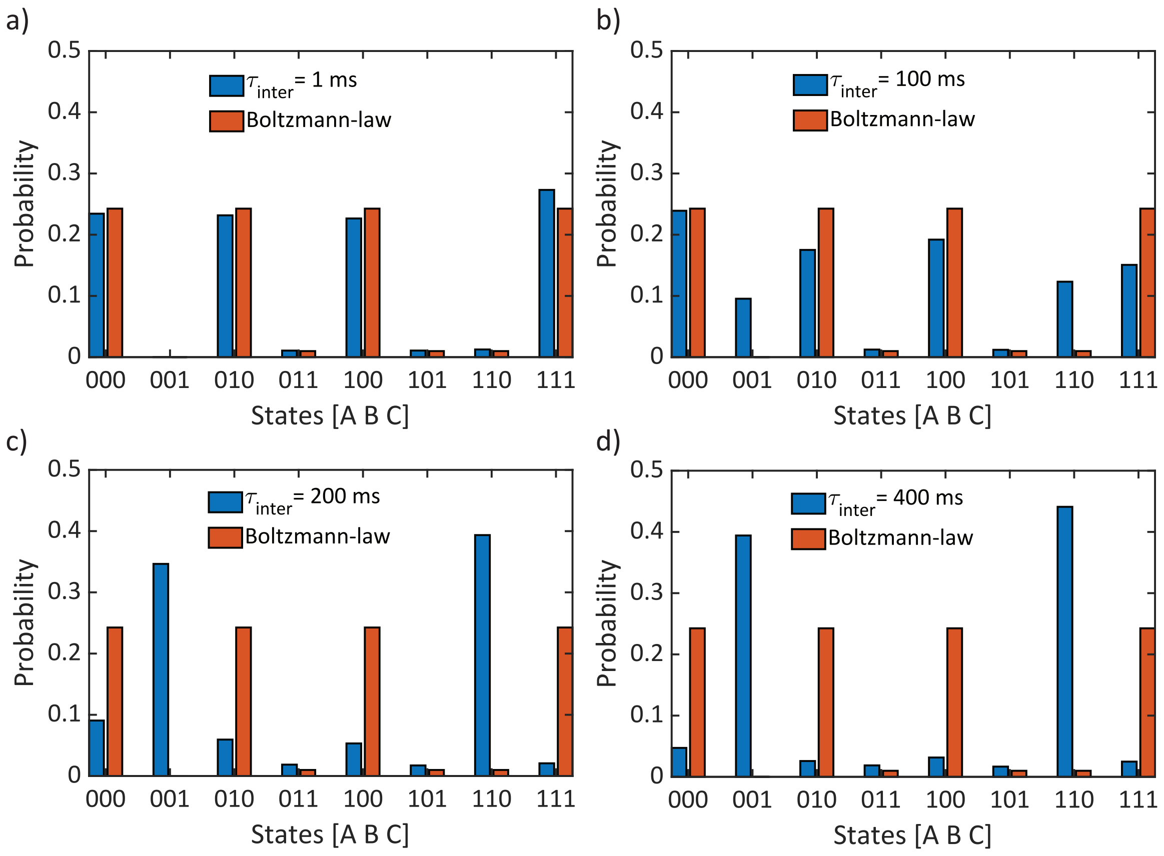


Fig. 2.4. **Interconnect delay.** An important design consideration in p-circuits is the interplay between the interconnect delay τ_{inter} and the retention time τ_N of p-bits. **(a)-(d)** shows the statistics collected as the p-circuits are slowed down or as the interconnect delay increases which is done by changing delay τ_D in Alg. 2. As the gate is slowed down, the system operations starts to breakdown.

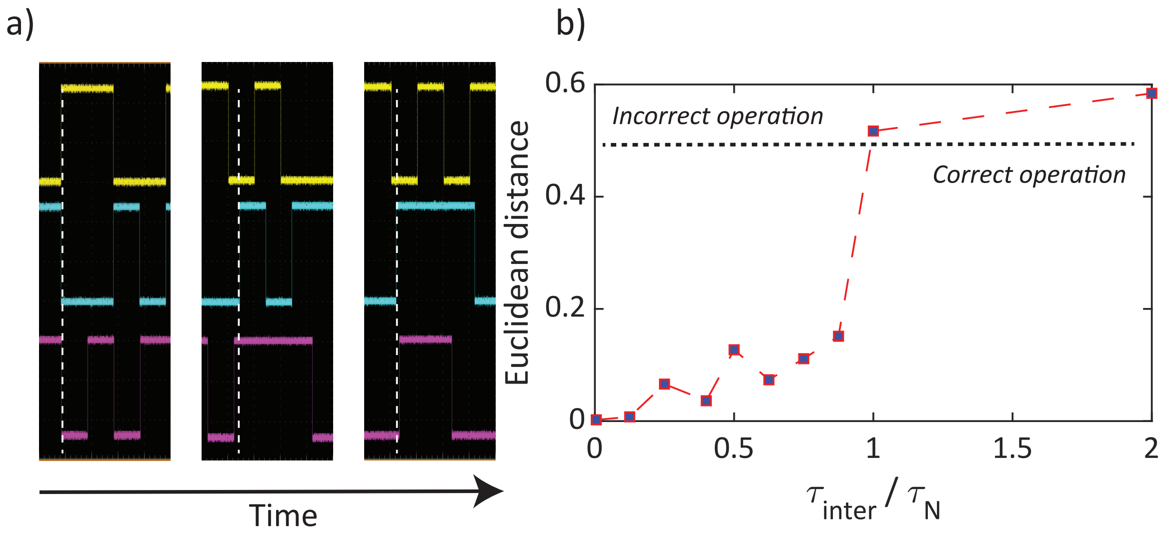


Fig. 2.5. ***Normalized* interconnect delay** $\left(\frac{\tau_{inter}}{\tau_N}\right)$**.** **(b)** shows that the system operation of the AND gate continuously deteriorates as the interconnect delay is increased. There seems to be a hard boundary for the interconnect delay beyond which the system completely breaks down. This *hardness* is most likely due to the use of constant retention times τ_N in the experiments. A key requirement for p-circuits in simulation is the need for serial updating which comes naturally in the demonstration as shown in **(a)**. Initially all 3 p-bits are well aligned with minor phase differences in between them. These phase differences are broken with time allowing each p-bit to update separately which leads to the system naturally having serial updates.

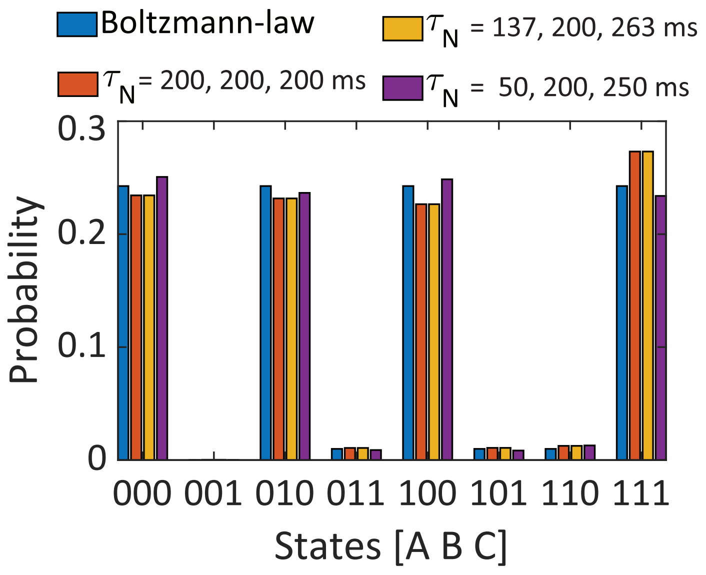
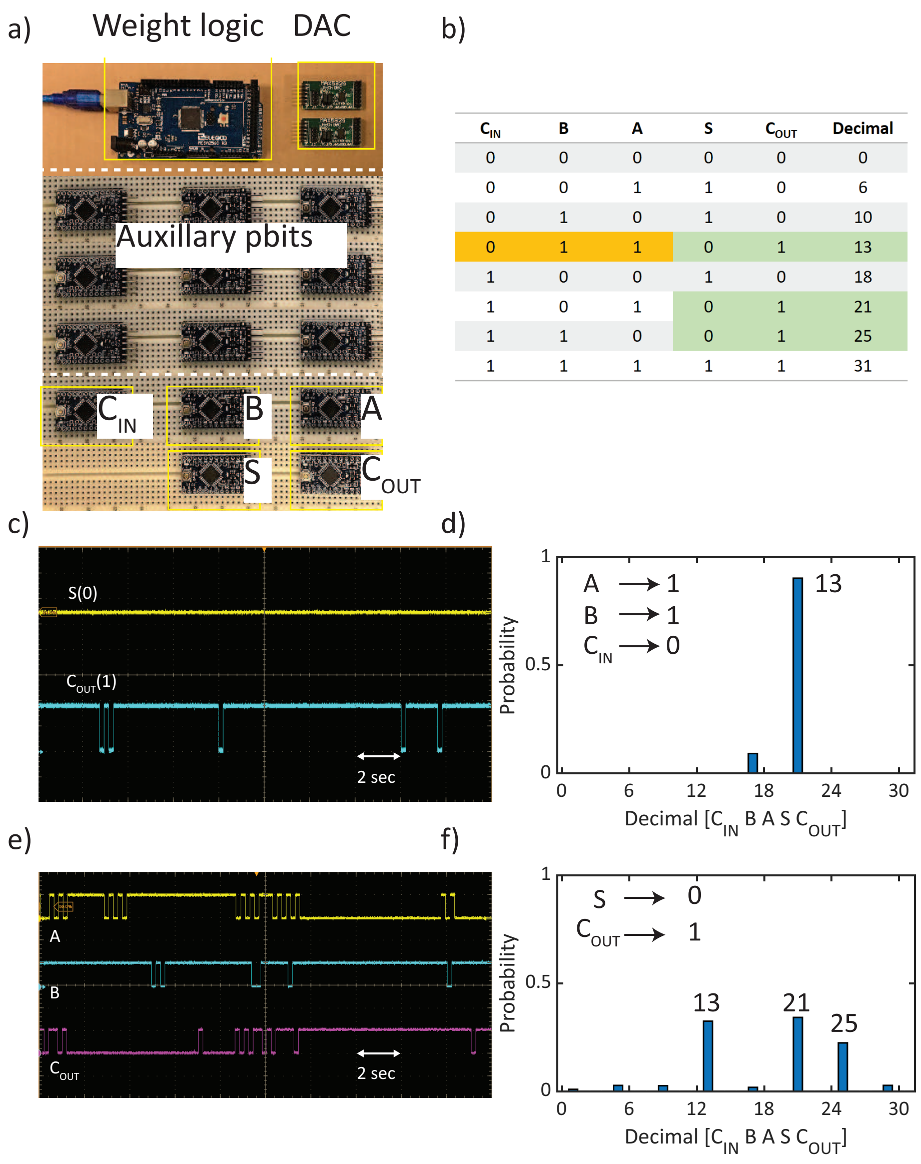


Fig. 2.6. **Variations in retention times τ_N of p-bits.** Variations in the retention time τ_N of the p-bits is expected in nano-device level implementations. These are investigated here by varying the retention times of p-bits across a wide distribution while keeping the interconnect delay significantly low, i.e τ_{inter} $\tau_N \ll 1$. As long as the interconnect delay is greater than the smallest retention time, the system will operate correctly.



C_{IN}	B	A	S	C_{OUT}	Decimal
0	0	0	0	0	0
0	0	1	1	0	6
0	1	0	1	0	10
0	1	1	0	1	13
1	0	0	1	0	18
1	0	1	0	1	21
1	1	0	0	1	25
1	1	1	1	1	31

Fig. 2.7. **Full Adder.** 14 p-bits are used to implement a full adder shown in **(a)**. 5 of these are the terminals of the full adder (truth table shown in **(b)**) while the remaining 9 are auxiliary p-bits. The full adder works both in the direct (**(c)**-**(d)**) and inverted (**(e)**-**(f)**) mode of operation similar to the AND gate.

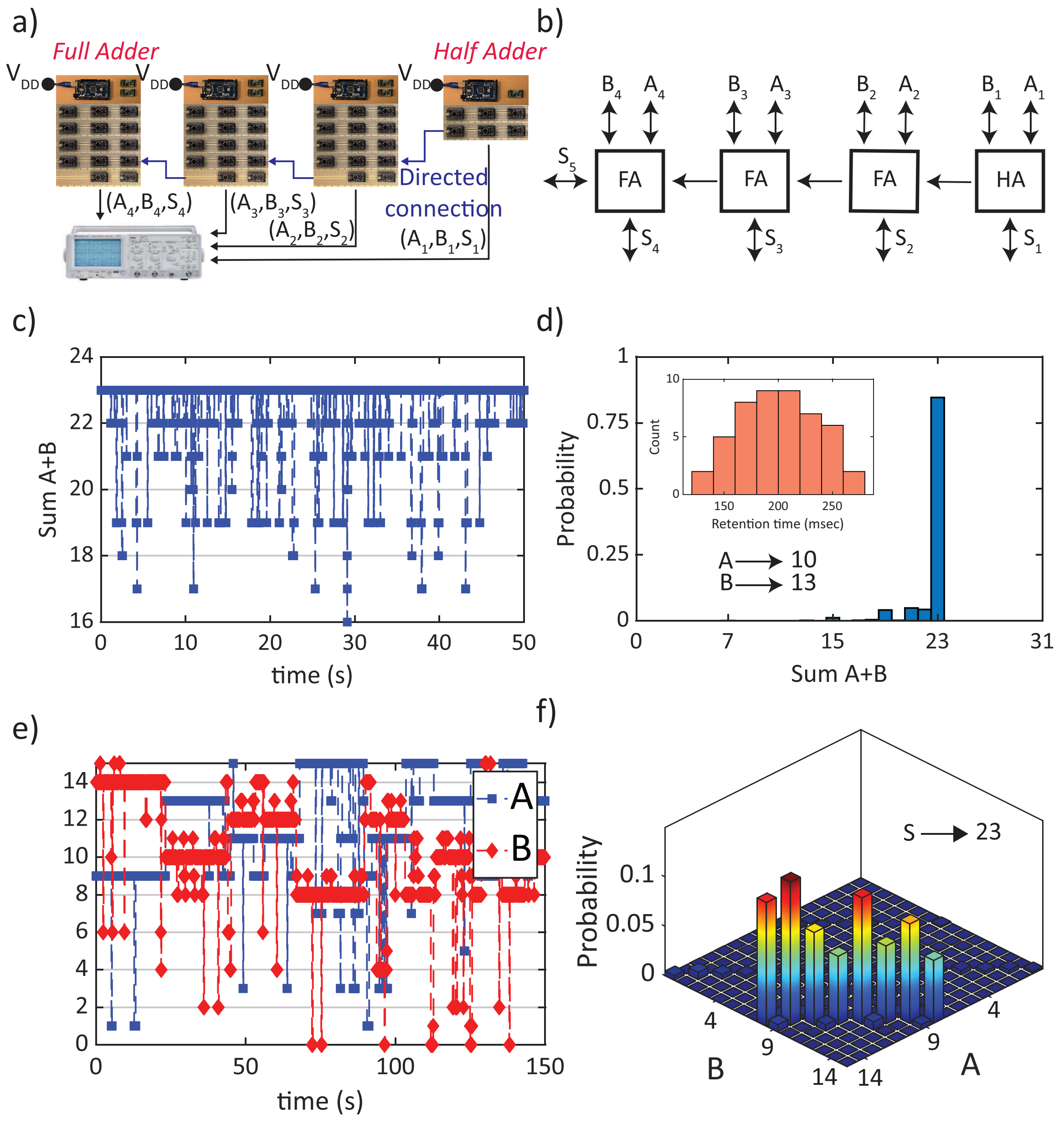


Fig. 2.8. **4-bit Ripple Carry Adder (RCA).** A 4-bit adder is implemented using 4 autonomous p-circuits using a total of 48 p-bits (3 full adders and one half adder) shown in the schematic in **(a)** and block diagram in **(b)**. Each of the 48 p-bits is given a slightly different retention time τ_N (inset of **(d)**). **(c-d)** shows the 4-bit adder working in the standard direct mode of operation where it adds two numbers. However, **(e-f)** shows the inverted mode in which the output S is clamped to 23. The inputs A and B fluctuate between all 8 combinations consistent with a sum of 23.

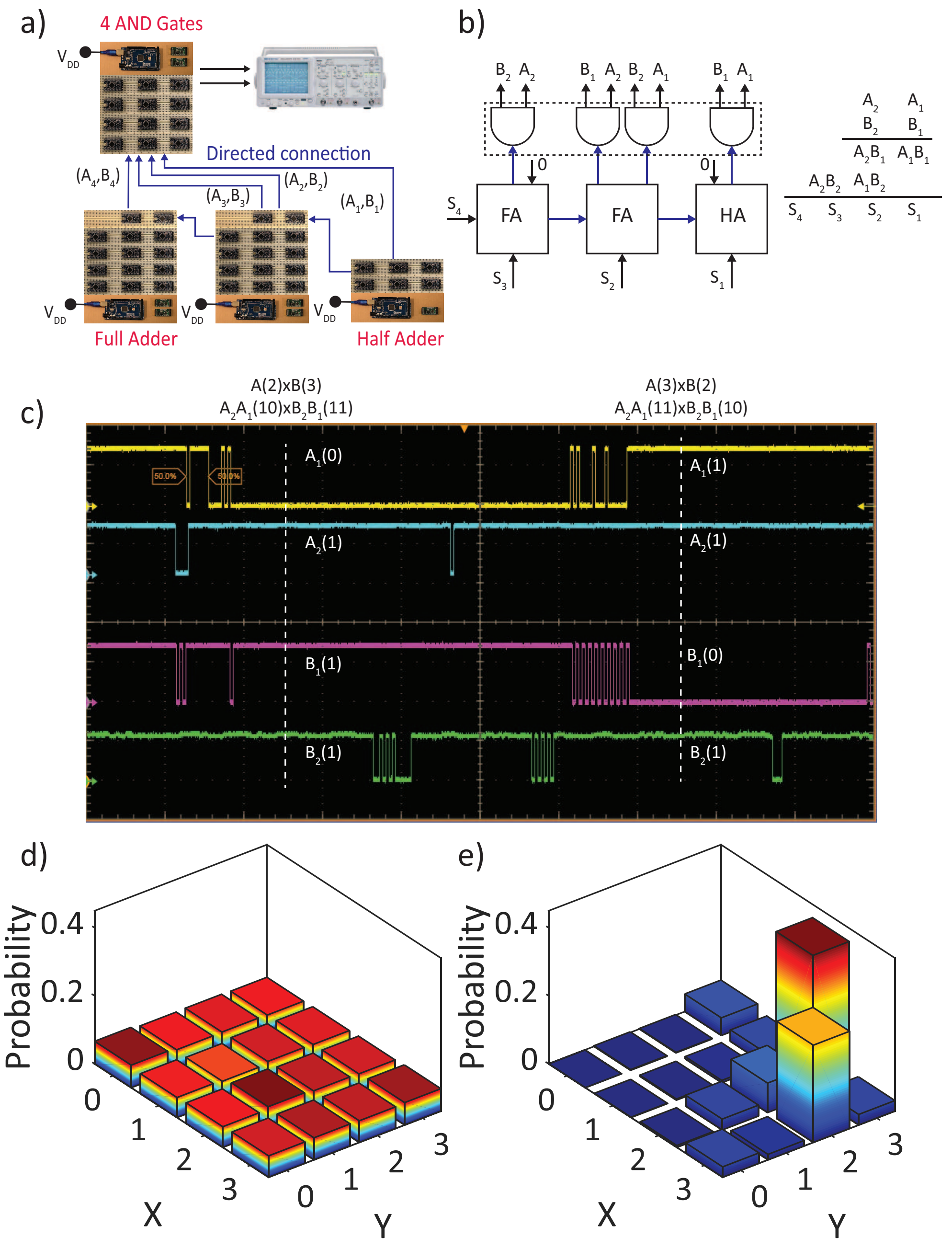


Fig. 2.9. **4-bit multiplier/factorizer.** A 4-bit multiplier is constructed out of 3 Full Adders and 4 AND gates. The schematic and a block diagram are shown in **(a)** and **(b)**. The multiplier works in the inverted mode operates as a factorizer. When the multiplier is left in the uncorrelated state the system fluctuates between the 2^4 states. When a product of 6 is clamped the multiplier now fluctuates between the only two combinations ($2 \times 3 = 3 \times 2$) consistent with a product of 6.

3. INTEGER FACTORIZATION USING STOCHASTIC MAGNETIC TUNNEL JUNCTIONS

Materials in this chapter have been extracted verbatim from the paper: W. A. Borders, **A. Z. Pervaiz**, S. Fukami, K. Y. Camsari, H. Ohno & S. Datta (2019). Integer Factorization using stochastic Magnetic Tunnel Junctions. Nature 573, no. 7774 (2019): 390-393.

3.1 Introduction

The field of adiabatic quantum computing [15] (AQC) solves complex optimization problems by constructing networks of qubits in which the inter-qubit interactions are engineered to make the overall energy E reflect the cost function for the problem. One such algorithm [16] frames integer factorization of a given number F as an optimization problem by writing each of its factors X and Y in binary form and defining the cost function $E = (XY - F)^2$

$$E(x_p, \cdots, x_1\,; y_Q, \cdots, y_1) = \left[\Big(\sum_{p=0}^{P} 2^p x_p\Big)\Big(\sum_{q=0}^{Q} 2^q y_q\Big) - F\right] \tag{3.1}$$

with $x_0 = 1$, $y_0 = 1$ and P, Q denoting the number of bits needed to represent X and Y, respectively, so that the lowest energy state corresponds to the configuration of qubits $\{x_p, \cdots, x_1, y_q, \cdots, y_1\}$ that makes XY equal to F.

In general, E involves terms of the form $x_p y_q x_r y_s$, requiring up to four-body interactions. This algorithm does not require coherence, but needs auxiliary bits to represent many-body interactions when implemented using AQC [12, 17]. In probabilistic computing, many-body interactions are implemented electrically, removing the need for extra components.

Individual p-bits are stochastic building blocks with a normalized output m_i that takes on the values 0 and 1 with probabilities P_0 and P_1, respectively. These probabilities are controlled by their normalized inputs I_i; for $I_i = 0$ they are equal $\big(P_0 = P_1 = 0.5\big)$, large $+I_i$ pins the output m_i to 1 $\big(P_0 = 0, P_1 = 1\big)$ and large $-I_i$ pins m_i to 0 $\big(P_0 = 1, P_1 = 0\big)$. This is similar to the behavior of a binary stochastic neuron, a well known concept in the field of stochastic neural networks and machine learning [18], which has an inputoutput relation $m_i = \vartheta\big[\sigma(I_i) - r\big]$, where ϑ is the unit step function, σ is the sigmoidal function, r is a random number uniformly distributed between 0 and 1, and the input I_i is obtained from the synaptic function (described below). Thus, the p-bit requires a natural element that is substantially unstable but controllable. A magnetic tunnel junction (MTJ), widely recognized as a critical building block of nonvolatile magnetoresistive random-access memory (MRAM) [16, 19], has potential to be used as the stochastic element in p-bits [20] if its thermal stability can be sufficiently reduced. In this work, we build stochastic MTJs and demonstrate an experimental proof of concept of probabilistic computing, in which an eight-p-bit network performs integer factorization of values up to 945.

The building block of the p-bit, the MTJ, comprises ferromagnetic free and reference layers separated by an insulating tunnel barrier Fig. 3.1a. Previous studies have used the switching probability [21] and fluctuation rate [22] of the free-layer magnetization of separate MTJs to show random-number generation and population coding, respectively. Here we show that complex optimization problems can be generally addressed using the correlation among multiple naturally stochastic MTJs. The stack consists of a CoFeB/MgO structure with a perpendicular magnetic easy axis [23], a de facto system of MRAM technology (see Methods section MTJ fabrication). In general, an MTJ is characterized by its tunneling magnetoresistance, which switches between high and low values by varying the angle between the magnetization direction of the two ferromagnetic layers [24]. The high (antiparallel, AP) and low (parallel, P) resistance states (R_{AP}, R_P) are separated by an energy barrier E such that stored information is retained for a time $\tau = \tau_0 exp\big[E/(k_B T)\big]$ following Arrhenius law, where τ_0 is the attempt time $\big(\tau_0 \approx 1ns\big)$ [8], k_B is the Boltzmann constant and T is the temperature (Fig. 3.1b). Nonvolatile memory applications require stable MTJs with a retention time τ of the order of years [19], whereas our p-bit experiments require stochastic MTJs with retention times on the millisecond scale. Fig. 3.1c shows the measured τ as a function of the CoFeB free-layer thickness for different nominal diameters of the MTJ pillar. For each junction diameter D (CoFeB thickness tCoFeB), the timescale of stochasticity decreases with increasing t_{CoFeB} (decreasing D).

The behaviour is understood by considering the energy barrier for magnetization reversal. Because interfacial magnetic anisotropy is dominant in this system [23], increasing the free-layer thickness will reduce the total perpendicular magnetic anisotropy energy, mainly owing to an increase in the demagnetizing energy, which favours in-plane magnetization. Furthermore, decreasing D also decreases the energy barrier for magnetization reversal, as reported in previous studies [26]. Importantly, by varying only the ferromagnetic free-layer thickness for arbitrary sizes of the MTJs used in typical MRAM fabrication, we are able to manipulate the stochasticity of the MTJ so that it is suitable for p-bit experiments (see Methods section MTJ characterization).

To form the building block for stochastic neural networks, we connect the stochastic MTJs with standard n-type metal-oxide-semiconductor (NMOS) transistors to obtain a three-terminal p-bit (Fig. 3.2a). The output voltage for the i^{th} p-bit, $V_{OUT,i}$,

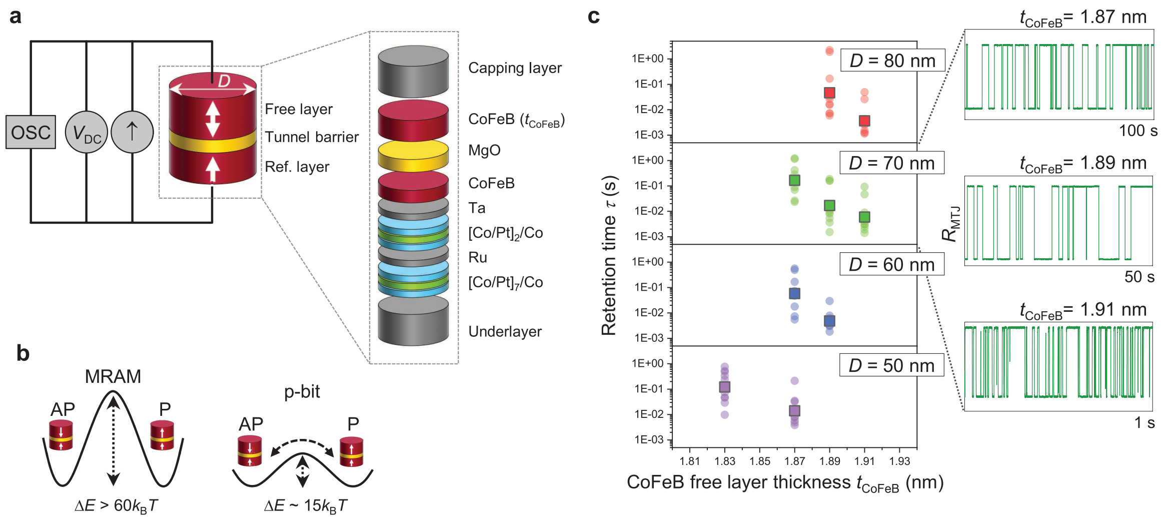


Fig. 3.1. **Characteristics of stochastic magnetic tunnel junctions.** **(a)** Measurement setup of a stochastic MTJ, with a stack structure that is only slightly modified from current MRAM technology. A current is passed from the free layer to the reference layer, a time-averaged signal is read on the voltmeter, and a time-domain signal is measured on the oscilloscope. **(b)** The energy profile between the P and AP states of the magnetization orientation of the MTJ for typical MRAM technology and for the MTJs used in the p-bits for this work. **(c)** Experimental results showing the retention time τ of MTJs with varying thickness of the CoFeB free layer t_{CoFeB} and diameter D. The retention time τ is determined at an applied current of $I_{50/50}$, which induces equal fluctuation time of the MTJ magnetization in the AP and P states. Square symbols represent the average of the retention time for 10 MTJs at each D and t_{CoFeB}. Transparent circles represent the retention time for each device. The right-most panels show the effect of varying the free-layer thickness on the stochasticity for devices of the same size. Note that reducing the thickness below 1.8nm results in a stable binary device suitable for nonvolatile memory applications [25]. The MTJs were prepared at Tohoku University by William A. Borders, Professor Shunsuke Fukami and Professor Hideo Ohno.

from this composite unit can be written in terms of the input voltage $V_{IN,i}$ in a form similar to the ideal binary stochastic neuron described above:

$$\frac{\overbrace{V_{out,i}}^{m_i}}{V_{DD}} \approx \vartheta\left\{\sigma\left(\overbrace{\frac{V_{in,i} - v_{0,i}}{V_{0,i}}}^{I_i}\right) - r\right\} \tag{3.2}$$

where V_{DD} is the supply voltage, $V_{0,i}$ is the scaling voltage determined by the transistor, $v_{0,i}$ is the offset voltage (1.95 V in this experiment). Fig. 3.2b shows the time-averaged output voltage as the input voltage is swept from 1.5 V to 2.4 V, where each point is averaged over 700 ms with a fixed input voltage. Fig. 3.2c shows the time-varying output voltage for specific input voltages, displaying stochastic behaviour centered at 1.95 V, but becoming deterministic as the input changes by about 75 mV, a consequence of spin-transfer torque [27–29] (see Methods section "p-bit construction").

These p-bits can be used to perform useful functions by interconnecting them so that the i^{th} p-bit is driven by a synaptic input I_i that is a function of all the other outputs $\{m_1, \cdots, m_N\}$. Boltzmann machines represent a subset of such networks for which I_i can be obtained from an energy function E using the relation $I_i = -\partial E(m_1, \cdots, m_N)/\partial m_i$. Such networks will visit different configurations with probabilities given by the Boltzmann law $P\big(m_1, \cdots, m_N\big)$, which are proportional to $exp\big[-E(m_1, \cdots, m_N)\big]$, so configurations with the lowest energy E occur with the highest probability. This property makes the networks naturally suited for solving optimization problems, similar to the way that AQC solves them, where the correct solution minimizes a cost function identified for E and is used to calculate the synaptic inputs I_i. Unlike in machine-learning schemes, these synaptic inputs are analytically deduced and not learned.

Experimentally we connect eight p-bits following a general architecture presented previously [30] (Fig. 3.3a). A microcontroller reads the output voltage of each p-bit and is programmed to calculate the inputs I_i for a given cost function E. The result is converted into analogue voltages using a digital-to-analogue converter (DAC). Together, the microcontroller and DAC function as the synaptic weight logic that determines I_i, reading in digital outputs from the p-bits and feeding back analogue

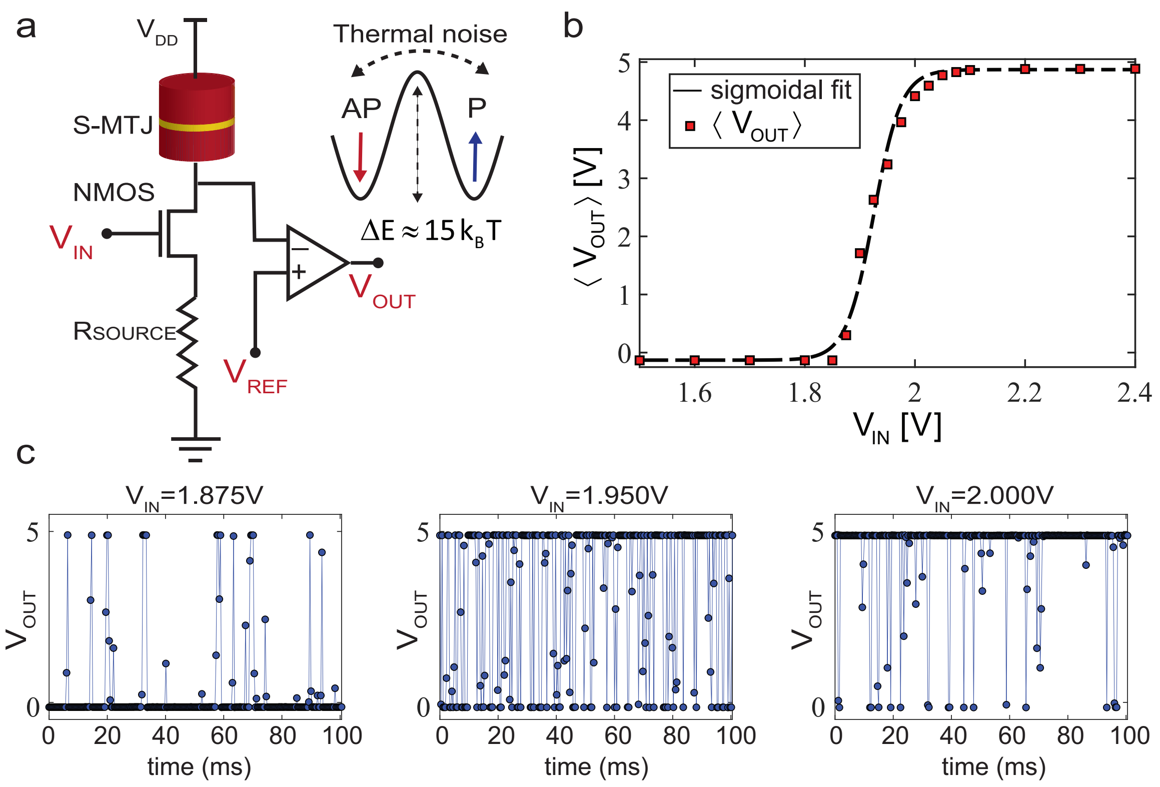


Fig. 3.2. **Experimental demonstration of a p-bit.** **(a)** Electrical schematic of a p-bit using a stochastic MTJ with an NMOS transistor, a comparator and a resistor, extending the design presented in ref. [20] to handle device specific variations. A stochastic MTJ (s-MTJ) has a free layer with a relatively low energy barrier $\left(\Delta E \approx 15k_BT\right)$ so that thermal noise makes it fluctuate between its stable states, one being parallel (P) to the fixed layer and the other being anti-parallel (AP). **(b)** Time-averaged V_{OUT}, $\langle V_{OUT}\rangle$, as a function of the applied input, fitted to the sigmoidal function. Each point is averaged over 700 ms with $2,000$ or more sampling points for each data point shown. **(c)** Time snapshots of V_{OUT} for three different inputs V_{IN}, showing the preferred state of a p-bit (high or low) as a function of its input voltage.

inputs (see Methods section "p-circuit construction"). Although the main experiment that we describe here demonstrates integer factorization, this methodology can be applied to other optimization problems, such as invertible Boolean logic, for which the objective is to determine all the possible inputs when the logic output is known (see Methods section "p-bit-based implementation of an invertible AND gate").

In the case of integer factorization, we use the cost function represented by equation (1) to evaluate the input functions. We first test the factorization of 35 using four p-bits $\big(P = 2, Q = 2\big)$ (see Methods section "Factorization algorithm"). In our algorithm, the synaptic inputs include nonlinear terms that effectively enforce both three p-bit and four p-bit interactions, in addition to the customary linear terms arising from two p-bit interactions. Accordingly, an integer up to 2^{n+2} can be encoded according to Eq. 3.1 with n p-bits using the current algorithm, a relation that requires fewer bits than current AQC schemes, mainly owing to the added flexibility provided by nonlinear synapses [17] that could be useful in other optimization problems as well. Figure 3b gives the three-dimensional histograms of the time fluctuations (see Methods section "Factorization algorithm") for pairs of numbers $\big\{x_2, x_1, 1\big\}$ and $\big\{y_2, y_1, 1\big\}$, depicted below the uncorrelated state that is obtained when all input functions are set to zero. Although the p-bits fluctuate independently in the uncorrelated state (top panel), non-zero input to the network results in two peaks observed at $\big(5, 7\big)$ and $\big(7, 5\big)$, showing that 35 is factorized into 5 and 7 correctly (bottom panel). Figure 3c shows the three-dimensional histogram obtained with the input functions appropriate for factorizing 161 using six p-bits with $P = 4$ and $Q = 2$, where the correct factor $(23, 7)$ shows a prominent peak (bottom panel). Similarly, Fig. 3d shows an eight-p-bit network factorizing 945 $\big(P = 5, Q = 3\big)$. Using p-bit models, we also simulate the factorization process and obtain agreement with experimental results using a single fitting parameter (see Methods section "Experiment versus simulation"). We also investigate the influence of varying MTJ parameters such as R_P, R_{AP}, $I_{50/50}$, shift and distortion of the response of MTJs, and retention time τ. Response variations are corrected by adjusting the bias voltage $v_{0,i}$ (see Methods section "Factorization experiment calibration") an variations in the retention time of the MTJs have little effect provided that the synapse is faster than the fastest p-bit (see Methods section

“Effect of p-bit parameter variation on system performance”). Owing to the relative ease of these methods, we expect robust and repeatable results for networks on even larger scales.

Next, we compare the demonstrated probabilistic computing system with its quantum counterpart. The present approach uses an algorithm that is similar to AQC but does not perform annealing, which normally requires coherence. Compared to AQC, the present scheme has a threefold advantage: it operates at room temperature, it can be implemented using existing highly scalable MRAM technology and it is relatively easy to incorporate complex many-body interactions into the scheme. Further, we note that for a subclass of quantum systems, quantum annealing can be approximated with replicated p-bit networks [31]. This class of systems is commonly referred to as stoquastic [15]. The approximation becomes systematically more accurate upon increasing the number of replicas. The increased number of p-bits is offset by their comparably lower implementation costs (see Methods section “Comparison between p-bit and quantum computing”).

Probabilistic computing can also be executed using conventional complementary metal-oxide-semiconductor (CMOS) circuits. Our p-bit implementation uses three transistors and one MTJ, whereas CMOS based probabilistic computing with digital random-number generators (RNGs) requires more than a thousand transistors to perform the same function. A quantitative comparison shows an energy advantage by a factor of 10 and an area advantage by a factor of 300 (see Methods section “Comparison between MTJ-based p-bit and CMOS-based alternatives”).

We should note that there are deterministic algorithms implemented on a fully digital CMOS system that specializes in performing factorization. However, this system takes a substantially greater amount of time to reach the exact solution as the problem size increases [32]. On the other hand, when algorithms that produce approximate solutions are acceptable, there is interest in hardware that enables probabilistic computing methods. Because the purpose of this study was to establish a system that is suitable for solving optimization problems in general, these factors mentioned above are very attractive, particularly considering the energy and surface area advantages.

In summary, this work serves as a proof-of-concept demonstration of an asynchronous probabilistic computer similar to the one envisioned by Feynman [1], which is realized through a slight modification of embedded MRAM technology currently at the level of 8 Mb and above [33] and which could find applications in the areas of optimization, sampling, and machine learning. An important aspect of this demonstration is the asynchronous operation of p-bits without any forced sequencing, unlike typical software implementations of Boltzmann machines, which require individual neurons or p-bits to be updated sequentially [34]. This asynchronous feature allows the parallel operation of a large number of p-bits, leading to an unconventional computing paradigm.

3.2 Methods

3.2.1 MTJ fabrication

The MTJs are fabricated with a stack structure as follows, from the substrate side: Ta(5)/Pt(5)/[Co(0.3)/Pt(0.4)]$_7$/Co(0.3)/Ru(0.45)/[Co(0.3)/Pt(0.4)]$_2$/Co(0.3)/ Ta(0.3)/$Co_{18.75}Fe_{56.25}B_{25}$(1)/MgO(1.1)/$Co_{18.75}Fe_{56.25}B_{25}$($t_{CoFeB}$)/Ta(5)/Ru(5)/ Ta(50) Fig. 3.1a. The numbers in parentheses are the nominal thicknesses in nanometres. The thickness of the free layer of CoFeB, t_{CoFeB}, is adjusted to view the change in the fluctuation of the MTJ magnetization. All films are deposited on a thermally oxidized silicon substrate by d.c. and radio frequency magnetron sputtering at room temperature. The stacks are then processed into circular MTJs with nominal junction

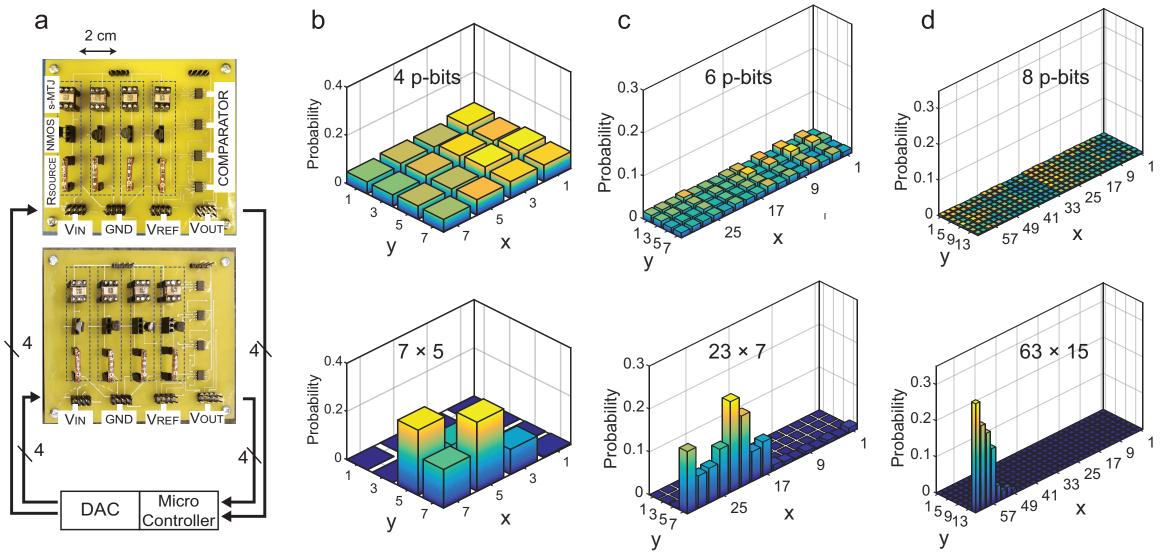


Fig. 3.3. **Experimental demonstration of integer factorization.** **(a)** A photograph of a printed circuit board for an eight-p-bit circuit, interconnected through a microcontroller and a DAC. **(b-d)**, The uncorrelated (top) and correlated (bottom) state of the system when four, six and eight p-bits are used to factorize $35 = 5 \times 7 = 7 \times 5 (P = 2, Q = 2$ with four p-bits) **(b)** $161 = 23 \times 7 (P = 4, Q = 2$ with six p-bits) **(c)** and $945 = 63 \times 15 (P = 5, Q = 3$ with eight p-bits) **(d)**. The x and y axes show the factors X and Y (see Methods section Factorization algorithm). All statistics are taken over a window of 15 s with over 2,000 sampling points. Each separate factorization experiment was performed more than twice to ensure reproducibility.

size varied from 40 to 80 nm in diameter by electron beam lithography and argon ion milling. The samples are annealed at 300°C in vacuum for an hour under a 1.2 T perpendicularly applied magnetic field. MTJs are then cut out from wafers and bonded with wires to IC sockets to be placed in the p-bit circuit board. The MTJs were prepared at Tohoku University by William A. Borders, Professor Shunsuke Fukami and Professor Hideo Ohno.

3.2.2 MTJ characterization

First, the MTJ resistance is measured by sweeping the current from negative to positive values, and the time-averaged and high-frequency signals are read across a voltmeter and oscilloscope, respectively (Fig. 1a). We measured an approximate tunnel magnetoresistance ratio of 100% fluctuating between $R_P = 7 - 11k\Omega$ and $R_{AP} = 12 - 19k\Omega$. The current at which the resistance switches by half is determined to be $I_{50/50}$, which is the bias current at which the MTJs will spend equal time in the AP and P states. To determine τ, we perform retention time measurements [35] when the MTJ is in either the AP or the P state using voltage measurements from the oscilloscope (Fig. 3.1b). To ensure reliable collection of data, each point is measured with a constant current on the oscilloscope at a sampling rate set ten times faster than the fastest fluctuation time of the MTJ. The retention time values are determined from approximately 1,000 to 10,000 switching events per device. The retention times used in this work range from 1 ms to 100 ms, which is suitable to match with the sampling rate of the microcontroller and DAC used to determine the inputs for each p-bit. For these purposes, we choose a free-layer thickness of 1.9 nm and different MTJ diameters (Fig.3.1c). The MTJs were characterized at Tohoku University by William A. Borders and Professor Shunsuke Fukami.

3.2.3 p-bit construction

The p-bit is constructed following the circuit proposed previously [20] with two changes to the design: First, we use an additional resistance R_{SOURCE} attached to the source of the NMOS transistor to restrict the current through the MTJ branch to values in the stochastic range around $I_{50/50}$ (which is around $510\mu A$). This produces voltage fluctuations $V \approx 3050mV$. On the basis of measured values of $I_{50/50}$, and R_P, R_{AP} for every MTJ, an R_{SOURCE} value for each MTJ is calculated according to:

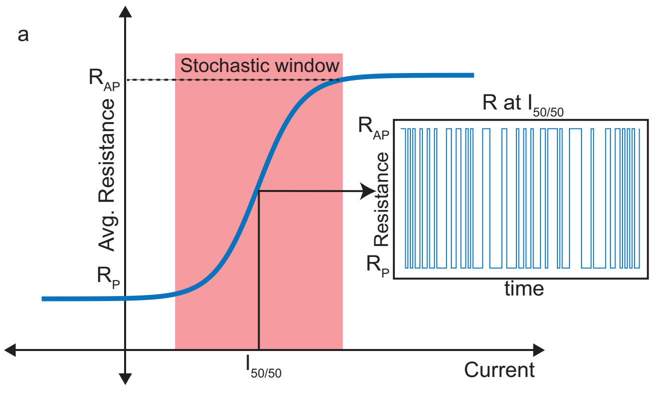
$$R_{SOURCE} = \frac{V_{DD}}{I_{50/50}} - R_{NMOS} - \frac{R_{AP} + R_P}{2} \tag{3.3}$$

where V_{DD} is the supply voltage and R_{NMOS} represents the drain-to-source resistance of the NMOS transistor.

Extended Data Fig. 1b shows the measured R_{NMOS} versus V_{IN} characteristics for a 2N7000 (T0-92-3 package) NMOS with drain resistance $R_D = 9.8k\Omega$, source resistance $R_S = 9.6k\Omega$ and $V_{DD} = 200mV$ to mimic the p-bit circuit used in our experiment. The value of R_{NMOS} is chosen so that the p-bit is centred at $V_{IN} = 1.95V$, as shown in Fig. 3.2b. This value of V_{IN} is optimized considering the transistor characteristics. A smaller value of V_{IN} makes the sigmoidal characteristics sharper because the current through the MTJ changes rapidly for small changes in V_{IN}, pinning the MTJ. If we choose values of V_{IN} greater than 1.95 V, the p-bit does not get saturated properly to VDD.

Second, to achieve better gain, we use comparators (AD8692, 8-SOIC package) instead of the inverters used previously [20]. The drain of the NMOS is connected to the negative terminal of the comparator and a voltage V_{REF} is given as an input to the positive terminal. The comparator has a biasing current of 1 pA, which is 3-4 orders of magnitude lower than the current passing through the MTJ, ensuring that it does not load the MTJ branch. V_{REF} is chosen so that when $I_{50/50}$ is flowing through the MTJ branch, V_{REF} is centered at the drain voltage V_{DRAIN}. Under these conditions, V_{REF} can be calculated according to:

$$R_{REF} = V_{DD} - I_{50/50}\left(\frac{R_{AP} + R_P}{2}\right) \tag{3.4}$$



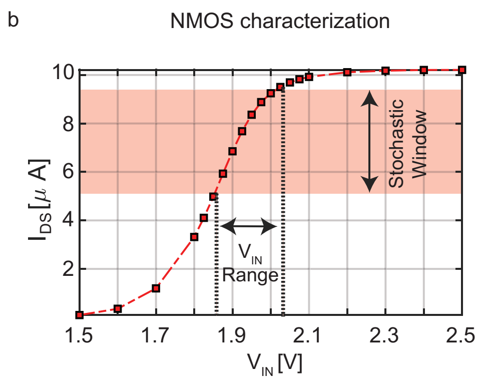


Fig. 3.4. **p-bit construction.** **(a)** A diagram of the ideal response of a stochastic MTJ as used in this work and the parameters used to characterize the MTJ. **(b)** The measured drain current I_{DS} as a function of V_{IN} of a 2N7000 NMOS transistor used in our p-bit demonstration.

3.2.4 p-circuit construction

We have constructed our p-circuits following the general architecture described previously [30] which is shown in Fig. 3.5. An Arduino microcontroller (Mega 2560) is used to read the output voltages of each p-bit as binary inputs and is programmed to implement the synaptic weights. These are then converted into analogue voltages using a DAC (PMOD DA4) that has eight channels, each with 12-bit resolution. The DAC also has an internal $2.5V$ reference allowing a resolution of $2.5/4096 \approx 6.1mV$. An important design consideration is to ensure that the interconnect delay—that is, the time it takes to update the inputsis shorter than the retention time of the p-bits [30] (see Methods section “Effect of p-bit parameter variation on system performance”). The DAC uses a Serial Peripheral Interface (SPI) protocol to communicate with the microcontroller and has a worst-case interconnect delay of $150\mu s$ for eight p-bits, which is lower than the retention time of the MTJs used in this manuscript. We use an oscilloscope (MSO-X-3014T, Keysight) to collect the output voltages for

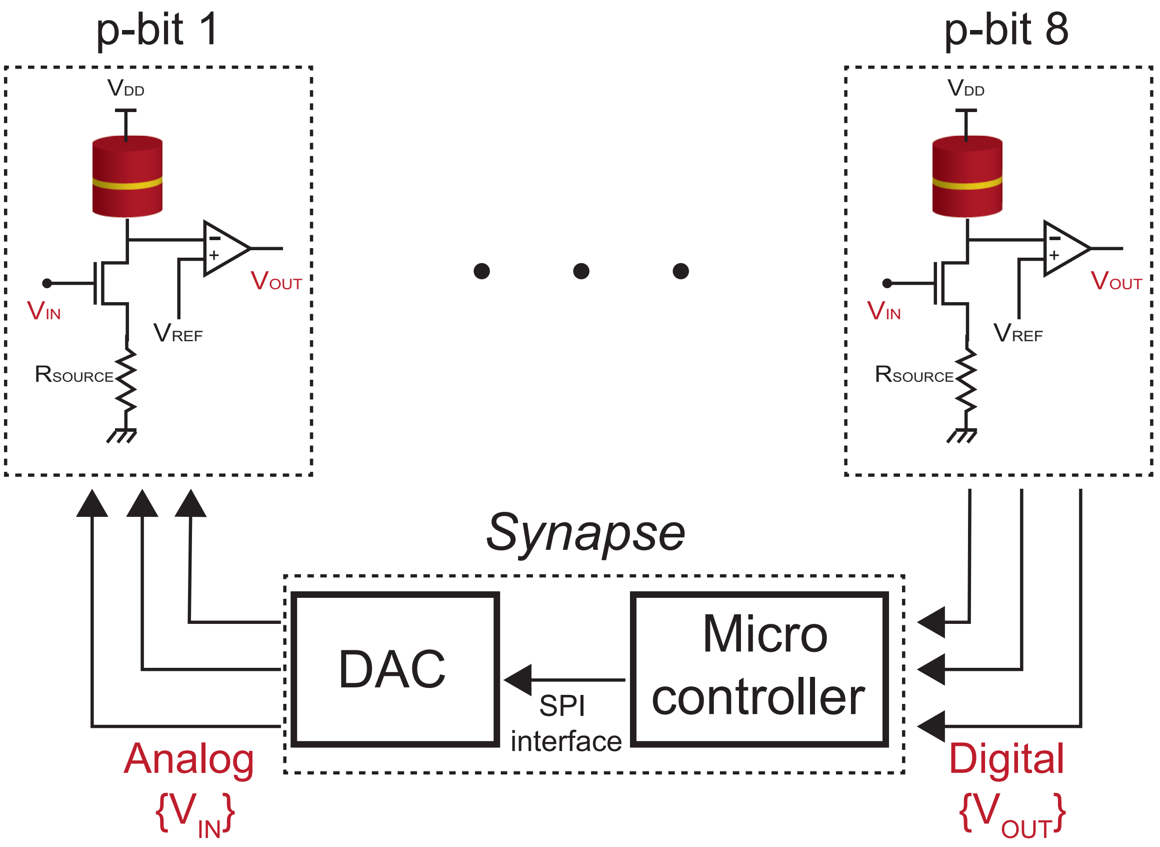


Fig. 3.5. **Block diagram of an asynchronous p-circuit. (a)** A microcontroller reads the outputs voltages V_{OUT} of all p-bits and computes the synaptic weights, which are then converted to the analogue input voltages V_{IN} for each p-bit, using a DAC that communicates with the microcontroller.

all p-bits. The oscilloscope can read up to 16 digital voltages and is connected to a computer using the Keysight BenchVue oscilloscope software.

3.2.5 Factorization algorithm

To minimize the cost function E, we construct a network of binary stochastic neurons with the *ith* neuron driven by an input I_i obtained from evaluating $-\partial E(m_1, \cdots, m_N)/\partial m_i$, where m_i is the output of the i^{th} neuron. This approach is similar in spirit to AQC [16] and a large amount of effort has gone into identifying appropriate cost functions for different problems of interest [36]; many of these formulations can also be adapted to design p-bit networks. The optimization-problem-based approach in this scheme is different from those in previous studies [30, 37], in which integer factorization is cast as an inverse multiplication problem, which typically requires more p-bits to factor numbers of the same size.

For each number that is factored, the corresponding function is programmed into the synaptic function Ii, as explained below.

We start from a cost function of the form in equation (1) [17, 38–41], which is simplified to:

$$E\big(x_p,\cdots,x_1;y_Q,\cdots,y_1\big) = F^2 + \sum_{p,q} x_p y_q\big(2^{2p+2q} - 2^{p+q+1}F\big) + \\ \sum_{p,q,s\neq q} 2^{2p+q+s} x_p y_q y_s + \sum_{p,q,r\neq p} 2^{p+2q+r} x_p x_r y_q + \sum_{p,q,r\neq p,s\neq q} 2^{p+q+r+s} x_p y_q x_r y_s \quad (3.5)$$

using the property of binary digits that $b^2 = b$.

In this cost function, the numbers X and Y are assumed to be odd numbers, because large semiprimes of interest are always odd; this is implemented by setting x_0 and y_0 to 1. For a four-p-bit network, P = 2 and Q = 2 so that the cost function for F = 35 from Eq. 3.5 is obtained as below, where I_0, an arbitrary constant that controls overall strength of coupling, is chosen to be 1.

$$E = -0.3x_1 - 0.7x_2 - 0.3y_1 - 0.7y_2 - x_2y_1 - 1.4x_2y_2 - 0.6x_1y_1 \\ - x_1y_2 + 0.3x_1y_1y_2 + x_2y_1y_2 + x_2y_1y_2 + 0.3x_1x_2y_1 + x + 1x_2y_2 + 0.7x_1x_2y_1y_2 \quad (3.6)$$

where the coefficients are rounded off to have one significant digit. By evaluating $-\partial E(m_1,\cdots,m_N)/\partial m_i$, we obtain the input functions I_i:

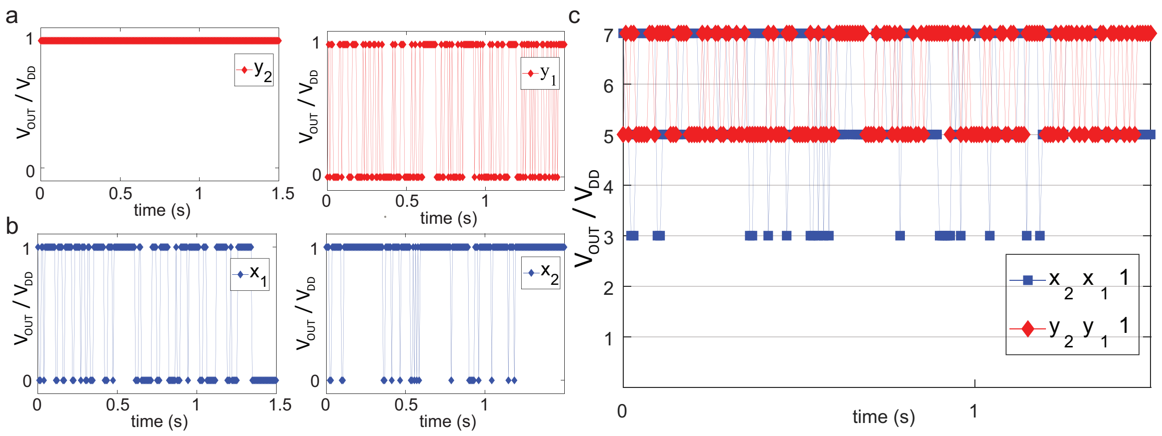


Fig. 3.6. **Experimentally observed time snapshots.** **(a-c),** Experimentally observed time snapshots of the four p-bits used to factorize 35 **(a,b)**. These snapshots are combined to create x and y **(c)**, which fluctuate between 7×5 and 5×7.

$$I_{x2} = 0.7 + 1.0y_1 + 1.4y_2 - 1.0y_1y_2 - 0.3x_1y_1 - 1.0x_1y_2 - 0.7x_1y_1y_2 \tag{4.7a}$$

$$I_{x1} = 0.3 + 0.6y_1 + 1.0y_2 - 0.3y_1y_2 - 0.3x_2y_1 - 1.0x_2y_2 - 0.7y_1x_2y_2 \tag{4.7b}$$

$$I_{y1} = 0.3 + 0.6x_1 + 1.0x_2 - 0.3x_1y_2 - 1.0x_2y_2 - 0.3x_1x_2 - 0.7x_1x_2y_2 \tag{4.7c}$$

$$I_{y2} = 0.7 + 1.0x_1 + 1.4x_2 - 0.3y_1x_1 - 1.0y_1x_2 - 1.0x_1x_2 - 0.7x_1y_1x_2 \tag{4.7d}$$

Similar cost functions—but with many more termscan be obtained for the eight p-bit experiment in which P = 4 and Q = 4. These cost functions and the resulting input functions are not listed here but are available upon request from the authors. Fig. 3.6 shows the output of four p-bits x_2, x_1, y_2, and y_1 as a function of time, which are then used to collect the statistics shown in Fig. 3.3b.

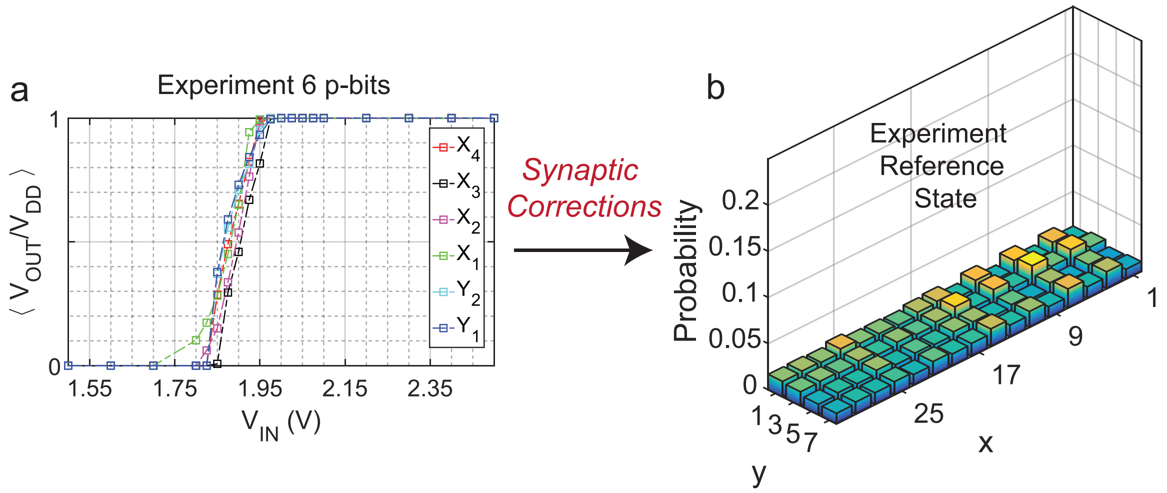


Fig. 3.7. **Calibrating the experimental system.** **(a)** Calibrating a reference state using synaptic weights. **(a)** The experimentally observed time-averaged output of six p-bits versus applied inputs (which are misaligned).**(b)** The output is corrected using synaptic biases leading to the reference state shown. Each data point in a and b are taken as an average over a time window of 15 s with 2,000 or more sampling points.

3.2.6 Factorization experiment calibration

We begin by establishing an uncorrelated state for the p-circuit as a reference for the experiment. To offset variations, we first measure the average sigmoidal response of each p-bit used in our experiment. Fig. 3.4 shows six such responses (15 second averages per point) for the six p-bits used in our experiment. Initially, we choose a value for R_{SOURCE} so that each sigmoid is centered at 1.95 V, and measure the average output. Any shifts in the average outputs from 1.95 V (due to variations in transistor characteristics and MTJ parameters) are adjusted as individual synaptic biases to center the average response. Once these are set to obtain average responses that are aligned, they are not varied and an uncorrelated state for the system is established, as shown in Fig. 3.2 and in Fig. 3.7b. After establishing the reference state, only the interconnect strengths between p-bits are changed for the remainder of the experiment.

3.2.7 Comparison between MTJ-based p-bit and CMOS-based alternatives

As noted in equation (2), the MTJ-based p-bit used in this work evaluates the function $mi = \vartheta\big(\sigma(I_i) - r\big)$. Below we compare this evaluation to a digital-CMOS-based evaluation of the same function. As mentioned in the main text, the problem of factorization can be addressed with fully digital deterministic algorithms that do not require this function. However, the aim of this work is to demonstrate a broad approach to optimization and sampling problems using a network of p-bits interacting asynchronously, in which high precision is not the primary figure of merit. With this in mind, we do not consider the deterministic algorithm and present below a functionality-based comparison between MTJ-based and CMOS-based probabilistic computers. To evaluate the same function $mi = \vartheta\big(\sigma(I_i) - r\big)$ digitally using CMOS, one could use [42,43] an RNG for r, a look-up table for $\sigma(I_i)$ or a comparator for the step function ϑ.

In this section, we compare the energy and area of a CMOS-based pseudo-random-number generator (PRNG) to the MTJ-based p-bit (Extended Data Fig. 5). The look-up table and comparator would further add to the area of the CMOSbased p-bit. However, we note that the MTJ-based p-bit requires a DAC to interface with digital synapses. In principle this would not be needed for synapses implemented with analogue devices.

Fig. 3.8 shows that the CMOS-based PRNG requires an energy consumption an order of magnitude higher and requires an area several orders larger compared to the MTJ-based p-bit in this work. Details of the models used are described below.

CMOS-based RNG. True RNGs operate specialized circuits using thermal noise from CMOS-based sources such as cross-coupled inverter pairs to produce true random bits [44]. However, inducing true randomness in conventional hardware typically requires high levels of energy consumption and large cell area. On the other hand, a PRNG-based approach that uses linear-feedback shift registers (LFSRs) offers a low-cost solution at the expense of reduced random bit quality [42].

We implement a 32-bit LFSR to form the PRNG that is composed of 32 D-type flip flops with three separate two-input XOR gates. Each XOR requires 14 transistors. Each D-type flip flop is composed of 36 transistors, which includes eight NAND gates (four transistors each) and two inverters (two transistors each). Therefore, the 32-bit LFSR requires 1,194 transistors in total. Each transistor is implemented using a minimum size (nfin = 1) 14 nm high performance fin field effect transistor HP-FinFET model obtained from a predictive technology model [45]. The details of the LFSR are shown in Fig. 3.8. This circuit is simulated in the HSPICE circuit-simulator software with a clock frequency of 10 GHz ($\tau_{CLK} = 100ps$). We note that because we are computing the energy per random bit, we average the active power over many clock cycles and so the exact clock frequency that is used in the circuit becomes irrelevant. The energy per random bit is obtained by integrating the total supply current (multiplied by the supply voltage) over one clock cycle. The energy per random bit for the 32-bit LFSR is about 20 fJ, as shown in Fig. 3.8.

MTJ based p-bit For the MTJ-based p-bit simulation, we use the design proposed previously [46] with an MTJ of negligible energy barrier and with an autocorrelation time of about 100 ps for an arbitrarily chosen magnetization direction denoted as m(t). The MTJ is modelled as a variable conductance with $G_{MTJ}(t) = G_0\big[1+m(t)TMR/(2+TMR)\big]$, where TMR is the tunnelling magnetoresistance with a value of around 110%, close to the experimental value of TMR in our experiments. The average MTJ conductance G_0 (where $G_0^{-1} = 23.4k\Omega$) is chosen to match the transistor conductance when $V_{IN} = 0$. This makes the sigmoidal response of the p-bit symmetric around zero. The instantaneous magnetization $m(t)$ is calculated by a stochastic Landau-Lifshitz-Gilbert (LLG) equation solver as a separate circuit in HSPICE. The stochastic LLG solver takes spin current as an input and produces $m(t)$ at each time step. The spin current is assumed to be proportional to the instantaneous charge current flowing through the MTJ, multiplied by a spin polarization P that in turn is assumed to be related to TMR by $TMR = 2P^2/(1 - P^2)$ ref. [47].

The energy per random bit for the MTJ-based p-bit is calculated by computing the average power drawn from the supplies, $V_{DD} \times (I_{SUPPLY1} + I_{SUPPLY2})$, for a given period (t = 100 ns) and multiplying this average by the autocorrelation time of the low-barrier magnet to estimate the energy per random bit to be about 2 fJ per random bit. Fig. 3.8c shows the difference in energy per random bit and the transistor count for the p-bit-based and hardware CMOS-based schemes.

3.2.8 Comparison between p-bit and quantum computing

The optimization algorithm used in this work is similar to an AQC algorithm that can run on quantum computing hardware. It has been shown [31] that a system of x qubits, if they belong to a class of stoquastic [15] systems, can be efficiently emulated with $x \times r$ p-bits when using the Suzuki-Trotter decomposition, where r (about 10-100) is the number of replicas, each comprising x p-bits. Increasing the number of replicas systematically reduces the error compared to the exact solution; the increased number of p-bits is offset by their relative cheapness. Although many groups are working towards implementing 1,000 qubits, p-computers with density around 1 Gb could be a relatively near-term goal using embedded MRAM technology operating at room temperature. However, we note that the replicated p-bit approach to quantum computing is established only for a subset of quantum Hamiltonians that do not suffer from the 'sign-problem' associated with negative probabilities, and are commonly referred to as 'stoquastic' [31].

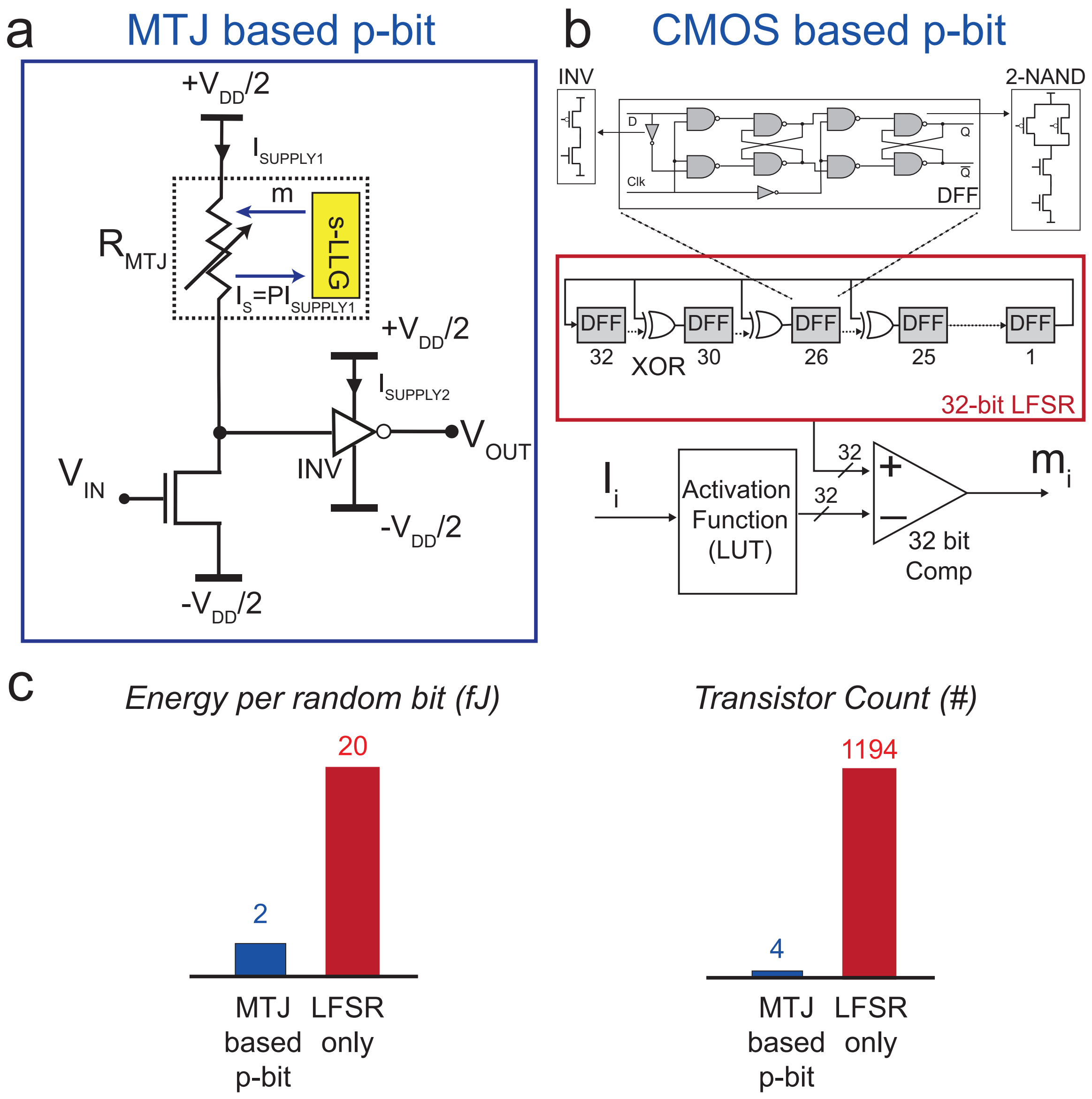


Fig. 3.8. **Comparison between the MTJ- and CMOS-based energy per random bit and cell area. (a)** An MTJ-based p-bit simulated with the stochastic LLG model (s-LLG, dotted box). **(b)** A 32-bit LFSR. The look-up table (LUT) and the digital comparator of the CMOS p-bit are not included in the comparison. INV, inverter; DFF, D-type flip flop.

A recent experiment [17] performed on a D-Wave machine (D2000Q) used the same factorization algorithm—but with additional qubits to reduce the problem to two-body interactions—and factored 15 and 21 using four logical qubits, and factored

143 using 12 logical qubits. In general, $O(log_2(F))$ logical qubits are required to factor an integer F. The increased number of qubits is a result of additional logical qubits in the Hamiltonian used to reduce the problem. By contrast, our demonstration factors numbers up to 945 with eight p-bits at room temperature and is estimated to be able to factorize 2^{n+2} sized integers, with n p-bits.

Comparison of AQC and p-bits We first describe the typical system—the transverse Ising Hamiltonian—that demonstrates an AQC algorithm for factorization and then present an emulation of this system with p-bits.

We show in Fig. 3.9 that the results of an exact solution of the quantum many-body Hamiltonian can be accurately obtained by a replicated network of p-bits. The transverse Ising Hamiltonian for the factorization problem H_Q is given as:

$$H_Q = \Bigg(\sum_{i<j} J_{ij}\sigma_i^z\sigma_j^z + \sum_{i<j<k} K_{ijk}\sigma_i^z\sigma_j^z\sigma_k^z + \sum_{i<j<k<l} L_{ijkl}\sigma_i^z\sigma_j^z\sigma_k^z\sigma_l^z \\ + \Gamma_X \sum_i \sigma_i^x \tag{4.8}$$

where J_{ij}, K_{ijk} and L_{ijkl} represent the interactions obtained from the cost function $E = \big(XY - F\big)^2$ in equation (1), and Γ_X is the (dimensionless) transverse magnetic field that is used as an annealing parameter. The quantum system described in Eq. 4.8 can be mapped to a classical system with networks of p-bits. The classical Hamiltonian H_C for a classical system with r replicas is expressed as:

$$H_C = -\Bigg(\sum_{n=1}^{n=r}\sum_{i<j} \frac{J_{ij}}{r} m_{i,n}m_{j,n} + \sum_{n=1}^{n=r}\sum_{i<j<k} \frac{K_{ijk}}{r} m_{i,n}m_{j,n}m_{k,n} \\ + \sum_{n=1}^{n=r}\sum_{i<j<k<l} \frac{L_{ijk}}{r} m_{i,n}m_{j,n}m_{k,n}m_{l,n} + \sum_{n=1}^{n=r}\sum_{i} J_{\perp} m_{i,n}m_{i,n+1} \Bigg) \tag{4.9}$$

where $J_\perp$ is the local transverse coupling between replicas with periodic boundary conditions; $J_\perp = -1/(2\beta)ln\big[tanh(\Gamma_X\beta/r)\big]$ where β is the dimensionless inverse temperature.

In AQC, the system is prepared at a low temperature and the transverse magnetic field starts from a high value to initialize the system in its ground state. The magnetic field is then slowly reduced to keep the system in its ground state so that the ground state of the classical Ising Hamiltonian is reached.

Here, instead of performing annealing that requires a continuous change of the transverse magnetic field, we perform two static simulations, for factoring $161 = 23\times 7$ using a small Γ_X (corresponding to a 'cold' system close to the ideal solution) and using a large Γ_X (corresponding to a 'hot' system close to thermal equilibrium).

We compare the results obtained by exactly solving the quantum system with those obtained by a classical simulation of p-bits. We note that quasi-static quantum annealing can also be performed using p-bits, but our purpose here is to show the correspondence between the exact quantum and the replicated classical system. *Exact quantum solution.* For a small number of qubits, the many-body quantum Hamiltonian described in Eq. 4.8 can be solved exactly by methods of equilibrium statistical quantum mechanics:

$$\langle S\rangle = \frac{tr\big[S_{op}exp(-\beta H_Q)\big]}{tr\big[exp(-\beta H_Q)\big]} \tag{4.10}$$

where $\langle S\rangle$ is the expectation value of an observable corresponding to the operator S_{op} 'tr' represents trace. In this case, we choose S_{op} to correspond to all possible spin configurations corresponding to the different factors of the problem. We choose an inverse temperature of $\beta = 25$ and two magnetic fields $\Gamma_X = 0.1$ and $\Gamma_X = 0.5$. For each spin configuration $\big[y_2\ y_1\ x_4\ x_3\ x_2\ x_1\big]$, where $\big(y_i, x_i\big) \in \big\{-1, +1\big\}$, we compute the corresponding operator Sop to calculate the equilibrium probability.

Replicated p-bit simulation The mapped classical system is simulated by first obtaining the current vector I_i for the i^{th} p-bit in the system from the classical Hamiltonian in Eq. 4.9 by $I_i = -\partial H_C/\partial m_i$. The same inverse temperature, $\beta = 25$ is chosen with $r = 45$ replicas and all p-bits are sequentially updated according to $m_i = sgn\big[tanh(\beta I_i) - rand(-1, 1)\big]$, where rand is a number that that is uniformly distributed between -1 and $+1$. For each magnetic field $\Gamma_X = 0.1$ and $\Gamma_X = 0.5$ that enters J_{perp}, $N = 2 \times 10^6$ time steps are chosen and a probability of each state is obtained using time averaging of the state of the system for the entire duration N of the simulation over all replicas r. Although the exact solution and the replicated p-bit simulation do not seem to show complete agreement at each state, the error can be systematically reduced by choosing a larger number of replicas; the error of the system scales as $O\big(1/r^2\big)$.

3.2.9 Experiment versus simulation

In this section, we compare our experimental work with ideal simulations performed using software. The simulation updates all p-bits every Δt, flipping the i^{th} p-bit with probability $P_i = 1 - exp\big(-\Delta t/\tau_i\big)$, where the dwell time τ_i of the i^{th} p-bit depends on the inputs I_i obtained from the synaptic function: $\tau_i = \tau_{0,i} exp\big(\pm I_i\big)$. Here $\tau_{0,i}$ is the zero-bias dwell time, and I_i is positive if it is parallel to the state of the p-bit and negative if it is anti-parallel. Fig. 3.10a shows six simulated p-bits of an ideal system in which the average outputs versus inputs for all p-bits are identical. The retention times of the p-bits are much greater than the interconnect delay (about 1,000 times greater) such that $\tau_{inter} \ll \tau_N$, where τ_N is the smallest zero-bias dwell time among all p-bits.

By contrast, Fig. 3.10b shows experimentally observed average behaviour of six p-bits where the device variations of the MTJs affect the alignment and shape of the

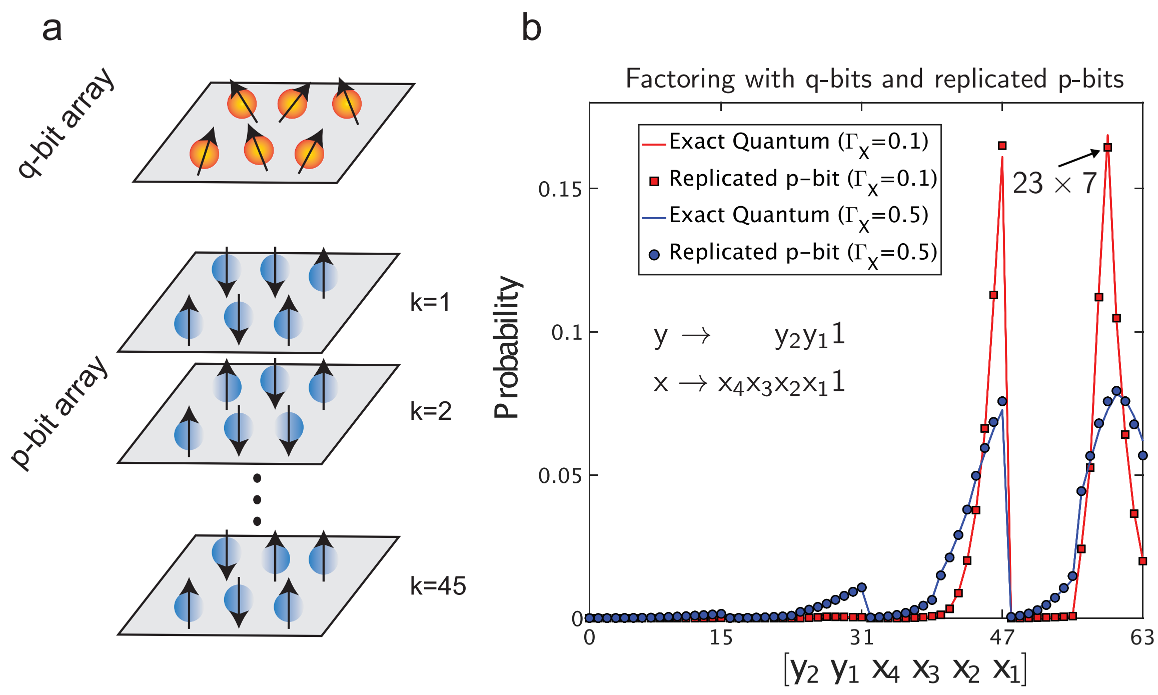


Fig. 3.9. **Computing with p-bits versus AQC. (a)** A representation of how an array of six Ising spins in a qubit array can be replicated with an array of p-bits. **(b)** A comparison of both approaches for factoring 161 = 23 × 7. For a system of six Ising spins, there are 64 states. At higher magnetic fields ($\Gamma_X = 0.5$) both systems are 'disordered' and the correct peak is not pronounced. At lower magnetic field ($\Gamma_X = 0.1$) the correct peaks emerge with a high probability. The states (y_i, x_i) have been converted to binary variables si from the bipolar variables mi by defining $s_i = (m_i + 1)/2$ and the states $\left[y_2 y_1 x_4 x_3 x_2 x_1\right]$ are expressed in decimal on the x axis.

average response. Using a simple correction in the synaptic weights (see Methods section "Factorization experiment calibration"), experimental results of factorizing 161 (shown in Fig. 3.10d) are fitted to computer simulations (Fig. 3.10c) using a single fitting parameter $I_0 = 5$.

3.2.10 Effect of p-bit parameter variation on system performance

We investigate simulations using device parameter variations obtained from our experiments and elaborate on how to effectively mitigate them within certain limits.

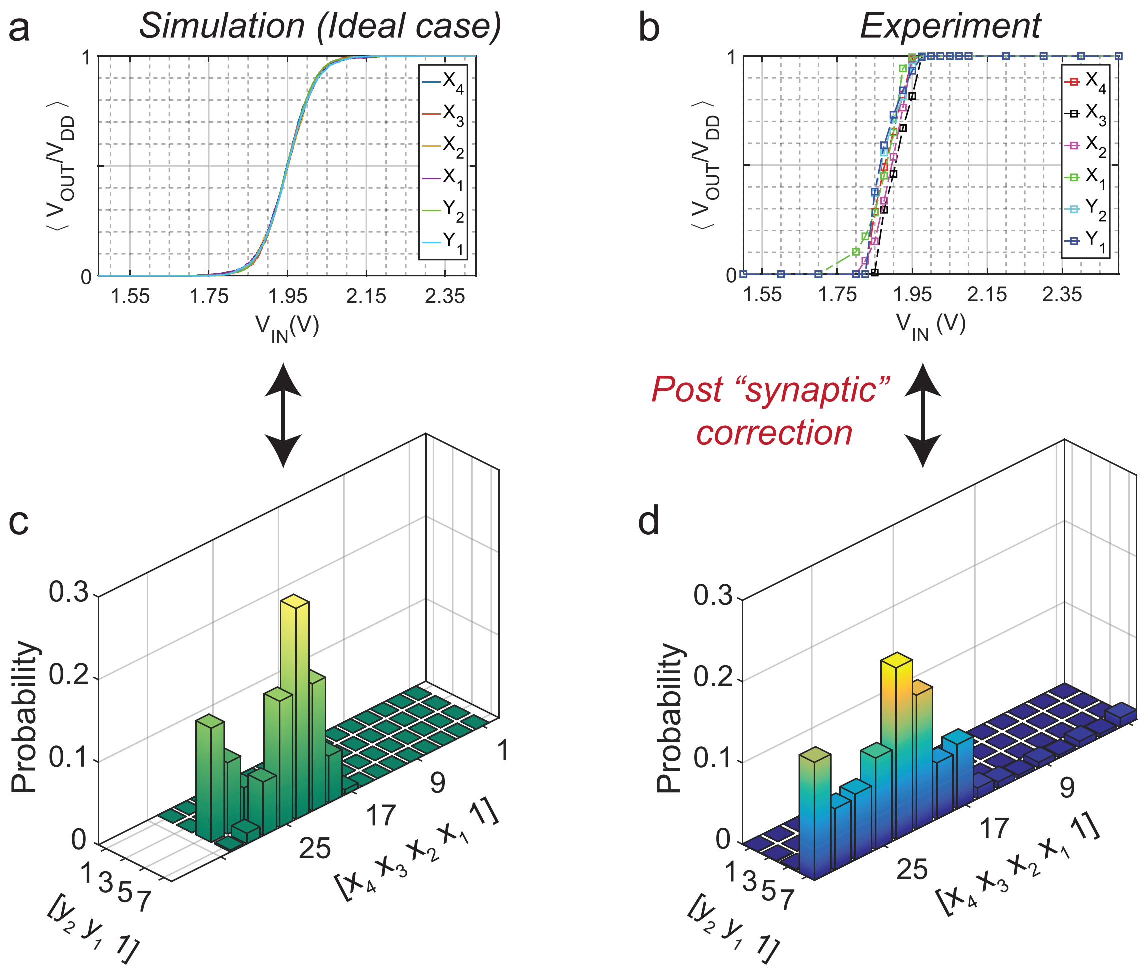


Fig. 3.10. **Simulation versus experiment.** **(a-d)**We simulate the ideal experiment when all p-bits are perfectly aligned **(a)** using an idealized p-bit model which produces the results shown in **(c)**. Each data point is taken as an average over a time window of 15 s with 2,000 or more sampling points. The presence of device variations leads to a non-ideal system of misaligned p-bits **(b)**, which is corrected using synaptic biases, allowing the experiment to approach the correct results **(d)**. The time averaged statistics in **(d)** are collected over a time window of 15 s with 2,000 or more sampling points.

Fig. 3.11 shows the effect of variations in retention times of the free layer on the overall performance of the system. In these simulations, the retention times for p-bits is varied from τ_N to $4\tau_N$ in all of the three cases shown. We conclude from our simulations that in general, for all p-bits that have retention times much slower than the interconnect delay ($\tau_{inter} \ll \tau_N$), the system will operate properly.

Fig. 3.11c suggests that when $\tau_N = 10^1 \times \tau_{inter}$ the system fails to operate correctly. The exact boundary where the system stops working is a function of the type (linear versus nonlinear) and size (fan-in) of the synapse and the overall size (number of p-bits) of the system and in general requires a systematic study using a large number of p-bits.

Fig. 3.12 shows the effect of variations of other MTJ parameters $\big(R_P, R_{AP}, TMR, I_{50/50}\big)$ that are important for p-bit operation. Variations manifest themselves as either a misaligned average response of the p-bits or a distorted shape of the average behaviour of a p-bit. We correct the former in our experiments by measuring this shift and by adding an appropriate constant d.c. bias to the synaptic weights for each p-bit. The results of this procedure are simulated in Fig. 3.12d-f. For all our experiments this procedure was performed to achieve an "unbiased reference state", which is the first step of the factorization process. This process can be automated, for example using a control loop feedback mechanism such as a proportional-integral-derivative (PID) controller. The latter variation—the distortions in the shape of the average behaviour—are harder to correct, but in general their adverse effects on system operation seem minimal.

Owing to the ease of implementing compensation for device variations, as well as the recently reported market-ready MRAM showing lower levels of variation [48] compared to the experimental values obtained in this work, variation effects are not expected to become an issue as the size of the p-bit network scales.

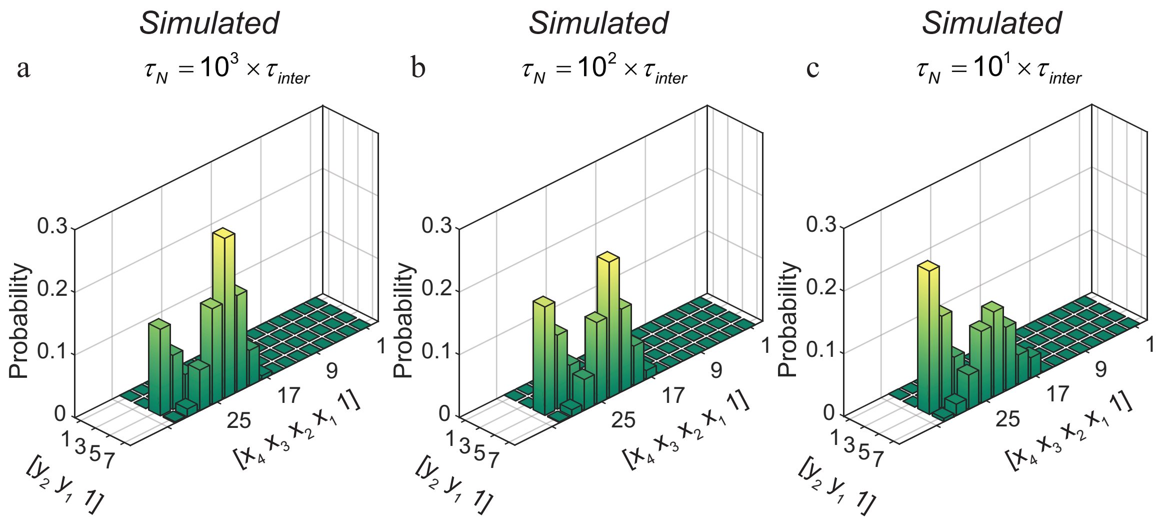


Fig. 3.11. **Simulation of variations of τ_N.** The τ of six p-bits is varied from a minimum value of τ_N to a maximum value of $4\tau_N$. Variations between p-bits do not affect system operation providing that $\tau_{inter} = \tau_N$.

3.2.11 p-bit-based implementation of invertible AND gate

A three-p-bit circuit of the type shown in the main text can implement an AND gate using x_2, x_1 as input p-bits and y_1 as an output p-bit with a cost function of the form [12, 49]

$$E(x_1, x_2, y_1) = I_0\big(3y_1 + x_1x_2 - 2x_12y_1 - 2x_2y_1\big) \tag{4.11}$$

which minimizes the energy for configurations x_2, x_1, y_1 that satisfy the truth table. We use the same method as the main text to obtain the inputs I_{x2}, I_{x1}, I_{y1}:

$$I_{x2} = I_0(-x_1 + 2y_1)$$

$$I_{x1} = I_0(-x_2 + 2y_1)$$

$$I_{y1} = I_0(-3 + 2x_1 + 2x_2)$$

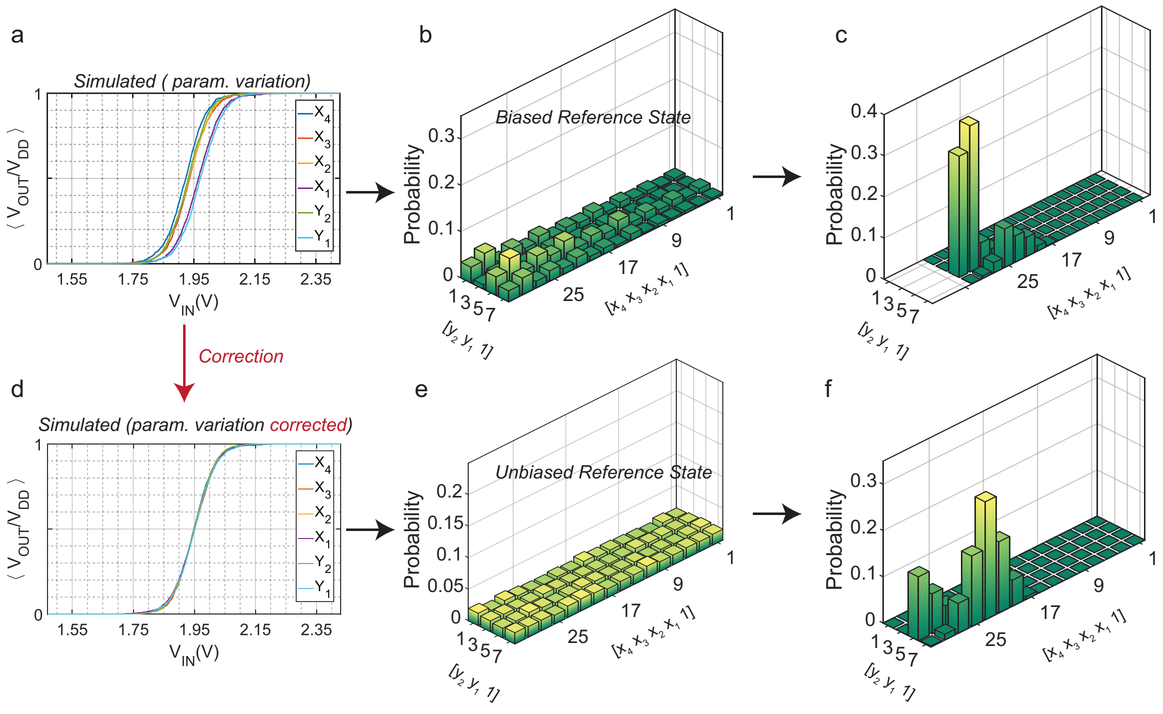


Fig. 3.12. **Simulations of variations of MTJ parameters. (ac)** The variation of MTJ parameters results in the misalignment of the average responses of the p-bits **(a)**, which results in a biased reference state **(b)**. When such a system is used for factorizing 161 the observed results are incorrect **(c)**. **df**, The shifts in the average responses are corrected using synaptic biases **(d)**, which correct the reference state **(e)** and factorization results **(f)**.

Fig. 3.13a, b shows the direct mode of operation for the AND gate, with applied inputs leading to an output consistent with the inputs of any CMOS-based Boolean gate. Fig. 3.13a, b shows a time snapshot and statistics for the three p-bits when both inputs are pinned to 1 by adding a large input voltage. The statistics for the direct mode of operation match well with the Boltzmann law (see main text) with the constant I_0 adjusted to 0.25.

A more interesting case is the inverted mode, in which an output is pinned and the inputs resolve themselves to be consistent with the applied output. Fig. 3.13c shows a time snapshot of the p-bits when the output p-bit is pinned to 0. In this case, all three possible combinations of inputs appear, as shown by the statistics in Fig. 3.13d.

The final case is when all p-bits are left floating. Fig. 3.13e shows a time snapshot acquired for such a case, and Fig. 3.13f shows the statistics. In this case the system goes through the four states consistent with the truth table of an AND gate.

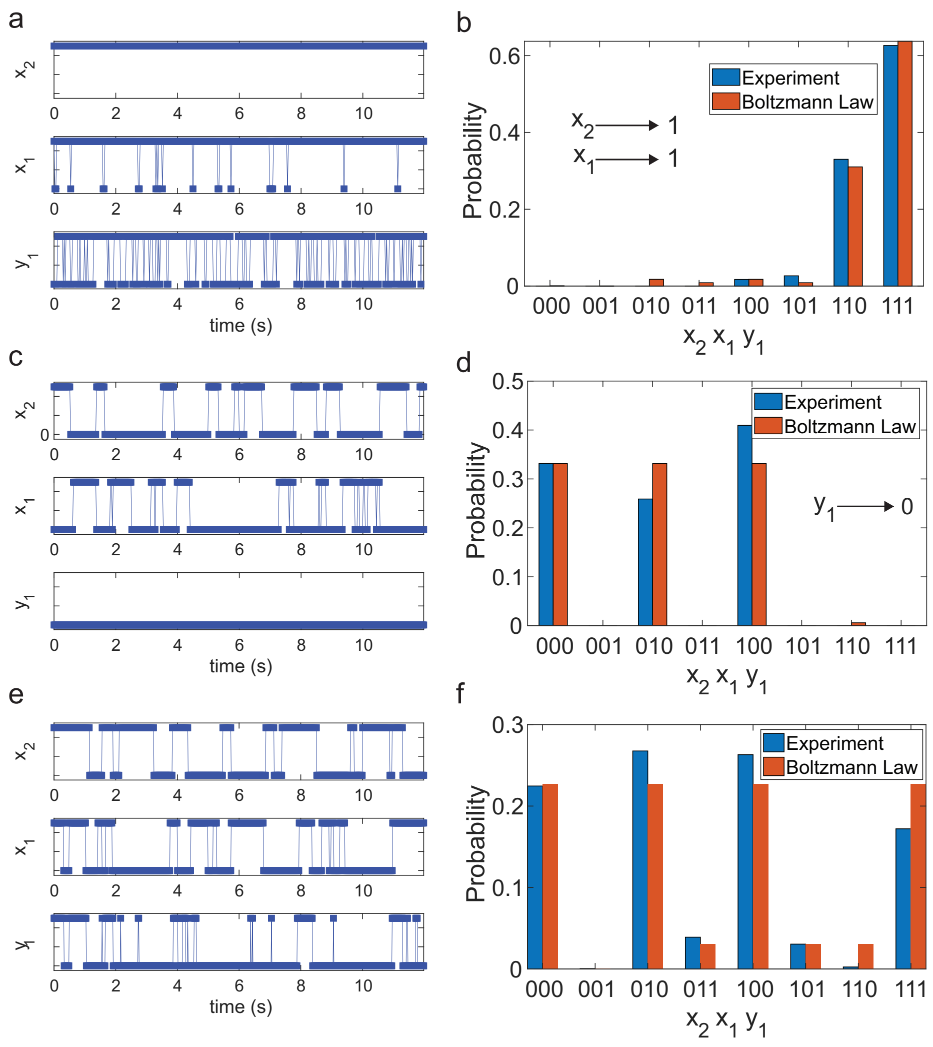


Fig. 3.13. **Invertible AND gate operation.** **(a-b)** Time snapshot for the direct mode of operation when the inputs x_2 and x_1 have both been pinned to 1 **(a)**; the statistics collected for 60 s **(b)**. **(c-d)** Time snapshot for the p-bits operating the AND gate in inverted mode when the output y_1 is pinned to 0 **(c)**; the statistics collected for 60 s **(d)**. **(e-f)** Time snapshot for the p-bits operating the AND gate in floating mode **(e)**; the statistics collected for 60 s **(f)**. All statistics shown are collected over a time window of 60 s with 2,000 or more sampling points.

4. HARDWARE EMULATION OF STOCHASTIC P-BITS FOR INVERTIBLE LOGIC

Materials in this chapter have been extracted verbatim from the paper: **A. Z. Pervaiz**, K. Y. Camsari & S. Datta (2019). Probabilistic computing with Binary Stochastic Neurons. **In press: BCICTS 2019**

4.1 Introduction

In the previous chapter, an autonomous probabilistic computer was demonstrated. This proof-of-concept demonstration was realized by making slight modifications to market ready MRAM technology to realize a classical fluctuating resistance. In this chapter we present a low-level emulation of MRAM based p-bits and build an 8-bit p-computer which resembles the autonomous p-computer shown before. Many results from the s-MTJ based p-computer are reproduced using this low-level emulation. The s-MTJs require some special handling to prevent loss of devices due to environmental factors such as damage due to electrostatics. The purpose of this chapter is primarily to develop an all weather p-computer which could be used for prototyping new algorithms and studying more complex phenomenons such as the role of pinning and time variations in s-MTJs.

In Section 4.2, we show a mapping from an ideal p-bit to the low-level emulation of MRAM based p-bit. In Section 4.3 we describe the system architecture while in Section 4.4 we illustrate key results obtained using the emulated p-bits.

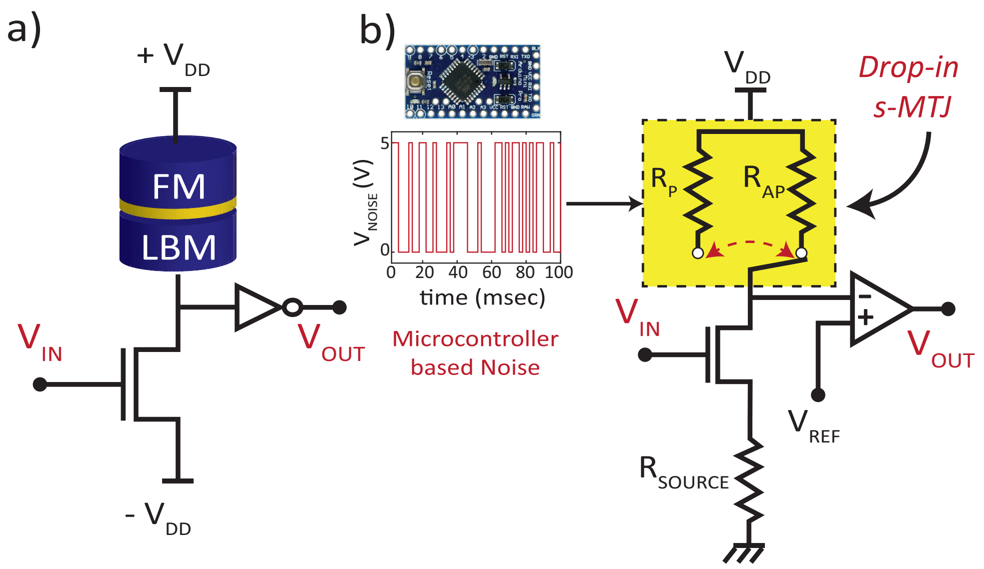


Fig. 4.1. **Emulating MRAM p-bit. (b)** shows a mapping from the MRAM based design to its emulation. An analog multiplexer emulates a stochastic MTJ fluctuating between R_P and R_{AP}, where the statistics of the fluctuation is controlled by a noise signal which is generated using a microcontroller.

4.2 Stochastic MRAM based p-bit and its emulation

Fig. 4.1b shows the schematic of the emulated p-bit, which closely resembles the MRAM based p-bit. The s-MTJ has been replaced by an analog multiplexer which fluctuates between two fixed resistances R_P and R_{AP} based on an applied noise signal. This applied signal has the same statistics of a stochastic magnet for a given energy barrier whose attempt time is given by a Neel-Arrhenius theory [9].

Our source of noise also introduces pinning which is a function of the voltage across (or current through) the multiplexer branch. For example if the multiplexer is meant to be completely stochastic for a voltage of V_0, then for a measured voltage of V across the multiplexer, the average retention time in the parallel state is given by $\tau_P = \tau/(1 + exp\big((V - V_0)/\Delta\big)\big)$, where Δ is the width of the intended stochastic region and τ_N is the average retention time of the emulated s-MTJ. In Fig. 4.1b we add a source resistance R_{source} similar to what was shown previously. This resistance

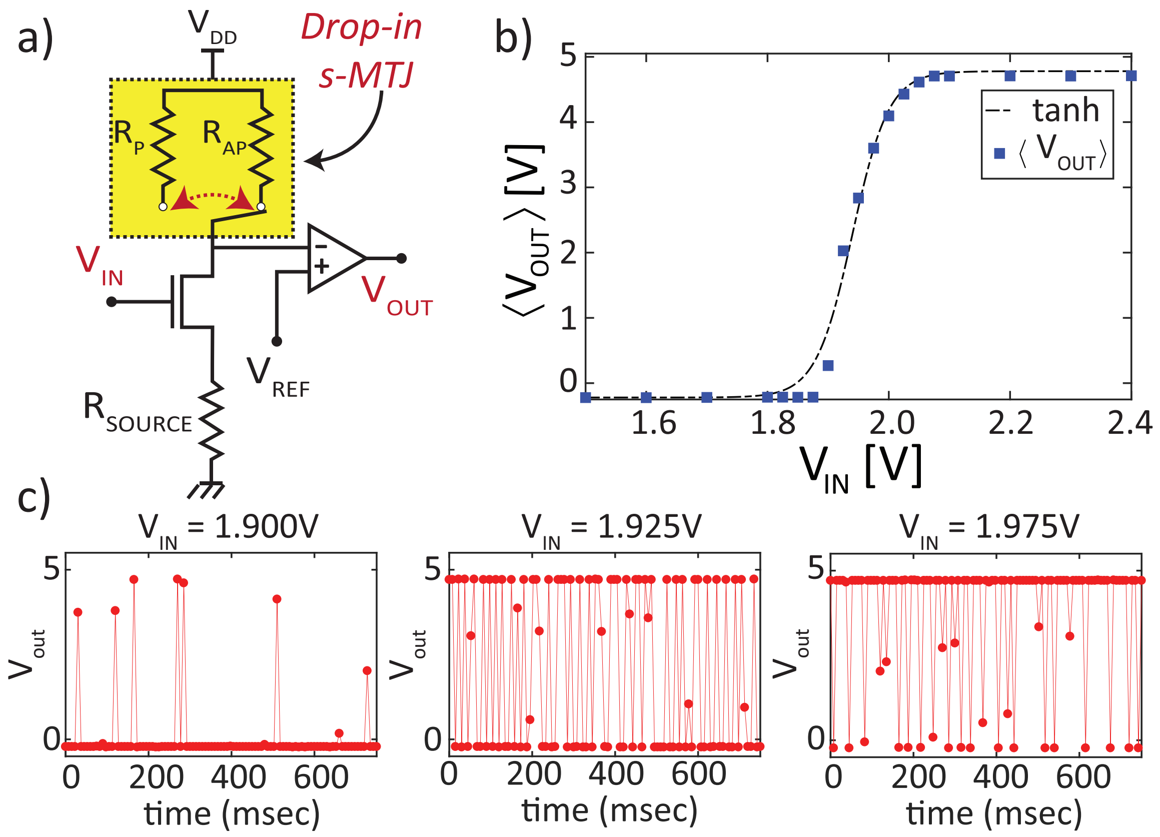


Fig. 4.2. **Emulated p-bit.** **(a)** Electrical schematic of a emulated p-bit which uses a multiplexer, NMOS and a comparator. The multiplexer emulates a low barrier magnet having a retention time $\tau = 50ms$. **(b)** shows the average output as a function of applied input with each point being a 15 second average. **(c)** shows time snapshots for three sets of applied input voltages.

is added for the purposes of unpinning a drop-in PMA s-MTJ. We do this since we are using off-the-shelf small signal transistors and from working previously with s-MTJ's, we recognize that the resistance of the transistors in the ON region will not be sufficient to unpin a s-MTJ (should one be used here), which we assume requires current in the $\approx 10\mu A$ region as observed previously as well. Also similar is the use of a comparator instead of an inverter since the voltage swing at the drain of the transistor is insufficient for driving off-the-shelf CMOS inverters. This requires

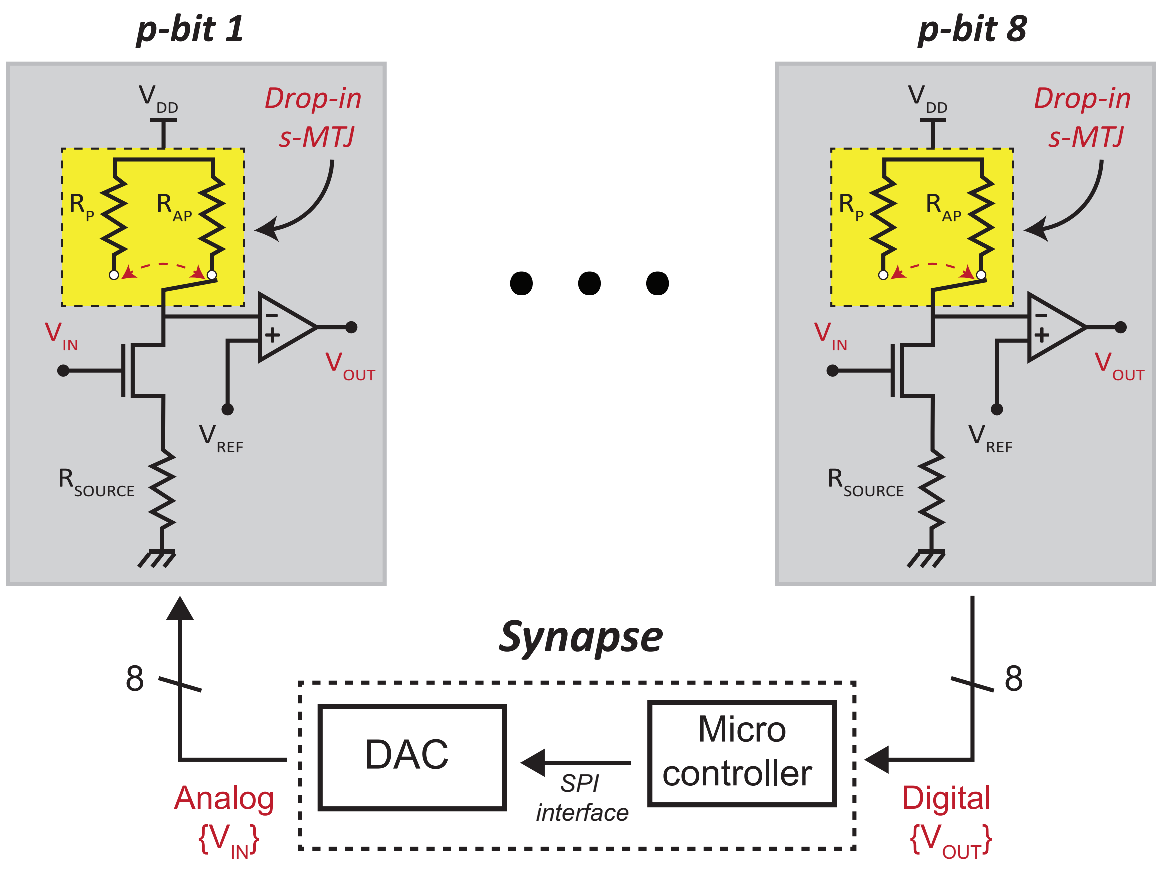


Fig. 4.3. **Architecture of p-computers.** A system level schematic for a p-circuit is shown. A microcontroller reads the output voltages of all p-bits, calculates the inputs corresponding to Eq. 4.1 and uses a DAC to set analog input voltages to the p-bits.

an additional applied voltage V_{REF} which is the average drain voltage when the multiplexer is fluctuating roughly equally between R_P and R_{AP}.

Fig. 4.2b shows the average behavior of the emulated p-bit as a function of applied input, while Fig. 4.2c shows time snapshots for a set of three applied inputs voltages. In these experiments we have used $\tau_N = 50\ ms$, $R_P = 8\ k\Omega$, $R_{AP} = 16\ k\Omega$ and $R_{source} = 5\ k\Omega$. We use 2N7000 NMOS transistors, AD8694 quad OP-AMPs as comparators and MAX 394 quad Analog multiplexers.

4.3 Optimization using Probabilistic computers

Interconnected networks of p-bits are connected in such a way that the i^{th} p-bit is driven by an input I_i that is a function of all other outputs $\{m_1, \cdot\cdot, m_N\}$. Symmetric networks represent a subset of such networks for which the inputs I_i can be obtained from an energy function E:

$$I_i = -\frac{\partial E(m_1, \cdot\cdot, m_N)}{\partial m_i} \tag{4.1}$$

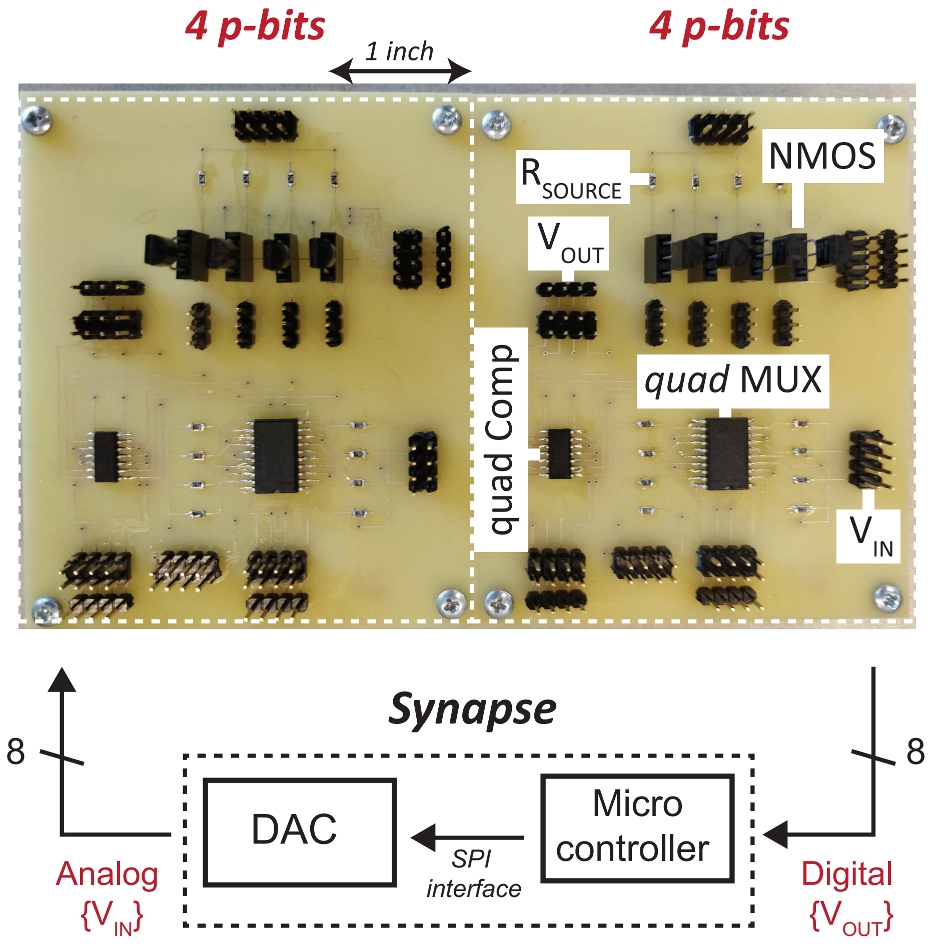


Fig. 4.4. **Photograph of p-circuit.** A photograph of a printed circuit board is shown that houses 4 emulated p-bits. We use two boards each housing 4 p-bits to put together an 8-bit p-computer.

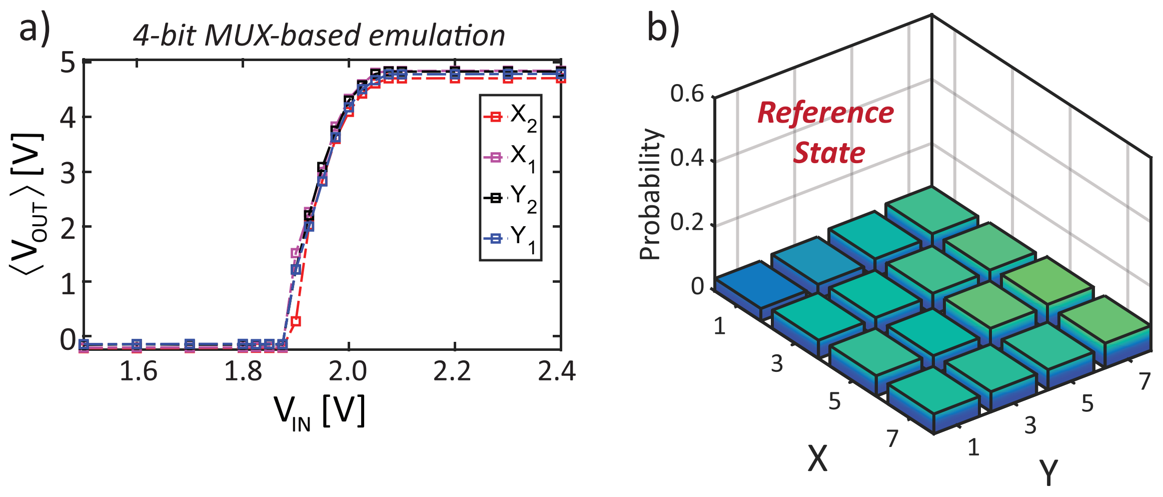


Fig. 4.5. **Reference state calibration.** **(a)** shows the time average response of four p-bits which are slightly misaligned. These misalignments are corrected by adding a constant synaptic bias to each p-bit individually leading to a reference state shown in **(b)** where all p-bits are left uncorrelated.

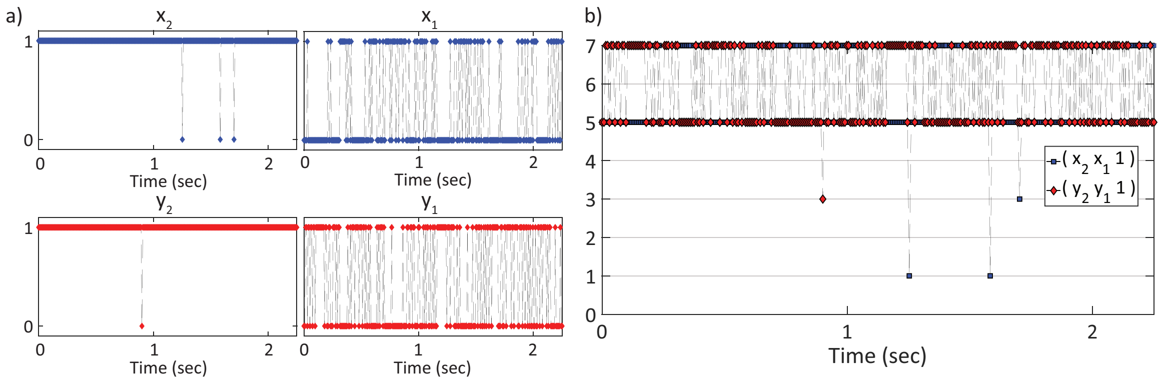


Fig. 4.6. **Time snapshot for factorizing 35.** **(a)** shows individual time snapshots of 4 p-bits that are fluctuating when the synapse is programmed to factorize 35. **(b)** shows the time snapshots of both factors X and Y while the synapse is programmed to factorize 35. The system spends most of its time in two states; 7×5 and 5×7.

Such networks will occupy different configurations with probabilities given by the Boltzmann distribution at equilibrium:

$$P(m_1, \cdot\cdot, m_N) \propto \exp\left[-E(m_1, \cdot\cdot, m_N)\right] \tag{4.2}$$

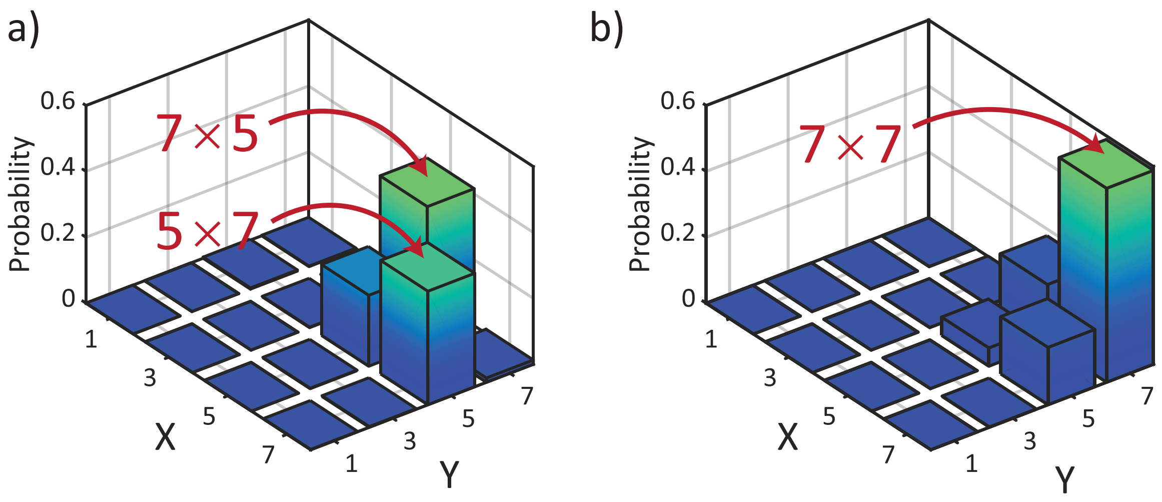


Fig. 4.7. **Statistics for factorizing 35 and 49.** **(a)** shows the statistics of the system when the synapse is programmed to factorize 35. In this case the two peaks corresponding to the correct factors 7×5 and 5×7 are highlighted by the system. **(b)** shows the statistics for the case when the system is programmed to find factors of 49. In this case only the state 7×7 is highlighted by the system.

so that configurations with the lowest energy E will have the highest probability.

This property makes the networks naturally suited for solving optimization problems where the correct solution minimizes a *cost function* which can be identified with E and used to calculate the synaptic functions from Eq. 4.1. This is similar in the spirit to adiabatic quantum computing (see for example, Ref. [50]) which is based on engineering a network of quantum spins ($\vec{\sigma}$'s: Pauli spin matrices) having an energy $E = \sum J_{ij}\vec{\sigma}_i \cdot \vec{\sigma}_j$. A large amount of work has gone into identifying appropriate cost functions for different problems of interest [36] and many of these formulations can also be adapted to design p-bit networks (see for example, Ref. [51–53]). In the previous chapter we demonstrated how the integer factorization problem can be cast as an optimization problem. We now use the very same approach, with our low-level p-bit emulations and reproduce all the results from before.

4.4 Integer Factorization

Fig. 4.3 shows a schematic of an architecture of our system while Fig. 4.4 shows a photograph of a table top 8-bit asynchronous p-computer built using emulated p-bits. The synapse is constructed using an Arduino MEGA microcontroller which reads the output voltage of each p-bit, calculates the inputs to each p-bit and uses an 8 channel PMOD DA4 Digital-to-Analog converter to set analog voltages. The details for programming the synapse and the DAC interface are provided in detail in chapter. 2.

For proper operation the first step is to establish an unbiased *reference* state. This is the state of the system when all p-bits have been left uncorrelated and behave as independent random number generators. Due to mismatches in device characteristics, the average response of the each p-bit is slightly different from those of others. These device variations can manifest in three different ways.

- **Variations in average retention time τ_N:** This leads to p-bits fluctuating at different speeds with respect to one another. In p-bits the the primary cause of this would be variations in energy barrier of the free layer of the s-MTJ.
- **Linear shift in average response of the p-bits:** Any variations in transistor characteristics or s-MTJ resistances can shift the average response of p-bits.
- **Shape distortion of the average response:** It is also possible that the shape of average p-bit responses be different across p-bits. This non-linear effect is a function of both s-MTJ and transistor characteristics.

If the system is used without correcting these variations, it will lead to improper operation. Variations in retention times can be dealt by ensuring that the synapse is faster than the fastest p-bit. Variations which lead to a shift in the average response of p-bits can be addressed by adjusting the on-site bias h_i for the i^{th} p-bit. This process can be automated at a system level using control loop mechanisms such as a proportional-integral-derivative controller. Shape distortion of the average responses is harder to correct, but it is does not seem to effect system performance as much as the other two. Our emulator based p-circuits allow us to study many of these phenomena in a systematic manner.

Fig. 4.5a shows the average response of four p-bits which have small linear shifts with respect to one another. We use a Keysight MSO-X-3014T Oscilloscopes for measuring and recording the states of p-bits. Fig. 4.5b shows the starting point of the experiment, the reference state, which is obtained by correcting the linear shifts shown in Fig. 4.5a. This leads to a state where all p-bits are uncorrelated, resulting in roughly 2^N equal states, where $N = 4, 6$ & 8 is the total number of p-bits used. For integer factorization we use the cost function given by Eq. 4.7a using 4 $(P = 2, Q = 2)$, 6 $(P = 4, Q = 2)$ and 8 $(P = 5, Q = 3)$ p-bits. For more detailed derivations please see section. 3.2.5.

Fig 4.6a shows a time snapshot of all four p-bits (x_2, x_1, y_2, y_1) when the synapse is programmed for factorizing 35 while Fig. 4.6b shows a time snap shot of the factors $X = (x_2, x_1, 1)$ and $Y = (y_2, y_1, 1)$ constructed from the p-bits. The factors X and Y are locked into the two possible states that factorize 35, which are 7×5 and 5×7. This can be seen more clearly by statistics shown in Fig. 4.7a, where the two peaks corresponding to the right factors are highlighted. These statistics and all others presented in this work are taken across 15 seconds for an average retention time of 50 ms for all p-bits. Fig. 4.7b shows the statistics gathered for when the system is programmed to factorize 49. In this case only one peak corresponding to 7×7 is highlighted by the system.

Fig. 4.8a shows the average response of 8 p-bits which are first corrected to get a proper reference state as shown in Fig. 4.8b. This process is similar to that described above. Once a good reference state is achieved the synapse is programmed to factorize 161 using 6 $(P = 4, Q = 2)$ p-bits. p-bits which are not used for an experiment are simply clamped to 0 by applying a high negative on-site bias h_i. Fig. 4.8c shows the statistics gathered for factorizing $161 = 23 \times 7$. Fig. 4.8c shows the results of factorizing $945 = 63 \times 15$ using all 8 p-bits.

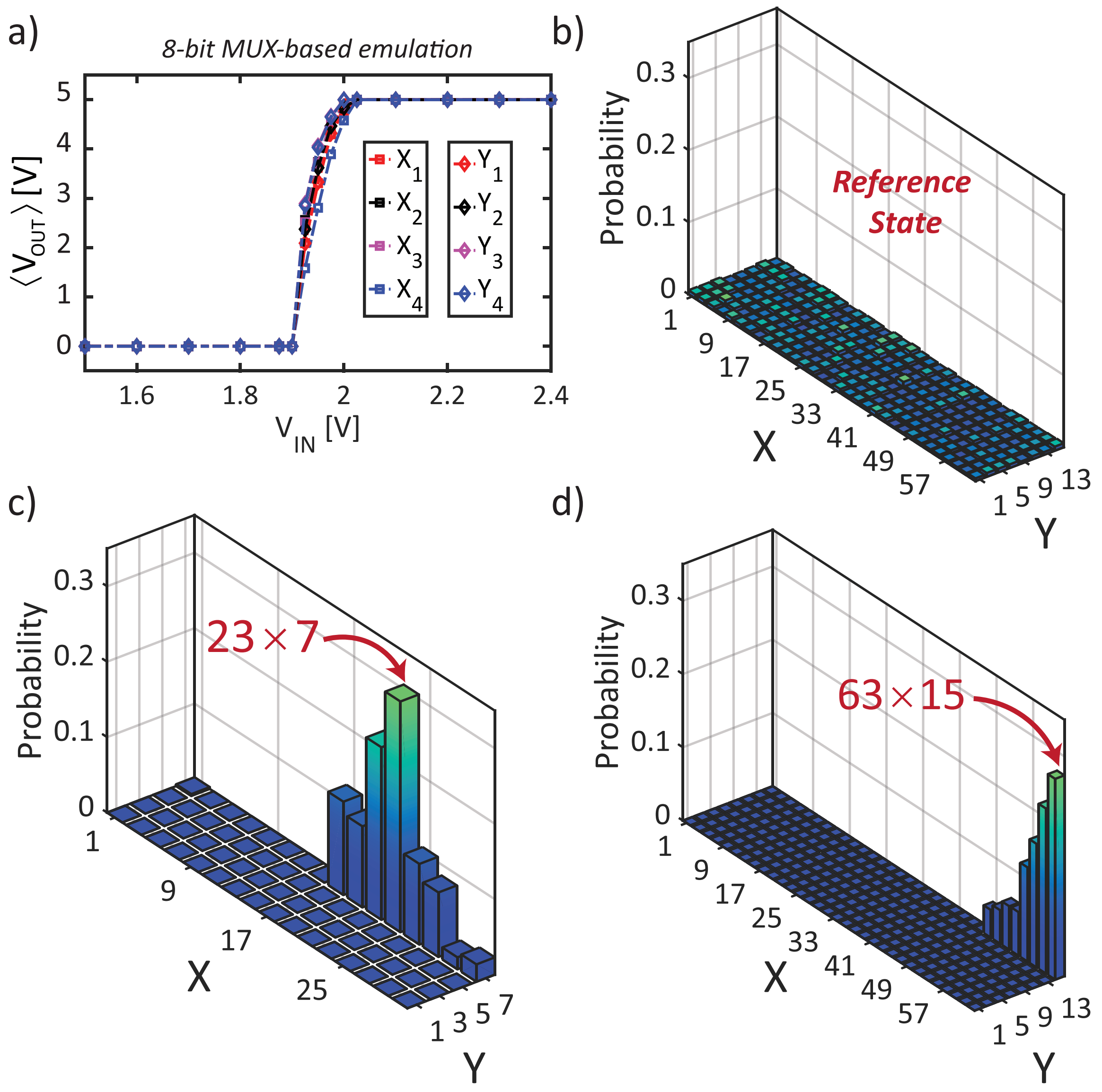


Fig. 4.8. **Factorizing 161 and 945 with 6 and 8 p-bits.** **(a)** shows the average response for 8 p-bits which are corrected by adding synaptic biases to produce an 8-bit reference state as shown in **(b)**. The synapse is then programmed to factorize $161 = 23 \times 7$ and $945 = 63 \times 15$ as shown by the statistics in **(c-d)**.

5. WEIGHTED P-BITS FOR FPGA IMPLEMENTATION OF PROBABILISTIC CIRCUITS

Materials in this chapter have been extracted verbatim from the paper: textbfA. Z. Pervaiz, B. M. Sutton, L. A. Ghantasala & K. Y. Camsari (2018). Weighted p-bits for FPGA implementation of probabilistic circuits. IEEE transactions on neural networks and learning systems. vol. 30, no. 6, pp. 1920-1926, June 2019.

5.1 Introduction

In the previous chapters s-MTJ based implementations of p-bits have been demonstrated, and interconnected into what may be considered a cyber-physical architecture. These nanodevice based implementations offer advantages in area (300$\times$ or more) and energy (10$\times$ or more), however, realizing scaled p-computers using these nanodevices is difficult at present.

In this chapter we present a digital, tiled FPGA based implementation of large p-circuits using a *weighted* p-bit structure which combines the functionality of Eq. 1.1 and Eq. 1.2 into a single composite unit where each weighted p-bit is a CMOS based tunable random number generator with its own *local* CMOS based synapse that weights the outputs of other such weighted p-bits in the p-circuit. The specific 4×4 p-bit tile can map any $n^2 \times n^2$ $[J]$ matrix. This 4$\times$4 array can support a fully connected reciprocal network.

One significant difference between the approach present in this chapter is the clocked or sequential operation of our weighted p-bits. This is different from the autonomous p-circuits demonstrated in previous chapters, in which the serial updating of p-bits came naturally due to the asynchronous operation of p-bits. To do so, we use within a tile, a *sequencer* that produces a set of enable signals for each weighted p-bit in the 4×4 tile. As noted in the introduction, it is envisioned that large scale p-circuits would greatly benefit from autonomous operation, as opposed to the clocked mode of operation presented here. In this chapter we compose our larger p-circuits using these smaller 4×4 tiles of weighted p-bits and interconnect the tiles using "directed" connections. This approach has been further improved by others [54] who introduced into these weighted p-bits the autonomous operation by adding an "attempt" logic module to the tunable random number generator sub-module of the weighted p-bit.

The organization of this chapter is as follows: In Section 5.2 we describe the FPGA implementation of the weighted p-bit. In Section 5.3 we demonstrate examples of p-circuits realizing invertible Boolean logic starting from simple Boolean gates that are then interconnected to construct an N-bit invertible Ripple Carry Adder in Section 5.3.3 and a small instance solver for the NP-complete Subset Sum Problem in Section 5.3.4.

5.2 Weighted p-bit

Fig. 5.1 shows the block diagram of an FPGA implementation of a weighted p-bit. There are two major sub-blocks of the weighted p-bit: (a) The weight matrix which implements Eq. 1.2 and (b) the tunable RNG which implements Eq. 1.1. We describe both components below.

5.2.1 Weight Matrix

Each weighted p-bit can take the outputs of others and weight them according to the interconnect matrix $[J]$. This is done by every weighted p-bit locally using a weight matrix block that takes the i^{th} row of the $[J]$ matrix and stores it in registers local to the weighted p-bit along with the i^{th} entry of the self-bias vector, $\{h\}$. The local presence of these registers allows a compact implementation of Eq. 1.1-1.2 in a single unit. These registers can also be made user accessible, however in our demonstration this functionality is not needed since all our $[J]$ and $\{h\}$ entries are calculated offline without any online learning [55]. Moreover, the examples discussed in this paper do not make use of "annealing" [52, 54], which would also require user accessibility.

Fixed point arithmetic: To perform all arithmetic operations in a weighted p-bit, we use a fixed point notation of s[x][2], where integer x is chosen based on the requirements of the p-circuit. This allows a range of $-|2^x|$ to $(2^x - 1)$ for the integer part. For example, the $[J]$ and $\{h\}$ matrices used for the Full Adder shown in Fig. 5.8 use weights that require s[4][2] while for the AND gate shown in Fig. 5.6, the use of s[3][2] is sufficient. In general, $[J]$ and $\{h\}$ registers allow different problems to be mapped onto the system.

Thresholding: Given that each weighted p-bit has multiple inputs, the worst case for the weighted sum $I_0(Jm+h)$ can exceed the allowed input range of s[x][2] notation or the allowed input range of the *Activation Function*, which uses the output of the weighted sum block to calculate $tanh()$ as explained in the subsequent subsection. To prevent this, an overflow detection and numerical clamping system is used that compare the bit extended result from the *Sum* block to the maximum and minimum allowed numbers for the *Activation Function*. The result of the sum is clamped to the the maximum or the minimum number that can be read by the *Activation Function*. For example, the s[4][2] notation has a maximum and minimum limit of 15.75 and -16, however the input of the lookup table for the *Activation Function* need not be any less than -8 and any greater than 7.75 as shown in (Fig. 5.1).

MUX: A multiplexer is used to perform both the thresholding and clamping of weighted p-bits (Fig. 5.1). Table 5.1 shows the truth table of the multiplexer.

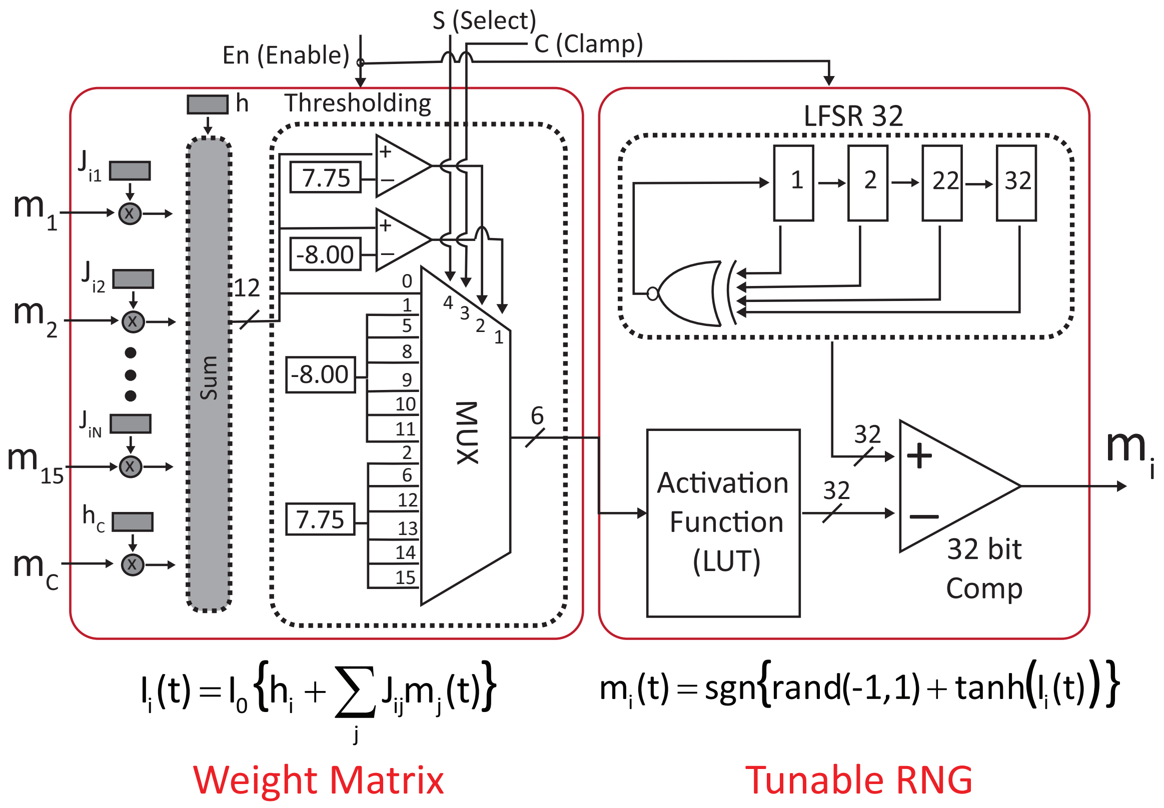


Fig. 5.1. **Weighted p-bit.** A weighted p-bit consists of two major subblocks, a *Weight Matrix* and a *Tunable RNG* implementing Eq. 1.2 and Eq. 1.1 as a composite unit. The weight matrix implements one column of Eq. 1.2 and adds overflow protection and clamping capabilities to the weighted p-bit while the tunable RNG subblock implements Eq. 1.1 whose terminal characteristics are further shown in Fig. 5.2. See text for a detailed description.

Four signals are used as inputs, “S(Select)”, which is high if the weighted p-bit is to be clamped to the “C(Clamp)” signal. The other two are the outputs of signed comparison between the Sum and the maximum/minimum numbers to be passed on to the *Activation Function*.

5.2.2 Tunable random number generator

The output of the weight matrix is applied as input to the tunable RNG. Fig. 5.2 shows the average time characteristics of the tunable RNG block. As shown in the inset of Fig. 5.2, when the input I_i is 0, the m_i randomly fluctuates between 0 and 1 as a function of time, leading to a long time average of 0.5. As the applied input is increased above (below) 0, the average increases (decreases) and saturates to 1 (-1). The tunable randomness allows weighted p-bits to become correlated with each other. We describe the sub-modules of the tunable RNG block below.

Activation Function: We use lookup tables (LUT) to implement the $tanh()$ function. The domain of the $tanh()$ is $(-\infty, +\infty)$ and its range is $(-1, 1)$. To allow a comparison between the output of the pseudo random number generator and that of the LUT, we first transform $tanh()$ to $z = (\tanh + 1)/2$ and then use a s[0][31] bit fixed-point representation, where 0 represents the integer part and 31 represents the fractional part. We choose an s[3][2] representation for the input of the LUT, which translates to the interval $(-8, 7.75)$ with a resolution of 0.25 between successive data points. Several methods of implementing sigmoid like functions have been studied [56] and lookup tables are naturally suited for implementation in FPGAs. The use of lookup tables will result in an approximation error. Ref. [56] looks at the average and the maximum errors for various approximation methods. For the lookup table an error arises due to the difference in absolute real number value of tanh and its

Table 5.1. Truth Table for the weight matrix multiplexer

S (Select) (4)	(C) Clamp (3)	$\mathrm{I_{IN}} > \mathrm{max_{tanh}}$ (2)	$\mathrm{I_{IN}} < \mathrm{min_{tanh}}$ (1)	Output
0	x	0	0	$\mathrm{I_{IN}}$
0	x	0	1	$\mathrm{min_{tanh}}$
0	x	1	0	$\mathrm{max_{tanh}}$
1	0	x	x	$\mathrm{min_{tanh}}$
1	1	x	x	$\mathrm{max_{tanh}}$

truncated representation which is shown in Ref. [56] (Eq. 9) as $E_{trunmax} = 2^{-(b+1)}$ for a fixed point representation of s[a][b].

Pseudo-random number generation: We use a 32-bit Linear feedback shift register (LFSR) with an XNOR feedback to the first register using taps from 32, 22, 2 and 1 position in the LFSR [57]. Given a seed value, this produces a maximal length pseudo-random stream of size $2^{32} - 1$ with the all 1's being the only state that is not part of the stream. It is important to note that each weighted p-bit in a p-circuit must have a unique seed value, otherwise the p-bits may have unintentional strong correlations resulting in incorrect system operation. In this paper, the use of more complex pseudo-RNGs were avoided due to the complexity of implementation and size. In practice, LFSR based pseudo-RNG worked well and are naturally suited for digital implementation. The LFSR provides the same functionality as the s-MTJ in the p-bit presented in chapter. 3, with two important distinctions. First, the randomness here is pseudo-randomness and of significantly lower quality. Secondly, thousands of transistors are now needed to replace the s-MTJ, but do give the advantage of digital accuracy as opposed to the analog output of s-MTJs.

Comparator: A 32-bit comparator compares the outputs of the *activation Function* and the pseudo random number generator and produces 0 or 1 state at the output, as shown in the inset of Fig. 5.2.

5.2.3 System Tile

Serial updating: Our p-circuits within a tile are similar to reciprocal networks and in general reciprocal networks require all p-bits to be updated sequentially [58]. To ensure this requirement is met, a "sequencer" is present in each tile which generates an *Enable* signal for every p-bit in the p-circuit, ensuring no two p-bits are active simultaneously. At any given point in time, only one of these enable signals is high

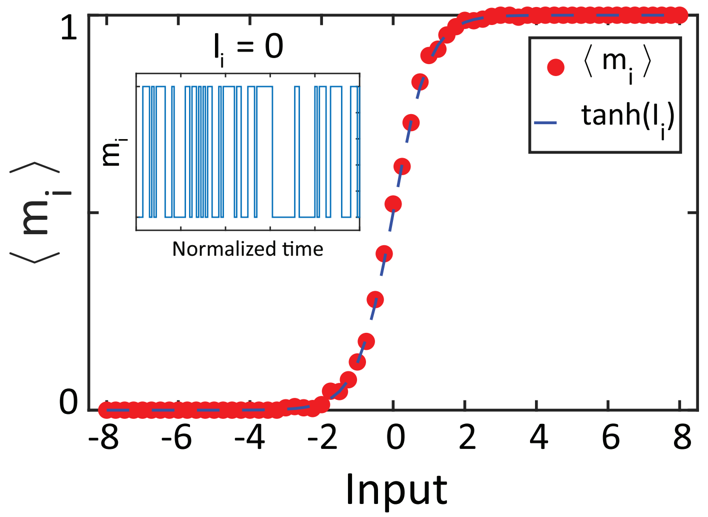


Fig. 5.2. **Sigmoid.** The time-averaged output $(\mathrm{m_i})$ of a weighted p-bit is shown as a function of the applied input $\mathrm{I_i}$. When the $\mathrm{I_i} = 0$ (inset), the output $\mathrm{m_i}$ shows equal amounts of 1's and 0's with a long-time average of 0.5. As $\mathrm{I_i}$ is increased above (below) 0, the average increases and saturates to 1 (-1). Here, the binary output of the FPGA $m_i \in \{0, 1\}$ is converted to a bipolar $\mathrm{m_i} \in \{-1, +1\}$ representation.

while all others are kept low, allowing only one p-bit to update. An AND gate (as implemented here) has 3 p-bits with each p-bit requiring 2 clock cycles for a complete update. To help manage timing constraints within the FPGA, a gap of 1 clock cycle is added between adjacent *Enable* signals. In this case the update order is $(A \rightarrow B \rightarrow C)$ but this sequence itself could have been randomized at each iteration as a method to avoid unwanted correlations due to the update order. In general, the updating sequence could also influence the average time it takes for the p-circuit to settle to its steady-state, but this is not discussed further. A specific example which illustrates the connectivity within a tile (such as the 4×4 shown in Fig. 5.3) is shown in Fig. 5.5, with all the connections presented.

Mapping problems Another important aspect of the system tile is to allow mapping of different problems [59] [60]. In this chapter we demonstrate simple boolean gates such as an AND gate and a Full Adder which require 3 and 5 (or 14) p-bits for functioning. For every problem there is a necessary requirement of serial updating which is fulfilled using the sequencer present within a tile, but larger problems such as the 32-bit Ripple Carry Adder can be implemented by cascading system tiles that implement smaller p-circuits. This cascading in a parallel manner results in a serial-parallel architecture. The presence of this serial-parallel architecture preserves the invertibility of the boolean gates all the while allowing a speed up as the problems are scaled. The ideal size of a system tile is dictated by the maximum network size of coupled p-bits which require serial updating. Note that if the network size is larger than the tile size, multiple tiles can be joined using minor graph embedding [59] [60], though with some trade-offs. For example a p-circuit that requires 5 p-bits with p-bits being updated serially, needs only a system tile of size 5, not any larger. Many such 5 p-bit systems tiles can be put together to build a larger instance of the same problem. However, note that it is also possible that some problems would require all p-bits to be updated serially, in which case the entire problem can only be mapped onto a single system tile. Note that in such a case the solution will inherently be slower in a synchronous implementation, since the time for a complete update of a system tile increases linearly with the number of p-bits. For example, the AND gate presented in the section III-A needs $3 \times (2 + 1)$ clock cycles for a complete update, while the Full Adder requires $5/14 \times (2 + 1)$ clock cycles (since we present a 5 and a 14 p-bit Full Adder design) for a complete update. On the other hand, the 32-bit Ripple Carry Adder presented in the section III-C requires the same $(5, 14) \times (2 + 1)$ clock cycles (depending on which Full Adder design is used) for an update since it is using 32 individual Full Adders connected in parallel. In general, the size of the system tile and the speed of one complete update depend on the details of how the problem is mapped.

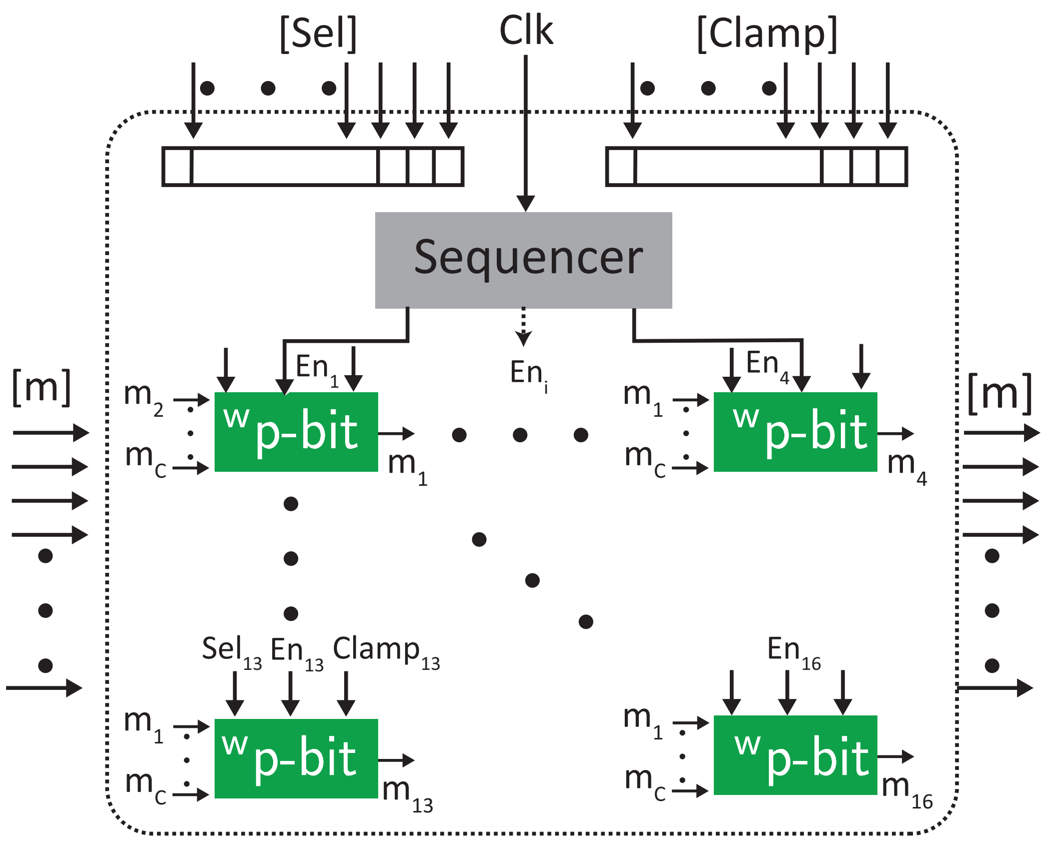


Fig. 5.3. **4 x 4 system tile.** A 4 × 4 block of weighted p-bits (denoted by ʷp-bit) can be used to implement [J] matrices with a dimension of $4^2 \times 4^2$ and the sequencer block allows each of the 16 p-bits to be updated sequentially for proper system operation. Different problems can be mapped through a choice of suitable [J] and {h} matrices to construct larger p-circuits.

Interconnecting System tiles For certain problems, scaling to larger instances will be possible by interconnecting system tiles. The Ripple Carry Adder and the Subset Sum solver implemented in this chapter are two such examples where Full Adders realized within a system tile are interconnected to form larger more complex systems. In such problems, the system tiles need to be connected in a "directed" manner where the strength of the connection can be manipulated. For example in the 32-bit RCA, the Carry out of a Full Adder is connected to the Carry In of the proceeding Full Adder. This connection can be done via the following

1. The Select and Clamp signals shown in Fig. 5.1. For example, for the 32-bit Adder one could Clamp the Select line of all Carry-in p-bits and clamp them to the Carry-out line from the preceding Adder with the exception of the First Full Adder which has its Carry-in clamped to 0.

2. By the m_C terminal. This terminal allows the output of a p-bit to be weighted by an interconnect strength h_C such that when $h_C \to \infty$ the p-bit is effectively cloned to the signal coming into m_C, while for $h_C \to 0$, the signal m_C has no effect on the p-bit operation. Note that $h_C \to \infty$ is the same as using the Select and Clamp signal as presented previously.

5.2.4 FPGA

I/O Architecture for FPGA: We use the Xilinx Kintex ultrascale XCKU040-1FBVA676 FPGA. The Xilinx Vivado Design Suite was used to synthesize and implement the Verilog RTL for the FPGA. As shown in Fig. 5.4, I/O operations with the p-circuits was accomplished by memory-mapping the p-circuits using AXI peripheral logic. Once wrapped, a number of standard interfaces can be used to control and extract data from the FPGA. Herein, we used a standard UART connection coupled to a Xilinx MicroBlaze softcore processor. For simplicity, we targeted a base operating frequency of 100 MHz for the design, as the principle objective was to explore invertible logic using p-circuits. For high-performance boolean gates, we believe an optimized CMOS design would be more appropriate, however, since there is no equivalent of invertible Boolean logic in CMOS, we believe that the real application space for p-circuits lies in this domain. Table 5.2 presents a summary of resource utilization of the various designs that have been implemented in this paper.

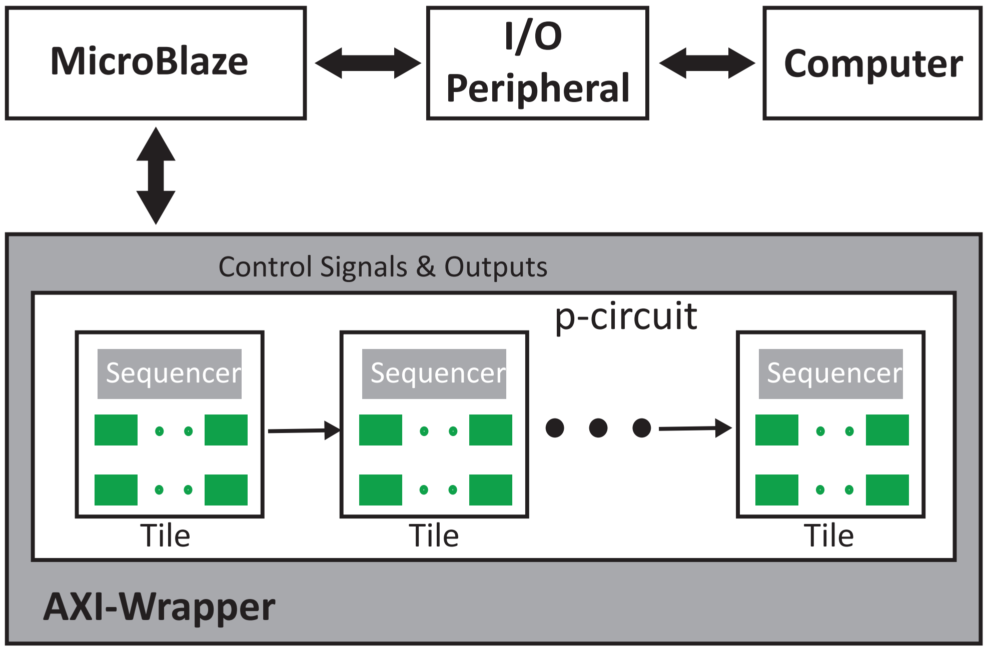


Fig. 5.4. **I/O Architecture for FPGA.** We communicate with the input and output terminals of weighted p-bits using an I/O architecture whose block diagram is shown above. Any p-circuit (tile or collection of tiles) can be converted into an AXI (universal serial bus architecture) peripheral, which can then communicate with a computer via a MicroBlaze processor that allows the collection of data from p-circuits.

Table 5.2. FPGA resource utilization of the p-circuits that have been implemented in this paper.

	Total Weighted p-bits	**Slice LUTs**	**Slice Registers**
Kintex Ultrascale XCKU040-1FBVA676		242400	484800
Tunable RNG	1	42	33
AND Gate	3	156	123
Full Adder	14	1345	586
15-bit SSP problem	155	14931	7083
32-bit Ripple Carry Adder	434	38814	18071

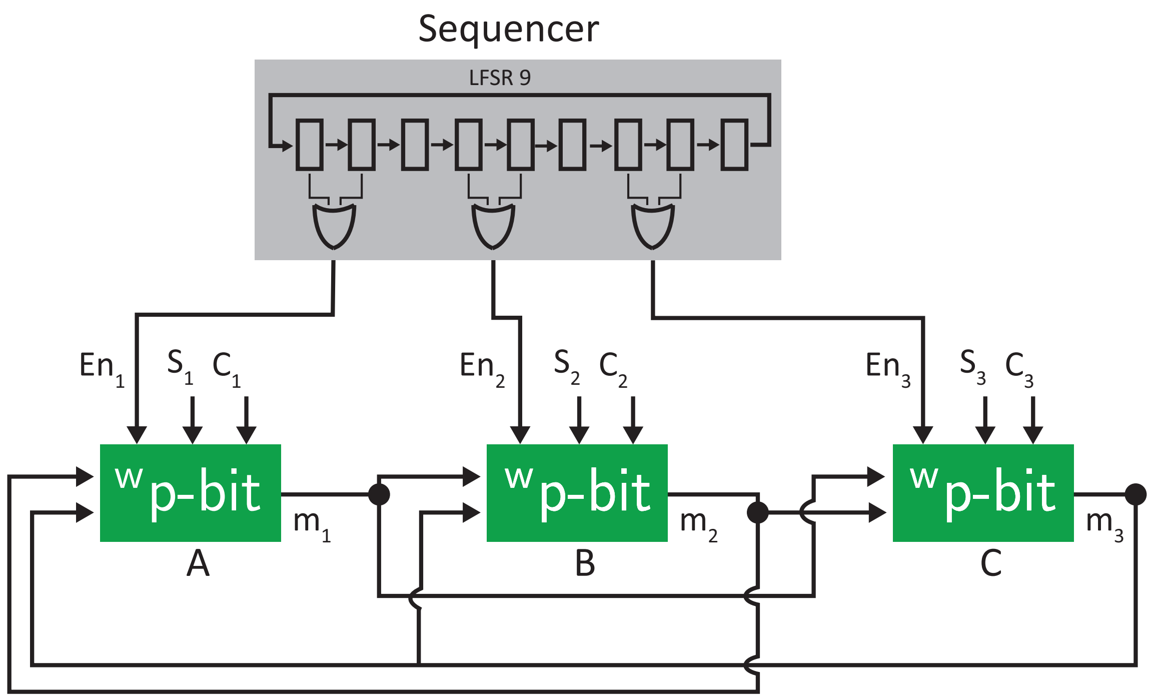


Fig. 5.5. **AND Gate.** Three weighted p-bits are needed to implement an AND gate whose [J] matrix is obtained from Ref. [12]. A sequencer circuit is used to force an updating sequence of (A → B → C).

5.3 Results

5.3.1 AND Gate

Fig.5.5 shows the block diagram of an AND gate that is implemented using 3 p-bits with each p-bit having two inputs. The weighting matrices [J] and $\{h\}$ are from Ref. [12] and shown below:

$$J_{AND} = \begin{array}{c} \begin{array}{ccc} A & B & C \end{array} \\ \begin{pmatrix} 0 & -1 & 2 \\ -1 & 0 & 2 \\ 2 & 2 & 0 \end{pmatrix} \end{array}, \quad h^T = \begin{pmatrix} 1 & 1 & -2 \end{pmatrix} \tag{5.1}$$

The p-circuit architecture of Ref. [7] forces m_i to be bipolar, i.e. $m_i \in \{1, -1\}$. It is more convenient to work with a binary representation of 1 and 0, i. e $m_i \in \{0, 1\}$, in the FPGA which requires that the $[J]$ and $\{h\}$ matrices be mapped to binary bases. This can be accomplished by the following transformation: $J_{binary} = 2 \times J_{bipolar}$ and $h_{binary} = h_{bipolar} - J_{bipolar}\mathbf{1}$, where $\mathbf{1}$ is an all ones vector of size $N \times 1$.

Floating mode (AND): Fig. 5.6 shows the operation of an AND gate with all weighted p-bits left floating, where the states $[ABC]$ corresponding to the truth table $(A \cap B = C)$ of an AND gate are visited with high probability. Note that this is a unique property of p-circuits with no counterpart in a digital CMOS implementation of an AND gate. In reciprocal networks with symmetric $[J]$ matrices, an energy functional E for the state $\{m\} = [m_i, \cdots, m_N]$ can be defined as [7, 58]:

$$E(\{m\}) = -I_0 \Bigg\{ \sum_{i,j} \frac{1}{2} J_{ij} m_i m_j + \sum_i h_i m_i \Bigg\} \tag{5.2}$$

Then, Boltzmann Law describes the steady state probabilities for each configuration {m} according to,

$$P(\{m\}) = \frac{\exp\big(-E(\{m\})\big)}{\sum_{i,j} \exp\big(-E(\{m\})\big)} \tag{5.3}$$

Fig. 5.6 shows the steady state statistics of the AND gate in excellent agreement with the Boltzmann law for a total of 10^6 samples.

Forward / Invertible mode (AND): Fig. 5.7(a) shows the statistics for the system when both inputs A and B have been clamped to 1 through the Select and Clamp signals that control the self-bias vector $\{h\}$. In this case, for the chosen I_0, the output C mostly stays high (1) which means the circuit is operating like a standard digital AND gate. The output bits (C) can also be clamped and in this case the inputs (A,B) fluctuate among combinations consistent with the clamped output. Fig. 5.7(b)

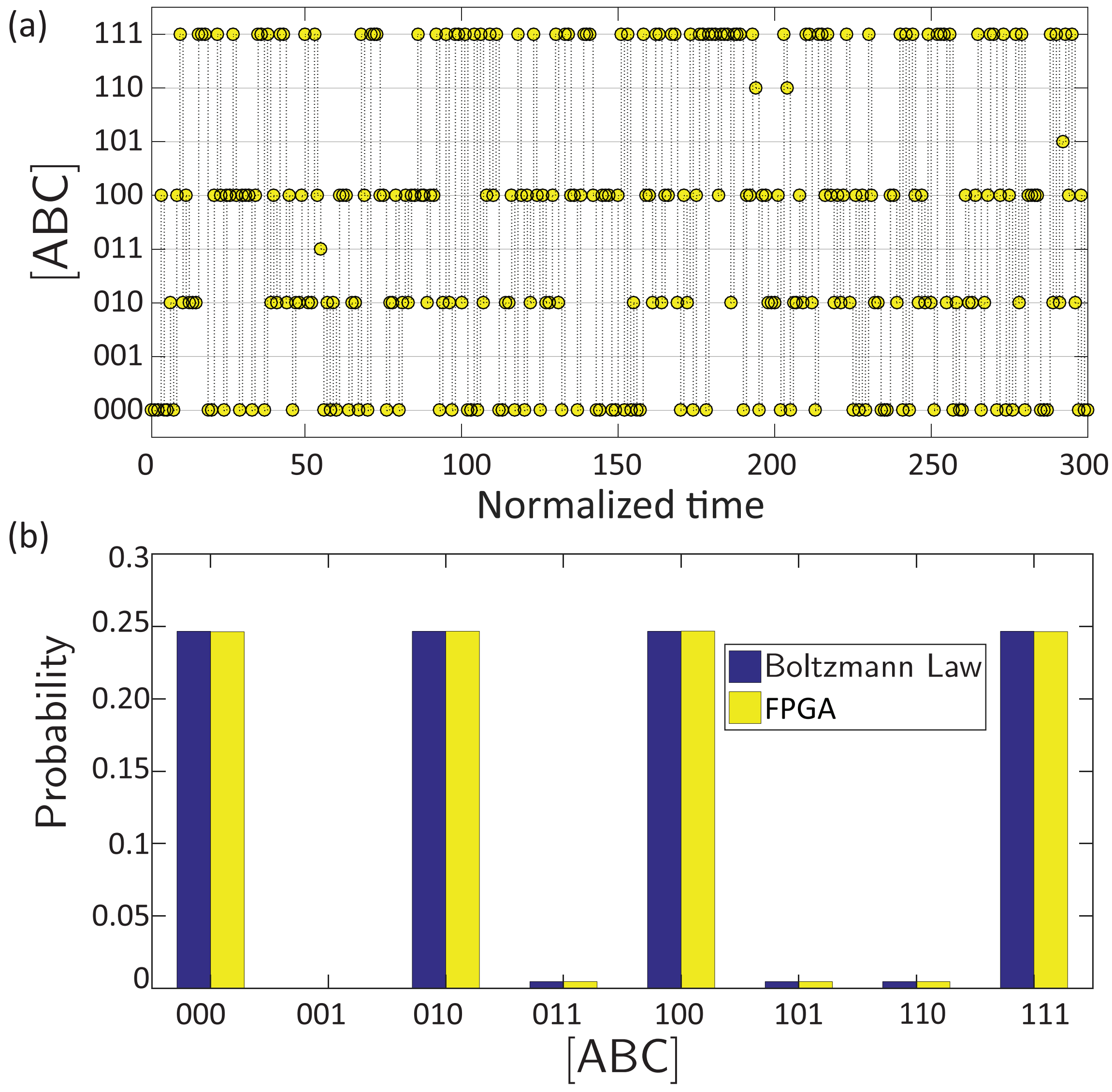


Fig. 5.6. **Floating mode (AND). (a)** Time dependent outputs of [ABC] for the AND gate are shown as a function of samples collected from the serial port of the FPGA. **(b)** The weighted p-bits are correlated and when left floating they reproduce the truth table of the AND Gate as shown by the time-averaged statistics which are collected using 10^6 samples. The FPGA results are in excellent agreement with the Boltzmann Law of Eq. 5.3.

shows the long time statistics of the system when the output C has been clamped to 0. It can be seen that the system spends an equal amount of time visiting three possible combinations of (A,B), namely (0,0), (0,1) and (1,0). This basic example

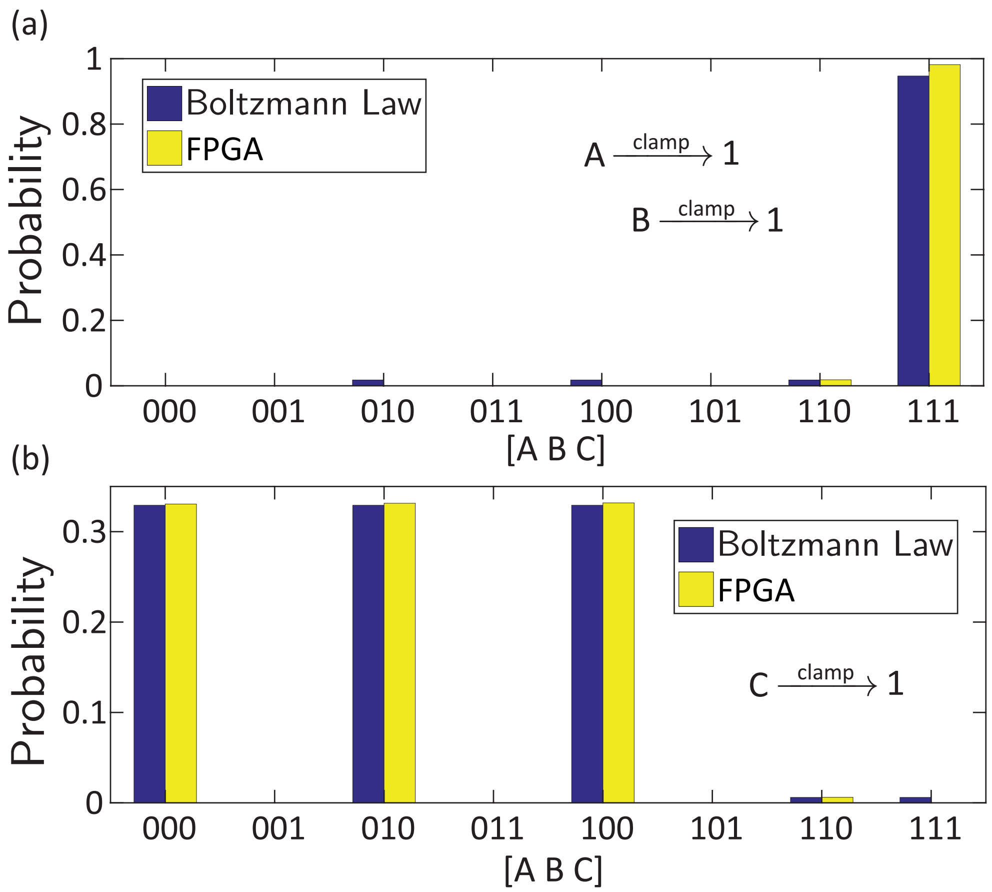


Fig. 5.7. **Forward / Invertible mode (AND).** Any weighted p-bit of the AND gate, input (A, B) or output (C) can be clamped using the bias vector {h} through the Select and Clamp signals. **(a)** shows the long time statistics when the inputs A and B have been clamped to 1, while **(b)** shows the long time statistics when the output C has been clamped to 0. In both cases 10^6 samples have been used.

can be imagined to be 1-bit factorization of an AND gate where the factors of the product 0 are identified.

5.3.2 Full Adder

A Full Adder was implemented as a p-circuit following the architecture of Ref. [7]. In Ref. [7] 14 p-bits are used to build a Full Adder, of which only 5 constitute input/output terminals, namely $\mathrm{C_{IN}}, \mathrm{A}, \mathrm{B}, \mathrm{Sum}$ and $\mathrm{C_{OUT}}$. The remaining 9 are known as “auxiliary” p-bits. In this paper, we improve the 14 p-bit implementation of the invertible Full Adder (FA) in Ref. [7] and implement the same functionality using 5 p-bits. This is achieved by first noting that the first half of the truth table is complementary to the second half for the FA (Fig 8). The first 4 lines in the truth table are then turned into an orthonormal set by a Gram-Schmidt process and a [J] matrix is obtained using Eq. 12 in Ref. [7] which is finally rounded to integer values, with diagonal entries replaced by zeroes

$$J_{FA} = \begin{array}{c} \begin{array}{ccccc} C_{in} & B & A & S & C_{out} \end{array} \\ \begin{pmatrix} 0 & -1 & -1 & 1 & 2 \\ -1 & 0 & -1 & 1 & 2 \\ -1 & -1 & 0 & 1 & 2 \\ 1 & 1 & 1 & 0 & -2 \\ 2 & 2 & 2 & -2 & 0 \end{pmatrix} \end{array} \tag{5.4}$$

These designs for the Full Adder fit within the 4×4 tiles that were defined previously with less packing efficiency, but since the design is re-configurable, appropriate changes can be made at a relatively low cost by scaling the tile size up or down. Similar to the AND gate we convert the weight matrix into their binary equivalents using the transformation shown earlier. A summary of resource utilization for the 14 p-bit Full adder is given in Table 5.2.

Fig. 5.8 shows the state of a 14 p-bit Full Adder when all the p-bits have been left floating. The truth table of the Full Adder is highlighted in floating mode and can be seen by the statistics shown in Fig. 5.8 that are collected using 10^6 samples, once again in excellent agreement with the Boltzmann Law. Due to its sequential updating, this Full Adder design requires $14 \times [2 + 1(gap)] = 42$ clock cycles for one complete update. The Full Adder is the largest p-circuit that we have built within

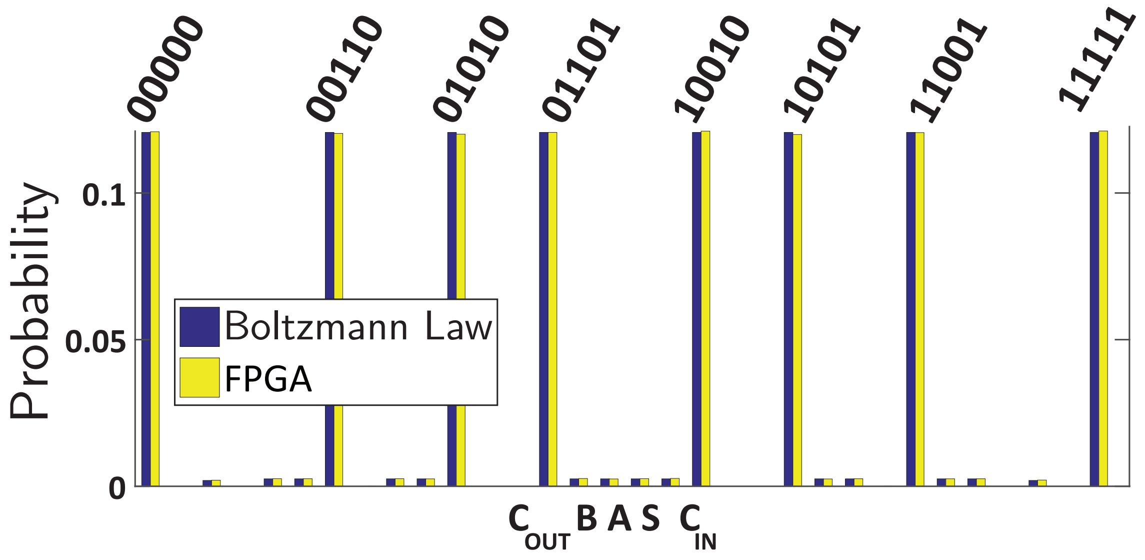


Fig. 5.8. **Floating mode Full Adder.** Long time statistics of the 14 weighted p-bit Full Adder using 10^6 samples when all the terminals have been left floating are shown in the figure above. Similar to the AND gate, the Full Adder reproduces its truth table when all p-bits are left floating. Results show excellent agreement with Boltzmann Law (Eq. 5.3).

a 4×4 tile, since each weighted p-bit within the Full Adder needs to be updated sequentially. In the next section we use this 14 p-bit Full Adder to construct a N-bit Ripple Carry Adder, while in section III-D we use a 5-bit Full Adder to solve a small instance of the SSP.

5.3.3 N-bit Ripple Carry Adders

Unlike the reciprocal networks ($J_{ij} = J_{ji}$) we have shown so far, we now construct a *directed* p-circuit by cascading the symmetric Full Adders in a *parallel* architecture without any global sequencer circuit. This is very different from the AND gate and Full Adder presented in sections III-A and III-B which are designed within a 4×4 tile, where each p-bit is updated sequentially. This serial-parallel update scheme significantly speeds up convergence time.

Fig. 5.9a shows the block diagram of multiple tiles, each designed as a Full Adder, that are interconnected in a directed way to form an N-bit Ripple Carry Adder (RCA). What makes the RCA directed is the fact that the carry out bit of the Full Adders is connected only from the least significant bit to the most significant bit but not vice versa. Note that this constitutes a significant difference from the AND and Full Adder because the Boltzmann Law is not applicable to this system anymore.

In this design we chose a fully directed connection between Full Adders. However, in general, tiles can be interconnected via an adjustable connection that can be partially (or fully) bidirectional using the terminal m_c shown in Fig. 5.1. This concept of directionality is key to building larger p-circuits as shown in Ref. [7] where a strong degree of bi-directional interconnections can lead to erroneous results unless a very large number of time samples are obtained.

The system shown Fig. 5.9a in general produces a sum (S) consistent when the inputs A, B are clamped to an N-bit number functioning as an adder. However, the system can also function as a subtractor when an N-bit input (A or B) and the sum are clamped to a given number, even though the system is no longer completely bidirectional. Fig. 5.9b shows the long time statistics for the N-bit Ripple Carry Adder when all N-bit terminals (S=sum, A and B) have been left floating and, remarkably, the system correlates in such a manner to select a single state (S$-$A$-$B=0) with $\approx 20\%$ probability out of 10^5 samples. This feature can be used to solve hard problems such as the 3-sum problem that is concerned with finding a set of inputs (A, B, C) that add up to a given sum S [61]. With minor modifications, the invertible Full Adders could also be used to solve the Subset Sum Problem, using similar adder architectures shown in [62]. A digital implementation such as our invertible N-bit adder could be used to solve such problems in hardware very efficiently.

The N-bit Ripple Carry Adder presented in Fig. 5.9 is *not* a true sequentially updated machine because while each adder takes 42 clock cycles to produce one complete update, the adders themselves do not wait $42 \times N$ for one complete update. In this way the N-bit Adder presented in Fig. 5.9 is a serial-parallel architecture different from the serially updated N-bit adders presented in Ref. [7]. This serial-parallel update for the N-bit RCA seems to operate accurately for the deterministic update sequence we chose, but it is not clear if this approach would be generally applicable for any problem, which is beyond the scope of this paper. While we do not give a quantitative analysis of the speed up from this serial-parallel architecture, we note that in the case of the invertible N-bit RCA, this serial-parallel architecture combined with fast clock speeds of the FPGA should allow considerably faster operation of this large scale p-circuit as compared to computer simulations.

5.3.4 Subset Sum Problem

In this subsection we show how the Full Adder block and the serial-parallel architecture of the N-bit Ripple Carry Adder can be used to solve a small instance of the NP-complete Subset Sum Problem (SSP) [61]. In this problem, a set G with a finite number of positive numbers is defined, and from this set the problem is to determine whether there exists a subset S' such that $S' \subseteq G$ has elements which sum to a specific target S. Figure. 5.10 shows a circuit that can be programmed to select a 17-bit sum S, while the 15-bit inputs are constrained to particular sets. In the example shown in Fig.5.10, the sum S is set to 3584 while the inputs A, B and C are constrained to the sets $\{0, 512\}$, $\{0, 512\}$ and $\{0, 512\}$ respectively. Note that in Fig. 5.10(b) we show terminals h_{XX} for ease of visualization where $h = -V$ means that the *select* and *clamp* lines are connected to 1 and 0 respectively, while $h = +V$ means that the *select* and *clamp* lines are connected to 1 and 1 respectively, and $h = 0$ means that

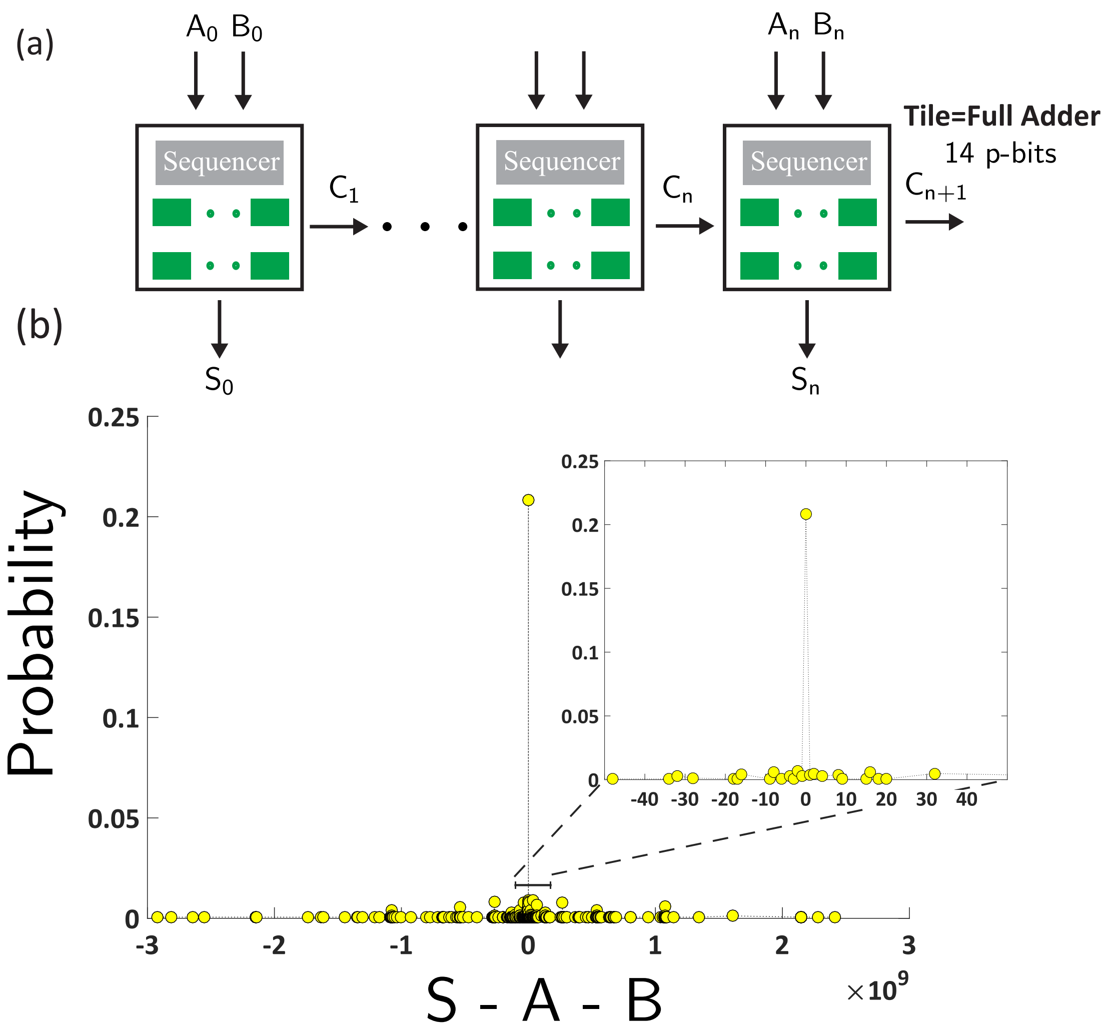


Fig. 5.9. **N-bit Ripple Carry Adder.** **(a)** Reciprocal networks of individual tiles programmed as Full Adders are interconnected in a directed manner to construct an N-bit Ripple Carry Adder (RCA). **(b)** The RCA is left floating and the long time statistics of the N-bit sum (S) and the inputs (A, B) get correlated in such a way to make a *single state* (inset) S−A−B = 0 appear with ≈ 20% probability out of 10^5 samples among billions of states ($\pm 2^{32}$), as can be seen in the x-axis. Only ≈ 1500 samples are shown for clarity.

the select line is connected to 0. One striking feature of this circuit is that the flow of information in the structure shown in Fig. 5.10(a) is upwards, i.e. information flows from the Sum to inputs A and B.

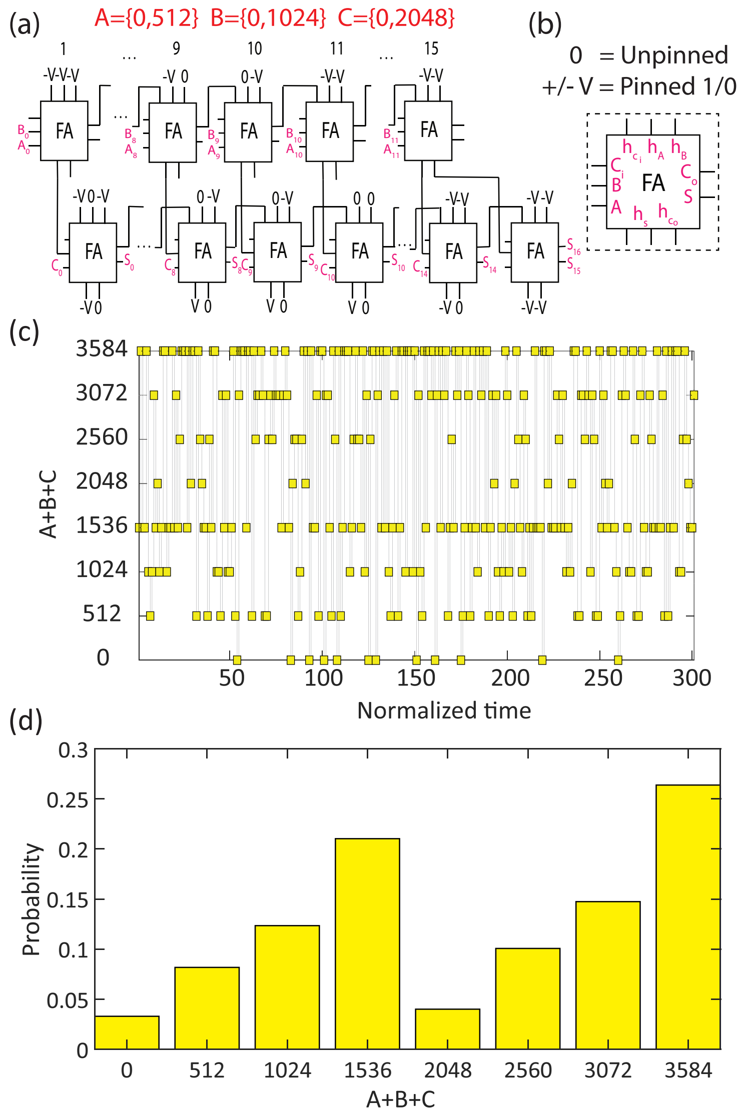


Fig. 5.10. **Subset Sum Problem.** **(a)** An adder that adds three 15-bit numbers A,B and C to give a 17-bit Sum S. The Sum S is first clamped to a particular number which in this case is 3584. The inputs A, B and C are constrained to particular sets using a scheme shown in **(b)** for each bit of the inputs. Note how the connections from the bottom layer of adders are *directed* where the sum is clamped to the top layer where the inputs A and B are added. In this example A is $\{0, 512\}$, B is $\{0, 1024\}$, and C is $\{0, 2048\}$. **(c)** shows 300 time samples taken from a data sequence of 10^5 from which two values of 3584 and 1536 appear more then the other 6 possible states. **(d)** shows the histogram corresponding to 10^5 samples.

The inputs can be constrained to sets by clamping certain bits to 0 or 1 depending on the choice of the set. For the input A used in Fig. 5.10, all the bits except the 9^{th} from the LSB side are clamped to 0. Clamping the bits hence allows the inputs to be *constrained* to particular sets, forcing the circuit to look within a certain configuration consistent with the members of the set. Fig. 5.10(c) shows 300 time samples of A+B+C taken from a data sequence of 10^5 when the Sum is clamped to 3584. In this case where the correct inputs are A=512, B=1024, and C=2048, it can be seen that two states one correct 3584 and another incorrect 1536 appear closer to each other and far removed from the other 6 possible combinations that the system could be in. The relative probabilities of different peaks in the solution (such as 1536 and 3584) is a function of the inverse pseudo-temperature (I_0 in Eq. 4) and could be made larger by increasing this value. However, in practice this could cause the system to get stuck in a meta-stable state for a long time. Therefore, for this example we have chosen a relatively small $I_0 = 1$ that does not create a large difference in probabilities. Fig. 5.10(d) shows the statistics for the entire 10^5 samples from which the state 3584 has a higher peak than 1536. We note that this particular example of the SSP is easily solvable and does not constitute a hard instance. Our main purpose is to illustrate how invertible Full Adders can be interconnected to design a hardware solver for this problem, similar in spirit to the approach described in Ref. [62].

6. CONCLUSION

6.1 Summary

Fig. 6.1 shows a microcontroller based emulation of p-bits where the inputs and outputs are actual voltages and the time constants τ_N (Retention time) and τ_{inter} (Interconnect delay) are user programmable. This allowed us to study the interplay between the time constants and develop a hardware platform for future nano-devices. We constructed large circuits with upto 48 p-bits working in invertible mode even in the presence of retention time variations. One striking finding of this work was how asynchronous operation of p-bits naturally allowed for serial updating of p-bits; a well-established requirement in purely software implementations.

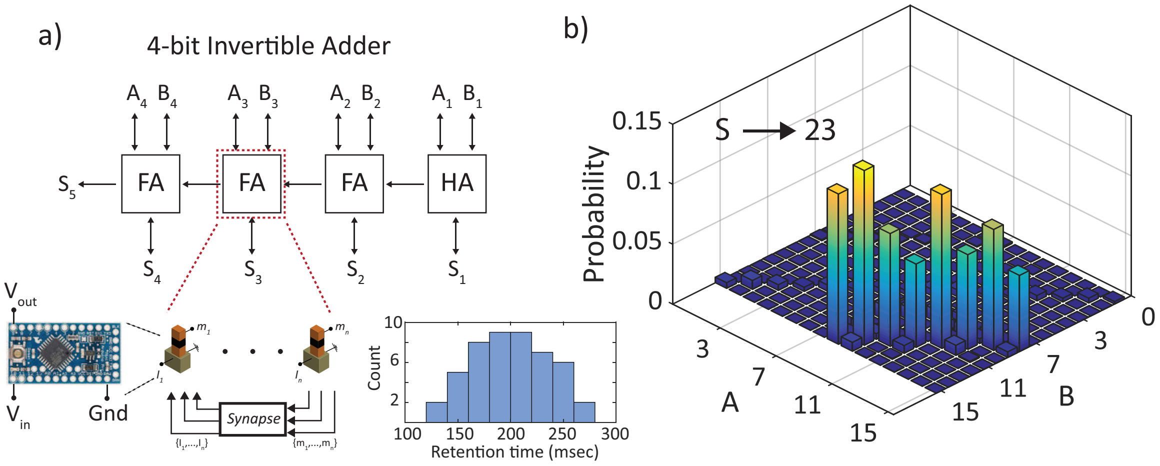


Fig. 6.1. **Microcontroller based p-bits.** **(a)** 4-bit Invertible Adder built using 3 full adders and a half adder using 48 total p-bits, with the retention time of p-bits distributed from 120-28 ms. **(b)** shows the long time average statistics when the sum is clamped to 23.

Fig. 6.2(a) shows a proof-of-concept probabilistic computer built using a slight modification to market ready STT-MRAM. The energy barries ΔE in STT-MRAM based memory devices is lowered by reducing the magnetic anisotropy of the free layer which leads to a stochastic MTJ fluctuating between R_P and R_{AP}. This proof-of-concept demonstration addresses many important questions such as how the system copes with parameter variations among MTJs and the area/energy improvements over CMOS based implementations. Fig. 6.2(b) shows a table top spintronics based probabilistic computer which is used to solve an integer factorization problem shown in c).

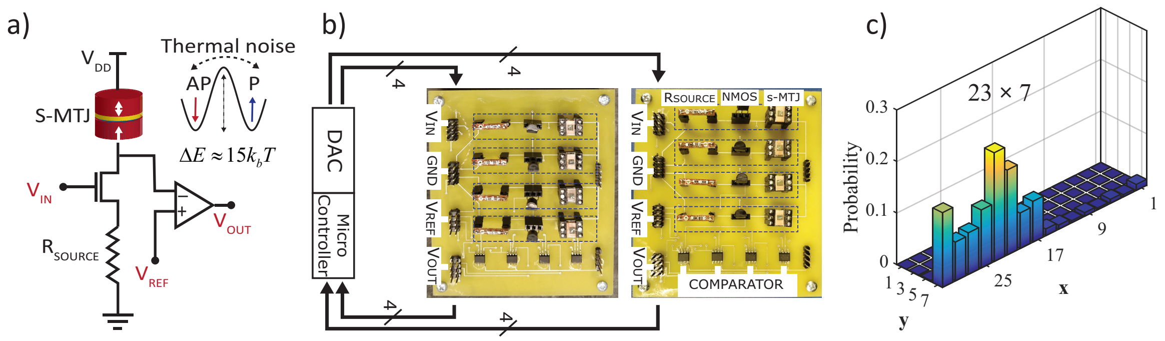


Fig. 6.2. **Probabilistic computing using STT-MRAM devices.** **(a)** shows an electrical schematic of a experimental p-bit which uses 40-60 nm stochastic MTJs. **(b)** shows a photograph of a printed circuit board which uses 8 p-bits to realize to a probabilistic computer. **(c)** shows integer factorization of $161 = 23 \times 7$

Fig. 6.3 shows a lower level implementation of a p-bit, where a multiplexer is used as a drop in replacement for a stochastic MTJ. This implementation could be used to study various interesting phenomenons such as the effect of pinning in stochastic MTJs and variations in resistances among MTJs. Fig. 6.3(b) shows an integer factorization problem cast as an optimization problem, where a cost function corresponding to the number 35 was minimized to obtain two factors X and Y.

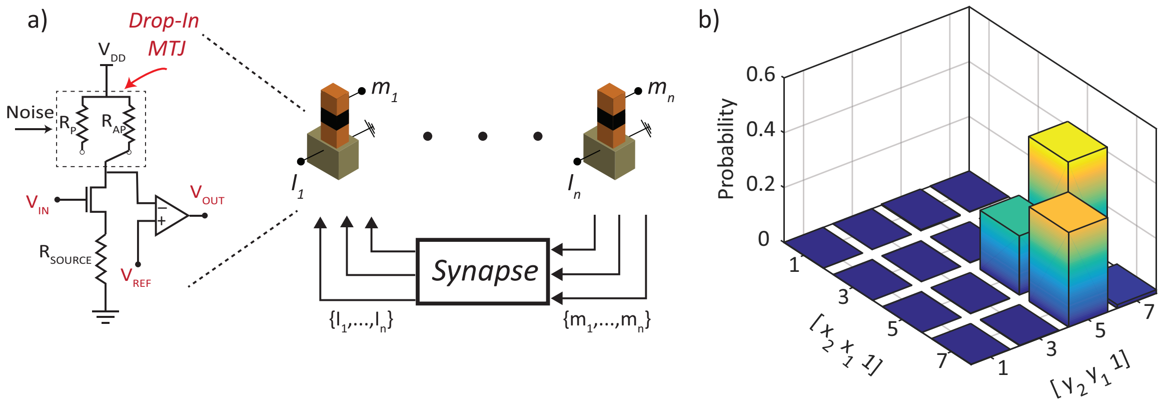


Fig. 6.3. **Analog multiplexer based p-bits.** **(a)** shows an analog multiplexer that functions as a drop in replacement for an MTJ. **(b)** shows a 4 bit probabilistic computer which is used to factorize 35 using 4 p-bits.

While nano-device based implementations offer significant improvements in area and energy, scaled implementations are not possible currently. To realize large scale probabilistic computers one could utilize off the shelf FPGA technology. Fig. 6.4(a) shows a p-bit realized on an FPGA and Fig. 6.4(b) shows a 32-bit invertible adder which uses approximately 500 of such p-bits. This FPGA based implementation can be used to study large scale problems in present while we work towards nano-device based implementations of stochastic computers.

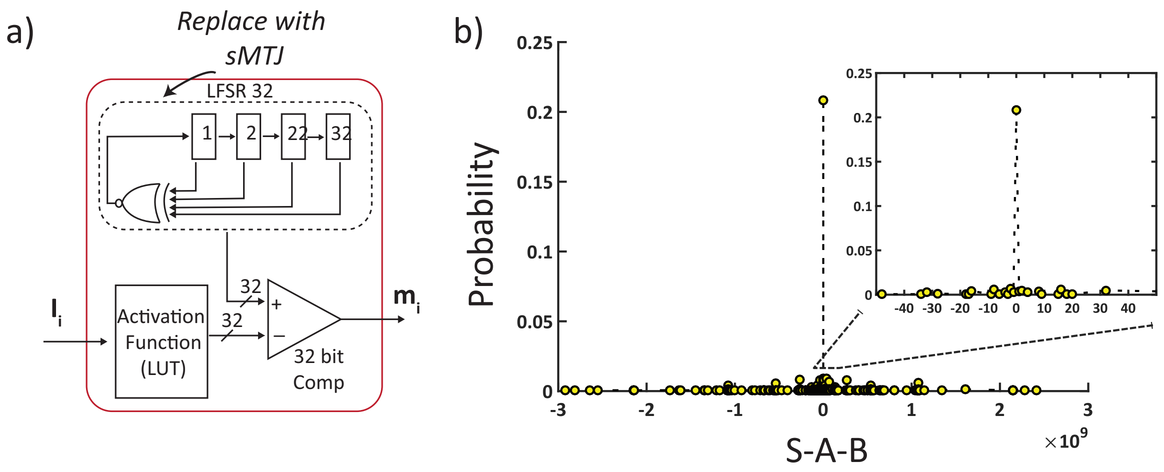


Fig. 6.4. **FPGA based probabilistic computer.** **(a)** shows a block diagram of FPGA based p-bit. A 32-bit invertible adder was built using 500 p-bits as shown in **(b)**.

6.2 Future Work

In this thesis we have presented an autonomous probabilistic computer. At the heart of this demonstration is the probabilistic bit. A classical resistance fluctuating back and forth between two states. These fluctuations have been introduced naturally by using an unstable stochastic MTJ; a functionality which would otherwise require thousands of transistors. MRAM based memory technology is currently available at the Gb density levels. Now assuming a power budget of 10 W for p-bits, only a million p-bits can be realized on a chip, which is 3 orders of magnitude smaller in density than memory chips. Firstly, improvements in the p-bit design are needed to ensure scaling of p-computers. Secondly, decreasing the retention time of p-bits would increase the number of effective flips per second of the p-computer, greatly improving speed.

REFERENCES

[1] R. P. Feynman, "Simulating physics with computers," *International journal of theoretical physics*, vol. 21, no. 6-7, pp. 467–488, 1982.

[2] J. Von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," *Automata studies*, vol. 34, pp. 43–98, 1956.

[3] B. R. Gaines *et al.*, "Stochastic computing systems," *Advances in information systems science*, vol. 2, no. 2, pp. 37–172, 1969.

[4] W. J. Poppelbaum, C. Afuso, and J. W. Esch, "Stochastic computing elements and systems," in *Proceedings of the November 14-16, 1967, Fall Joint Computer Conference*, ser. AFIPS '67 (Fall). New York, NY, USA: ACM, 1967, pp. 635–644. [Online]. Available: http://doi.acm.org/10.1145/1465611.1465696

[5] Y. V. Pershin and M. Di Ventra, "Experimental demonstration of associative memory with memristive neural networks," *Neural Networks*, vol. 23, no. 7, pp. 881–886, 2010.

[6] "Arduino - www.arduino.cc."

[7] K. Y. Camsari, R. Faria, B. M. Sutton, and S. Datta, "Stochastic p-bits for invertible logic," *Phys. Rev. X*, vol. 7, p. 031014, Jul 2017. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevX.7.031014

[8] W. F. Brown Jr, "Thermal fluctuations of a single-domain particle," *Physical Review*, vol. 130, no. 5, p. 1677, 1963.

[9] L. Lopez-Diaz, L. Torres, and E. Moro, "Transition from ferromagnetism to superparamagnetism on the nanosecond time scale," *Physical Review B*, vol. 65, no. 22, p. 224406, 2002.

[10] W. A. Borders, A. Z. Pervaiz, S. Fukami, K. Y. Camsari, H. Ohno, and S. Datta, "Integer factorization using stochastic magnetic tunnel junctions," *Nature*, vol. 573, no. 7774, pp. 390–393, 2019.

[11] "Maxim dac - www.maximintegrated.com."

[12] J. Biamonte, "Nonperturbative k-body to two-body commuting conversion hamiltonians and embedding problem instances into ising spins," *Physical Review A*, vol. 77, no. 5, p. 052331, 2008.

[13] D. J. Amit, *Modeling brain function: The world of attractor neural networks.* Cambridge University Press, 1992.

[14] H. Suzuki, J.-i. Imura, Y. Horio, and K. Aihara, "Chaotic boltzmann machines," *Scientific reports*, vol. 3, p. 1610, 2013.

[15] T. Albash and D. A. Lidar, "Adiabatic quantum computation," *Reviews of Modern Physics*, vol. 90, no. 1, p. 015002, 2018.

[16] X. Peng, Z. Liao, N. Xu, G. Qin, X. Zhou, D. Suter, and J. Du, "Quantum adiabatic algorithm for factorization and its experimental implementation," *Physical review letters*, vol. 101, no. 22, p. 220405, 2008.

[17] S. Jiang, K. A. Britt, T. S. Humble, and S. Kais, "Quantum annealing for prime factorization," *arXiv preprint arXiv:1804.02733*, 2018.

[18] D. H. Ackley, G. E. Hinton, and T. J. Sejnowski, "A learning algorithm for boltzmann machines," *Cognitive science*, vol. 9, no. 1, pp. 147–169, 1985.

[19] A. D. Kent and D. C. Worledge, "A new spin on magnetic memories," *Nature nanotechnology*, vol. 10, no. 3, p. 187, 2015.

[20] K. Y. Camsari, S. Salahuddin, and S. Datta, "Implementing p-bits with embedded mtj," *IEEE Electron Device Letters*, vol. 38, no. 12, pp. 1767–1770, 2017.

[21] A. Fukushima, T. Seki, K. Yakushiji, H. Kubota, H. Imamura, S. Yuasa, and K. Ando, "Spin dice: A scalable truly random number generator based on spintronics," *Applied Physics Express*, vol. 7, no. 8, p. 083001, 2014.

[22] A. Mizrahi, T. Hirtzlin, A. Fukushima, H. Kubota, S. Yuasa, J. Grollier, and D. Querlioz, "Neural-like computing with populations of superparamagnetic basis functions," *Nature communications*, vol. 9, no. 1, p. 1533, 2018. [Online]. Available: http://doi.org/10.1038/s41467-018-03963-w

[23] S. Ikeda, K. Miura, H. Yamamoto, K. Mizunuma, H. Gan, M. Endo, S. Kanai, J. Hayakawa, F. Matsukura, and H. Ohno, "A perpendicular-anisotropy cofeb–mgo magnetic tunnel junction," *Nature materials*, vol. 9, no. 9, p. 721, 2010.

[24] M. Julliere, "Tunneling between ferromagnetic films," *Physics letters A*, vol. 54, no. 3, pp. 225–226, 1975.

[25] M. Endo, S. Kanai, S. Ikeda, F. Matsukura, and H. Ohno, "Electric-field effects on thickness dependent magnetic anisotropy of sputtered mgo/co 40 fe 40 b 20/ta structures," *Applied Physics Letters*, vol. 96, no. 21, p. 212503, 2010.

[26] G. D. Chaves-OFlynn, G. Wolf, J. Z. Sun, and A. D. Kent, "Thermal stability of magnetic states in circular thin-film nanomagnets with large perpendicular magnetic anisotropy," *Physical Review Applied*, vol. 4, no. 2, p. 024010, 2015.

[27] J. C. Slonczewski, "Current-driven excitation of magnetic multilayers," *Journal of Magnetism and Magnetic Materials*, vol. 159, no. 1-2, pp. L1–L7, 1996.

[28] L. Berger, "Emission of spin waves by a magnetic multilayer traversed by a current," *Physical Review B*, vol. 54, no. 13, p. 9353, 1996.

[29] A. Brataas, A. D. Kent, and H. Ohno, "Current-induced torques in magnetic materials," *Nature materials*, vol. 11, no. 5, p. 372, 2012.

[30] A. Pervaiz, L. Ghantasala, K. Camsari, and S. Datta, "Hardware emulation of stochastic p-bits for invertible logic." *Scientific reports*, vol. 7, no. 1, p. 10994, 2017.

[31] K. Y. Camsari, S. Chowdhury, and S. Datta, "Scalable emulation of sign-problem–free hamiltonians with room-temperature p-bits," *Physical Review Applied*, vol. 12, no. 3, p. 034061, 2019.

[32] T. Kleinjung, K. Aoki, J. Franke, A. K. Lenstra, E. Thomé, J. W. Bos, P. Gaudry, A. Kruppa, P. L. Montgomery, D. A. Osvik *et al.*, "Factorization of a 768-bit rsa modulus," in *Annual Cryptology Conference.* Springer, 2010, pp. 333–350.

[33] Y. K. Lee, Y. Song, J. Kim, S. Oh, B.-J. Bae, S. Lee, J. Lee, U. Pi, B. Seo, H. Jung *et al.*, "Embedded stt-mram in 28-nm fdsoi logic process for industrial mcu/iot application," in *2018 IEEE Symposium on VLSI Technology.* IEEE, 2018, pp. 181–182.

[34] G. O. Roberts and S. K. Sahu, "Updating schemes, correlation structure, blocking and parameterization for the gibbs sampler," *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, vol. 59, no. 2, pp. 291–317, 1997.

[35] E. C. I. Enobio, M. Bersweiler, H. Sato, S. Fukami, and H. Ohno, "Evaluation of energy barrier of cofeb/mgo magnetic tunnel junctions with perpendicular easy axis using retention time measurement," *Japanese Journal of Applied Physics*, vol. 57, no. 4S, p. 04FN08, 2018.

[36] A. Lucas, "Ising formulations of many np problems," *Frontiers in Physics*, vol. 2, p. 5, 2014.

[37] K. Y. Camsari, R. Faria, B. M. Sutton, and S. Datta, "Stochastic p-bits for invertible logic," *Physical Review X*, vol. 7, no. 3, p. 031014, 2017.

[38] N. Xu, J. Zhu, D. Lu, X. Zhou, X. Peng, and J. Du, "Quantum factorization of 143 on a dipolar-coupling nuclear magnetic resonance system," *Physical review letters*, vol. 108, no. 13, p. 130501, 2012.

[39] C. J. Burges, "Factoring as optimization," *Microsoft Research MSR-TR-200*, 2002.

[40] P. Henelius and S. Girvin, "A statistical mechanics approach to the factorization problem," *arXiv preprint arXiv:1102.1296*, 2011.

[41] R. Dridi and H. Alghassi, "Prime factorization using quantum annealing and computational algebraic geometry," *Scientific reports*, vol. 7, p. 43048, 2017.

[42] A. Z. Pervaiz, B. M. Sutton, L. A. Ghantasala, and K. Y. Camsari, "Weighted p-bits for fpga implementation of probabilistic circuits," *IEEE transactions on neural networks and learning systems*, vol. 30, no. 6, pp. 1920–1926, 2018.

[43] R. Zand, K. Y. Camsari, S. Datta, and R. F. DeMara, "Composable probabilistic inference networks using mram-based stochastic neurons," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 15, no. 2, p. 17, 2019.

[44] S. K. Mathew, D. Johnston, S. Satpathy, V. Suresh, P. Newman, M. A. Anders, H. Kaul, A. Agarwal, S. K. Hsu, G. Chen *et al.*, "μrng: A 300–950 mv, 323 gbps/w all-digital full-entropy true random number generator in 14 nm finfet cmos," *IEEE Journal of Solid-State Circuits*, vol. 51, no. 7, pp. 1695–1704, 2016.

[45] W. Zhao and Y. Cao, “New generation of predictive technology model for sub-45 nm early design exploration,” *IEEE Transactions on Electron Devices*, vol. 53, no. 11, pp. 2816–2823, 2006.

[46] K. Y. Camsari, S. Salahuddin, and S. Datta, “Implementing p-bits with embedded mtj,” *IEEE Electron Device Letters*, vol. 38, no. 12, pp. 1767–1770, Dec 2017.

[47] D. Datta, B. Behin-Aein, S. Datta, and S. Salahuddin, “Voltage asymmetry of spin-transfer torques,” *IEEE Transactions on Nanotechnology*, vol. 11, no. 2, pp. 261–272, 2011.

[48] C. Park, H. Lee, C. Ching, J. Ahn, R. Wang, M. Pakala, and S. Kang, “Low ra magnetic tunnel junction arrays in conjunction with low switching current and high breakdown voltage for stt-mram at 10 nm and beyond,” in *2018 IEEE Symposium on VLSI Technology.* IEEE, 2018, pp. 185–186.

[49] S. Geman and D. Geman, “Stochastic relaxation, gibbs distributions, and the bayesian restoration of images,” *IEEE Transactions on pattern analysis and machine intelligence*, no. 6, pp. 721–741, 1984.

[50] M. W. Johnson, M. H. Amin, S. Gildert, T. Lanting, F. Hamze, N. Dickson, R. Harris, A. J. Berkley, J. Johansson, P. Bunyk *et al.*, “Quantum annealing with manufactured spins,” *Nature*, vol. 473, no. 7346, p. 194, 2011.

[51] M. Yamaoka, C. Yoshimura, M. Hayashi, T. Okuyama, H. Aoki, and H. Mizuno, “A 20k-spin ising chip to solve combinatorial optimization problems with cmos annealing,” *IEEE Journal of Solid-State Circuits*, vol. 51, no. 1, pp. 303–309, 2016.

[52] B. Sutton, K. Y. Camsari, B. Behin-Aein, and S. Datta, “Intrinsic optimization using stochastic nanomagnets,” *Scientific Reports*, vol. 7, 2017.

[53] B. Behin-Aein, V. Diep, and S. Datta, “A building block for hardware belief networks,” *Scientific reports*, vol. 6, p. 29893, 2016.

[54] B. Sutton, R. Faria, L. A. Ghantasala, K. Y. Camsari, and S. Datta, “Autonomous probabilistic coprocessing with petaflips per second,” *arXiv preprint arXiv:1907.09664*, 2019.

[55] C. D. Schuman, T. E. Potok, R. M. Patton, J. D. Birdwell, M. E. Dean, G. S. Rose, and J. S. Plank, “A survey of neuromorphic computing and neural networks in hardware,” *CoRR*, vol. abs/1705.06963, 2017.

[56] M. Tommiska, “Efficient digital implementation of the sigmoid function for reprogrammable logic,” *IEE Proceedings-Computers and Digital Techniques*, vol. 150, no. 6, pp. 403–411, 2003.

[57] www.xilinx.com/support/documentation/application_notes/xapp210.pdf.

[58] G. E. Hinton, “Boltzmann machine,” *Scholarpedia*, vol. 2, no. 5, p. 1668, 2007.

[59] W. Vinci, T. Albash, G. Paz-Silva, I. Hen, and D. A. Lidar, “Quantum annealing correction with minor embedding,” *Physical Review A*, vol. 92, no. 4, p. 042310, 2015.

[60] V. Choi, "Minor-embedding in adiabatic quantum computation: Ii. minor-universal graph design," *Quantum Information Processing*, vol. 10, no. 3, pp. 343–353, 2011.

[61] T. H. Cormen, *Introduction to algorithms.* MIT press, 2009.

[62] F. L. Traversa and M. D. Ventra, "Polynomial-time solution of prime factorization and np-complete problems with digital memcomputing machines," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 27, no. 2, p. 023107, 2017.

A. EXPERIMENTAL DATA

A.1 Experimental data for s-MTJ based p-computer and its emulation

In this section we list down all of the experimental data for the s-MTJ based p-computer and its low-level emulation using analog multiplexors. The data was recorded in MATLAB readable format via the Benchvue software suite which connects directly to the Keysight Oscilloscope. All experimental data has been packed with MATLAB wrapper files for easy viewing and comments have been added to facilitate the process.

Nature experimental data

Sub-Folder	Experiment	MATLAB File
Figures_4pbits	Factorizing 35 and 49 (Fig. 3.3)	Tomography.m
Figures_6pbits	Factorizing 161 and 213 (Fig. 3.3)	Tomography.m
Figures_8pbits	Factorizing 705 and 945 (Fig. 3.3)	Tomography.m
Figures_Sigmoid	Sigmoid (Fig. 3.2)	Sigmoid.m
Figures_ANDgate	AND Gate (Fig. 3.13)	BenchVueReading.m

Analog multiplexer experimental data

Sub-Folder	Experiment	MATLAB File
Figures_4pbits	Factorizing 35 and 49 (Fig. 4.6)	Tomography.m
Figures_6pbits	Factorizing 161 (Fig. 4.8)	Tomography_4plus2.m
Figures_8pbits	Factorizing 945 (Fig. 4.8)	Tomography_5plus3.m
Figures_Sigmoid	Sigmoid (Fig. 4.2)	Sigmoid.m

A.2 Arduino Sketches for synapse

A folder titled "Arduino_files" has been provided along with the experimental data. To use the sketches for the synapse, appropriate libraries for the micro-controller needs to added which have been provided within the folder "Arduino_files"

Working with digital-to-analog converters: The DACs used in this thesis either used the I_2C or the *SPI* protocol. Libraries for each of the two types is provided for use with the Arduino Mega2560 microcontroller. Consider for example using the MAXIM 5825 DAC which communicates over the I_2C protocol.

- **System initialization:** Each DAC needs to be addressed by a physical address which is set physically using jumpers on the IC itself.
- **Setting a reference voltage:** Each DAC needs to have a reference voltage which is used for setting analog voltages. In our experiments we always use the internal references available on the DAC since they are low noise signals.
- **Setting the analog voltages:** This is done by following the byte sequence listed in the programming specifications of the particular DAC being used. For example, to write a voltage of 2.5 V to channel 4 of the DAC whose address is set at “0x20”, we could send the following 4 bytes over the I_2C interface: byte1 [0010000], byte2 [10110011], byte3 [10000000], byte4 [00000000].

Libraries were written to internalize these operations, allowing the user to simply set voltages using a single write command that only uses the channel number and voltage for operation. These have been provided as part of the data files.