A Fully Automated Technique for Constructing FSM Abstractions of Non-Ideal Latches in Communication Systems

Karthik .V. Aadithya*‡ Yingyan Lin† Chenjie Gu* Aolin Xu† Jaijeet Roychowdhury* Naresh Shanbhag†
*Department of Electrical Engineering and Computer Sciences, The University of California at Berkeley, CA, USA
†Department of Electrical and Computer Engineering, The University of Illinois at Urbana-Champaign, Illinois, USA
‡Contact author. Email: aadithya@berkeley.edu

***Abstract*—The design of a communications system is typically most effective only when each of its components can be accurately represented by a discrete, symbolic behavioural abstraction. Such abstractions, in addition to providing valuable design intuition, also enable highly efficient and scalable system-level simulation. However, given a SPICE-level description for a subsystem such as a latch, it is a challenge to come up with a discrete, symbol-level abstraction that accurately captures its continuous-time dynamics. Indeed, the manual construction of such an abstraction requires deep knowledge and understanding of the operation of the module in question; moreover, it is very time-consuming, tedious, error-prone and not easily scalable to larger designs.**

In recent work [1], we adapted methods from computational learning theory to develop an automated technique, DAE2FSM, that produces binary finite state machine (FSM) abstractions of non-linear analog/mixed-signal (AMS) circuits. In the present paper, we demonstrate the application of the DAE2FSM technique to automatically derive FSM abstractions for a mixed-signal communications circuit component, namely a current mode latch (CML) designed in IBM's 90nm LP process technology. We show that the FSMs learned by DAE2FSM not only capture the essence of the latch's behaviour during normal conditions, but also faithfully mimic its behaviour under adverse operating conditions (e.g., under lowered supply voltages). Moreover, in addition to a stand-alone CML, we also generate FSMs for cascades of two and three latches (such topologies are used in the design of power-efficient, bit-error optimised analog-to-digital converters). In spite of the inherent non-linearity of such systems, and in spite of the pronounced "analog-ness" of the waveforms in question, our FSM abstractions are able to produce discrete-time symbol sequences that closely match the data points obtained by sampling from continuous-time SPICE simulations.

I. Introduction

Analog/Mixed-Signal (AMS) modules play a crucial role in today's high-speed (i.e., several Gb/s), low power, bit-error optimised digital communication systems. In particular, for delivering high performance at low power, one needs to impose stringent design constraints on AMS modules such as the analog-to-digital converters (ADCs) and digital-to-analog converters (DACs) that make up the front and back ends of high-speed communication links. Most other modules in a digital communications system can be analysed/simulated very efficiently; either they are continuous-time linear systems (e.g., anti-aliasing filters), or they can be accurately described by convenient discrete-time abstractions (e.g., FIR/IIR filters, FFT modules). By contrast, the AMS blocks in the design pose a significant challenge both for analysis and for system-level simulation: these subsystems are highly non-linear, and it is also not straight-forward to derive discrete-time abstractions that accurately capture the rich set of continuous-time dynamics exhibited by these subsystems. Accordingly, this paper presents a fully automated technique, DAE2FSM, that is able to generate discrete, symbol-level abstractions (in the form of finite state machines, or FSMs) for AMS blocks used in high-speed communication systems.

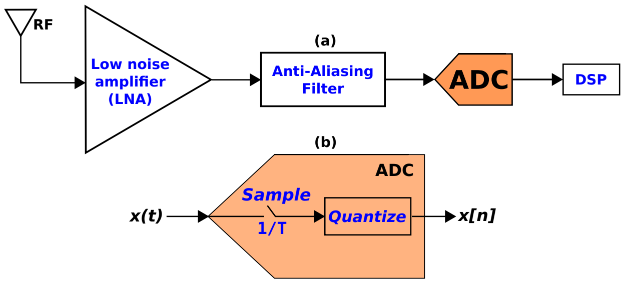

Fig. 1. Role of an AMS block (i.e., an ADC) in an RF receiver.

To set the context of this paper in more concrete terms, Fig. 1 (a) shows a block diagram of the front end of an RF receiver. As the figure shows, the ADC represents the first major module through which any received signal has to pass, well before any digital processing can even begin. Thus, the ADC's performance has a strong bearing on the performance of the entire communication link. Fig. 1 (b) shows an abstract model for how the ADC is supposed to function: the incoming signal $x(t)$ is first sampled at a suitable frequency $1/T$ (which is at the Nyquist rate or higher), and the sampled analog sequence is then quantized to discrete levels, which results in the output sequence $x[n]$.

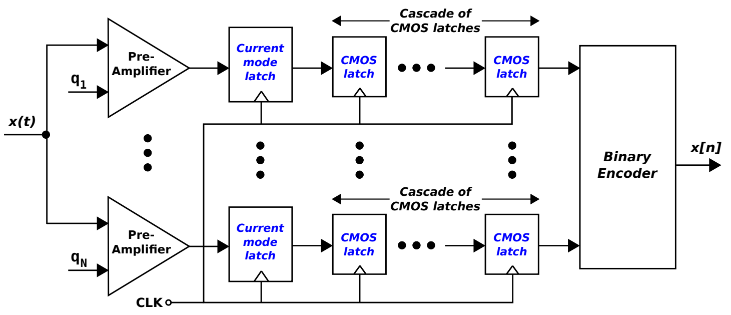

Fig. 2. Schematic of a bit-error optimised flash ADC.

Fig. 2 shows a schematic design of a low power flash ADC suitable for use in high-speed communication links [2]. As is typical in a flash ADC, the input signal $x(t)$ is compared against N quantization thresholds $\{q_1, q_2, \ldots, q_N\}$. This is accomplished with N pre-amplifiers, each of whose outputs is then passed through a cascade of latches. The first latch in each of these cascades is a current mode latch (please see Fig. 6 (a) for a circuit diagram), and all subsequent latches are CMOS latches.

The basic structural and functional unit in the above ADC topology is a cascade of latches. To design the ADC, therefore, it is necessary to develop an in-depth understanding of how a latch cascade works, both under ideal and non-ideal operating conditions. However, there are several difficulties in analysing and simulating the behaviour of such a cascade. Firstly, each latch exhibits non-ideal behaviour characterised by minimum setup times and hold times. Secondly, each latch is subject to minimum and maximum "clock-to-Q" delays that may affect subsequent latches in the cascade. Thirdly, each latch exhibits a region of metastability that has to be carefully designed around. Fourthly, the impact of initial conditions, parameter variability and supply voltage variability can adversely affect the performance of both individual latches and the cascade as a whole. Therefore, for purposes of system-level analysis and symbol-level simulation, one needs to develop a discrete-time model that accurately captures this entire spectrum of complex, continuous-time dynamics exhibited by a cascade of latches.

Of course, an *ideal* latch can be described by a simple *functional abstraction*: it thresholds its input into a binary output when the clock is high, and retains its previous value when the clock is low. However, such an abstraction is too simplistic for today's nanoscale latches; it does not capture the effects of setup time, hold time, latch delays, metastability, sensitivity to parameter/supply voltage variability, etc. In actual design, these are the key considerations that can make or break an ADC.

On the other hand, one could use full-blown SPICE-level simulation with sophisticated transistor-level models (e.g., BSIM3, BSIM4) to capture the complete range of possible latch behaviours, including the above effects. However, for designs of moderate to large size, this can be so time-consuming and inefficient that it becomes completely impractical. Also, SPICE-level simulations cannot directly be used for higher level analysis (e.g., they do not provide intuition about system behaviour at the level of discrete communication symbols), and therefore cannot be integrated easily into a symbol-level simulation framework.

In this paper, we apply a recent technique, DAE2FSM [1], to produce an *intermediate level of abstraction* for latches that, (a) captures a richer set of dynamics than ideal latch models, and (b) simultaneously delivers better intuitive understanding and greater simulation efficiency compared to overly complex SPICE-level models. We use finite state machines (FSMs) for such an intermediate level of abstraction. In §II, we provide a brief description of DAE2FSM. In §III and §IV, we describe applications of DAE2FSM to latches and latch cascades within a communication link.

II. The Core Technique: DAE2FSM

Fig. 3 below illustrates the high-level architecture behind DAE2FSM [1], our automated technique to generate symbol-level FSM abstractions for AMS circuits (such as latches and cascades of latches). As the figure shows, DAE2FSM takes as input (a) an analog circuit (such as a current mode latch) described as a SPICE netlist, and (b) a SPICE simulator such as SPECTRE that has access to process parameters of the circuit.

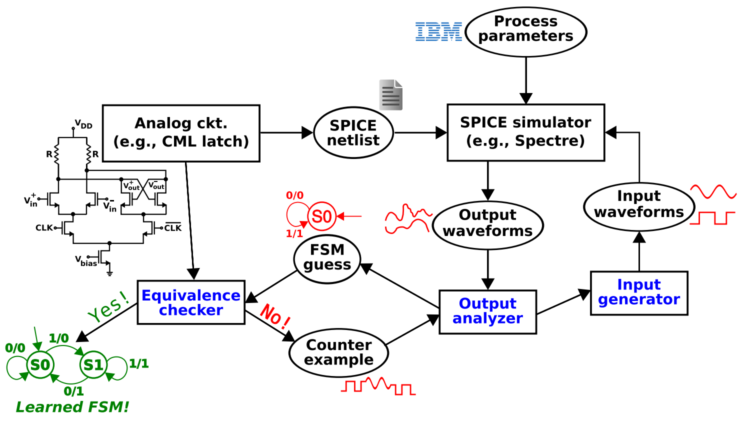

Fig. 3. The architecture behind DAE2FSM. Software modules/CAD techniques are shown in rectangular boxes, and data interchanges between modules are shown in circular boxes. The components making up the core DAE2FSM engine (i.e., the output analyzer, the equivalence checker, and the input generator) are indicated in blue font, while external dependencies (e.g., SPECTRE) are shown in black font.

DAE2FSM works by issuing multiple calls to the SPICE simulator, each time requesting the circuit's output waveform for a specific input waveform. These input waveforms are produced by a carefully crafted *input generator*, one of whose goals is to minimise the total number of SPICE runs required. Every output waveform returned by the simulator is processed by an *output analyzer*, which tries to learn the circuit's behaviour from the input/output traces collected thus far. The output analyzer can also issue specific queries to the input generator, which are taken into account while generating the next input waveform for the simulator. Eventually, the output analyzer learns enough about the circuit to propose a *candidate FSM* abstraction, which is passed to an *equivalence checker*. The equivalence checker then attempts to generate, either deterministically or probabilistically, a *counter example* to show that the candidate FSM does not accurately reflect the given circuit's behaviour. If such a counter example is found, the output analyzer uses it to refine its guess about the FSM. If no such counter example is found, the candidate FSM is taken to be a realistic abstraction for the given circuit, and the algorithm terminates.

The core technique behind DAE2FSM is a well-known algorithm from the field of computational learning theory, namely Angluin's algorithm [3]. In previous work [1], we have shown how Angluin's algorithm can be modified to produce binary FSM abstractions for AMS circuits, where the input and output alphabets are restricted to two level signals. For this work, we have re-architected the internal structure of DAE2FSM, and completely re-written the code from scratch. This resulted in several improvements over the original version: (a) much faster runtimes[1], (b) an improved output analyzer, and (c) applicability to real-world designs, including a current mode latch (CML) and a cascade of latches designed for IBM's 90nm LP process technology [2].

[1]All the FSMs in this paper have been generated by DAE2FSM within 5 minutes of execution time on a 2.4GHz laptop computer with 2GB of memory.

Due to space constraints, we do not describe these technical improvements in detail here, but move on to present the results obtained by running DAE2FSM on the latches mentioned above.

III. DAE2FSM Applied to a CML

Fig. 6 (a) shows the circuit diagram of a CML. As mentioned in the introduction, the CML constitutes the first stage in a cascade of latches that forms the basic structural and functional unit of a low-power ADC in a high-speed communication link.

The CML used in this paper has been designed in IBM's 90nm LP process technology, fabricated and tested in the context of a 1.2V, 6Gb/s communication link that employs a flash ADC [2]. We now present the results obtained by applying DAE2FSM to automatically generate FSM abstractions of this CML for a range of supply voltages $V_{DD} = \{0.8V, 1.0V, 1.2V\}$. (Although the CML has been designed only for a 1.2V supply voltage, we deliberately lowered the supply voltage to generate FSMs that capture the latch's behaviour under non-ideal operating conditions.)

Before applying DAE2FSM, we first need to map the FSM's symbolic 0/1 inputs and outputs to actual voltage levels exhibited by the circuit. For the FSM inputs, this choice is straightforward: the 0 and 1 inputs to the FSM correspond to circuit inputs of $-V_{DD}$ and $+V_{DD}$ volts respectively. The choice of output levels is a little more complicated: the 0 and 1 FSM output levels have to correspond to the minimum and maximum differential outputs produced by the circuit, sampled at some chosen instant of the clock cycle (in general, these are different from $-V_{DD}$ and $+V_{DD}$ volts). This leads to another choice; we need to determine a set of evenly spaced points at which to sample the input and output sequences (because the FSM corresponds to a discrete time representation). In this paper, we have selected two natural candidates for such sampling instances, namely the positive and negative edges of each clock cycle.

Having fixed the input/output levels and the sampling instance, the objective now is to generate an FSM that satisfies what we call a *fundamental test*: for any given input sequence (comprising a string of 0s and 1s), the output sequence predicted by the FSM should tally closely with the (sampled) output sequence obtained by SPICE simulation of the circuit on a similar input sequence. For example, suppose we have a negative-edge sampled FSM whose output levels 0 and 1 correspond to circuit (differential) outputs of -0.3V and +0.6V respectively. And suppose this FSM predicts that the input sequence $[0,\ 1,\ 1,\ 0]$ will result in the output sequence $[1,\ 0,\ 1,\ 1]$. Then, a SPICE simulation of the circuit on a corresponding input (i.e., an input that is -V_{DD} volts for the first clock cycle, V_{DD} volts for the second and third clock cycles, and $-V_{DD}$ volts again for the fourth clock cycle) should result in an output, whose samples (taken at the negative edge of 4 consecutive clock cycles) should be closely approximated by $0.6V,\ -0.3V,\ 0.6V,$ and $0.6V$. If this *fundamental test* holds for every input sequence, then the generated FSM is considered a sufficiently faithful abstraction of the original circuit.

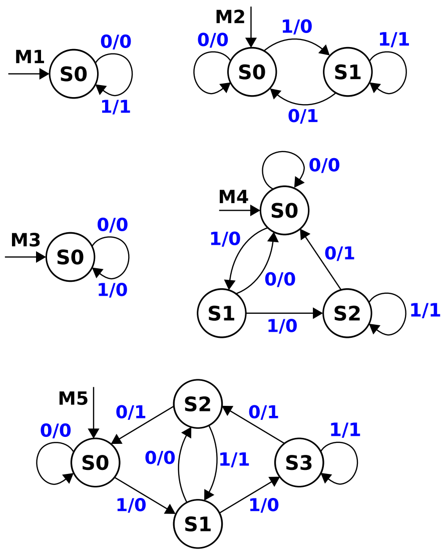

Fig. 4. FSM abstractions automatically generated by DAE2FSM for the CML, and for cascades of two and three latches, operating under V_{DD} = 0.8V, 1.0V, and 1.2V. For more details, please see Fig. 5.

Circuit	V_{DD} (volts)	Sampling instant	Learned FSM	FSM input levels	FSM output levels
Single CML	1.2	-ve clock edge	M1	$[0,1] \equiv [-1.2\text{V}, +1.2\text{V}]$	$[0,1] \equiv [-0.28\text{V}, +0.28\text{V}]$
		+ve clock edge	M2	$[0,1] \equiv [-1.2\text{V}, +1.2\text{V}]$	$[0,1] \equiv [-0.67\text{V}, +0.67\text{V}]$
	1.0	-ve clock edge	M1	$[0,1] \equiv [-1.0\text{V}, +1.0\text{V}]$	$[0,1] \equiv [-0.24\text{V}, +0.24\text{V}]$
		+ve clock edge	M2	$[0,1] \equiv [-1.0\text{V}, +1.0\text{V}]$	$[0,1] \equiv [-0.56\text{V}, +0.56\text{V}]$
	0.8	-ve clock edge	M1	$[0,1] \equiv [-0.8\text{V}, +0.8\text{V}]$	$[0,1] \equiv [-0.18\text{V}, +0.18\text{V}]$
		+ve clock edge	M2	$[0,1] \equiv [-0.8\text{V}, +0.8\text{V}]$	$[0,1] \equiv [-0.40\text{V}, +0.40\text{V}]$
Cascade of 2 latches	1.2	-ve clock edge	M2	$[0,1] \equiv [-1.2\text{V}, +1.2\text{V}]$	$[0,1] \equiv [-1.2\text{V}, +1.2\text{V}]$
		+ve clock edge	M2	$[0,1] \equiv [-1.2\text{V}, +1.2\text{V}]$	$[0,1] \equiv [-0.18\text{V}, +0.18\text{V}]$
	1.0	-ve clock edge	M2	$[0,1] \equiv [-1.0\text{V}, +1.0\text{V}]$	$[0,1] \equiv [-0.99\text{V}, +0.99\text{V}]$
		+ve clock edge	M3	$[0,1] \equiv [-1.0\text{V}, +1.0\text{V}]$	$[0,1] \equiv [-0.09\text{V}, +0.09\text{V}]$
	0.8	-ve clock edge	M4	$[0,1] \equiv [-0.8\text{V}, +0.8\text{V}]$	$[0,1] \equiv [-0.15\text{V}, +0.15\text{V}]$
		+ve clock edge	M3	$[0,1] \equiv [-0.8\text{V}, +0.8\text{V}]$	$[0,1] \equiv [-0.05\text{V}, +0.05\text{V}]$
Cascade of 3 latches	1.2	-ve clock edge	M2	$[0,1] \equiv [-1.2\text{V}, +1.2\text{V}]$	$[0,1] \equiv [-0.24\text{V}, +0.24\text{V}]$
		+ve clock edge	M5	$[0,1] \equiv [-1.2\text{V}, +1.2\text{V}]$	$[0,1] \equiv [-0.81\text{V}, +0.81\text{V}]$
	1.0	-ve clock edge	M2	$[0,1] \equiv [-1.0\text{V}, +1.0\text{V}]$	$[0,1] \equiv [-0.16\text{V}, +0.16\text{V}]$
		+ve clock edge	M5	$[0,1] \equiv [-1.0\text{V}, +1.0\text{V}]$	$[0,1] \equiv [-0.75\text{V}, +0.74\text{V}]$
	0.8	-ve clock edge	M3	$[0,1] \equiv [-0.8\text{V}, +0.8\text{V}]$	$[0,1] \equiv [-0.12\text{V}, -0.09\text{V}]$
		+ve clock edge	M3	$[0,1] \equiv [-0.8\text{V}, +0.8\text{V}]$	$[0,1] \equiv [-0.55\text{V}, -0.35\text{V}]$

Fig. 5. Details about the FSM abstractions generated by DAE2FSM for a single CML, and cascades of two and three latches, operating under 3 different supply voltages: 1.2V, 1.0V and 0.8V. This table refers to FSMs M1 to M5 shown in Fig. 4. The FSMs have been automatically generated for two different sampling instances (at the +ve and -ve clock edges). The last two columns of this table provide a mapping between the 0/1 FSM I/O symbols and actual voltages in the circuit.

Fig. 4 and Fig. 5 provide details about the FSM abstractions automatically generated by DAE2FSM for the output differential $(V_{out}^{+} - V_{out}^{-})$ of the CML, for three different supply voltages (0.8V, 1.0V and 1.2V). This CML is positive-edge triggered, i.e., it has been designed to sample the input when the clock is high, and amplify its previous value when the clock is low. Therefore, ideally, the output sequence sampled at the negative edge of the clock should be the same as the input sequence (corresponding to FSM M1), whereas the output sequence sampled at the positive edge of the clock should be the input sequence delayed by one clock cycle (corresponding to FSM M2). Indeed, from Fig. 5, we find that this is exactly the case for all values of V_{DD} that we tested. However, it is observed from Fig. 5 that as V_{DD} is lowered, the voltage difference between the FSM output levels 0 and 1 keeps shrinking, making the latch's state more difficult to resolve at lower supply voltages.

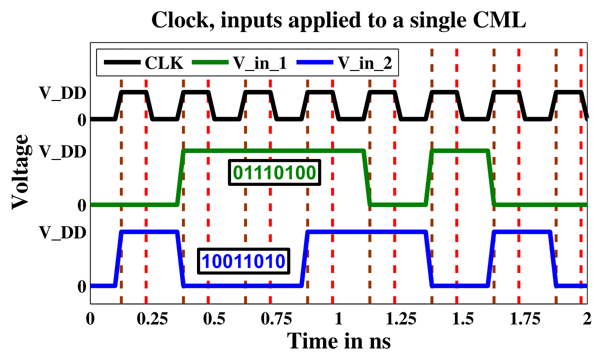

Fig. 7. Clock (black) and input (blue, green) waveforms applied to a CML, a 2-latch cascade and a 3-latch cascade for generating the output waveforms of Fig. 6. The vertical, dashed red (brown) lines represent the negative (positive) clock edges (where the FSM input/output sequences are sampled).

Finally, we would like to show that the learned FSMs are indeed a faithful abstraction of the original circuit. For this, we demonstrate that the learned FSMs all pass the *fundamental test* (described above) for two example input sequences, namely [0, 1, 1, 1, 0, 1, 0, 0] and [1, 0, 0, 1, 1, 0, 1, 0].[2]

[2]We have verified that the FSMs pass the fundamental test for not only these particular input sequences, but for *every* possible input sequence of length 8 and below.

Fig. 7 shows the (analog) clock and input waveforms that correspond to these particular input sequences. The responses produced by the circuit for these input sequences are shown in the first column of Fig. 6 (i.e., plots (b) to (d)). These plots include the response obtained by SPICE simulation (the blue and green waveforms), as well as the response predicted by the FSM simulation (the black waveforms with magenta markers). From the plots, it is clear that: regardless of supply voltage, the output sequence samples always lie close to the predictions made by the FSM simulation. This testifies to the accuracy of the FSM abstraction.

IV. DAE2FSM Applied to Cascades of Latches

We now apply DAE2FSM to cascades of two and three latches. Just as we treated the single latch case, we used DAE2FSM to generate FSM abstractions for two (and three) latch cascades operating under a range of supply voltages. The FSMs so generated are presented in Fig. 4 and Fig. 5 respectively.

Let us first consider the two latch cascade (please refer to Fig. 6 (e) for a circuit schematic). At $V_{DD} = 1.2$V, we find from Fig. 5 that the two-latch cascade behaves as an ideal delay unit (corresponding to FSM M2 for both negative and positive edge sampling). However, as V_{DD} gets lowered, the system's behaviour becomes non-ideal: for $V_{DD} = 1.0$V and below, the positive edge sampled FSM (M3) becomes severely limited in the its output swing, to the point that the only symbol it can output is 0, regardless of input. Even the negative edge sampled FSM (M4) becomes non-ideal at $V_{DD} = 0.8$V: this FSM indicates that, for the latch cascade to produce a 1 at the output, at least two consecutive 1s are needed at the input, whereas to produce a 0, it is enough if the input has a single 0. In other words, the transmission of a 1 from the cascade's input to its output has been slowed down significantly.

Similar to the single CML case, the second column in Fig. 6 (i.e., plots (f) to (h)) shows the results obtained by running both SPICE and FSM simulations of the two latch cascade on the inputs of Fig. 7. As before, we see that the generated FSMs, across supply voltages, all pass the *fundamental test* for these particular input sequences (additionally, we have verified this for *every* input sequence of length 8 and below).

We now turn to the results obtained for a three latch cascade (whose circuit schematic is shown in Fig. 6 (j)). Fig. 5 shows that even at $V_{DD} = 1.2$V,

Bottomline: ***The analog circuit output sequences (sampled at the positive and negative edges of each clock cycle) always lie close to the values predicted by the FSM abstractions! This holds true for a single latch, and for cascades of 2 and 3 latches, across a range of supply voltages.***

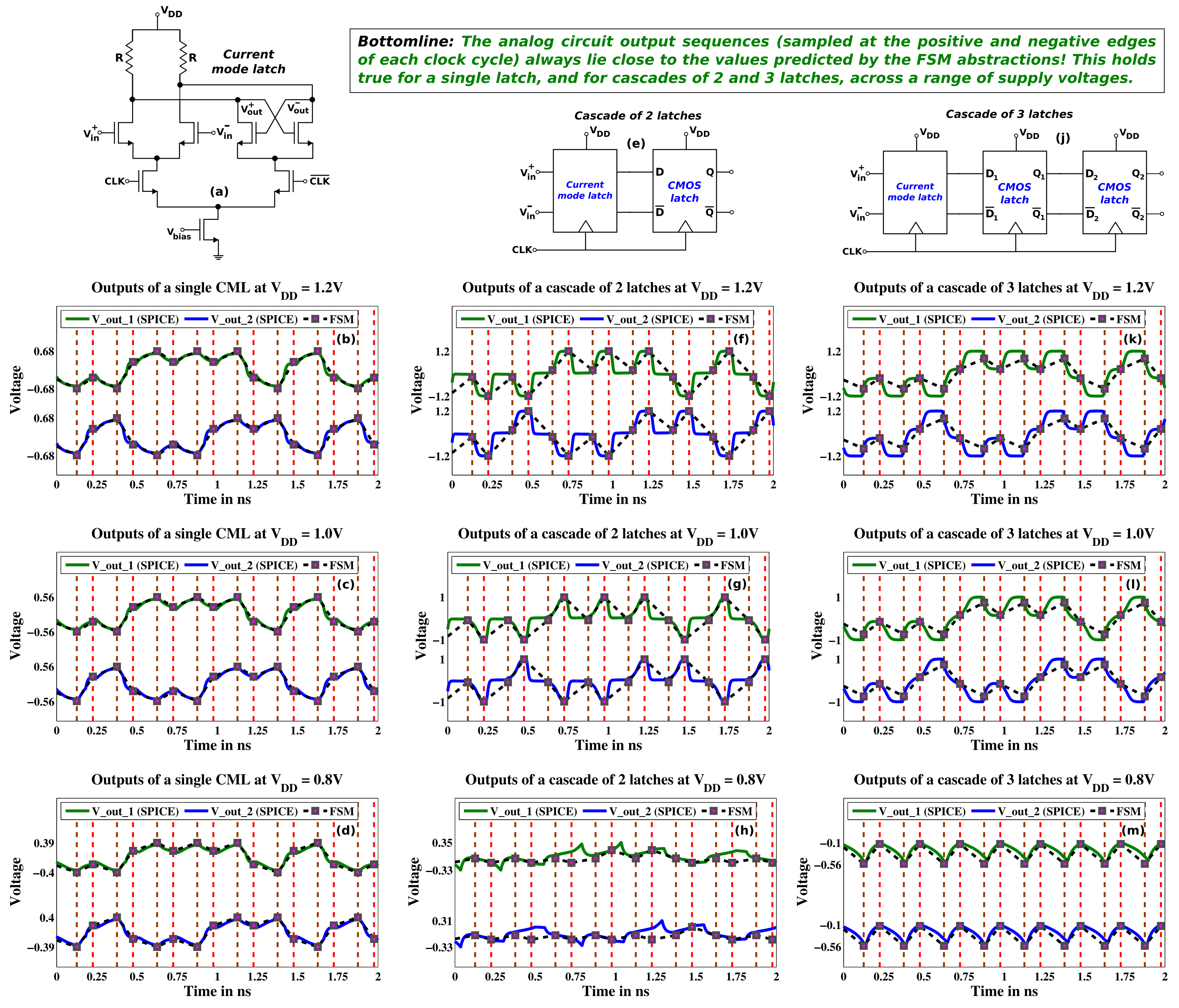

Fig. 6. Plots obtained by simulating a CML (first column), and cascades of 2 and 3 latches (second and third columns respectively), on two input sequences ([0 1 1 1 0 1 0 0] and [1 0 0 1 1 0 1 0]), for V_{DD} values 0.8V, 1.0V and 1.2V. Each plot shows the differential voltage at the circuit's output (for both these input sequences) obtained from SPICE simulations (the solid blue and green curves) as well as the output sequences predicted by FSM simulations (the dashed black curves with magenta markers). The vertical red and brown dashed lines show two different sets of FSM sampling instances, i.e., the negative and positive edges of each clock cycle respectively. It is seen that, for every simulation, in spite of the pronounced "analog-ness" of the SPICE waveforms, the analog output samples *always* lie close to the values predicted by the FSM abstraction at the sampling instances.

the positive edge sampled FSM (M5) is non-ideal; it represents a delay unit with a very slow response, i.e., to guarantee a 0 (1) at the output, the input must contain at least two consecutive 0s (1s). In other words, both 0 and 1 propagate through the cascade very slowly. And as V_{DD} gets further reduced to 0.8V, the reduction in output swing is so dramatic that the cascade becomes capable of producing only one output symbol (0), regardless of the input (corresponding to FSM M3).

As before, we can verify from the third column of Fig. 6 (i.e., plots (k) to (m)) that the generated FSMs all pass the *fundamental test* for the inputs in Fig. 7. Additionally, we have verified this for *every* input sequence of length 8 and below.

V. Summary and Conclusions

To summarise, we have demonstrated DAE2FSM, a fully automated technique to generate symbol-level FSM abstractions for latches (and cascades of latches) used in high-speed communication systems. Our technique is able to produce FSMs that offer (a) considerable intuition about the behaviour/dynamics of the latch/cascade in question, both under ideal and non-ideal operating conditions, and (b) dramatic improvements over SPICE in the context of simulation efficiency (while a SPICE simulation could take days, a comparable FSM simulation would take mere seconds). Therefore, we believe that our technique holds significant promise in the context of system-level analysis and symbol-level simulation of high-speed communication systems.

References

[1] C. Gu and J. Roychowdhury. FSM model abstraction for analog/mixed-signal circuits by learning from I/O trajectories. In *ASP-DAC '11: Proceedings of the 16th Asia and South Pacific Design Automation Conference*, pages 7–12, 2011.

[2] Y. Lin, A. Xu, N. Shanbhag, and A. Singer. Energy efficient high-speed links using BER-optimal ADCs. In *EDAPS '11: Proceedings of the IEEE Electrical Design of Advanced Packaging and Systems Symposium (to appear)*, 2011.

[3] D. Angluin. Learning regular sets from queries and counter examples. *Information and Computation*, 75:87–106, November 1987.