Functional Modeling of RSFQ Circuits Using Verilog HDL

Kris Gaj, Chin-Hong Cheah[†], Eby G. Friedman, and Marc J. Feldman
Department of Electrical Engineering, University of Rochester, Rochester, NY 14627

Abstract— **Circuit level simulation is too slow to be used for verification of function and timing of large RSFQ circuits. The alternative, known from semiconductor digital circuit design, is simulating at the logic (gate) instead of the circuit (transistor or junction) level. Using a hardware description language (HDL) such as Verilog, it is possible to write a functional model of each of the RSFQ basic gates. A large RSFQ circuit composed of hundreds gates and thousands Josephson junctions can then be simulated using standard semiconductor industry CAD tools. We have developed a library of Verilog models for over 15 basic RSFQ gates. We describe in detail our model for the DRO RSFQ cell. We show how this model can be generalized for other more complex cells. Our library has been verified by employing it in the design of timing for three large RSFQ circuits.**

I. INTRODUCTION

RSFQ (Rapid Single Flux Quantum) superconductive technology has become sufficiently matured to allow the development of medium to large scale integrated circuits with hundreds of RSFQ gates and thousands of Josephson junctions [1]–[3]. Circuit level simulation, although indispensable for accurate simulation and optimization of individual RSFQ gates, is too slow to verify the function and timing of large RSFQ circuits [4], [5]. Large scale circuits must therefore be simulated and optimized at the gate (logic) level rather then at the junction (circuit) level. Unfortunately, tools for logic level simulation of semiconductor circuits, which are based on a voltage state logic convention, cannot be directly applied to the pulse-based RSFQ logic.

The first attempt to calibrate standard CAD tools for logic level simulation of RSFQ circuits is reported in [6]. It is based on *structural* modeling of the RSFQ gates in which the logic and timing of the RSFQ cells is achieved using the appropriate combination of standard voltage state gates and flip-flops. This approach, while viable, has clear disadvantages: the development of the gate models is cumbersome, the models lack accuracy, and the models are not easily transferable to other CAD systems. These drawbacks can be overcome by using a hardware description language (HDL) for *functional* modeling of the RSFQ cells. In a functional model, the behavior of the gate is described using textual C-like notation, which is easy to edit, verify, and transfer between systems.

To our knowledge, standard HDLs have not been used in the design of RSFQ circuits to date. Proprietary notation called SFQHDL reported in [5] and [7] is not used for either logic simulation or automated logic synthesis. It describes the behavior of the circuit at the junction level by defining the correct order in which junctions switch, and is used primarily to describe the circuit pass/fail criteria for the purpose of optimization performed using a circuit level simulator [5].

Two standard HDLs coexist today: VHDL (Very high speed integrated circuit Hardware Description Language) and Verilog HDL [8]. VHDL became the IEEE standard in 1987; Verilog in 1995. Both HDLs have been in use for a long time; in fact, Verilog became the de-facto industrial standard years before official recognition by IEEE. The entire top-down design process of current large scale and very large scale *semiconductor* digital circuits is based on the use of HDLs. They are employed to describe the behavior and structure of the circuit at all design levels for the purpose of simulation and automated synthesis.

For logic level simulation of RSFQ circuits we chose to use Verilog HDL over VHDL. We believe that Verilog is more widely used today, and is easier to learn. Also, the efficient and user-friendly Verilog simulator, *Verilog-XL*, is available within our Cadence-based CAD environment [9].

We have developed a library of over 15 functional models of basic RSFQ gates in Verilog HDL. This library is continuously updated as new RSFQ gates are designed and implemented. It can be employed to simulate large RSFQ circuits using standard CAD simulators. Such simulations confirm the circuit function and timing, and may be used to explore various design concepts and trade-offs. Three large RSFQ circuits composed of several thousands Josephson junctions have been simulated, redesigned, and verified using our Verilog library and Cadence Verilog-XL HDL simulator. The simulation time for these circuits was found to be over two orders of magnitude shorter compared to circuit level simulation with JSPICE [4].

II. MODELING TIMING PARAMETERS OF RSFQ GATES

The design of a functional model of an RSFQ gate requires knowledge of the gate logic function, and the timing characteristics of the gate. The logic function of the cell is described using the cell state-transition diagram or the present-state/next-state table. Knowledge of an internal structure of the cell is not required. Models that represent only a logic function of the cells are extremely easy to write, and may be used to verify the function of a large circuit composed of basic RSFQ cells. An example of such a model for a DRO (Destructive Read-Out) cell is shown in Fig. 1.

Unfortunately, such models are not suitable for the verification of the timing in the circuit. Most RSFQ gates impose requirements on the minimum interval between the clock and the data pulses at the inputs of the gate. In addition, some cells impose requirements on the minimum interval between two data pulses at the same or related inputs of the cell. These requirements can be specified using timing parameters such as the hold time (*t_hold*), the setup time

Manuscript received August 25, 1996

[†] currently at Intel Corp.

Financial support for this research was provided in part by the University Research Initiative at the University of Rochester, sponsored by the Army Research Office under Grant No. DAAL03-92-G-0112.

1051-8223/97$10.00 © 1997 IEEE

```
// Verilog HDL for "gates.lib", "dro_cell" "_functional"
module dro_cell (d, clk, out);
input
    d, clk;
output
    out;
reg
    out;
parameter
    delay = 10,

reg
    d_state;          // internal state at the input d

always @(posedge d)   // execute at positive edge of d
      d_state <= d | d_state;

always @(posedge clk) // execute at positive edge of clk
    begin
       out <= #delay d_state;
       d_state <= 0;
    end
```

Fig. 1. Functional model of the DRO cell without full timing information.

(*t_setup*), and the minimum separation time (*t_separation*), as shown in Fig. 2. The *hold time* is the minimum interval between the clock pulse and the following data pulse; the *setup time* is the minimum interval between the data pulse and the following clock pulse [6]. The hold time and the setup time bound the range of the data pulse positions (marked as light dotted zones in Fig. 2) for which *either* the output is logically incorrect *or* the output timing is incorrect.

The necessary and sufficient condition for correct operation of most clocked cells is:

$$t\_hold \le data.position \le T_{CLK} - t\_setup, \qquad (1)$$

where *data.position* is the position of a data pulse within the clock cycle, and T_{CLK} is the length of the clock period. For certain cells, the minimum distance *t_separation* between two pulses at the same or related inputs must be preserved. The corresponding range of positions around the data pulse at the first input that is forbidden for a data pulse at the second input is marked in Fig. 2 as a dark dotted zone.

The functional description of each RSFQ cell must check for timing violations, and thus include values of timing parameters. The timing parameters are determined using JSPICE simulations. This process is automated using our custom designed software. Calculation of the hold and setup times for synchronous RSFQ cells requires a *sequence* of JSPICE simulations with the relative position of the data and clock pulses changed according to a binary search algorithm. The input test stimuli used in simulations must be chosen carefully to represent an exhaustive test sequence.

The clock-to-output delay in our models is assumed to be independent of the position of the data pulses within the clock cycle. Additionally, there is no clear distinction between violating the setup time in a given clock cycle and violating the hold time in the next clock cycle. Thus, the whole interval between the time determined by the setup time in a given clock cycle and the hold time in the next clock cycle is treated identically. The occurrence of a data pulse within this interval may lead to an undetermined output in a given clock cycle and an undetermined output in the next clock cycle. Timing diagrams obtained from the *Verilog-XL* simulator for an RSFQ half-adder illustrate a violation of the timing constraints in Fig. 3. Undetermined outputs are represented on the waveform display with *shaded pulses*.

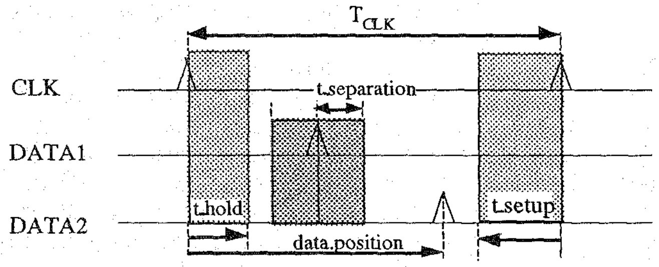


Fig. 2. Timing parameters describing requirements on the position of the data pulse within the clock cycle.

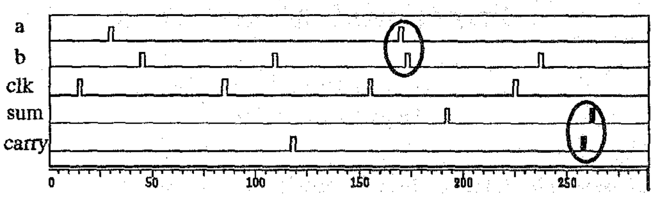


Fig. 3. Verilog-XL simulation of an RSFQ half-adder; *sum=a⊕b, carry=a·b.*

The functional view of each cell in Verilog HDL is verified using the Cadence Verilog-XL simulation environment for an exhaustive set of test sequences. These sequences include correct input stimuli and input stimuli with timing violations (*e.g.*, a violation of the hold time or setup time). For the purpose of testing, timing parameters are set to different arbitrary values, and the circuit function is verified for each set of timing parameters. The only constraints assumed in our models on the values of timing parameters are described by the following equations:

$$delay \ge 0 \qquad (2)$$

$$t\_hold + t\_setup \ge 0 \qquad (3)$$

$$t\_hold \le delay \qquad (4)$$

These constraints come directly from the definition of timing parameters in RSFQ cells [6].

As a result, the same model can be used to represent various specific implementations of a given RSFQ cell. When the circuit parameters of a gate are changed to improve its operation, or to adapt it for use as a component of some other circuit, the HDL model is updated simply by changing the values of the timing parameters.

III. Verilog Models for Clocked RSFQ Gates

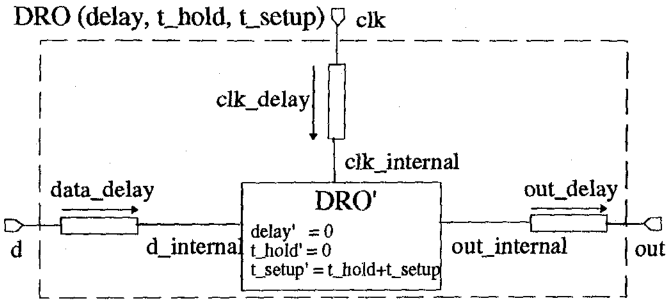
All functional models of RSFQ gates have been designed using the same structure. We discuss this structure by describing in detail the Verilog model of the simplest RSFQ clocked gate - DRO. A functional model of the DRO cell in Verilog HDL is shown in Fig. 4.

Lines (2)-(8) define the DRO interface; *d* and *clk* are declared as the inputs to the cell, and *out* is declared as the output from the cell. The timing parameters of the cell, the hold time (*t_hold*), the setup time (*t_setup*), and the delay (*delay*) are specified in lines (9)-(12). Their values may be overwritten in a higher-level module to make them specific for a given instance of the cell. A multichannel descriptor *warning_file* specifies where to direct the warning messages. The auxiliary signals in the circuit are defined as registers, and the auxiliary variables as integers in lines (15)-(25).

In our model, the DRO cell is treated as composed of a simpler dummy cell DRO' and three delay lines *data_delay,*

```
// Verilog HDL for "gates.lib", "dro_cell" "_functional"                    (1)
module dro_cell (d, clk, out);                                               (2)
input                                                                        (3)
    d, clk;                                                                  (4)
output                                                                       (5)
    out;                                                                     (6)
reg                                                                          (7)
    out;                                                                     (8)
parameter                                                                    (9)
    t_hold    = -3,                                                          (10)
    t_setup   = 8,                                                           (11)
    delay  = 9,                                                              (12)
    warning_file=3;  // multichannel description of a warning file          (13)
                                                                             (14)
reg                                                                          (15)
    d_internal, clk_internal,                                                (16)
    d_state,          // internal state at the input d                       (17)
    d_set;            // signal determining the moment when                  (18)
                      // the state of the d input changes to "1"             (19)
integer                                                                      (20)
    data_delay,  // delay between d and d_internal                          (21)
    clk_delay,   // delay between clk and clk_internal                      (22)
    out_delay,   // delay between clk_internal and out                      (23)
    out_value,   // output value in a given clock cycle                     (24)
    last_clk_time;   // time when the last clock pulse appeared             (25)
                                                                             (26)
initial                                                                      (27)
    begin                                                                    (28)
        if(t_hold<0)                                                         (29)
          begin                                                              (30)
            data_delay = -t_hold;                                            (31)
            clk_delay  = 0;                                                  (32)
            out_delay  = delay;                                              (33)
          end                                                                (34)
        else                                                                 (35)
          begin                                                              (36)
            data_delay = 0;                                                  (37)
            clk_delay  = t_hold;                                             (38)
            out_delay  = delay-t_hold;                                       (39)
          end                                                                (40)
        d_internal = 0;                                                      (41)
        clk_internal = 0;                                                    (42)
        last_clk_time = 0;                                                   (43)
        d_state = 0;                                                         (44)
        d_set = 0;                                                           (45)
        out = 0;                                                             (46)
    end                                                                      (47)
                                                                             (48)
always @(posedge d)   // execute at positive edge of d                       (49)
      d_internal <= #(data_delay) d;                                         (50)
                                                                             (51)
always @(posedge clk)  // execute at positive edge of clk                    (52)
      clk_internal <= #(clk_delay) clk;                                      (53)
                                                                             (54)
always @(posedge clk_internal)                                               (55)
    begin                                                                    (56)
        if (clk_internal === 1'bx)                                           (57)
          out_value = 1'bx;                                                  (58)
        else                                                                 (59)
          out_value = d_state;                                               (60)
        out <= #(out_delay) out_value;                                       (61)
        out <= #(out_delay+2) 0;                                             (62)
        d_state  <= 0;                                                       (63)
        clk_internal <= 0;                                                   (64)
        last_clk_time = $stime;                                              (65)
    end                                                                      (66)
                                                                             (67)
always @(posedge d_internal)                                                 (68)
    begin                                                                    (69)
        d_state <= d_state | 1'bx;                                           (70)
        if (d_internal === 1)                                                (71)
          d_set <= #(t_hold+t_setup) 1;                                      (72)
                                                                             (73)
        d_internal <= 0;                                                     (74)
    end                                                                      (75)
                                                                             (76)
always @(posedge d_set)                                                      (77)
    begin                                                                    (78)
        if ($stime - last_clk_time >= t_hold+t_setup)                        (79)
            d_state = 1;                                                     (80)
        else                                                                 (81)
          begin                                                              (82)
            d_state = 1'bx;                                                  (83)
            $fwrite(warning_file, "Violation of timing in module %m.\n");    (84)
          end                                                                (85)
                                                                             (86)
        d_set <= 0;                                                          (87)
    end                                                                      (88)
                                                                             (89)
endmodule                                                                    (90)
```

Fig 4. Functional model of the DRO cell in Verilog HDL.



if (t_hold ≥ 0) then
  data_delay = 0
  clk_delay = t_hold
  out_delay = delay - t_hold

if (t_hold < 0) then
  data_delay = - t_hold
  clk_delay = 0
  out_delay = delay

Fig. 5. The abstract structure of the DRO cell assumed in the Verilog HDL model.

clk_delay, and *out_delay* as shown in Fig. 5. The DRO has arbitrary values of timing parameters (*t_hold, t_setup,* and *delay*). The DRO' has zero delay (*delay'*) and zero hold time (*t_hold'*); its setup time (*t_setup'*) is equal to the sum of the hold time and setup time of the original gate. The input and output delays *data_delay, clk_delay,* and *output_delay* are chosen such that the timing parameters of both gates match. The formulas describing the input and output delays depend on the sign of the DRO hold time *t_hold,* as shown in Fig. 5. When *t_hold* is positive, the DRO is modeled using DRO' with the delay in the clock path equal to *t_hold,* and no delay in the data path. When *t_hold* is negative, the DRO is modeled using DRO' with the delay in the data path equal to *-t_hold,* and no delay in the clock path. In Fig. 6, we show the positions of the clock pulses and the timing violation intervals for the DRO (*clk* and *d* inputs) and the DRO' (*clk_internal* and *d_internal* inputs), for positive (Fig. 6a) and the negative (Fig. 6b) value of *t_hold.*

The variables describing the input and output delays are declared in lines (21)-(23), and initialized in lines (29)-(40). Lines (49)-(50) and (52)-(53) describe the propagation of the pulse through the data and the clock delay lines, respectively.

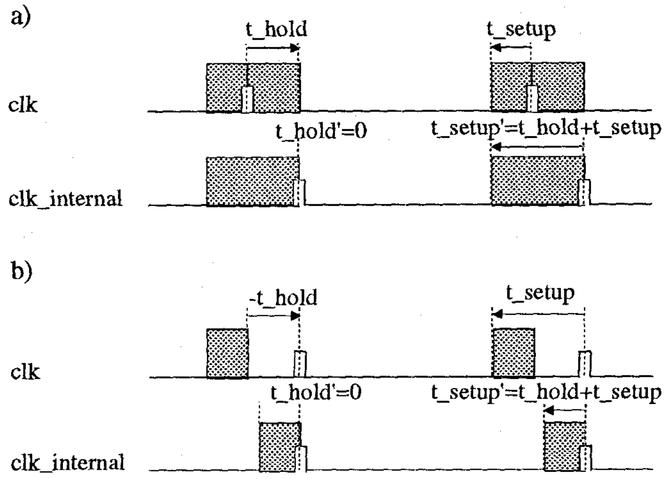


Fig. 6. The positions of the clock pulses and the timing violation intervals for the DRO and DRO' for two cases a) positive hold time, b) negative hold time.

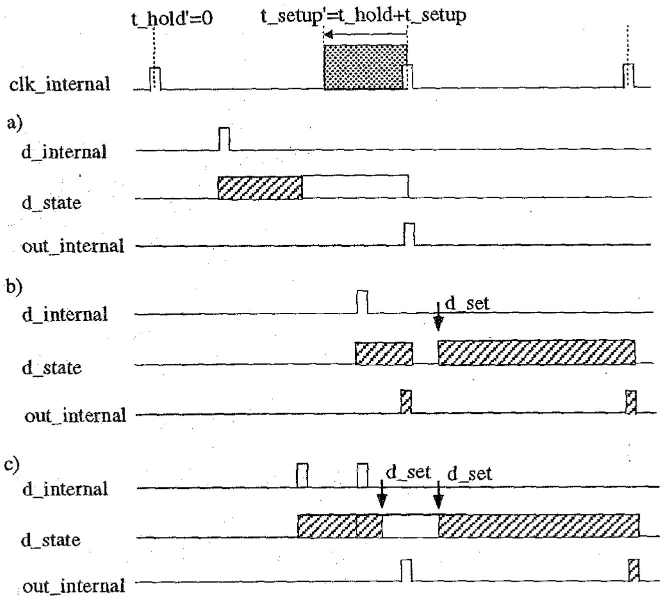


Fig. 7. Data input, output, and internal state of the DRO' for various positions of the data pulses within a clock period.

The rest of the code describes the operation of DRO'. When the clock pulse arrives at the *clk_internal* node, the code from lines (56)-(66) is executed. If the clock pulse is not an unknown-state (shaded) pulse, an internal state of the DRO' *d_state* is transferred to the output of the DRO after the *out_delay*, and the internal state of the DRO' is reset. Finally, the variable *last_clk_time* is set to the current simulation time. Notation 1'bx used in the Verilog model in Fig. 4 denotes a single-bit constant of the value *unknown*.

The remaining part of the code is used to set the internal state of the DRO' and to check for violations of the hold and setup times. The violation of the setup time of the DRO' appears when a clock pulse appears less than *t_setup'* after any data pulse (which is equivalent to a data pulse appearing less than *t_setup'* before a clock pulse). This condition can be detected by setting the internal state of the DRO' *d_state* to unknown for a period *t_setup'* (*t_hold+t_setup*) after a data pulse arrives at *d_internal*. When a clock pulse arrives within this period, the unknown state of DRO' will result in the unknown-state (shaded) pulse at the DRO output. If the clock pulse does not arrive within this interval, the internal state of the cell is set to one, and the next output of the DRO will be a correct pulse irrespective of further timing violations. All this is implemented in lines (68)-(88).

In Fig. 7, we show the data input, output, and internal state of DRO' for three varied positions of data pulses within a clock period. In case a), the data pulse arrives in the middle of the clock period and no timing violation appears. In case b) a single data pulse arrives close to the end of the clock period violating setup time in the given clock period and/or hold time in the next clock period. This results in the shaded pulses at the DRO output for two consecutive clock periods. In case c), one data pulse arrives close to the middle of the clock period, and the other violates setup and/or hold time. As a result, correct output appears at the end of the first clock period, and a shaded pulse at the end of the next clock period.

More complex logic gates can be modeled accordingly. For example, the only difference between the models of a DRO cell and an inverter cell is line (60). This line changes from

```
        out_value <= d_state
```

in a DRO model to

```
        out_value <= ~d_state.
```

in an inverter model.
For gates with two inputs such as AND gate, each data input is treated as was the data input *d* of the DRO. States at the inputs *a* and *b* are recorded using two distinct variables *a_state* and *b_state*, and the output is generated with the instruction:

```
        out_value <= a_state & b_state.
```

V. SUMMARY

We have developed functional models in Verilog HDL for most of the currently known RSFQ gates. Our models accurately describe the timing characteristics of basic RSFQ gates, and are used for simulation and timing analysis of large RSFQ circuits. The simulation can be performed using standard semiconductor-industry logic level simulators. Our models are not limited to any particular simulator and can be easily transferred between various CAD systems.

The Verilog models are easy to update, modify and extend. The models for more complex RSFQ gates can be easily created using models for simpler gates such as DRO.

All our models have been exhaustively verified using Cadence Verilog-XL simulation environment. The library of these models is currently being used for the design of three large scale digital and mixed-signal RSFQ circuits developed at the University of Rochester. The accuracy of modeling obtained with Verilog HDL appears to be satisfactory for the purpose of design of the clock distribution network in these circuits, whereas the simulation speed is over two orders of magnitude faster with Verilog-XL compared to JSPICE. For example, the simulation of a four-bit multiplier accumulator took 30 seconds in Verilog-XL compared to 4 hours in JSPICE. The logic simulation of large RSFQ circuits with Verilog HDL can therefore significantly increase the efficiency of the clocking design for medium to large RSFQ circuits.

REFERENCES

[1] O. A. Mukhanov, P. D. Bradley, S. B. Kaplan, S. V. Rylov, and A. F. Kirichenko "Design and operation of RSFQ circuits for digital signal processing," Proc. ISEC'95, pp. 27-30.

[2] V. K. Semenov, Yu. Polyakov, and D. Schneider, "Preliminary results on the analog-to-digital converter based on RSFQ logic," CPEM'96 Conf. Digest, Braunschweig, Germany, June 1996.

[3] Q. P. Herr *et al.*, "Low-speed operation of a four-bit RSFQ multiplier-accumulator," this conference.

[4] S. R. Whiteley, "Josephson junctions in SPICE3," *IEEE Trans. on Mag.*, vol. 27, pp. 2902-2905, March 1991.

[5] S. V. Polonsky, *et al.*, "PSCAN 96: New software for simulation and opimization of complex RSFQ circuits," this conference.

[6] A. Krasniewski, "Logic simulation of RSFQ circuits," *IEEE Trans. Appl. Supercond.*, vol. 3, pp. 33-38, March 1993.

[7] A. V. Rylyakov and S. V. Polonsky, "All digital 1-bit RSFQ autocorrelator for radioastronomy applications: design and experimental results," *IEEE Trans. Appl. Supercond.*, vol. 7, 1997.

[8] D. J. Smith, "VHDL & Verilog compared & contrasted - plus modeled example written in VHDL, Verilog and C," Proc. 33rd Design Automation Conf., 1996.

[9] V. Adler, C.-H. Cheah, K. Gaj, D. K. Brock, and E. G. Friedman, "A Cadence-based design environment for single flux quantum circuits," this conference.