Review and comparison of RSFQ asynchronous methodologies

L C Müller, H R Gerber and C J Fourie
Department of Electric and Electronic Engineering, University of Stellenbosch, South Africa
E-mail: lcmuller@sun.ac.za

Abstract. The various design limitations imposed on RSFQ circuits through synchronous clocking schemes, make the use of asynchronous clocking attractive. This paper critically reviews and compares a wide range of asynchronous methodologies that have thus far been proposed in the published literature. Circuits are optimized using a Genetic Algorithm optimizer and are then compared in terms of yield, critical margins and latency. A full-adder is also designed and simulated for each methodology to obtain information regarding implementation above the primitive cell level.

1. Introduction
Synchronous clocking and transistor technology would seem to complement each other naturally, whilst extra care and time has to be taken to create their asynchronous counterparts. This is not the case for RSFQ technology, a novel implementation for digital superconducting circuits proposed by [1]. Clocking for RSFQ circuits remains challenging due to extremely fast propagation delays, causing chip wide problems such as clock skew and timing violations. However, RSFQ can quite naturally be used with asynchronous methodologies to create a far more robust system, and possibly even a friendlier design environment.

This paper will review and compare a wide range of asynchronous design methodologies proposed in the published literature in the 15 odd years since RSFQ has been published widely. They are, in no particular order, Data-Driven Self Timed (DDST), Dual-Rail RSFQ (DR-RSFQ), RSFQ Asynchronous Timing (RSFQ-AT) and Delay Insensitive Dual-Rail (DI). All the methodologies are discussed in Section 3. With the discussions, a table containing the simulated parameters for the cell primitives are shown.

In Section 4 of the paper Full Adders of all the methodologies are compared.

Due to the ambiguity of measuring stability and robustness in RSFQ circuits (yield, critical margin etc.), it was decided to optimize all cell primitives with the same Genetic Optimizer using the same optimization parameters. The Genetic Optimizer and test setups are discussed in section 2.

In the final section conclusions are drawn as to which methodology should be used in different situations.

Take note that the $\beta_c$ value for all circuits was chosen as 1.

2. Genetic Algorithm, Optimization and Test Setup
For optimization a Genetic Algorithm meta-heuristic technique was used [2] [3].

© 2008 IOP Publishing Ltd
A multi-objective genetic algorithm was written for optimization, using an aggregate method to create single fitness values. Circuits were optimized with regards to yield, critical margin, leakage current and latency. A variable mutation rate was implemented to facilitate a high mutation rate at the start of the algorithm and ever decreasing mutations as the algorithm progresses. The parameters governing the algorithms operations are displayed in Table 1. The weight assignment for the aggregate calculations are 0.45, 0.15, 0.30 and 0.10 for yield, critical margin, leakage current and latency respectively. The values for crossover and reinsertion are the fractions of a new population created using the respective operators.

| Crossover | Reinsertion | Population Size | Generations | Monte-Carlo runs |
|-----------|-------------|-----------------|-------------|------------------|
| 0.8       | 0.2         | 15              | 35          | 100              |

All circuits were analysed and optimized using WRSpie or JSpice3. Each of the circuit’s external connections were connected to standard 250\(\mu\)A JTLs to facilitate circuit interconnectivity. To save computational time a DC/SFQ circuit was not used for the optimization algorithm but instead a voltage pulse was generated using a piece-wise linear model. Process variations for the use of Monte Carlo simulations were based on the 1\(kA/cm^2\) Hypress process. The tolerance values were obtained from [4] and are shown in Table 2. These values represent the Gaussian 3\(\sigma\) tolerances of the process.

| Parameter                  | Global Tolerance | Local Tolerance |
|----------------------------|------------------|-----------------|
| Critical Current Density   | 10%              | 5%              |
| Junction Area              | 40%              | 5%              |
| Resistance                 | 20%              | 5%              |
| Inductance                 | 10%              | 15%             |
| Junction Capacitance       | 5%               | 0%              |

3. The Methodologies
A thorough overview on general asynchronous methodologies can be found in [5]. One can roughly divide the asynchronous methodologies that could possibly be used for RSFQ circuits into three main groups: bounded delay, delay insensitive and micro-pipelines.

Bounded delay methodologies assume that gate and propagation delays are known throughout the circuit. Thus by using delay elements where necessary, the designer is able to avoid logical hazards.

The delay insensitive methodology assumes that no information is available concerning propagation delay and gate latencies. A circuit is defined as being delay insensitive if proper functionality holds for any order of input pulse arrival. Dual-Rail techniques fall naturally under this methodology. Hand-shaking can also be described as being delay insensitive. Though some Hand-Shaking methodologies have been proposed for use in RSFQ, for example [6], they will not be discussed in this paper.
Micro-pipelines can be loosely defined as a mixture of bounded delay data paths and delay insensitive control circuitry. An example of micro-pipelines in RSFQ is [7].

RSFQ-AT and DDST both use standard RSFQ primitives to construct their respective circuits modularly. Because of this the standard RSFQ library needed to be optimized. Almost all the circuits were obtained from the Stony Brook University Cell Library except for the RSFQ Muller-C element (Coincidence Buffer) which was taken from the Stellenbosch University cell library and the Complementary D-Flip Flop which was obtained from [8].

Simulated parameter values for the RSFQ primitives are shown in Table 3. Yield was calculated using 100 Monte Carlo simulations and the process variations given in Table 2. It should be noted that a very small emphasis was placed on latency, explaining the slow gate propagation times throughout this paper. Latencies are calculated as the relative time between the arrival of the clock pulse and the output pulse. For gates with no clock input the last input received serves as reference to the output. This convention explains why gates like RSFQ-OR would seem to have a smaller relative latency than non-clocked gates.

| Parameter       | AND | OR  | NOT | XOR  | Merge | Split | Muller-C | C-DFF |
|-----------------|-----|-----|-----|------|-------|-------|----------|-------|
| Yield           | 100%| 100%| 88% | 100% | 100%  | 100%  | 100%     | 100%  |
| Critical Margin | ≥ 30%| ≥ 30%| ≥ 8%| ≥ 30%| ≥ 30%| ≥ 30%| ≥ 30%     | 19%   |
| Latency         | 25ps| 16ps| 17ps| 10ps | 16ps  | 14ps  | 15ps     | 27ps  |
| Junction Count  | 13  | 9   | 8   | 9    | 5     | 3     | 5        | 8     |

3.1. Rapid Single Flux Quantum Asynchronous Timing (RSFQ-AT)

RSFQ-AT was introduced by [9] as a simple, yet functional, way to implement asynchronous circuits.

![Figure 1. RSFQ-AT Functional Diagram](image)

The same conventions are used in RSFQ-AT as was proposed in [1] where the presence of a fluxon or SFQ pulse has a logical value of ‘1’ and the absence of the pulse, the logical value ‘0’. For the circuit to function asynchronously a timing pulse is sent in parallel with each data pulse. This takes the physical form of a clock input for every data input in the logic gate as shown in Figure 1. The clock signals are fed into RSFQ Muller-C elements that release output pulses only when all input pulses have arrived. The output of the Muller-C element is split to the clock input of the RSFQ logic gate, and to serve as clock signal for the next RSFQ-AT gate. A delay element, denoted by a buffer in Figure 1, is used to synchronize the timing of the output data and clock signals. It is clear that for safe operation of the circuit that all the timing pulses
must arrive with, or after, the data pulses. It is because of this limitation that RSFQ-AT can not be classified as being delay insensitive.

This methodology can be scaled to larger bundled systems where a whole data bus can be accompanied by only one timing pulse as long as this pulse arrives after all the data pulses. An asynchronous CPU was also designed to demonstrate the implementation of the methodology in larger systems [10].

The simulated parameters of the optimized RSFQ-AT cells are shown in Table 4.

| Parameter      | AND | OR | NOT | XOR |
|----------------|-----|----|-----|-----|
| Yield          | 69% | 100% | 36% | 100% |
| Critical Margin| 6%  | ≥ 30% | 6%  | ≥ 30% |
| Latency        | 62ps | 44ps | 38ps | 43ps |
| Junction Count | 29  | 21  | 17  | 21  |

To construct the RSFQ-AT gates, optimized RSFQ primitives were used and thus these gates were not optimized as a unit. It is conjectured that this is the reason why the RSFQ-AT AND and NOT gates suffer from poor stability. Whole gate optimization would minimize leakage currents between elements and have an overall positive effect on yield and critical margin.

3.2. Dual Rail

Dual-Rail circuits can be constructed by using two input connections for every logical input to a gate. One connection represents a ‘True’ and the other a ‘False’.

A methodology using Dual-Rail circuits were proposed by [11]. AND and OR logical gates were constructed using complimentary D Flip-Flops whilst the standard RSFQ XOR cell was used for the Dual-Rail XOR function. Implementing the NOT operator in Dual-Rail methodologies is trivial in the sense that the True and False connections need only be swapped.

The simulated parameters of this methodology are presented in Table 5.

| Parameter      | AND | OR | XOR |
|----------------|-----|----|-----|
| Yield          | 97% | 95% | 100% |
| Critical Margin| ≥ 30% | ≥ 30% | ≥ 30% |
| Latency        | 67ps | 79ps | 62ps |
| Junction Count | 26  | 26  | 30  |

The difference obtained by optimizing whole gates can clearly be seen by comparing this methodology parameters to those of RSFQ-AT.

3.3. Data-Driven Self-Timed

Data-Driven Self-Timed is a Dual-Rail methodology proposed by [8] [12]. Like RSFQ-AT, this methodology uses mostly existing RSFQ primitives to implement their asynchronous circuits. The workings of DDST will be explained with reference to Figure 2.
DDST gates are constructed modularly. The arrival of data pulses on both Dual-Rail inputs of a logical circuit is detected using a Muller-C element. This output is then split for use as a clock signal for a logic gate, as well as the complementary D-type Flip-Flop that stores the logic gate’s result. It is necessary to slow down the propagation of the clock signal to the D Flip-Flop so that the output is not clocked before the result enters the Flip-Flop.

The simulated parameters of the DDST logical cells are shown in Table 6.

Once again the penalty of not optimizing whole gates is apparent. DDST is also the largest methodology tested and thus also most prone to leakage current influence. These gates can also be optimized further for junction count as many RSFQ primitives contain buffer junctions not needed by this implementation. This would improve all gate parameters.

3.4. Delay Insensitive-RSFQ

The delay insensitive methodology proposed by [13] does not make use of normal RSFQ logical circuits. A set of universal primitives were introduced based on the work by [14] and [15] and can be thought of as finite state machine implementations using synchronous circuits. DI-RSFQ makes use of Dual-Rails for its timing requirements.

Two of the universal primitives are shown in Figure 3. A \( m \times n \)-join can be described as having \( m \) row inputs, \( n \) column inputs and a \( m \times n \) matrix that contains a corresponding output for each input pair. As can be seen in Figure 3, the \( 1 \times 2 \)-join primitive has one row input, designated as \( a^? \), and two column inputs, \( b_0^? \) and \( b_1^? \). For the \( a^? - b_0^? \) pair a pulse is released at \( c_0! \) whilst for the pair \( a^? - b_1^? \) a pulse is released on output \( c_1! \). Note that these inputs can arrive in any sequence making the design delay insensitive. The \( 2 \times 2 \)-join works on the same principle but with two row inputs and two column inputs. Table 7 shows the functionality of this circuit.

Only the \( 1 \times 2 \)-join and \( 2 \times 2 \)-join primitives were optimized. The Fork and Merge primitives correspond to the RSFQ-Split and RSFQ-Merge elements respectively. Values for the optimized primitives are shown in Table 8.
Figure 3. Delay Insensitive RSFQ Primitives

Table 7. Functionality of 2x2-Join primitive

| Row | Column | Output |
|-----|--------|--------|
| a?  | c?     | p!     |
| a?  | d?     | r!     |
| b?  | c?     | q!     |
| b?  | d?     | s!     |

Table 8. Parameters for selected DI primitives

| Parameter       | DI-1 × 2-Join | DI-2 × 2-Join |
|-----------------|---------------|---------------|
| Yield           | 100%          | 100%          |
| Critical Margin | ≥ 30%         | ≥ 30%         |
| Latency         | 15ps          | 32ps          |
| Junction Count  | 7             | 28            |

4. Full-Adders

Full-Adders for RSFQ-AT, Dual-Rail and DDST were all implemented using a general design as shown in Figure 4. DI-RSFQ, being a wholly different methodology, had another implementation as presented in [13]. This implementation is shown in Figure 5, where the thick arrows denote Dual-Rail input lines.

Table 9. Parameters for Full-Adders

| Methodology | RSFQ-AT | DR-RSFQ | DDST | DI |
|-------------|---------|---------|------|----|
| Latency     | 147ps   | 252ps   | 297ps| 71ps |
| Junction Count | 145  | 174     | 287  | 98  |

Clearly DI has a simple natural implementation of the Full-Adder circuit. However, this
does not mean that all of the DI circuits will be smaller and faster than their counterparts. It remains to be seen if other, possibly more complex, circuits can be constructed efficiently using DI primitives.

Latency for all circuits were calculated from last input to last output.

5. Conclusion
The Delay-Insensitive methodology appears to be more robust, faster and to consume less space than its counterparts. More complex circuits will have to be constructed to see if this trend continues.

For small purpose applications, RSFQ-AT delivers reasonable speed with relatively small layout. The logic gates will have to be optimized as a whole to raise circuit yield and critical margin. For large systems the process variations might prove to be a problem for the clock-follow-data constraint.

Dual-Rail methodologies appear to be the most beneficial for large scale integration. The logic gates, if designed and optimized correctly, are intrinsically stable in regards to process variations. Although DR-RSFQ has a stable design, it lacks cell variety. DDST can thus be very useful for use with arbitrary RSFQ circuits, though as has been stated before, optimization will have to be performed on complete circuits.

The effect of leakage current on larger systems appear to be substantial, as was shown with the RSFQ-AT and DDST methodologies. More emphasis will have to be put on leakage current as RSFQ enters the LSI arena.

**Figure 4.** General Full-Adder Design

**Figure 5.** Delay-Insensitive Full-Adder
References

[1] Likharev K K and Semenov V K 1991 IEEE Transactions on Applied Superconductivity 1 3–28
[2] Mühlenbein H 1998 Local Search in Combinatorial Optimization ed Aarts E and Lenstra J K (Wiley-Interscience) chap 6, pp 137–172
[3] Reeves C 2003 Handbook of Metaheuristics ed Glover F and Kochenberger G A (Kluwer Academic) chap 3, pp 55–82
[4] Fourie C J, Perold W J and Gerber H R 2005 IEEE Transactions on Applied Superconductivity 15 384–387
[5] Hauck S 1995 Proceedings of the IEEE 83 69–93
[6] Ito M, Kawasaki K, Yoshikawa N, Fujimaki A, Terai H and Yorozu S 2005 IEEE Transactions on Applied Superconductivity 15 255–258
[7] Teh C K and Okabe Y 2003 IEEE Transactions on Applied Superconductivity 13 543–546
[8] Deng Z J, Yoshikawa N, Whiteley S R and van Duzer T 1997 IEEE Transactions on Applied Superconductivity 7 3634–3637
[9] Gerber H R, Fourie C J and Perold W J 2005 IEEE Transactions on Applied Superconductivity 15 272–275
[10] Gerber H R, Fourie C J, Perold W J and Müller L C 2007 IEEE Transactions on Applied Superconductivity 17 490–493
[11] Maezawa M, Kurosawa I, Aoyagi M and Nakagawa H 1997 IEEE Transactions on Applied Superconductivity 7 2705–2708
[12] Deng Z J, Yoshikawa N, Whiteley S R and van Duzer T 1999 IEEE Transactions on Applied Superconductivity 9 7–17
[13] Patra P, Fussell D S and Polonsky S 1997 Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems (IEEE Computer Society Press) pp 42–53 URL citeseer.ist.psu.edu/patra96delay.html
[14] Keller R M 1974 IEEE Transactions on Computers 23 21–33
[15] Patra P 1995 Approaches to Design of Circuits for Low-Power Computation Ph.D. thesis University of Texas at Austin