Available online at www.sciencedirect.com

SciVerse ScienceDirect

Physics Procedia 36 (2012) 59 – 65

Physics Procedia

Superconductivity Centennial Conference

20 GHz operation of an asynchronous wave-pipelined RSFQ arithmetic-logic unit

Timur V. Filippov[a]*, Anubhav Sahu[a], Alex F. Kirichenko[a], Igor V. Vernik[a], Mikhail Dorojevets[b], Christopher L. Ayala[b], and Oleg A. Mukhanov[a]

[a]*HYPRES, Inc, 175 Clearbrook Road, Elmsford, NY 10523, USA*
[b]*Department of Electrical and Computer Engineering, Stony Brook University, Stony Brook, NY 11794, USA*

Abstract

We have designed and tested at high frequency an RSFQ-based Arithmetic-Logic Unit (ALU), the critical component of an 8-bit RSFQ processor datapath. The ALU design is based on a Kogge-Stone adder and employs an asynchronous wave-pipelined approach scalable for wide datapath processors. The 8-bit ALU circuit was fabricated with HYPRES' standard 4.5 kA/cm^2 process and consists of 7,950 Josephson junctions, including input and output interfaces. In this paper, we present chip design and high-speed test results for the 8-bit ALU circuit.

Keywords: RSFQ; ALU; microprocessor, datapath.

1. Introduction

A high-performance arithmetic-logic unit (ALU) is a fundamental building block for any special- or general-purpose processor. The reported ALU is a key processing component for the RSFQ-based [1] 8-bit processor datapath [2]. This is the first attempt to build a superconductor parallel processor in contrast to the bit-serial approaches [3, 4].

The ALU design is based on Kogge-Stone adder (KSA) [5]. A set of logic operations is integrated into the adder structure. The ALU is switched between arithmetic and logic operations by control signals. A similar approach to build an adder-based ALU was reported in [6]. However, that ALU was based on a simple ripple-carry adder and, therefore, was hardly scalable to a large number of bits.

The current ALU employs a wave-pipeline synchronization approach [7]. According to this approach, a pipeline stage is allowed to start its operation on two independent data operands as soon as both operands arrive. There is no clock pulse used to advance the computation from one stage to another. Instead, a clock pulse that follows data is used to reset cells in the stage to make it ready to process the next data wave. This type of synchronization makes it different from the previous RSFQ-based pipeline ripple-carry adder [6] and KSA [8], where a co-flow timing technique was used to clock data throughout the entire adder requiring a clock distribution tree for every stage.

We have already reported low-frequency functionality test results of the 8-bit ALU in [9]. This paper focuses on high-speed test results.

* Timur V. Filippov, *E-mail address*: tfil@hypres.com

1875-3892 © 2012 Published by Elsevier B.V. Selection and/or peer-review under responsibility of the Guest Editors.
doi:10.1016/j.phpro.2012.06.130

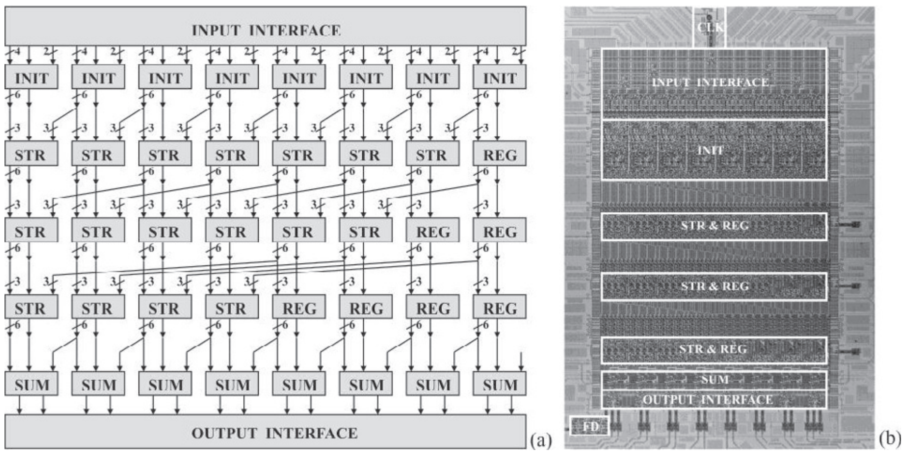

Fig. 1. An 8-bit ALU: (a) block-diagram; (b) microphotograph of the chip

2. Design of the 8-bit ALU

The block-diagram of the 8-bit ALU is shown in Fig. 1a and consists of 5 stages formed by four building blocks in accordance with the KSA algorithm [5]. All four blocks and their functions are listed in Fig. 2. The described ALU can be switched between arithmetic and logic operations by applying control signals to the INIT blocks. All control signals and corresponding instructions are listed in Table 1.

While performing arithmetic operations, the INIT blocks produce bit-wise *generate* (*G*), *propagate* (*P*), and *partial sum* (*pi*) signals under control of *Ready* (*R*) pulses. The INIT blocks also route these signals to group prefix stages (three stages in the case of the 8-bit adder) formed by STR and REG blocks. The *pi* signals propagate only inside the bit-slice while *G*, *P*, and *R* pulses are also copied into other bit-slices in accordance with the KSA algorithm. The last stage of SUM blocks completes the summation. While executing logic operations, the INIT blocks perform bit-wise logic operations and route results to outputs inside the bit-slice through STR, REG and SUM blocks.

Fig. 3 shows schematics of the blocks. The most complex block (INIT) is shown in Fig. 3a. It consists of the following logic cells: a D flip-flop (D) [1], a dual-port D flip-flop (D2) [10], a D flip-flop with complementary outputs (DC) [1], an XOR cell [1], and a dynamic AND cell [11]. The top layer of the INIT block is formed by D flip-flops to store control signals *ctrl_xor* and *ctrl_add,* and XOR cells to store direct or inverted (depending on *inv_a/inv_b* signals) input data. All cells of the top layer are clocked by a *Ready* pulse. Then, *P* and *G* signals are calculated by XOR and AND cells of the second layer and buffered by D2 cells of the third layer. In the presence of *ctrl_add* signal, the results are sent by the *R* signal to the group prefix stages. Note that *G*, *P*, and *R* signals are split into two copies to propagate inside the bit-slice (*v*) and be routed to the other bit-slice (*h*). The *P* signal is replicated as the *pi* signal for propagation inside the bit-slice.

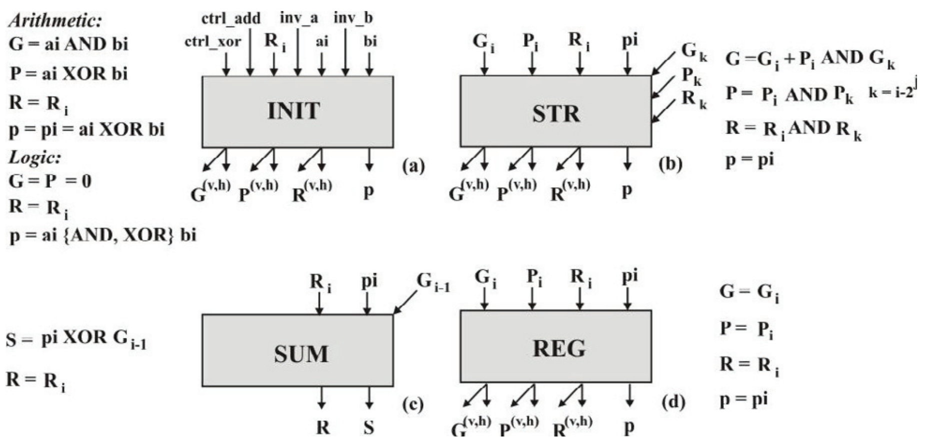

Fig. 2. ALU building block symbols and their functions