

## WP 4.7: A Reconfigurable Multiprocessor IC for Rapid Prototyping of Real-Time Data Paths

Dev C. Chen and Jan M. Rabaey

Electrical Engineering and Computer Science, University of California, Berkeley, CA

In real-time digital-signal-processing systems, data often enters or leaves the computationally intensive parts at small integer multiples of the clocking interval. In these cases, traditional microprocessor-based architectures are inadequate to meet the throughput requirements, and so clusters of dedicated data paths, hard-wired to closely match the algorithmic data flow, are often used. These architectures typically contain multiple concurrently-operating data-path pipelines. The circuit reported here targets the rapid implementation and prototyping of such architectures using reconfigurable multiprocessors. The basic concept of this work is presented in Reference 1. Other programmable approaches include FPGAs, VSPs and multi-processor configurations based on commercial DSPs [2]. The circuit presented here contains 8 execution units (EXUs) connected via a dynamically controlled crossbar switch. It can operate at 25MHz (200MIPs) with a data I/O bandwidth of 400MB/s and a typical power consumption of 0.45W. It contains 140,106 transistors on a 8.9x9.5mm<sup>2</sup> die, in 1.2µm CMOS technology.

In the algorithms examined, the signal flow graphs tend to exhibit a spatial and temporal locality in the usage of variables. The hardware mapping of these types of algorithms is supported extremely well by clusters of EXUs, each containing local register files and connected by a flexible high-bandwidth network. The combination of flexible local interconnect and distributed memory supports direct mapping of flowgraphs to EXUs, as well as the multiplexing of more than one operation onto a given EXU. The architecture of this chip is outlined in Figure 1. It contains a cluster of 8 EXUs connected by a dynamically-configurable crossbar. Clusters can communicate with other clusters with high bandwidth via 128 dedicated I/O pins.

Figure 2 shows the internal architecture of an EXU. It supports unconditional pass, addition, subtraction, comparison, maximum-minimum, arithmetic right shift, and accumulation. Pairs of EXUs can be configured to be 32b wide. Arithmetic automatically saturates and can be in two complement or unsigned format. Two dual-ported register files, each with six registers, are used for temporary data buffering. They can also be configured as delay lines to support operation pipelining and retiming. In each file, one of the six registers is implemented as a scan-register. It can be initialized as a constant variable, serve as a regular register, or be used for scan-testing. Arithmetic functions are implemented using a fast carry-select adder and logarithmic shifter. A pipeline register is available at the output of each EXU for optional use. A status flag ( $a > b$ ) is provided for other EXUs and the external world.

To ensure flexible and high-bandwidth data routing, a crossbar network interconnects the processors. This network routes both data and status flags. Data routing is under program control and can be changed in each program cycle. The routing of the status flags is static and set at compile time, reducing the area required for control store (nanostore). The main challenge in the design of the crossbar network is to ensure pitch-matching between the crossbar switches and the EXUs. Therefore, a layered crossbar structure is used, as shown in Figure 3. A detail of the data routing bit-slice connecting

EXUs A and E to each other, to other EXUs in the cluster, and the I/O buses, is pictured. The layered switch implementation is organized as follows: The Type I switch connects the input of an EXU to either one of its neighbors (B,C,D for EXU A) or to the busses or the other half of the cluster via a Type II switch (shown in detail). The squares and the circles represent inputs and outputs to the switches respectively. Data lines flow horizontally and control lines vertically. The major advantage of this approach is that it allows fitting of all horizontal busses within the pitch provided by the EXUs, yielding substantial area savings.

Each EXU has an SRAM-based nanostore that is serially configured at set-up time. At run-time, an external sequencer broadcasts to each nanostore a 3b global address that is locally decoded into a 53b instruction word. This scheme saves on instruction pins, allowing more pins for data I/O. Each instruction completes in one clock cycle. Figure 4 shows a section of the SRAM. Shown are the row decode, six transistor cell, and read/write circuitry. Master-slave scan registers at its I/O are connected to a global shift register for serial configuration of the SRAM (and the chip). EXUs and the external controller communicate status information over the crossbar switch to affect both the local and global control flows. By setting one or both of its interrupt state fields, an EXU can accept interrupts from other EXUs or the external world and vector to the appropriate address contained in its pre-compiled interrupt vectors. Two on-board FSMs generate the configuration control signals that interface directly to standard EPROMs.

Figure 5 is a chip micrograph. The chip was functional on first silicon executing a variety of programs. Figure 6 shows an example: the assembly code for a counter (mapped to 2 EXUs). The block depth counter counts from 0 and increments by 1 each clock cycle. Its output is continually monitored by the block depth compare that signals a reset at value 1, taking effect in the next cycle. Thus, the block depth counter cycles between 0, 1, 2. Figure 7 shows scope traces of the 25MHz clocks and the lsb's of the counter output. Due to test-setup limitations, the quality of the clock signals at 25MHz were non-ideal so it is likely that the chip can operate at higher speed. An assembler and simulator are completed and a high-level-language compiler is being developed to facilitate the programming task. An expanded chip, and an MCM with several chips are being designed to provide more computational power.

#### Acknowledgements

This project was sponsored by DARPA and Sharp Microelectronics Technology, Inc. The authors thank R.W. Brodersen, A. Lee, S. Li, E. Ng, D. Schultz, C. Yu and the members of the BJ-Group and MOSIS for support.

#### References

- [1] Chen, D. C., and J. M. Rabaey. "PADDI: Programmable Arithmetic Devices For Digital Signal Processing". In VLSI Signal Processing IV, pp. 240-249. IEEE Press, Nov. 1990.
- [2] van Roermund, A. H., et al., "A General Purpose Programmable Video Signal Processor" IEEE Transactions on Consumer Electronics, pp. 249-258, August 1989.

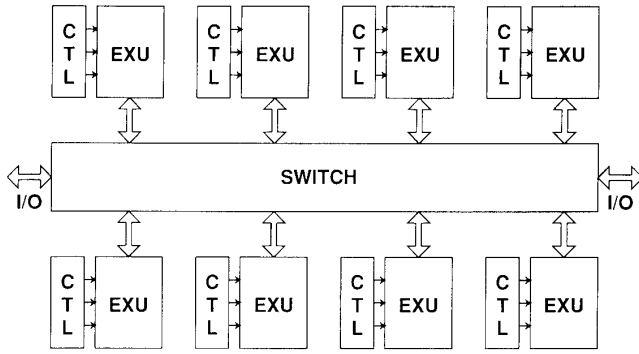


Figure 1: Prototype architecture.

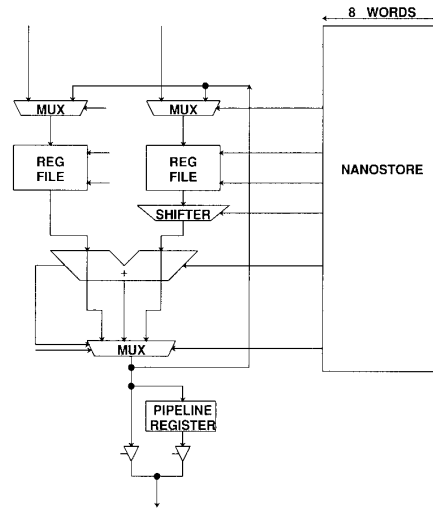


Figure 2: EXU architecture.

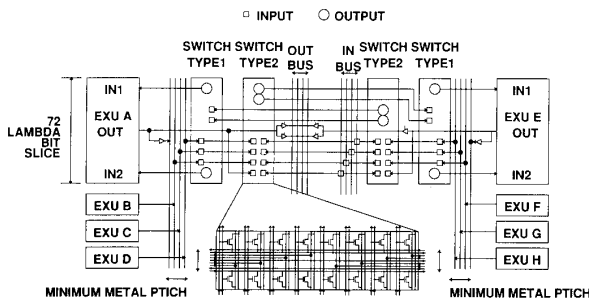


Figure 3: Simplified schematic of crossbar switch.

```

/* THE BELOW SETTINGS ARE THE 'DEFAULT' VALUES FOR
ALL EXECUTION UNITS; EACH OF THESE SETTINGS,
HOWEVER, CAN BE OVERRIDDEN LOCALLY FOR EACH EXU.
*/
DEFAULTS { A6=0, B6=1, NORMAL A, NORMAL B,
SIGNED, UNLINK, IEN1=0, IEN2=0 }

/* THE BELOW MAPPINGS ALLOW FOR MORE DESCRIPTIVE
NAMES FOR EACH OF THE EXUS. */
MAP { (BLOCK_DEPTH_COUNTER=XA),
(BLOCK_DEPTH_COMPARE=XB) }

/* INSTRUCTIONS 0 THROUGH 7 ARE SPECIFIED BELOW FOR EACH
EXU. UNSPECIFIED INSTRUCTIONS DEFAULT TO NOPS. */
BLOCK_DEPTH_COUNTER
/* VECTOR TO INSTRUCTION 1 WHEN
BLOCK_DEPTH_COMPARE ASSERTS ITS FLAG */
FLAG1=BLOCK_DEPTH_COMPARE, IVEC1=1
{ /* INSTRUCTION 0: INCREMENT A6 BY 1 AND STORE THE
RESULT IN A6 AND B1, ENABLE INTERRUPT IEN1, SEND
RESULT TO OUTPUT BUS O1 */
0: A6=THIS_EXU, B1=THIS_EXU, (A6+B6), IEN1, O1;

/* INSTRUCTION 1: SUBTRACT B1 FROM A6 AND STORE RESULT
IN A6 AND B1 (THIS RESETS A6 AND B1 TO ZERO), SEND
RESULT TO OUTPUT BUS 1 */
1: A6=THIS_EXU, B1=THIS_EXU, (A6-B1), O1; }

BLOCK_DEPTH_COMPARE
A6=0, B6=0, /* OVERRWRITE GLOBAL DEFAULT VALUES */
FLAGOUT1=1 /* ROUTE FLAG OFF-CHIP VIA BUS 1 (FOR
MONITORING) */
{ /* INSTRUCTION 0: LATCH OUTPUT OF BLOCK DEPTH COUNTER
INTO REGISTER B6, SUBTRACT B6 FROM A6. IF B6 IS
GREATER THAN OR EQUAL TO A6, ASSERT FLAG. SEND
RESULT OF A6-B6 TO OUTPUT BUS O2 (FOR MONITORING)*/
0: B6=BLOCK_DEPTH_COUNTER, (A6-B6), O2; }
    
```

Figure 6: 25MHz counter assembly code.

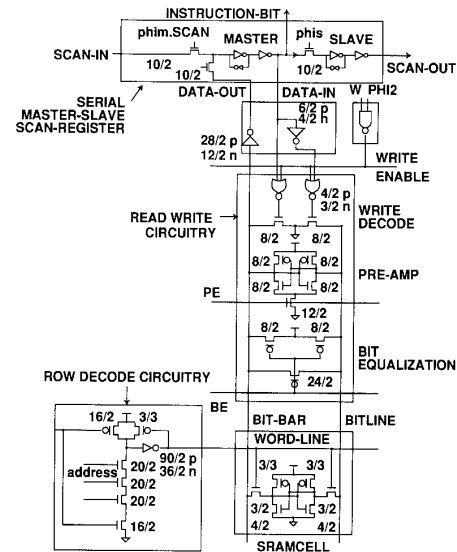


Figure 4: SRAM schematic.

Figure 5: See page 249.

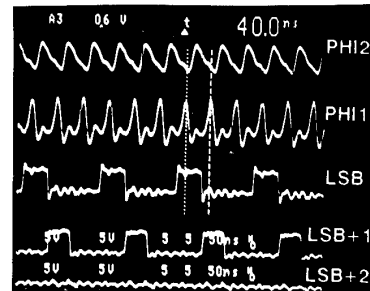


Figure 7: 25MHz counter waveforms.