RESST: A VLSI Implementation of a Record-Sorting Stack

Michael J. Carey
Paul M. Hansen
Clark D. Thompson

Computer Science Division
Department of Electrical Engineering and Computer Science
University of California
Berkeley, CA 94720

ABSTRACT

This report discusses a VLSI implementation of a record-sorting stack. Records are represented on the stack as (key, record-pointer) pairs, and the operations supported are PUSH, POP, and CLEAR. When records are POPped, they are returned in smallest-first order. The implementation allows the sorting of n records in $O(n)$ time, and the design is cascadable so that the capacity of a single VLSI chip does not limit the amount of data which may be sorted.

This report describes a paper design and evaluation, and thus serves two purposes: It describes one particular VLSI sorting circuit, and it also serves as a case study in VLSI design methodology. The algorithm is described, the overall chip organization and data flow are presented, and detailed circuits, layouts, and timing analyses are given.

This work was supported by the National Science Foundation Grant ECS-8110684, a Chevron U.S.A. Career Development Grant, a California MICRO Fellowship, the Air Force Office of Scientific Rese-:ch Grant AFOSR-78-3596, and the Naval Electronic Systems Command Contract NESC-N00039-81-C-0569.

RESST: A VLSI Implementation of a Record-Sorting Stack

Michael J. Carey
Paul M. Hansen
Clark D. Thompson

Computer Science Division
Department of Electrical Engineering and Computer Science
University of California
Berkeley, CA 94720

1. Introduction

This report describes RESST, a VLSI REcord Sorting STack. The chip operates in a stack-like manner, allowing (key, record-pointer) pair representations of records to be pushed and popped. Each key is an 8-bit integer, and each record-pointer is an 8-bit pointer value. When a (key, record-pointer) pair is popped from the RESST chip, the pair with the smallest key value is returned. Hence, RESST may be used to sort a group of records by pushing them and then popping them. We envision RESST chips being utilized as a form of hardware support for database systems, perhaps as disk buffer storage for providing automatic sorting of secondary indices for multi-relation query processing in a relational database system [Ull80].

The algorithm chosen for the VLSI RESST implementation, a parallel version of the classic bubblesort algorithm, is quite similar to other recent work in the area of hardware sorting devices. Armstrong and Rem [Arm82], Chung, Luccio, and Wong [Chu80], Leiserson [Lei79], Miranker, Tang, and Wong [Mir82], and Mukhopadhyay and Ichikawa [Muk72, Muk81] have all published papers on similar $O(n)$ time hardware sorting algorithms. The particular algorithm chosen for the RESST implementation is basically the Weavesort algorithm of Mukhopadhyay [Muk81], though it was independently developed when RESST was designed.

In the remainder of this report, we will describe the hardware sorting algorithm and a 32 data item VLSI implementation. The algorithm and the overall chip organization will be presented in section 2. Circuit schematics and cell layouts will be presented in section 3. Performance estimates for RESST, based on simple SPICE models for timing and power consumption, will be discussed in section 4. Section 5 contains a discussion of some possible enhancements for RESST. Finally, section 6 contains a summary of the main things that we have learned from our RESST experience.

2. High-Level Design Description

This section of our report discusses the high-level design issues involved in the RESST project. The RESST sorting algorithm is presented, and the overall structure and data flow of the RESST chip are described.

2.1. A Hardware Sorting Algorithm

The algorithm that we chose to use in the design of RESST is a parallel bubblesort algorithm. Let n be the number of items to be sorted, where each item has a fixed-length key (that is, key length is independent of n). Let N be the number of items that the chip can hold. This algorithm allows n items to be sorted in $O(n)$ time using $O(N)$ chip area. We chose this algorithm from the many possible VLSI sorting algorithms [Tho82] for several reasons, including simplicity, regularity, and extensibility.

The parallel bubblesort algorithm is two-phase in nature. That is, each step of the sorting process consists of two substeps. Let $key[i]<j>$ and $recPtr[i]<j>$, where $i = 0,1, \ldots, 31$ and $j = 0,1, \ldots, 7$, denote the jth bit of the ith word of the RESST key and record storage, respectively. (For the ith element, $key[i]<0>$ and $recPtr[i]<0>$ represent the least significant bits.) Let $key[-1]$ and $recPtr[-1]$ represent the values presented to/from the I/O pins of the chip. The two phases of the algorithm are given in terms of this notation in Figure 2.1.

The first substep of the algorithm (phase 1) involves shifting data in or out, clearing the chip's storage cells, or refreshing the chip's storage cells. The PUSH, POP, and REFRESH operations have the obvious meanings, while the CLEAR operation is somewhat more subtle. Associated with each 8-bit key is a 9th, hidden bit. This bit serves to distinguish real key values from empty cells, with a one in this most significant bit position representing an empty cell. This way, nonempty cells are always kept towards the left (I/O) side of the chip.

The second substep of the algorithm (phase 2) does the actual sorting, and involves comparing and conditionally exchanging the (key, record-pointer) pairs associated with keys 0 and 1, 2 and 3, and so on, up to $N-2$ and $N-1$. As denoted by the **forall...pardo...od** notation, these pairwise comparisons take

```
Phase 1:        { I/O Phase }

    case operation of

      PUSH:
            forall i in 0..31 pardo
              key[i] := key[i-1];
              recPtr[i] := recPtr[i-1];
            od;

      POP:
            forall i in 0..31 pardo
              key[i-1] := key[i];
              recPtr[i-1] := recPtr[i];
            od;

      CLEAR:
            forall i in 0..31 pardo
              key[i]<8> := 1;
            od;

      REFRESH:
            forall i in 0..31 pardo
              key[i] := key[i];
              recPtr[i] := recPtr[i];
            od;

    end;

Phase 2:        { Compare/Exchange Phase }

    forall i in 0..31 by 2 pardo
      if key[i] > key[i+1] then
        Exchange(key[i], key[i+1]);
        Exchange(recPtr[i], recPtr[i+1]);
      fi;
    od;
```

Figure 2.1: Hardware Sorting Algorithm.

place in parallel. An example of the operation of this algorithm is depicted in Figure 2.2.

The advantages of this algorithm for VLSI implementation should be immediately obvious. Because data only moves between adjacent storage cells, and because comparisons take place only between every other pair of adjacent cells, there is no need for global communication. As a result, the cell layout can be organized as a simple linear array of storage cells with a compare/exchange cell between every other pair of adjacent storage cells.

			REGISTER PAIRS						
operation	item	phase	0	1	2	3	4	5	
PUSH	1	1	1						. . .
		2	1						
PUSH	3	1	3	1					. . .
		2	1	3					
PUSH	9	1	9	1	3				. . .
		2	1	9	3				
PUSH	5	1	5	1	9	3			. . .
		2	1	5	3	9			
PUSH	2	1	2	1	5	3	9		. . .
		2	1	2	3	5	9		
PUSH	8	1	8	1	2	3	5	9	. . .
		2	1	8	2	3	5	9	
POP	1	1	8	2	3	5	9		. . .
		2	2	8	3	5	9		
POP	2	1	8	3	5	9			. . .
		2	3	8	5	9			
POP	3	1	8	5	9				. . .
		2	5	8	9				
POP	5	1	8	9					. . .
		2	8	9					
POP	8	1	9						. . .
		2	9						
POP	9	1							. . .
		2							

Figure 2.2. Example of Parallel Bubblesort Operation.

The extensibility of the design should also be clear at this point. With such a simple linear array of cells, it is quite easy to accommodate the sorting of more than *N* items by making RESST chips cascadable. Providing cascadability just involves buffering the right-hand outputs and inputs of the last storage cell in the array and providing off-chip connections for them.

2.2. Chip Structure and Data Flow

The actual structure and data flow for RESST follow naturally from the preceding discussion of the algorithm and its advantages. As described in the discussion, the algorithm is two-phase, so our design utilizes a two-phase clocking scheme. The chip is organized as a linear array of storage and compare/exchange cells. In our physical design, we chose to group each pair of storage cells and their associated compare/exchange unit into a single cell, called a COL cell. The overall RESST structure, shown in terms of this type of cell, is shown in Figure 2.3. As shown, data flows horizontally between adjacent COL cells.

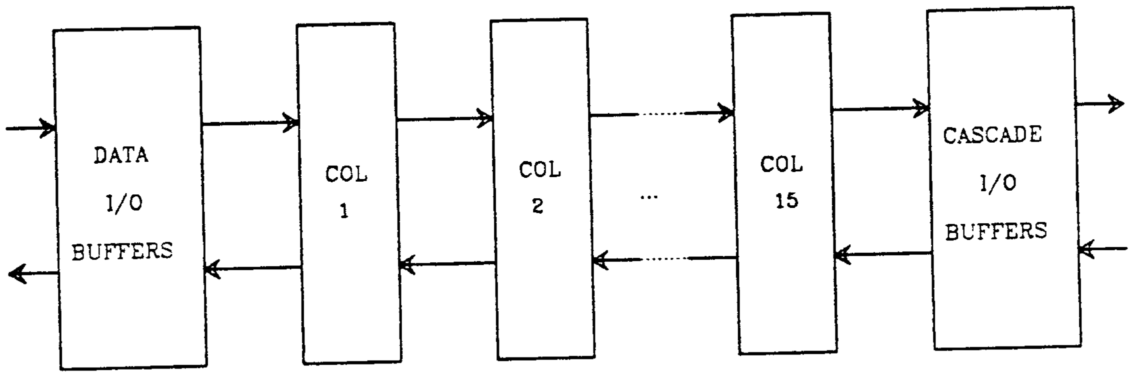


Figure 2.3. RESST Chip Structure.

Each COL cell contains a pair of 8-bit keys, a pair of 8-bit record-pointers, and logic for comparing keys and exchanging (key, record-pointer) pairs. There are 16 COL cells in our RESST implementation, so the kth COL cell, $k = 0,1, \ldots, 15$, contains $key[2k]$<0:8>, $key[2k+1]$<0:8>, $recPtr[2k]$<0:7>, and $recPtr[2k+1]$<0:7>.

Five types of cells are used for building a COL cell: a CE cell, which contains storage and compare/exchange logic for a pair of key bits, a TOPCE cell, which is a CE cell with preset capability (to support the CLEAR function), an RP cell, which contains storage and exchange logic for a pair of record-pointer bits, a CBUF cell, which buffers a Manchester-type carry chain used for word-parallel comparisons, and a PHI2SIG cell, which generates clocked exchange signals from the end of the carry chain for controlling the CE and RP cell exchange logic. The structure of a COL cell in terms of these five subcell types is depicted in Figure 2.4.

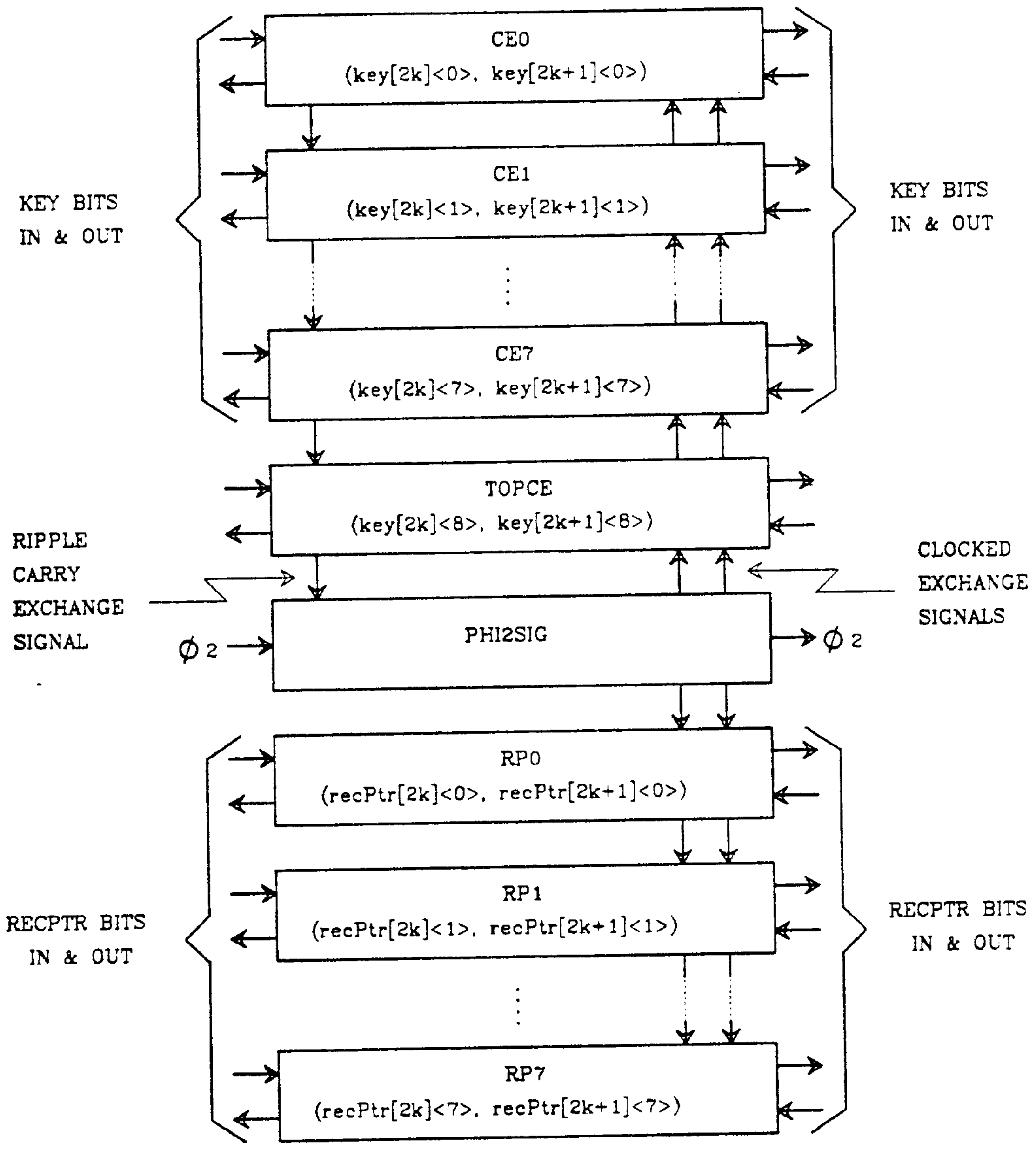

CE0
(key[2k]<0>, key[2k+1]<0>)
CE1
(key[2k]<1>, key[2k+1]<1>)
CE7
(key[2k]<7>, key[2k+1]<7>)
TOPCE
(key[2k]<8>, key[2k+1]<8>)
PHI2SIG
RP0
(recPtr[2k]<0>, recPtr[2k+1]<0>)
RP1
(recPtr[2k]<1>, recPtr[2k+1]<1>)
RP7
(recPtr[2k]<7>, recPtr[2k+1]<7>)
KEY BITS IN & OUT
KEY BITS IN & OUT
RIPPLE CARRY EXCHANGE SIGNAL
CLOCKED EXCHANGE SIGNALS
ϕ 2
ϕ 2
RECPTR BITS IN & OUT
RECPTR BITS IN & OUT

Figure 2.4. COL Cell Structure.

As shown in the figure, data flows horizontally, the carry chain signal flows vertically towards the PHI2SIG cell, the phase 2 clock signal flows horizontally through the PHI2SIG cell, and the clocked exchange signals flow vertically out from the PHI2SIG cell. There are also clocked PUSH, POP, CLEAR, and HOLD signals which flow vertically through the COL cell, and power and ground which flow horizontally through each subcell of the COL cell.

3. Low-level Design Description

This section of our report discusses the operation of the circuits employed to construct the basic cells of the RESST Chip, and is intended to answer the question *how does it do its job*?

3.1. Compare-Exchange Circuit (CE)

Figure 3.1 shows a block diagram representation of the basic compare/exchange circuit (CE) showing control and data signals. Functionally, it can be simply described as a pair of semi-static registers with some additional circuitry to provide a comparison of the information contained in the two cells and to either pass the carry-like exchange chain signal EXCHIN to the next most significant bit or to assert EXCHOUT high if an exchange is called for by a mismatch in the cell. Figure 3.2 shows the circuitry in mixed notation. The italicized signal names refer to signals which stay within the bounds of a single CE cell.

The controlling signals PUSH1 and POP1 function as you would expect. The user asserts either PUSH or POP in proper phasing with clock phase 1 (PHI1). PHI1 is ANDed to produce PUSH1 or POP1 which are then used to control writing into and reading from the RESST. Data present on BDIN will be passed to the gate of the first inverter of the cell during PUSH1. Data in the cell may also be passed to the left (read) via BDOUT by controlling POP1 in similar fashion. Obviously, POP1 and PUSH1 are mutually exclusive for proper circuit operation. During a PHI1 clock cycle in which neither reading nor writing of the RESST is desired, the signal HOLD1 guarantees that the information which is stored in the semi-static cell is refreshed.

Adjacent cells function in parallel, allowing true stack operations. Data is allowed to flow between the *i*th and *i*+1st CE cells by connecting UBDIN of the *i*th cell to BDOUT of the *i*+1st cell and, similarly, connecting UBDOUT of the *i*th

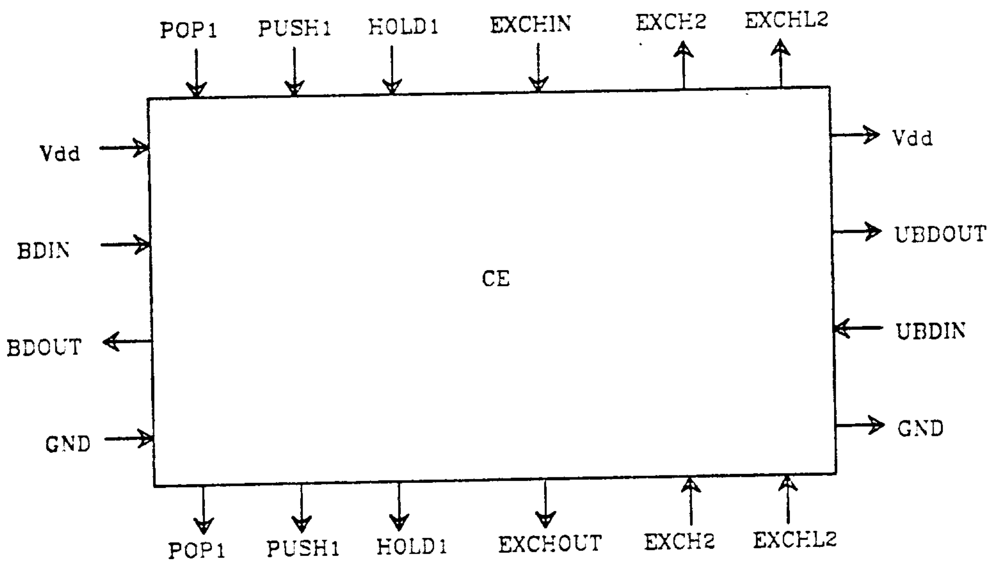


Figure 3.1. COMPARE/EXCHANGE Cell Block Diagram.

cell to BDIN of the $i+1$st cell. Hence, data flows to the "right" during PUSH operations, and to the "left" during POP operations.

As mentioned above, the contents of the two cells are compared during clock phase 2 (PHI2). If a match occurs, the EXCHIN signal is propagated. If a mismatch occurs, EXCHOUT will be the inverted value of the rightmost bit (signaling that an exchange is needed if the rightmost bit is "0"). An exchange is required if EXCH2 goes high after the carry-like EXCHIN/EXCHOUT signal has been fully propagated through all the CE cells in a word (going from the least to the most significant bit position). If an exchange is indeed called for, the two EXCH2-controlled pass transistors in the circuit serve to exchange the inverted data of the two storage cells. Thus, data is always applied to the leftmost inverter of a pair on PHI1 and to the rightmost inverter on PHI2. This is consistent throughout the design.

The cifplot layout of the cell is shown in Figure 3.3.

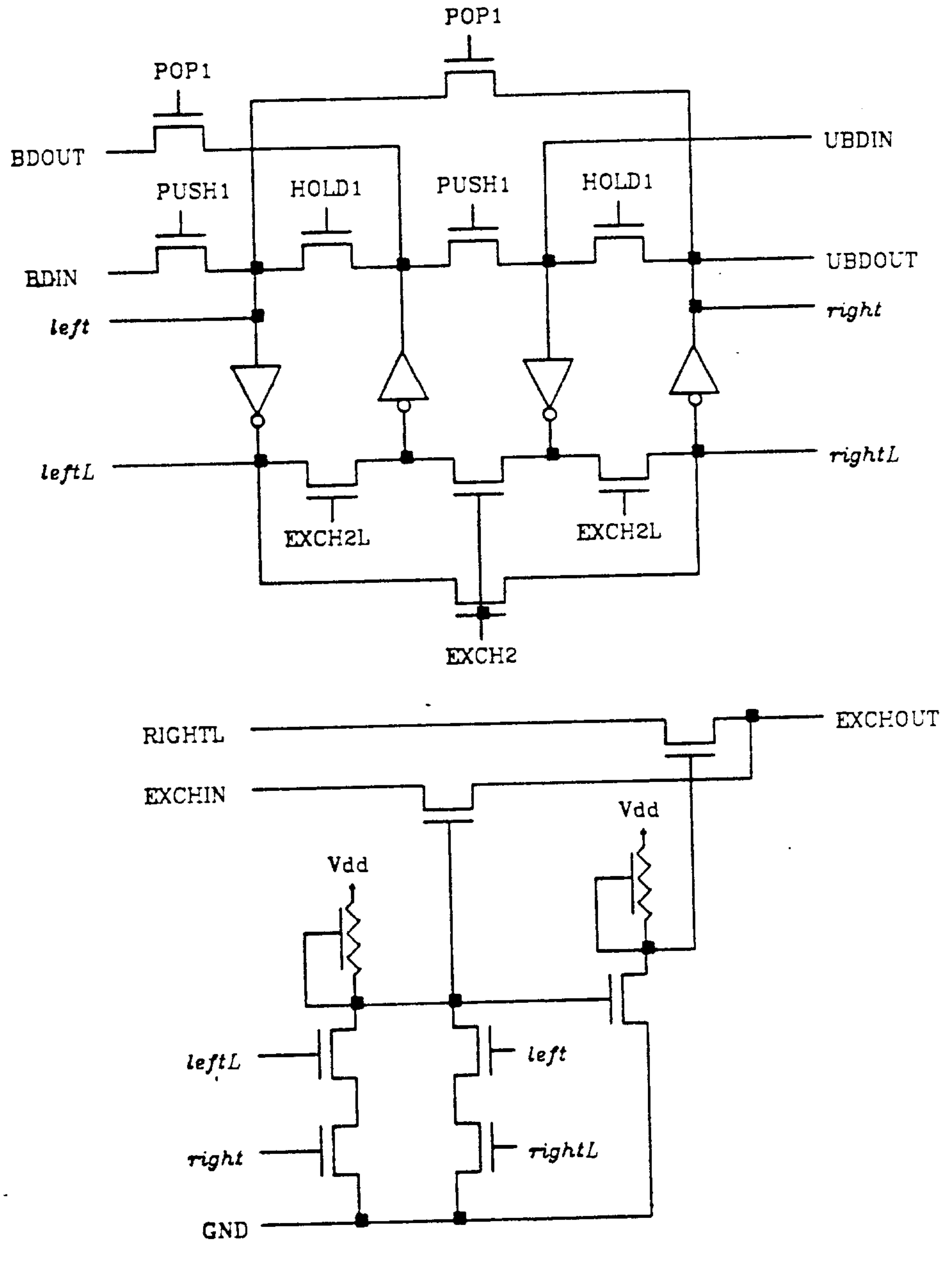


Figure 3.2. COMPARE/EXCHANGE Cell Schematic Diagram.

3.2. Record Pointer Circuit (RP)

As mentioned in the functional description of RESST, record pointers are entered simultaneously with keys to form (key, record-pointer) pairs. The record-pointer cell (RP) is identical to the CE cell, except that the compare circuitry is not necessary. (As you will read later in our conclusions, there is an interesting modification to this strategy which can provide some unique capabilities.) By extracting the compare circuitry, the cell is made less dense in terms

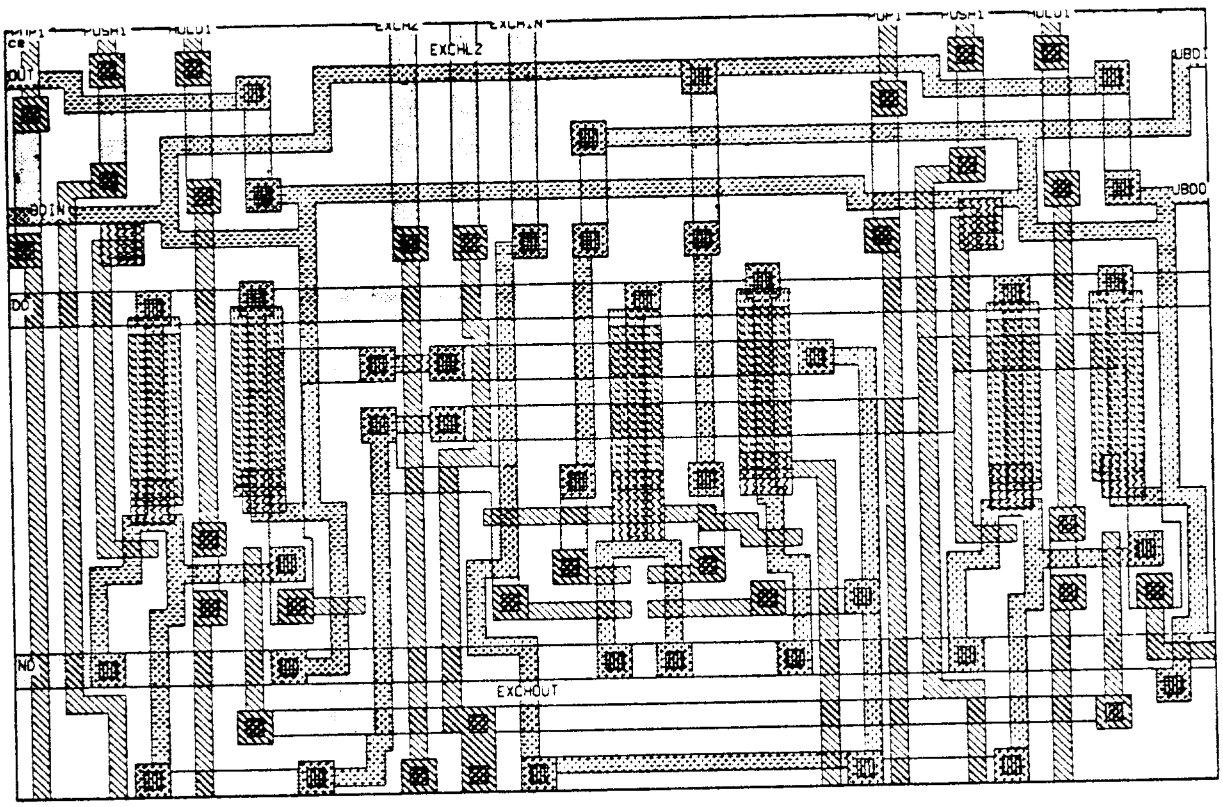


Figure 3.3. COMPARE/EXCHANGE Cell CIFPLOT Layout.

of the geometry. Since the overall width of a column of cells composing a compare/exchange word is determined by the CE cell, the reduction in width of the RP cell will gain nothing. By compressing the cell in height, a potentially *shorter* cell may result, with some space advantages being the net result. Due to time and resource constraints, we decided not to optimize the height of the RP cell for the purpose of the project. A refinement of the design would certainly take this into account. Figure 3.4 shows the layout for the RP cell.

3.3. Top COMPARE/EXCHANGE Circuit (TOPCE)

The TOPCE circuit is functionally identical to the CE circuit, with the minor addition of a *clear* capability. Following assertion of the clear signal, the entire array is supposed to contain null values. To achieve this, the CLEAR1 signal must cause the most significant bit of all keys to be set to "1" (by definition). Thus, the TOPCE circuit is simply a CE circuit with a pass transistor (controlled by CLEAR1) which gates Vdd into each bit. Figure 3.5 shows the layout for the

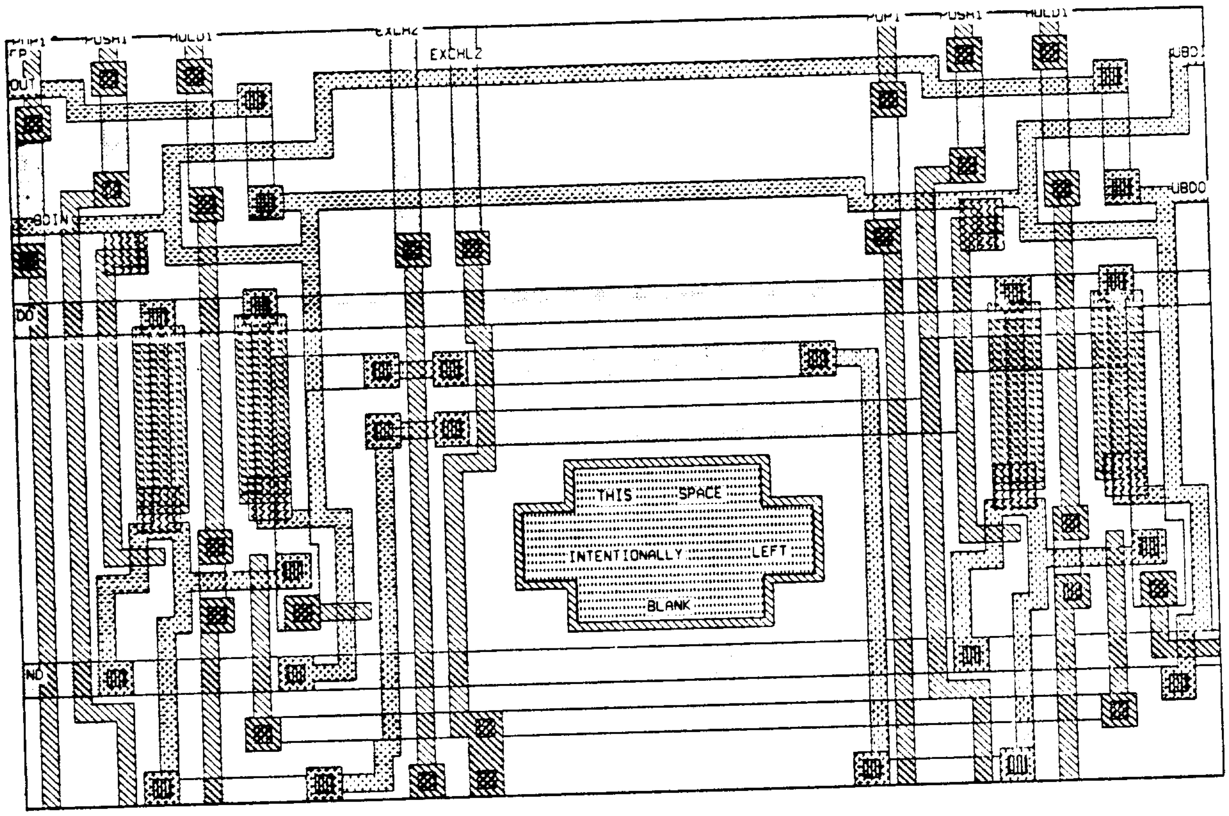


Figure 3.4. RECORD-POINTER CIFPLOT Cell Layout.

TOPCE cell.

3.4. Compare Buffer (CBUF)

As previously mentioned, a word-parallel key comparison involves a carry-like pass transistor chain to propagate the EXCHIN/EXCHOUT signals. Due to signal degradation in such pass transistor configurations, it is necessary to restore these types of signals with a buffer after every 3 or 4 stages [Mea80]. CBUF, a non-inverting super-buffer, performs this function in the RESST design. Figure 3.6 shows the schematic for the CBUF cell, and Figure 3.7 shows the layout.

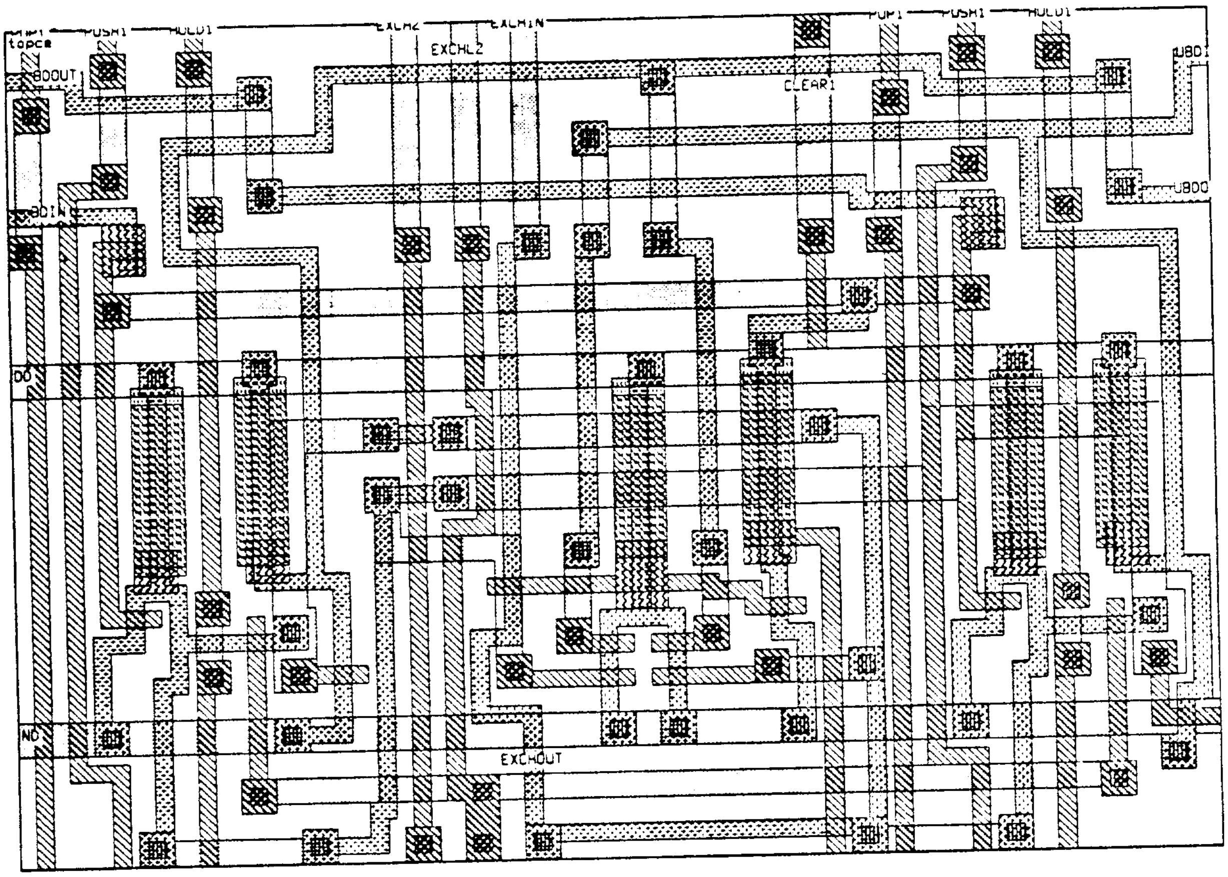


Figure 3.5. TOPCE CIFPLOT Cell Layout.

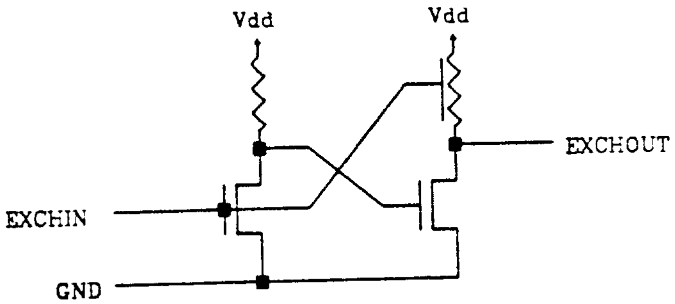


Figure 3.6. CBUF Schematic Diagram.

3.5. Phase 2 Clock Signal Cell (PHI2SIG)

In order to control the exchanging of data during phase 2, it is necessary to generate the clocked exchange signals EXCH2 and EXCHL2. These signals are formed by ANDing the TOPCE cell's EXCHOUT signal (PHI2SIG's EXCHIN input

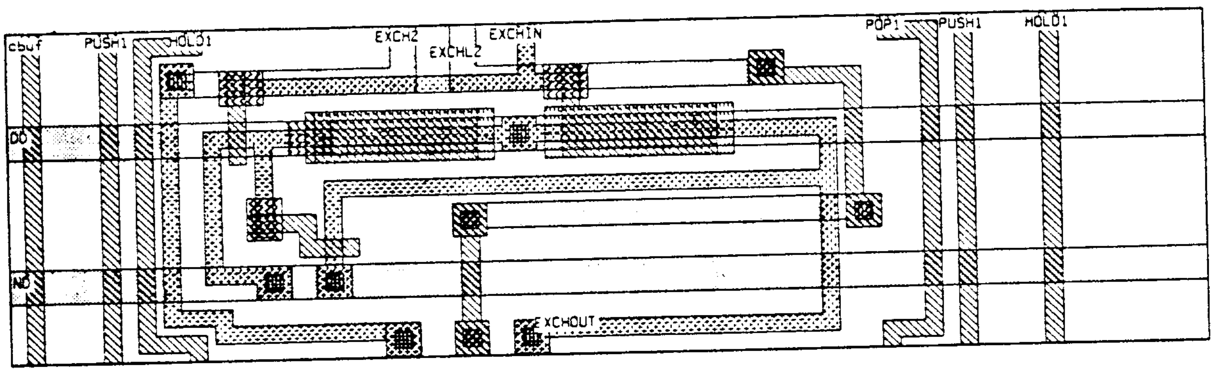


Figure 3.7. CBUF CIFPLOT Layout.

signal) and its complement with the phase 2 clock in the PHI2SIG cell. Also, since each such control signal must be propagated through an entire COL cell, these signals must be of super-buffer quality. The actual PHI2SIG circuit, shown in Figure 3.8, consists of a CBUF-type circuit to generate EXCHIN and EXCHINL signals of reasonable quality, and a super-buffer NOR circuit to combine them with the phase 2 clock. The layout for PHI2SIG is shown in Figure 3.9.

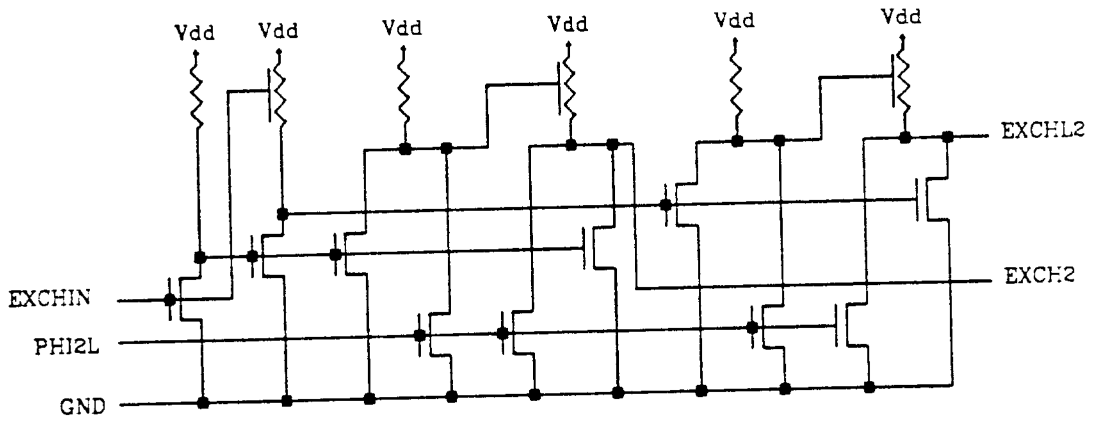


Figure 3.8. PHI2SIG Schematic Diagram.

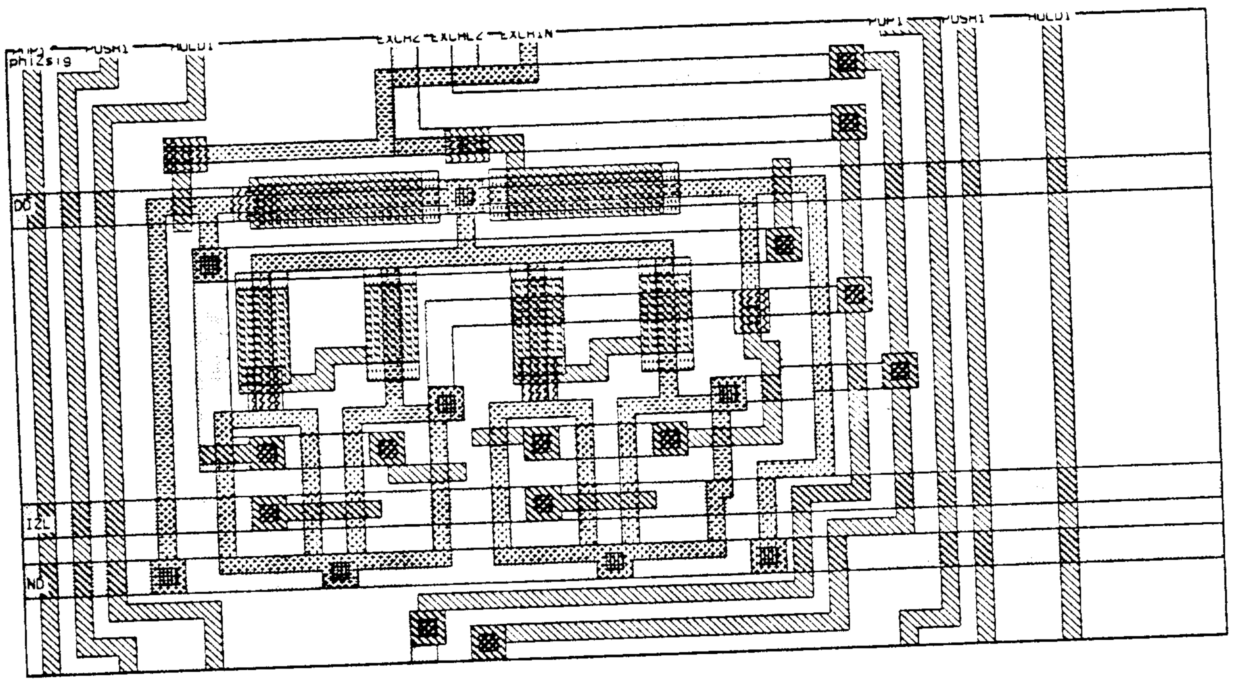
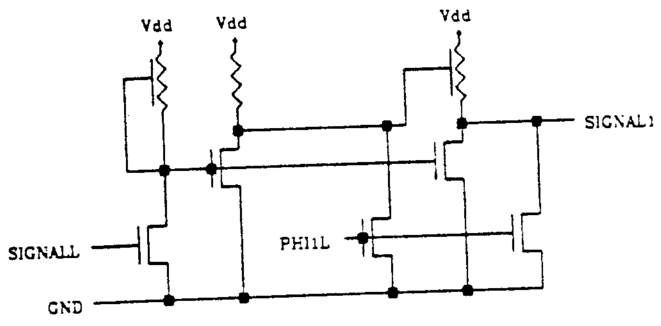


Figure 3.9. PHI2SIG CIFPLOT Layout.

3.6. Phase 1 Clock Signal Cell (PHI1SIG)

There are several clocked control signals required for phase 1 operations as well. These signals are: PUSH1, POP1, HOLD1, and CLEAR1. The PHI1SIG cell generates these signals in the same way that the PHI2SIG cell generates the clocked EXCH2 and EXCHL2 signals. Also, it generates the HOLD signal itself, which is high when none of PUSH, POP, or CLEAR are high. The PHI1SIG schematic is shown in Figure 3.10, and the layout for PHI1SIG is shown in Figure 3.11.



Note: Phi1sig actually contains 4 copies of this cell (for PUSH1, POP1, HOLD1, and CLEAR1) and a NOR circuit for HOLD1.

Figure 3.10. PHI1SIG Schematic Diagram.

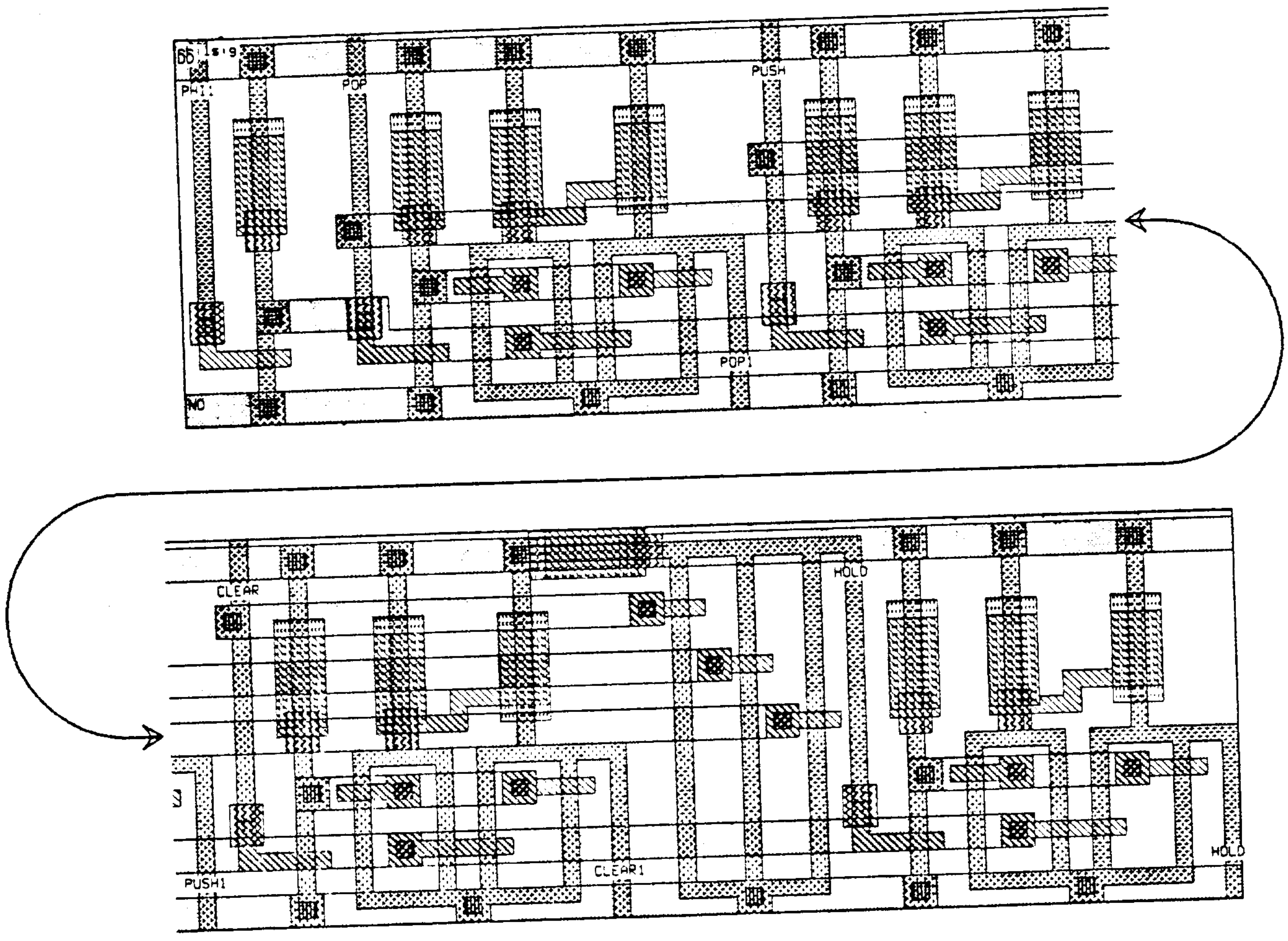


Figure 3.11. PHI1SIG CIFPLOT Layout.

3.7. Other Miscellaneous Signals

There are three other signals generated on our chip. These signals are PHI2L, PUSHL, and POPL. PHI2L is distributed to all of the PHI2SIG cells, and PUSHL and POPL are used to disable the data input/output and cascaded input/output tri-state pads when PUSH and POP operations are called for (respectively) to avoid having output drivers active while data is coming in. Each of these is generated using a standard inverting super-buffer circuit, ala [Mea80].

The remainder of our circuitry is accounted for by the standard library cells PadInBuff, PadOut, and PadTriState. PadInBuff is used for PUSH, POP, CLEAR, PHI1, and PHI2. PadOut is used for two special status signals, FULLL and EMPTY. The FULLL signal, low when all 32 of the storage cells on the chip contain valid data, is a buffered version of *key*[31]<8>. The EMPTY signal, high when none of the storage cells contain valid data, is simply a buffered version of *key*[0]<8>. PadTriState is used for the remaining signals, which are the key and pointer input/output and cascade input/output lines. Since these are standard cells, we will not describe them further here.

4. Performance Estimates

This section of our report discusses the performance of the RESST chip. Since RESST is a synchronous system, employing standard two-phase non-overlapping clocks, the key question is *how long are phase 1 and phase 2*? To determine the answer, we will examine what goes on in each of the clock phases using simple SPICE models of the salient portions of our circuit. In all of our models, we include maximally pessimistic parasitic effects (we rounded up all capacitance estimates and placed all diffusion resistances in front of line capacitances in our models). After we consider the speed of our design, we will consider how the speed might be improved. Finally, we will briefly consider the power consumption of the design. (The MOS parameters used in these performance estimates were taken from page 51 of Mead and Conway [Mea80], with the polysilicon resistivity assumed to be 50 ohms per square.)

4.1. Timing Estimates

4.1.1. Phase 1 Timing

During phase 1, data is PUSHed or POPped from the RESST chip. This involves getting data on and off the chip, getting phase 1 clocked signals distributed throughout the chip, shifting data from cell to cell, and allowing the carry-chain-like EXCHIN/EXCHOUT signal to ripple from the least to the most significant bit (worst case). Note that, for cascadability, it is necessary to allow sufficient time to shift between adjacent chips as well as adjacent cells within a single RESST chip.

For getting data on and off the chip, we simply rely on PadInBuff and PadTriState. Folklore has it that a reasonable estimate of delays for these pads is on the order of 35 nanoseconds. Since none of our pads are required to drive particularly large capacitances (the data pads each drive a single transistor gate, and the input pads drive our own logic/buffer circuit gates), we believe that it is reasonable to accept this estimate.

For generating and distributing phase 1 signals (PUSH1, POP1, HOLD1, and CLEAR1), we used SPICE to model the performance of our super buffer circuits driving the long polysilicon control lines. The model for the line parasitics was obtained by counting squares and using the result to compute the diffusion resistance and capacitance values from the Mead/Conway parameter table [Mea80]. The line resistance was 75K ohms (quite large), and the parasitic capacitance was 0.96 picofarads. The time required to drive the signals to the 90% point was found to be approximately 300 nanoseconds for going high and 240 nanoseconds for going low. (Our model and results are included in the Appendix as "SPICE #1: Phase 1 Signals".)

For exchanging data between adjacent cells within a single RESST chip, the critical quantity is the time it takes a static-storage inverter to drive another inverter through a pass transistor (including line parasitics, of course). For parasitics, we used the longest of the inverter-pass-inverter distances between cells, and came up with a parasitic resistance of 1.2K ohms and a parasitic capacitance of 0.192 picofarads. In our model, we first charged the input of the driven gate high, and then simulated the time required to drive it low and have it pull its output high. This time was determined to be roughly 8 nanoseconds. (Our model and results are included in the Appendix as "SPICE #2: On-Chip Shift".)

For the EXCHIN/EXCHOUT ripple delay, we simulated the time needed for driving a set of three pass transistors followed by a super-buffer. The parasitic resistance on this path was found to be about 4K ohms, and the parasitic capacitance was found to be 0.48 picofarads. The overall delay was determined to be approximately 100 nanoseconds. For a set of three such pass-transistor/super-buffer chains, which is what our 8-bit (plus hidden bit) key involves, the total ripple delay is thus about 300 nanoseconds. (Our model and results are included in the Appendix as "SPICE #3: EXCHIN/EXCHOUT Ripple".)

Based on these results, the phase 1 clock width is overwhelmingly determined by phase 1 signal distribution, off-chip shifts, and the EXCHIN/EXCHOUT ripple delay. Since all RESST chips in a cascaded configuration will presumably receive clock signals simultaneously, the overall phase 1 width must include one 35 nanosecond pad time for the phase 1 clock, one 300 nanosecond propagation time for the clocked control signals, two 35 nanosecond pad times for data to go from chip to adjacent chip, and then one 300 nanosecond ripple delay. Hence, phase 1 must be at least 705 nanoseconds long. (Note that the 8 nanosecond on-chip shift occurs in parallel with the 70 nanosecond chip-to-chip shift.)

4.1.2. Phase 2 Timing

During phase 2, keys are compared and exchanges are conditionally made. This involves time to get the phase 2 signal to the PHI2SIG cell, time for the PHI2SIG cell to generate and distribute clocked EXCH2 and EXCHL2 signals, and time to actually perform the exchanges.

For the time to get the phase 2 signal to its PHI2SIG destination, we simulated a super-buffer driving the PHI2L line parasitics. The resistance involved was about 10K ohms, and the parasitic capacitance was about 1.4 picofarads. The resulting signal delay was 210 nanoseconds. (Our model and results are included in the Appendix as "SPICE #4: Phase 2 Signals".) Added to the 35 nanosecond phase 2 pad delay, this produces a delay of 245 nanoseconds in all.

For the PHI2SIG exchange signal return path, we have a super-buffer driving a number of pass transistor gates and some parasitics. Through counting squares and gates, we found the driven resistance to be about 8.6K ohms and the driven capacitance to be 1.6 picofarads. This resulted in a signal delay of approximately 225 nanoseconds. (Our model and results are included in the Appendix as "SPICE #5: EXCH Signal Return".)

For the actual data exchange time, our previous model ("SPICE #2: On-Chip Shift") is applicable. Thus, the exchange operation takes about 8 nanoseconds.

The width of the phase 2 clock must be sufficient for all of these events to occur sequentially, so the total phase 2 time must equal or exceed the sum of the previous results. This sum is about 480 nanoseconds.

4.1.3. Overall Speed

Using pessimistic approximations and SPICE for circuit simulation, we determined each of the required clock widths. We obtained a width of 705 nanoseconds for phase 1 and a width of 480 nanoseconds for phase 2. The total cycle time for our present RESST design, then, is 1185 nanoseconds, or about 1.2 microseconds.

Two questions arise at this point. First of all, is 1.2 microseconds good enough for real applications? Secondly, what sort of times would have been produced with less than maximal pessimism?

To answer the first question, consider the use of RESST chips for disk buffering. A typical disk has a seek time of around 15 milliseconds and a transfer rate of around 1 megabyte per second. If each disk block read requires a seek, our 1.2 microsecond device is just fine. It is also capable of keeping up with sequential block reading, as 16 bits enter the RESST chip per sorting cycle, thus making anything faster than a 2 microsecond device sufficient.

As for the second question, suppose that our maximally pessimistic model of a diffusion-type resistance and capacitance is roughly a factor of 2 slower than an actual circuit. If this were the case, then our chip would actually have a cycle time of approximately 600 nanoseconds. However, since the RESST design is already capable of handling disk transfer rates under worst-case assumptions, the actual performance figures are of little interest.

The result of all this is that the performance of RESST seems quite satisfactory. Based on our performance estimates, RESST should be able to serve well as a disk block buffer for a database machine, and our design goal is thus satisfied.

4.1.4. Speed Enhancements

From the discussion above, it is clear that the dominant times are a result of driving long lines with super-buffer circuitry. In fact, the worst of the delays were encountered when driving materials other than metal. Thus, were it necessary to speed RESST up for even higher performance database applications, there are at least two things that we could do. First, we could make an effort to maximize the use of metal in long lines, using other materials only when metal crossovers are unavoidable. Second, we could utilize the Mead and Conway logarithmic scheme for optimal buffering [Mea80], especially for generating PUSH1, POP1, HOLD1, CLEAR1, and PHI2L, which are generated in less crowded regions of the chip (ie., outside the actual RESST array).

4.2. Power Estimates

An important aspect of any real VLSI design is that its power consumption must not exceed a level which can be dissipated safely on a chip. With today's technology, the power limitation for a chip the size of our project is around 1 watt. We used Mextra and Powest [Ber82] to extract and estimate power consumption for our various cells. According to Powest, our circuitry (array plus clocked signal logic), which occupies an area of 3358 microns by 4466 microns, requires a worst-case DC power of 0.182 watts. Similarly, our pads, laid out around a 7000 micron by 7000 micron square chip perimeter, require a worst-case DC power of 0.633 watts. Thus, the worst-case DC power consumption of a

RESST chip should be about 0.815 watts. Fortunately, this is within the allowable power consumption range.

5. Other Enhancements

In reflecting on our work, several possible enhancements come to mind. First, rather than have separate CE and RP cell types, we could have stored record-pointer bits in CE cells. If they were stored in the least significant bit positions, they would appear as "insignificant bits" in sorting, coming into play only with duplicate keys. The advantages of doing this would be one less cell type and variability in the boundary between where keys end and record-pointers begin. The disadvantages would be loss of the opportunity for RP cell height reduction, increased EXCHIN/EXCHOUT signal delays, and somewhat increased overall power consumption.

Another possible enhancement involves the external RESST control circuitry. An intelligent RESST controller could monitor the FULL pin on the leftmost RESST chip in a cascaded collection, and vary the clock cycle speed based on whether or not the leftmost chip is full. That is, when PUSHes and POPs can't cause data to shift from chip to chip, the phase 1 width can be smaller by 70 - 8 = 62 nanoseconds. (It isn't clear that the added controller complexity would be worthwhile given the small fractional speedup, though.)

6. Conclusions

This report discussed a VLSI implementation of a record-sorting stack. The implementation allows the sorting of n records, represented as (key, record-pointer) pairs, to be accomplished in $O(n)$ time. The design is cascadable so that the capacity of a single VLSI chip does not limit the amount of data which may be sorted.

The algorithm, a parallel version of the classic bubblesort algorithm, was described, the overall chip organization and data flow were presented, and detailed circuits, layouts, and timing analyses were given. It was shown that a RESST implementation can perform at disk transfer rates, making feasible its use as an enhancement to a database machine.

References

[Arm82] Armstrong, P., and Rem, M., "A Serial Sorting Machine", Computers and Electrical Engineering, Vol. 9, No. 1, Permagon Press, March 1982.

[Ber82] "Berkeley VLSI Tools", R. Mayo (ed.), Computer Science Division, University of California, Berkeley, 1982.

[Chu80] Chung, K., Luccio, F., and Wong, C., "On the Complexity of Sorting in Magnetic Bubble Memory Systems", IEEE Transactions on Computers, Vol. C-29, No. 7, July 1980.

[Lei79] Leiserson, C., "Systolic Priority Queues", CMU Technical Report No. CMU-CS-79-115, Department of Computer Science, Carnegie-Mellon University, 1979.

[Mea80] Mead, C., and Conway, L., "Introduction to VLSI Systems", Addison-Wesley Publishing Company, 1980.

[Mir82] Miranker, G., Tang, L., and Wong, C., "A 'Zero-Time' VLSI Sorter", IBM Research Report, IBM Thomas J. Watson Research Center, Yorktown Heights, NY, 1982.

[Muk72] Mukhopadhyay, A., and Ichikawa, T., "An n-Step Parallel Sorting Machine", Technical Report No. 72-03, University of Iowa, Iowa City, 1972.

[Muk81] Mukhopadhyay, A., "WEAVESORT - A New Sorting Algorithm for VLSI", Technical Report No. TR-53-81, University of Central Florida, Orlando, 1981.

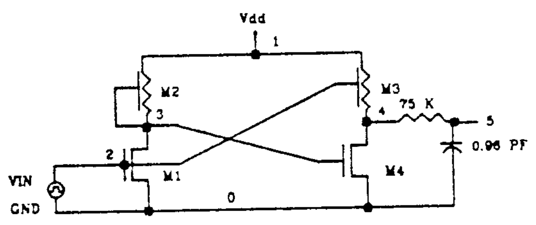
[Tho82] Thompson, C., "The VLSI Complexity of Sorting", ERL Memo No. UCB/ERL M82/5, University of California, Berkeley, 1982.

[Ull80] Ullman, J., "Principles of Database Systems", Computer Science Press, 1980.

Appendix

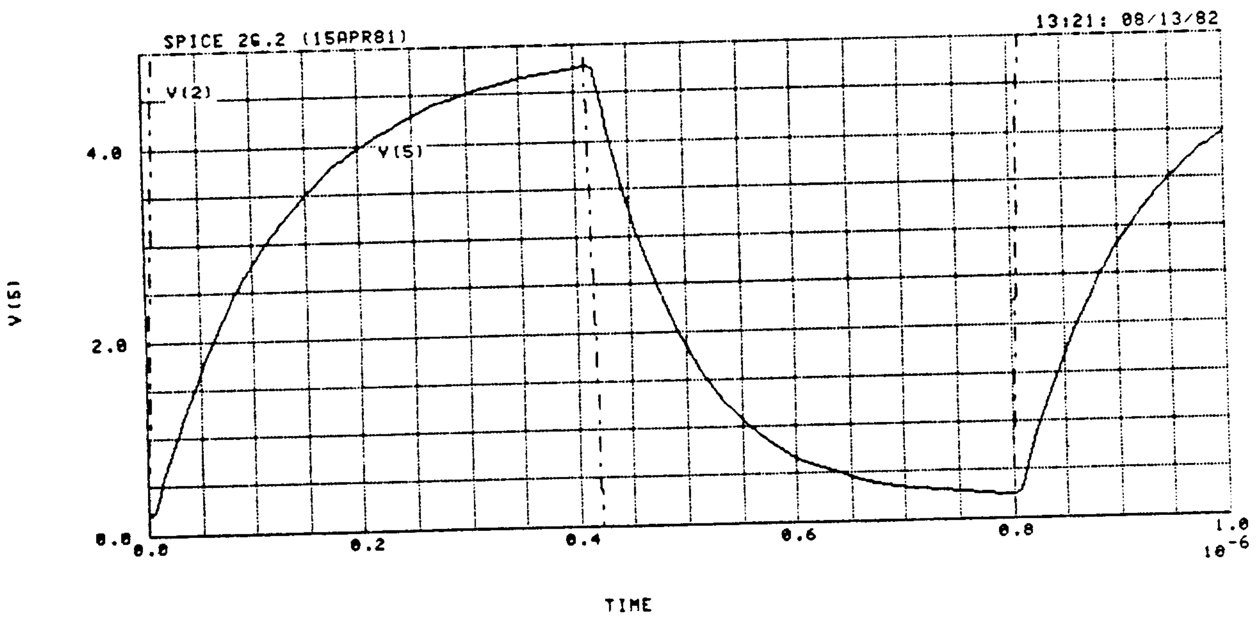
SPICE #1: Phase 1 Signals

```
1********08/13/82 ********  SPICE 2G.2 (15APR81)   ********13:21:09*****
0  SPICE *1 -- PHASE 1 SIGNALS
0****      INPUT LISTING                     TEMPERATURE =   27.000 DEG C
0***********************************************************************


 ** THE FOLLOWING LINES DEFINE THE NMOS MODELS AND VDD.

 .MODEL E NMOS(VTO=.9 NSUB=2E15 TOX=.1U UO=800 XQC=.5
  +LEVEL=2 VMAX=1E5 CGSO=245P CGDO=245P)
 .MODEL D NMOS(VTO=-3.2 NSUB=2E15 TOX=.1U UO=800 XQC=.5
  +LEVEL=2 VMAX=1E5 CGSO=245P CGDO=245P)
 .OPTIONS NOMOD
 .WIDTH OUT=80
 VDD 1 0 5V

 ** THE FOLLOWING LINES DEFINE THE CIRCUIT.
 M1 3 2 0 0 E L=4U W=4U
 M2 1 3 3 0 D L=16U W=4U
 M3 4 3 0 0 E L=4U W=4U
 M4 1 2 4 0 D L=16U W=4U
 RPAR 4 5 75KOHMS
 CPAR 5 0 .96PFARADS

 VIN 2 0 PULSE (0V 5V 0NS 0NS 0NS 400NS 800NS)
 .TRAN 10NS 1000NS
 .PLOT TRAN V(5) V(2) (0V,5V)
 .END
```



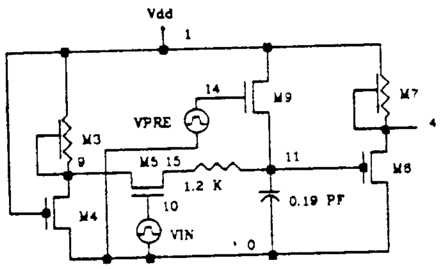
SPICE #1: Phase 1 Signals

SPICE #1 -- PHASE 1 SIGNALS

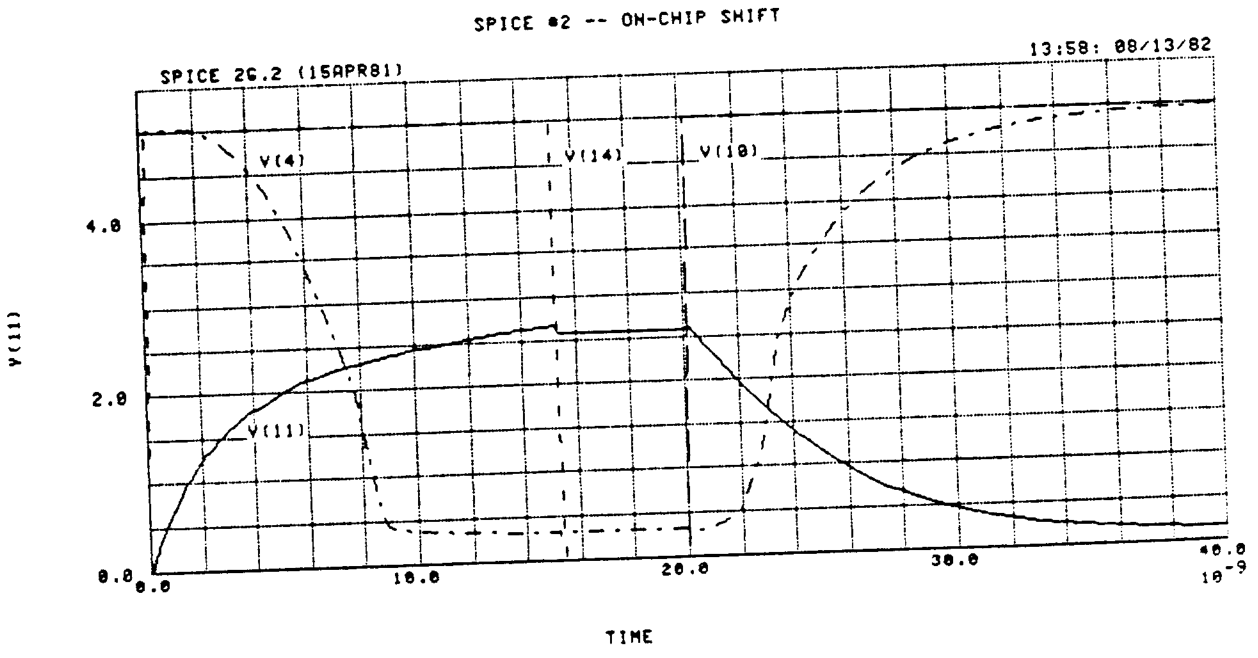


SPICE #2: On-Chip Shift

```
1********08/13/82 ********  SPICE 2G.2 (15APR81)  ********13:58:52*****
0   SPICE *2 -- ON-CHIP SHIFT
0****     INPUT LISTING                        TEMPERATURE =   27.000 DEG C
0**********************************************************************


 ** THE FOLLOWING LINES DEFINE THE NMOS MODELS AND VDD.

 .MODEL E NMOS(VTO=.9 NSUB=2E15 TOX=.1U UO=800 XQC=.5
  +LEVEL=2 VMAX=1E5 CGSO=245P CGDO=245P)
 .MODEL D NMOS(VTO=-3.2 NSUB=2E15 TOX=.1U UO=800 XQC=.5
  +LEVEL=2 VMAX=1E5 CGSO=245P CGDO=245P)
 .OPTIONS NOMOD
 .WIDTH OUT=80
 VDD 1 0 5V

 ** THE FOLLOWING LINES DEFINE THE CIRCUIT.
 M3 1 9 9 0 D L=32U W=4U
 M4 9 1 0 0 E L=4U W=4U
 M5 9 10 15 0 E L=4U W=4U
 M6 4 11 0 0 E L=4U W=4U
 M7 1 4 4 0 D L=32U W=4U
 M9 1 14 11 0 E L=4U W=4U
 CPAR 11 0 .192PFARADS
 RPAR 15 11 1.2KOHMS

 VIN 10 0 PULSE (0V 5V 20NS 0NS 0NS 100NS 100NS)
 VPRE 14 0 PULSE (0V 5V 0NS 0NS 0NS 15NS 100NS)
 .TRAN .2NS 40NS
 .PLOT TRAN V(11) V(4) V(10) V(14) (0V,5V)
 .END
```

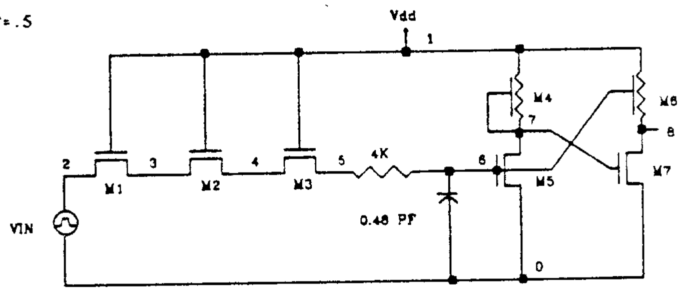


SPICE #2: On-Chip Shift



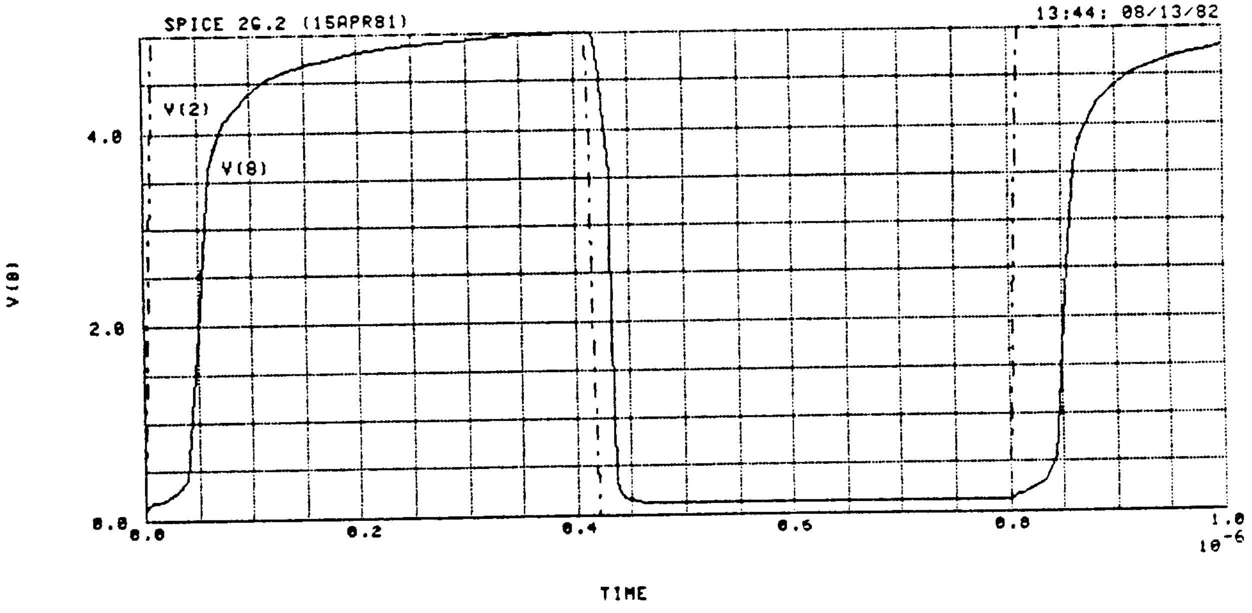
SPICE #3: EXCHIN/EXCHOUT Ripple

```
1*******08/13/82 ********  SPICE 2G.2 (15APR81)  ********13:44:46*****

0   SPICE #3 -- EXCHIN/EXCHOUT RIPPLE

0****      INPUT LISTING                     TEMPERATURE =   27.000 DEG C

0*******************************************************************************


 ** THE FOLLOWING LINES DEFINE THE NMOS MODELS AND VDD.

 .MODEL E NMOS(VTO=.9 NSUB=2E15 TOX=.1U UO=800 XQC=.5
  +LEVEL=2 VMAX=1E5 CGSO=245P CGDO=245P)
 .MODEL D NMOS(VTO=-3.2 NSUB=2E15 TOX=.1U UO=800 XQC=.5
  +LEVEL=2 VMAX=1E5 CGSO=245P CGDO=245P)
 .OPTIONS NOMOD
 .WIDTH OUT=80
 VDD 1 0 5V

 ** THE FOLLOWING LINES DEFINE THE CIRCUIT.
 M1 2 1 3 0 E L=4U W=4U
 M2 3 1 4 0 E L=4U W=4U
 M3 4 1 5 0 E L=4U W=4U
 M4 1 7 7 0 D L=32U W=4U
 M5 7 6 0 0 E L=4U W=4U
 M6 1 6 8 0 D L=32U W=4U
 M7 8 7 0 0 E L=4U W=4U
 RPAR 5 6 4.0KOHMS
 CPAR 6 0 0.4800PFARADS IC=0.0V

 VIN 2 0 PULSE (0V 5V 0NS 0NS 0NS 400NS 800NS)
 .TRAN 10NS 1000NS
 .PLOT TRAN V(8) V(2) (0V,5V)
 .END
```



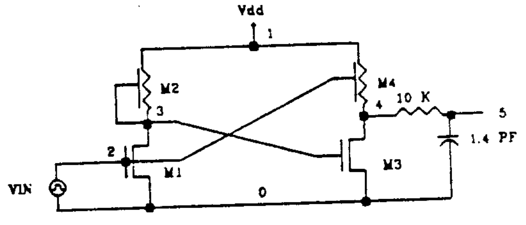
SPICE #3: EXCHIN/EXCHOUT Ripple

SPICE #3 -- EXCHIN/EXCHOUT RIPPLE



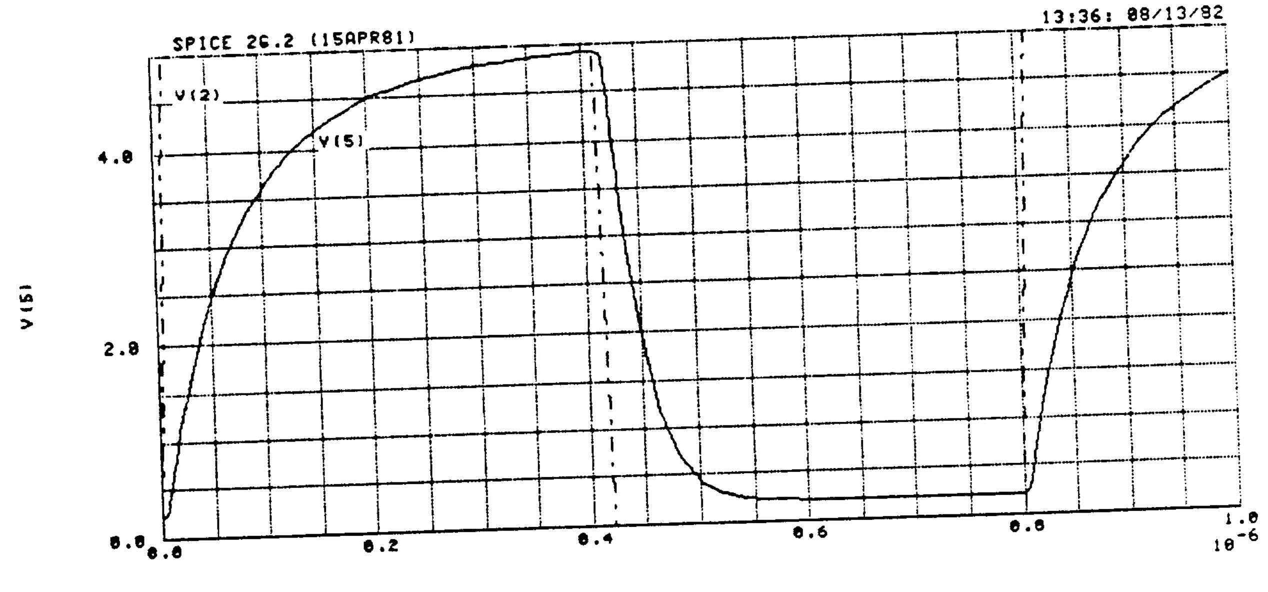
SPICE #4: Phase 2 Signals

```
1*******08/13/82 ********  SPICE 2G.2 (15APR81)  ********13:36:33*****
0  SPICE #4 -- PHASE 2 SIGNALS
0****      INPUT LISTING                       TEMPERATURE =   27.000 DEG C
0*********************************************************************


** THE FOLLOWING LINES DEFINE THE NMOS MODELS AND VDD.

.MODEL E NMOS(VTO=.9 NSUB=2E15 TOX=.1U UO=800 XQC=.5
 +LEVEL=2 VMAX=1E5 CGSO=245P CGDO=245P)
.MODEL D NMOS(VTO=-3.2 NSUB=2E15 TOX=.1U UO=800 XQC=.5
 +LEVEL=2 VMAX=1E5 CGSO=245P CGDO=245P)
.OPTIONS NOMOD
.WIDTH OUT=80
VDD 1 0 5V

** THE FOLLOWING LINES DEFINE THE CIRCUIT.
M1 3 2 0 0 E L=4U W=4U
M2 1 3 3 0 D L=16U W=4U
M3 4 3 0 0 E L=4U W=4U
M4 1 2 4 0 D L=16U W=4U
RPAR 4 5 10KOHMS
CPAR 5 0 1.4PFARADS

VIN 2 0 PULSE (0V 5V 0NS 0NS 0NS 400NS 800NS)
.TRAN 10NS 1000NS
.PLOT TRAN V(5) V(2) (0V,5V)
.END
```



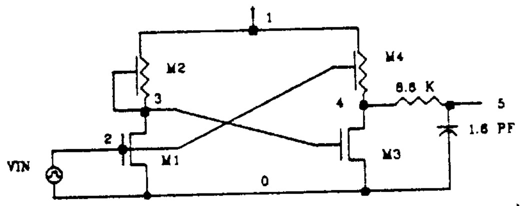
SPICE #4: Phase 2 Signals

SPICE #4 -- PHASE 2 SIGNALS



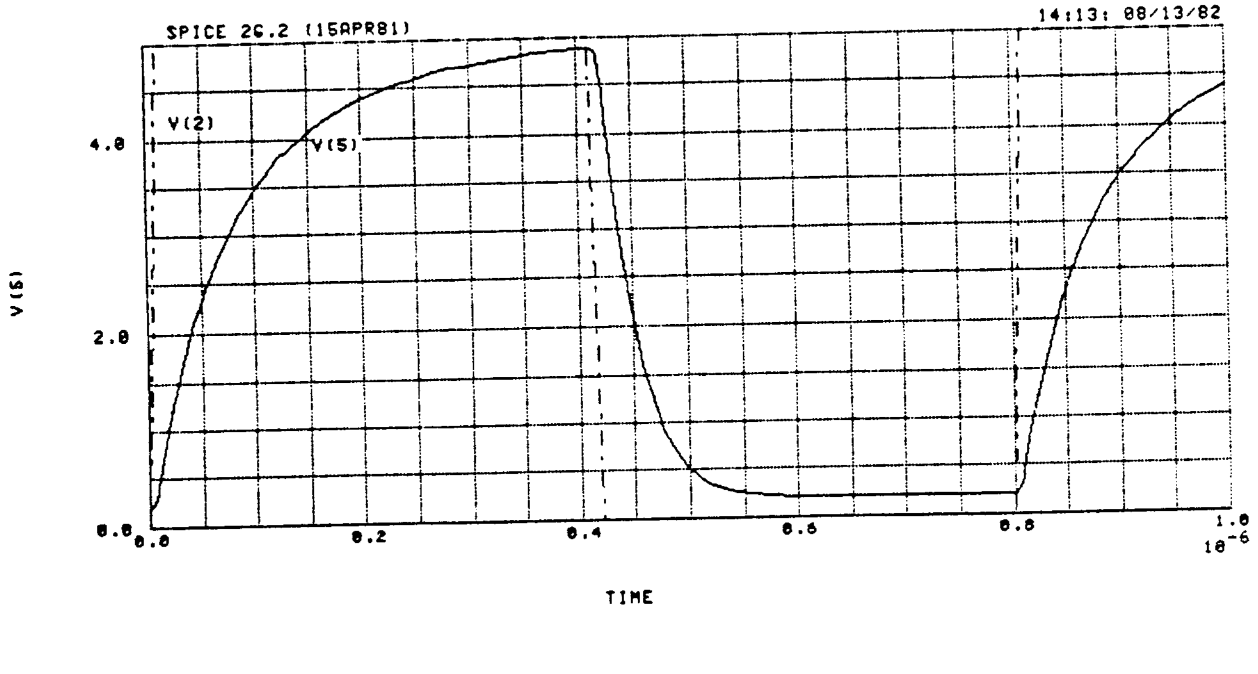
SPICE #5: EXCH Signal Return

```
1********08/13/82 ********  SPICE 2G.2 (15APR81)  ********14:13:26*****

0  SPICE #5 -- EXCH SIGNAL RETURN

0****      INPUT LISTING                    TEMPERATURE =   27.000 DEG C

0*******************************************************************


 ** THE FOLLOWING LINES DEFINE THE NMOS MODELS AND VDD.

 .MODEL E NMOS(VTO=.9 NSUB=2E15 TOX=.1U UO=800 XQC=.5
  +LEVEL=2 VMAX=1E5 CGSO=245P CGDO=245P)
 .MODEL D NMOS(VTO=-3.2 NSUB=2E15 TOX=.1U UO=800 XQC=.5
  +LEVEL=2 VMAX=1E5 CGSO=245P CGDO=245P)
 .OPTIONS NOMOD
 .WIDTH OUT=80
 VDD 1 0 5V

 ** THE FOLLOWING LINES DEFINE THE CIRCUIT.
 M1 3 2 0 0 E L=4U W=4U
 M2 1 3 3 0 D L=16U W=4U
 M3 4 3 0 0 E L=4U W=4U
 M4 1 2 4 0 D L=16U W=4U
 RPAR 4 5 8.6KOHMS
 CPAR 5 0 1.6PFARADS

 VIN 2 0 PULSE (0V 5V 0NS 0NS 0NS 400NS 800NS)
 .TRAN 10NS 1000NS
 .PLOT TRAN V(5) V(2) (0V,5V)
 .END
```



SPICE #5: EXCH Signal Return

SPICE #5 -- EXCH SIGNAL RETURN


SPICE 2G.2 (15APR81)
14:13: 08/13/82
V(2)
V(5)
V(5)
10^-6
TIME