Bit-serial architecture for optical computing

Vincent P. Heuring, Harry F. Jordan, and Jonathan P. Pratt

The design of a complete, stored-program digital optical computer is described. A fully functional, proof-of-principle prototype can be achieved by using LiNbO$_3$ directional couplers as logic elements and fiber-optic delay lines as memory elements. The key design issues are computation in a realm where propagation delays are much greater than logic delays and implementation of circuits without flip-flops. The techniques developed to address these issues yield architectures that do not change as their clocking speed is scaled upward and the size is scaled downward proportionally; these are called speed-scalable architectures. Signal amplitude restoration and resynchronization are accomplished by the novel technique of switching in a fresh copy of the system clock. Device characteristics that are important to the proof-of-principle demonstration are discussed, including the special properties and limitations that are important when designing with them. Design principles are exemplified by the design of an $n$-bit counter. Following this, the design for a stored-program bit-serial computer is described. We estimate that the described prototype architecture can be operated in the 100-MHz region with off-the-shelf components, and in the 0.1-1-THz region with foreseeable future components.

I. Introduction

We describe an architecture for a realizable, stored-program, general-purpose digital computer constructed from optical components. There have been a number of proposals for optical computer architectures in the literature, but most are either architectures for special-purpose machines, or they rely on devices that are in the research category. A key feature of this machine is that it incorporates a stored program that it can manipulate as data, which thus supports the compiler and high-level language hierarchy that forms the basis of computer science. For a computer architect, the stored program, not the arithmetic capability, distinguishes a computer from other digital systems. While a powerful, high-speed arithmetic unit may be an extremely valuable device, the ability to store and manipulate machine instructions as data is the foundation of the many levels of hardware and software complexities that characterize general-purpose computing. From past experience, quantitative advances in storage capacity and access speed, arithmetic capability, etc., follow the qualitative advance to self-contained, stored-program operation.

Our objective is to design a proof-of-principle machine that can be constructed from commercially available components. Such a design permits computer architects to become involved in optical computer design earlier than would be possible if they were to wait for the arrival of a more familiar-looking optical technology such as integrated optics. It thus puts the architect in a position to influence the design of future optical computing components. As for practical application, a bit-serial design such as the one proposed here should have immediate application in fiber-optic communications systems that already operate in the serial mode.

Given the goal of designing a realizable stored-program computer from commercially available components, we faced some of the same constraints that were faced by early architects of electronic computers: a limitation on the number and kinds of devices that could be used in a realizable implementation and the consequent need to implement memory with less expensive components. (Here we use the term expensive in its larger sense, i.e., the cost in terms of price, size, power dissipation, etc.) Practical electronic computers have been constructed with as few as 15 flip-flops and a diode array, with a rotating magnetic drum for memory. Circulating mercury delay lines were also employed. These early machines generally used a bit-serial approach, since, in

---

When this work was performed, the authors were with the Department of Electrical and Computer Engineering, Center for Optoelectronic Computing Systems, University of Colorado, Boulder, Colorado 80309-0425. J. P. Pratt is now with the University of Colorado Health Science Center, Department of Radiology, Box A0344, 200 East Ninth Avenue, Denver, Colorado 80262. Received 13 September 1991. 0003-6935/92/173213-12$05.00/0. © 1992 Optical Society of America.
general, a bit-serial design that uses $M$ devices for a certain functionality, for example, an adder, will require $M^*N$ devices to implement the functionality in parallel for word size $N$. Huang mentions the lack of cascadable logic gates and optical memory devices as two of the chief bottlenecks in building an optical computer. Our studies have shown that the LiNbO$_3$ directional coupler will serve as a logic device, and a glass fiber loop will serve as a delay-line memory. Subsections I.A and I.B describe these devices as used as components in an optical computer. Subsections I.C–I.E address some of the design issues inherent in architectures that are constrained by the speed of light, or more precisely, architectures that are confined to the surface of a relativistic light cone.

Section II describes the implementation of a counter as an example of a design that uses these components. Section III discusses the actual computer design. Finally, in Section IV we discuss the probable operating speed of the computer with present day components, and make estimates of possible future operating speeds.

A. LiNbO$_3$ Switch as a Logic Element

The LiNbO$_3$ electro-optic switch is a guided wave device that is produced by diffusing precisely placed Ti waveguides into high-purity LiNbO$_3$ crystals, as shown schematically in Fig.1(a). In the absence of an externally applied voltage $V_{AC}$, but with the correct bias voltages $V_{DC1}$ and $V_{DC2}$, light entering at point A will couple to the neighboring waveguide and emerge at E. Likewise, light entering at B will emerge at D. This is referred to as the cross state. Application of the correct voltage at $V_{AC}$ will cause the light entering at A to emerge at D and light entering at B to emerge at E. This is referred to as the bar state. Typical commercially available devices are several centimeters in length and are terminated in fiber pigtails for system interconnection. As systems designers, we are most interested in system level characteristics, rather than device physics, such as the distinction between AB and Mach–Zehnder switching. System specifications for a commercial switch are: $\lambda = 1500$ nm; switching speed 0.3 ns; switching voltage $V_{AC} = 4.5$ V; bias voltages $V_{DC1}$ and $V_{DC2} = 5$ V; on-off cross talk typically $> 25$ dB; and insertion loss $< 5$ dB. The switching is nonlinear with a functionality that depends on electrode configuration and geometry. It ranges from sinusoidal, $I = [1 - \cos(\pi V_{AC}/V_{max})]/2$, to a plateau function, in which $I$ is relatively insensitive to the value of $V_{AC} > V_{max}$. Even the sinusoidal characteristic provides relative insensitivity to $V_{AC}$ in the region of saturation, since $\partial I/\partial V_{AC}V_{max} \approx 0$. In our application $V_{AC}$ is generated by a photodiode–amplifier combination, thereby converting the switch to a five-terminal optical device, with optical terminals labeled A through E, as shown in Fig.1(b). In addition the amplifier was designed so that it saturates at $V_{max}$ at a relatively low-input light intensity. The cost of such a device, with its associated drive electronics, is $\sim $2000. (Although it might be argued that this is an optoelectronic device rather than an all-optical one, we take the view that, since the electronics are limited to one part of the device, the device can be considered as all-optical.) We do take advantage of the electronics to shape the incoming pulse in the manner described in Ref. 12.

From the point of view of a computer architect, the device can be modeled as two interconnected $1 \times 2$ multiplexers; its functionality and symbolism are described in Fig. 2. Figure 3 demonstrates the logic functions that can be synthesized from the directional coupler in a clocked system. As Fig. 3 shows, the switch has considerable utility as a design component and is, in fact, a complete logic element. This is fortunate, because there currently is a paucity of off-the-shelf optical components that are suitable for use in computer design. Because the use of a single component in so many different ways complicates the understanding of the circuit designs that employ it,
we take greater than usual pains in describing the circuits below. Figure 3 also shows two additional optical circuit elements, which consist of fixed 3-dB directional couplers. When used as a splitter, they permit fan-out of the optical signal. They can also serve as the optical equivalent of the wired OR gate. The splitter is essential to the design as the only means of fan-out. The wired OR can be replaced by a true OR gate made from an active switch when interference between coherent input signals is a concern.

B. Optical Delay-Line Memory

The optical delay-line memory is simply a loop of optical fiber whose length is \( l = nl_c \), where \( l \) is the loop length, \( n \) is the number of bits stored in the loop, and \( l_c \) is the distance traveled by the optical pulse during one clock period. Furthermore, \( l_c = \gamma v \), where \( \gamma \) is the clock period and \( v \) is the speed of light in the medium. \( \gamma \) is related to the speed of light in vacuo, \( c \), by \( v = c/\gamma \), where \( \gamma \) is the refractive index in the medium. In terms of the clock frequency \( \nu \), \( v = 1/\tau \), so \( l = nc/\gamma \). For glass fibers in which \( \gamma \approx 1.5 \), 1024 16-bit words are stored in a loop 3.3 m long at a clock frequency of 1 THz = 10^{12} Hz, and in a loop 33 km long at a clock frequency of 100 MHz. There is an upper bound to the number of bits that can be stored in such a loop before the thermal coefficients of the length and the refractive index of the fiber cause significant changes in the optical length of the fiber unless these effects are compensated for. A typical glass fiber has a linear thermal-expansion coefficient of \( \sim 1 \times 10^{-6} \) C^{-1}, and a thermal coefficient with a refractive index ranging from -10 to +19 \times 10^{-6} \text{ C}^{-1}, with values of +10 being typical of fused silica. Thus an uncompensated fiber loop has an upper bound on the number of bits that can be stored of \( \sim 10^6 \). This bound can be increased considerably by one or more of the following means: (a) careful temperature control, (b) encoding the clock signal in the data being circulated in the loop, (c) controlling the properties of the fiber so that length changes caused by thermal effects are compensated for by refractive index changes, and (d) measuring the drift and compensating for it by adjustments in the clock frequency. Changes in optical length may also be corrected by clocking the system with a clock signal that has been subjected to the same temporal perturbation as the signals in the memory loop. These matters are considered in more detail in Ref. 13.

C. Synchronization and Amplitude Restoration in the Delay-Line Memory

All passive devices exhibit loss. In commercial fibers, this loss may range from 0.5 to 10 dB/km. LiNbO_3 component insertion loss may range up to 5 dB. Thus some means of amplitude restoration must be provided. Furthermore a means must be provided to insert and retrieve information in the loop. The circuit of Fig. 4 provides both synchronization and amplitude restoration by interrupting the loop at switch S1 to permit the bit stream to refresh itself by switching a copy of the clock signal at A into the loop for every bit stored. The circled \( \Delta \) with the N beneath it represents a loop of N-bit length; splitter S2 permits sampling of the signal. Initializing data are written into the loop at S1B. The total loss around the loop is \( \sim 5 \) dB at S1 and 3 dB at S2, for a total loss of 8 dB. These losses are compensated for by the electronic gain at control terminal S1C. This technique of inserting a copy of the clock signal to restore amplitude and timing is extremely important to the entire design. The more usual technique employed in optical systems, that of using optical amplification, suffers from the disadvantage that cross talk and accumulated noise are also amplified. In addition, amplification does nothing to resynchronize the system. The technique is similar to clock gating used in some electronic systems for synchronization. The use of this technique to restore optical power level is novel.

The computation of power budgeting, loss estimation, and cross-talk influence is important in a design employing dozens or hundreds of components. We have developed algorithms for computing power loss and accumulated cross talk in complex optical systems such as the one described here.14 By using these algorithms, we find that the number of switches in the architecture described below must be increased...
from approximately 48 to 76 to restore power to adequate levels, assuming the cross talk and loss figures quoted above.

D. Clock

Clocking is provided by a 1300-nm laser that is modulated at the desired clock frequency. The duty cycle should be adjusted to <50%. The electronics at the control terminal of the LiNbO₃ switch will stretch the pulse somewhat. This is because the switch should be in the desired state before the arrival of an input pulse and should stay in the desired state until after the pulse has left the device. This prevents the occurrence of glitches and runt pulses. An apt analogy can be drawn with a railroad track switch, which also must be in the correct state before the arrival of the train and which must stay in the correct state until after the train has crossed the switch. A given device and clock speed will have a particular combination of duty cycle and delay that are optimum. The principal factors to be considered when determining duty cycle and delay are rise and fall times, propagation time through the electronics at terminal C, and pulse jitter. The latter becomes significant at clock rates of >100 MHz, at which detectors begin to exhibit data-dependent delays. Measurements of these parameters on prototype drive electronic modules show rise and fall times of <1 ns and end-to-end propagation times of ~4 ns (see Ref. 15).

E. Effect of Delay on Design

A key feature of speed-scalable architectures is that the design technique employed does not use flip-flops or latches for synchronization. Rather, temporal synchronization is included as a parameter during the design process. Fiber lengths are calculated so that pulses arrive at key places in the circuit (i.e., switches, splitters, and combiners) in synchronization. Although the lack of flip-flops may seem arbitrary or unusual to those accustomed to electronic design, it has a firm basis. It must be remembered that active flip-flop designs incorporate positive feedback loops, such as cross-coupled NAND gates. Information propagates around these loops no faster than the speed of light. The idea of static storage rests on the assumption that the propagation speed is much faster than that at which changes are made to the flip-flop's state. It is precisely this assumption that should be reinterpreted in an optical architecture. The representation of the memory function entirely in terms of recirculating loops is therefore justified on fundamental grounds.

Delays in fibers and devices can be computed from the dimensions and the refractive index of the material. Delays in the detector-amplifier circuits of the switch must be measured. In practice, an optical length is computed for each fiber of the device, including the control fiber, and is included in the design calculation. Delays for the control input include both the actual optical delay and the delay introduced by the drive electronics and detector. We find it expedient to do the first design iteration assuming no delays anywhere except where they are desired. The effect of delays is then taken into account when calculating fiber interconnect lengths by using an algorithm that represents the design as a directed graph.₁⁶

We have incorporated this algorithm and the one described above that calculates the effects of power loss and cross talk on system operation with a graphically oriented optical computer-aided design system that runs on the Apple Macintosh computer. Copies of the software are available upon request.

For any given set of components there is an upper bound on the clock speed. This upper bound may be fixed by clock rise and fall time, control electronics rise and fall time, device rise and fall time, or minimum delay. The minimum delay through a circuit becomes important when designing circuits with feedback, such as carry loops. In a design with such a loop, i.e., loop C in Fig. 5(a), the total delay around the loop must be exactly equal to the clock period 𝜏, since a bit in the delay loop must travel the loop and arrive back at S1 just in time to switch the next incoming pulse. Therefore the calculated delay-loop length must be greater than or equal to the optical lengths of the fibers. If the loop length is greater, then fiber must be added to the loop to make the length equal. Notice that the optical length of the path through S1 is greater than the path length through S2 because of the additional delay in the former that is due to the control electronics at terminal C.

![Fig. 5. (a) A 4-bit binary counter. (b) Simulation of the operation of the counter.](image-url)
This requirement that the carry-loop timing be such that the carry bit recirculates in exactly sufficient time to arrive at the switch just as the next bit arrives at the switch is analogous to carry propagation in a ripple-carry adder. An even more apt analogy is to the carry flip-flop in a serial adder. It may be noted in this connection that for arithmetic computations that take place on the surface of a light cone, as optical computations do, serial and parallel additions take exactly the same time to perform because of carry propagation and therefore, in this case, parallel computation offers no advantage over serial computation. This is not to say that there are no solutions to the carry propagation problem but rather that the speed of light offers fundamental limitations to the extent of parallelism that can be achieved. On the other hand optical computer designs such as the one described here are inherently pipelined, and thus can exploit all the advantages that pipelined architectures afford. In addition, where information does not interact, one can certainly parallelize large sections of such an architecture. For example the memory loop can be duplicated so as to provide access to an entire machine word simultaneously. In light of the discussion of carry propagation above, any fraction of the word parallelism can be moved between the time and space domains.

We have seen that creation of a serial memory loop requires only a single optical switch and delay line. However, the contents of such a loop should be readily alterable in order for the loop to be useful in computer design. The simple loop only allows bits to be cycled in by means of the terminal adjacent to the clock input. A more general memory loop is illustrated in Fig. 6. A second switch is added to provide for the insertion of new data into the loop. When the control input of switch 2 is not asserted, the resulting cross state of the switch completes the simple delay loop. However, when the control input is asserted new data from terminal B are switched into the loop. The timing of the new data must match that of the old in order for regeneration to continue.

II. Serial N-Bit Counter

Figure 5 shows the design and simulation of a 4-bit counter. Input pulses to be counted are expected to occur synchronously with the first pulse of each group of 4 clock pulses. Arrival of an input pulse switches a copy of the arriving clock pulse to fiber E of S1. This pulse is split at S4 and sent to S2 and S3. Accumulated bits circulate in loop M, which is a memory loop similar to that discussed above. An N-bit counter will have a delay loop containing N bits so, for this 4-bit counter, loop M has a delay of 4 Δ. Notice that if no bit is arriving at S3C from loop M when the input bit arrives at S3A the arriving bit is switched into loop M. However, if a bit is arriving at S3C from loop M while an input bit is arriving at S3A then the bit from M is not copied back into M, and the input bit at S2A is switched back into the carry loop, loop C, where it will emerge again one clock period later, and the process will repeat as shown in Fig.5(b). In the absence of input or carry bits the bit pattern at M will continue to circulate in the manner described in Subsection I.B. When the loop is filled with 1's, then the next arriving bit will overflow the loop, resetting it to all 0's, and the process will repeat. We describe the construction and operation of 50-MHz and 100-MHz counters in Refs. 18 and 19.

Section III shows how fiber delay loops, LiNbO₃ switches, splitters, and combiners are used in the design of a simple bit-serial computer.

III. SCAMP: A Simple Bit-Serial Optical Computer

A. General Description

The design goal of the SCAMP (Serial Computer Assuming Multiplexers Perfect; an early hedge introduced by the principal designer) is to create a stored-program optical computer with a general-purpose instruction set that uses few enough components to be realizable at the current state of optical technology. The architecture operates serially to reduce the component count to a surprisingly low number. Only ~50 optical switches are needed in the ideal design. The minimality of the design is underscored when it is realized that every simple logic function combining signals requires a switch, with the minor exception of the wired OR mentioned above. On the other hand, data storage is inexpensive, since it consists of an optical fiber with one switch for regeneration and one for writing into it. The switch count is independent of the size of the storage.

Five processor data structures are implemented with optical delay loops:

- Main memory (MEM), \(2^N\) words;
- Memory counter (MC), \(\text{WordSize}\) bits;
- Accumulator (ACC), \(\text{WordSize}\) bits;
- Instruction register (IR), \(\text{WordSize}\) bits;
- Program counter (PC), \(\text{WordSize}\) bits.

An interesting aspect of this serial design is that the word size may be changed without architectural modifications. Hence, wherever possible, the word size of the computer is generalized to \(\text{WordSize}\) bits. Since the SCAMP is a single-address machine, \(N\) bits of each instruction contain a memory address and 6 bits are used for control. This leads to a minor restriction on the word length:

\[
\text{WordSize} \geq N + 6 \text{ bits.}
\]

The data paths between the memory structures of the SCAMP are illustrated in Fig. 7. Arrows indicate the possible directions of transmission. The arith-
metic and logic unit (ALU) serves as a transfer point, routing the signals according to the control bits of the current instruction. In Fig. 8 all the connections between the SCAMP units are shown.

B. Instruction Set Processor Description of the SCAMP

An instruction set processor (ISP) description of the SCAMP is presented in Fig. 9. The registers and flags are defined under the heading Processor State. The main memory is defined under the heading Memory State. The 16-bit word of this example allows for a 10-bit address and thus, 1024 memory words. The fields of the instruction word are defined under the heading Instruction Format, and the sequence of events that produces instruction execution is described in the instruction interpretation section. At the bottom of Fig. 9 the nine instructions are named, and their effects are described.

C. Main Memory

Some discussion of memory access is required before the overall design is discussed. When a large quantity of data is circulating in a delay loop, there must be a means accessing a specific element or word. One possibility is to embed location information in the circulating data, a technique that is often applied to rotating magnetic media. Another option is to keep track of the currently accessible element through precise external timing. The latter method was chosen for the SCAMP because of its simplicity. The precise timing does not present a problem since the clock that regenerates the memory loop is the same one that drives the counting circuits.

Thus the SCAMP's memory consists of a $2^N$ word memory loop that is timed by an $N$-bit counter. The circulation of one group of WordSize bits in the memory loop corresponds to an increment in the counter. This memory counter is used to indicate which memory word is currently available at a given tap in the loop. Obtaining a word at a given address becomes a matter of waiting for the memory counter to equal that address. In the SCAMP the address comparator unit indicates when there is a match between the memory count and the desired address.

D. Instruction Execution

The SCAMP goes through four phases to execute each instruction: instruction search, instruction fetch, operand search, and instruction execution. During the instruction-search phase a match is sought between the program counter and the memory counter; the address fields of the MC and PC signals are compared until they are equal. A match is indicated by a pulse on the memory-found, or MEMF, signal line. When asserted during an instruction search, MEMF causes the current memory word to be fetched and sent to the instruction register.

After an instruction has been fetched the operand-search phase begins. This is similar to the instruction-search phase except that the address field of the instruction register rather than the program counter is compared with the memory counter address field. The operand/instruction (OPR/INST) signal is used to differentiate these two states. OPR/INST emanates from a circuit that is toggled, after a needed delay, by the MEMF pulse. When MEMF is asserted during the operand-search phase, instruction execution is initiated.

At the beginning of the instruction-execution phase the six control bits of the instruction are extracted and synchronized. Three of these bits are used to select a source signal, one bit is used to select a destination, and two bits are used to control the program counter. Table I summarizes the uses of these bits. The operand is fetched from memory (the MEM signal) and combined with the accumulator (the ACC signal) to produce eight source arithmetic signals. The three source bits and one destination bit are used to choose one of the ALU signals and send it to either the accumulator or the memory. In the latter case the original operand is replaced by the ALU result.

The methods of PC control are enumerated in Table I. For sequential operation the PC is incre-
Registers and flags are defined.

IR<15:0> The instruction register.
ACC<15:0> The accumulator.
PC<15:0> The program counter.
PCad<9:0> := PC<9:0> Address field of program counter.
result<15:0> Temporary ALU result state.
carryflag Temporary carry result state.
zeroflag Temporary zero result state.

Memory State
M[0:1023]<15:0> The main memory, 1024 sixteen bit words.

Instruction Format
op<5:0> := IR<15:10> The operation code of the instruction.
source<2:0> := op<5:3> Source calculation field.
dest := op<2> Destination bit.
PCcon<1:0> := op<5:4> Specifies PC control: branching, skipping, etc.
address<9:0> := IR<9:0> Address field of instruction.

Instruction Interpretation
Instruction cycle sequence is defined.

Fetch instruction, increment PC.
Do computation.
Addition determines carryflag state.
Result determines zeroflag state.
Destination is accumulator.
Destination is memory.
Skip on result not zero.
Skip on carry.

Instruction Set
Effects and mnemonics of instructions.

Clr (:= source = 0) → (result ← 0); Clear - set to zero.
Not (:= source = 1) → (result ← ¬M[address]); Invert Memory.
Or (:= source = 2) → (result ← ACC ∨ M[address]); OR Memory and Accumulator.
MoveA (:= source = 3) → (result ← ACC); Move Accumulator.
RoR (:= source = 4) → (result ← ACC <0> ACC <15:1>); Rotate Accumulator right.
Add (:= source = 5) → (result ← ACC + M[address]); Add Memory and Accumulator.
MoveM (:= source = 6) → (result ← M[address]); Move Memory.
And (:= source = 7) → (result ← ACC ∧ M[address]); AND Memory and Accumulator.
Jmp (:= op = 30) → (PC ← IR) Unconditional branch.

Table I. SCAMP Opcode Bits

<table>
<thead>
<tr>
<th>Source</th>
<th>Destination</th>
<th>PC Control</th>
</tr>
</thead>
<tbody>
<tr>
<td>548</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>000 Zero</td>
<td>0 MEM</td>
<td>00 PC ← PC + 1</td>
</tr>
<tr>
<td>001 MEM‘</td>
<td>1 ACC</td>
<td>01 IF DEST = 0 THEN PC ← PC + 1 ELSE PC ← PC + 2</td>
</tr>
<tr>
<td>010 ACC ∨ MEM</td>
<td></td>
<td>10 PC ← IR</td>
</tr>
<tr>
<td>011 ACC</td>
<td></td>
<td>11 IF CARRY THEN PC ← PC + 2 ELSE PC ← PC + 1</td>
</tr>
<tr>
<td>100 ACC rotated</td>
<td></td>
<td>right 1 bit</td>
</tr>
<tr>
<td>101 ACC + MEM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>110 MEM</td>
<td></td>
<td></td>
</tr>
<tr>
<td>111 ACC ∧ MEM</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In an unconditional branch the contents of the instruction register replace the contents of the PC. The control bits that go along have no effect since they are not in the address field. Finally, conditional branching can take place with skip-on-not-zero or skip-on-carry. Carry and zero conditions are computed for every instruction, and since they are transitory states they must be acted on as they are computed. While this has some obvious disadvantages, there are some advantages as well. For example, since ACC + MEM is always computed, it is possible to move an operand to the accumulator and skip on the carry that results from the add of ACC + MEM in one instruction.
E. System Timing

Figure 10 illustrates the word clock generator. This circuit produces a pulse once every WordSize (which is abbreviated as WS in the figure) clock cycles, where WordSize is the number of bits per word in the SCAMP. As its name implies, the word clock signal (WCK) is used primarily for signaling word boundaries in various data streams.

In Fig. 10, assuming that the clock begins cleanly at some initial time, switch 1 permits only the first pulse to pass. When applied to switch 2, which is initially in the cross state, this start pulse is injected into a word-sized fiber loop that terminates at the control input of switch 2. This loop is simply an application of the memory delay line described above; the single initial pulse is refreshed at each circulation, and the loop is tapped to provide the necessary number of WCK lines.

Figure 11 illustrates the pulse-driven toggle circuit that is used to differentiate between instruction and operand fetches. The delay introduced in the toggling MEMF pulse is 1 word period. This provides sufficient time for the address of the next phase to be made valid.

The toggle circuit is most easily viewed as a special case of the general memory structure of Fig. 6. When no T signal is applied, switch 4 remains in the cross state and a 1-bit loop is formed around switch 3. The complement of the output signal, which is a by-product of the regeneration at switch 3, is applied to switch 4 as an alternate input. A properly timed pulse at T sends the complement signal through to switch 3, where it becomes the new output.

Figure 12 is a simple circuit for converting a memory-found pulse into either an instruction-found or an operand-found pulse, depending on the machine state indicated by OPR/INST' . Switch 5 acts as a decoder; when OPR/INST' is low, the resulting cross state passes MEMF to the instruction-found (INSTF) line; when OPR/INST' is a pulse train, MEMF is routed to the operand-found (OPRF) line.

Figure 13 illustrates a pulse stretcher that generates a necessary timing signal. The effect is simply to take a single OPRF pulse and convert it into a sequence of WordSize pulses. The pulse-stretching element is represented by the square surrounding PS.

The stretched OPRF signal holds the switch in its bar state long enough to send the desired number of clock pulses to the output.

A sample timing diagram for integrating the timing signals into an overview of the instruction cycle is presented in Fig. 14. At the beginning of the instruction cycle OPR/INST' is low, indicating that an instruction is being sought. After an integral number of word cycles the address comparator indicates that the instruction has been found by sending a pulse on MEMF. This generates an INSTF pulse and causes OPR/INST' to go high at the beginning of the next word cycle. MEMF is pulsed again when the operand is found. This generates the pulse on OPRF and sends OPR/INST' low at the beginning of the following word cycle.

F. Memory Counter

Figure 15 shows the memory counter. The memory counter is a free-running version of the binary counter that is described above. WCK is used to increment the memory counter at the beginning of each word period. The least significant N bits of the count indicate which memory word is currently accessible. The counter has WordSize bits instead of N bits so that its increment time of WordSize cycles matches the WordSize cycles of each word in memory; N is unlikely to divide evenly into WordSize, so the extra bits are appended to maintain synchronization between the count and the circulating memory words.

G. Registers

The program counter is diagrammed in Fig. 16. The program counter holds the address of the next instruction to be executed. The OPRF line increments the count once during each instruction cycle to facilitate normal sequential operation. The SKIP line provides an additional increment (during a different word period) if a skip condition specified by an instruction succeeds. Branching is accomplished by forcing a different count value into the WordSize fiber loop. The CTL line is the source of the branch address.

The memory and accumulator loops are illustrated in Fig. 17. These are identical in operation to the structure of Fig. 6. The memory loop holds pro-
grams and data for the SCAMP and is \( \text{WordSize} \times 2^N \) bits long. The accumulator loop is a \( \text{WordSize} \) bit register whose function is typical of accumulators in single-address architectures. These loops are written to when \( \text{OPRFS} \) is asserted. The state of the \( \text{DSTSEL} \) line determines whether the memory loop or the accumulator loop receives the ALU result sent on \( \text{DEST} \).

The instruction loop of Figure 18 holds a copy of the current instruction during its execution. The instruction is obtained directly from the memory loop when \( \text{INSTF} \) is asserted. This register is another instance of the memory construct of Fig. 6.

H. System Operation

As described above, the address comparator, which is shown in Fig. 19, is used to indicate when a desired word in the memory loop has become accessible. In an instruction search the address field of the memory count is compared with the address field of the program count during each word period, and a match is signaled by a pulse on \( \text{MEMF} \). During an operand fetch it is the address portion of the current instruction that is compared with the memory count.

In Fig. 19 the \( \text{OPR/INST'} \) signal applied to switch 22 determines which comparison is to be made; in the cross state the \( \text{PC} \) is compared, whereas in the bar state \( \text{CTL} \) is passed. Switch 23 performs a bit-by-bit comparison of the \( \text{MC} \) and the control signal. Each matching bit passes a pulse on to switch 24. Switch 24 ands each input bit with its own result from the previous clock cycle. Any missing pulses in the data stream turn off all subsequent output. A comparison is initiated by the rising of the \( \text{WCK} \) to the control of switch 24. This passes the result of the first bit comparison. From then on the process must be self-sustaining; any failed bit comparisons turn off the output for the duration of the word cycle. Since an address consists of only the least significant \( N \) bits of a word, the \( \text{WCK} \) is applied to switch 25 after a delay of \( N \) bits to pass the result of the comparison as \( \text{MEMF} \). An additional delay of \( \text{WordSize} \times N \) is added to the latter to align the pulse on a word boundary. Figure 20 illustrates the control signal extractors. These have the effect of pulling the six serial opcode bits from the current instruction word and synchronizing them so that they may be applied in parallel. \( \text{OPRF} \), which indicates that an operand has been found, is used to initiate the extraction and subsequent instruction execution.

Synchronization of the opcode sequence is accomplished simply by sending delayed versions of the \( \text{CTL} \) signal to switches 26–30. The result is that the six sequential opcode bits are made to arrive simultaneously at the switches, where they may be clocked through by a single \( \text{OPRF} \) pulse.

I. Arithmetic and Logic Unit

Although it appears complex, the ALU pictured in Fig. 21 is actually a straightforward application of

---

```
Fig. 14. Timing example: \( \text{WordSize} = 16 \) bits, \( N = 10 \).

Fig. 15. Memory counter.

Fig. 16. Program counter.

Fig. 17. Memory and accumulator loops. \( \text{DSTSEL} \) is the destination selection line.
```
Boolean algebra. Figure 22 breaks down the addition circuit. It is seen that the basic logic functions AND and OR are required. Since these functions are themselves useful ALU outputs, it is desirable to share them in order to reduce the number of components. The ALU simultaneously performs eight operations on the current memory word and the accumulator. One of these eight results is sent out as the DEST signal. The carry from the addition and the zero state of DEST are computed so that skip-nonzero and skip-on-carry may be part of the program counter control.

Switches 34, 36, 39, 37, 40, and 41 in Fig. 21 make up an eight-to-one multiplexer. The three control bits SRCSEL0, SRCSEL1, and SRCSEL2 determine which of the eight computations—signals applied to the inputs of switches 34, 36, and 43 is routed to DEST. In order, the eight applied signals are MEM', MEM + ACC, ACC, MEM AND ACC, MEM, MEM OR ACC, 0, and ACC rotated right one bit. Switches 32 and 33 make up the adder. The previous carry is applied to the control input of switch 32. Switch 35 provides MEM AND ACC, a result that is also used in the carry computation. Switches 38 and 42 compute the carry. The WCK applied to the control input of switch 42 forces the initial carry to be zero. This also permits the final carry to be sent to the skip detection switches. Switch 44 is the nonzero detector. Any pulses in the DEST signal while OPRFS is asserted result in a constant select circuit. The latter is made up of switches 45 and 46. If the instruction commanded a skip on condition by sending a pulse on PCSEL1, the pulse is delayed so that it arrives at switch 46 at the correct time to clock the condition through. PCSEL0 applied to switch 45 selects between the nonzero and carry detection lines.

IV. Present and Future Operating Speed

The maximum operating speed of a machine that uses LiNbO$_3$ switches or devices of similar logical functionality is determined by two factors: switching time and switch latency. The switching time specified for the AT&T switch is 0.3 ns. However, this speed is considerably faster than the switching speed of the drive electronics. Measurements of switching times in the drive electronics indicate rise–fall times of approximately 1 ns (see Ref. 15). Therefore switch-
must be less than or equal to the delay imposed by the propagation time through a switch. We have designed a system to measure this maximum latency path through the switch. We have observed latencies of 4–6 ns in the drive electronics and 1–2 ns through fibers and the switch proper. This points to a maximum clock frequency in the 100-MHz region. As verification we constructed a counter similar to the one described in Fig. 5 and operated it at 50 MHz.

Estimates of possible future speeds and ultimate speed limitations are fraught with difficulty. Nevertheless we venture some speculation on the matter. One advantage of speed-scalable architectures such as the one described here is that no architectural change need be made to the design as the clock speed is raised, provided that switching time and switch latency are sufficiently fast and short, respectively. So switching time and switch latency provide the limiting factors on the system clock speed. We have discussed possible ultimate foreseeable switching speeds and switch latencies with several researchers in the field of directional coupler design and fabrication. The consensus was that the appropriate device would be directional couplers with integrated short-path interconnections and integrated drive electronics. Such designs could be envisioned as having both switching times and latencies in the 1–10 ps range. This would lead to clock speeds in the 0.1–1 THz range. A 1-bit loop would have a length in the 2–0.2 mm range at these clock frequencies; such a loop could be integrated easily with the switch. Even so, latency may well become the limiting factor. In latency-limited systems, it is possible to time-multiplex multiple systems on the same hardware, as discussed in Ref. 21.

A departure from the use of the direction coupler exchange switch as the logic element, and the use of other kinds of logic elements would lead to minor changes in the architecture. Such changes would open the design to other switching technologies. It is physically possible to produce and propagate 10-fsec pulses, which translates to a clock rate of 100 Tbits/sec. Haner and Warren have actually demonstrated a 100-fsec resolution in a time-compressed waveform, which suggests that 10 Tbits/sec may actually be achieved in practice. Islam et al. have demonstrated optical switching with solitons of ~300 fsec. Thus optical switching and transmission may not be too disparate in attainable speeds.

V. Summary and Conclusions

We have described an optical computer that is based on fiber-optic delay-line storage, optoelectronic directional coupler logic and switching, and fiber interconnections. Such a system bears a distinct similarity to early electronic computers that is due, in both cases, to the problems involved in building and reliably operating large numbers of active devices in an immature technology. Several new aspects of the architecture are dictated by the use of optics. The fact that signal propagation is of the same speed as switching gives a major geometrical component to the architecture. The signal propagation time is used even as the basis for information storage, so that each signal path in the machine must be considered as a storage element, or, at least, as modifying the behavior of a storage element that is the source or destination of the signal.

The use of optoelectronic directional couplers as the active elements of the machine leads to a unique solution of the signal amplitude and timing restoration problem. Since the switching of a signal is easier than its amplification, the technique of supplying multiple copies of the system clock, and switching one of these into a signal path by using a degraded control signal, is used to restore both amplitude and timing characteristics of signals. Another systemspecific technique of interest in the optical design world is the matching of optical speeds to electronic I/O speeds by transferring a single bit per memory cycle between electronic and optical systems.

The impetus for the design and implementation of an all-optical, stored-program computer arises from the long-term goal of developing architectures and design techniques that scale with speed, that is, speed-scalable architectures: architectures that are...
invariant to an increase in system speed coupled with a corresponding decrease in size. In addition there are two shorter term subgoals. It is a short route to an operating optical computer within the current technological constraints, which thus allows computer architects to begin doing optical computer design and thus to obtain feedback essential to the design process from operating prototypes. Such a bit-serial system would also have immediate application in the control of fiber-optic communications systems, which already operate in a serial mode. Current techniques of controlling such systems involve converting the serial information to electronic form, processing it at electronic speeds, and converting it back to optical form for transmission. The results of both computer architecture and high-speed optical component research need to be combined to make an all-optical, high-speed controller a reality.

The weak link in the current system design is the optoelectronic logic element. Detecting an optical signal, electronically amplifying it, and using it to control the state of a directional coupler is a significant problem. Electronics is at least limited in extent, and its natural pulse-spreading tendency can be used to make the control pulse overlap the duration of the signals being switched. But it is clear that this work should supply stimulus to, and anticipate results from, the many high-speed, optically switched, optical elements currently being developed in the research laboratories. Present devices limit operating speeds to the 100-MHz region; however, future devices could increase this speed by a factor of 1000 or more. The devices used in these experiments are not ready for use in systems, but it is clear that any system operating at rates that are near 1 Tbit/sec will pose serious competition to electronic processors. Though, in one sense, it is too early to design commercially successful optical computers, this work is aimed at helping develop a body of computer architecture knowledge in order to successfully exploit the new devices as they move out of the research laboratories.

This work was supported in part by National Science Foundation grant CDR 8622236 and by the Colorado Advanced Technology Institute, an agency of the state of Colorado (grant 270150).

References