Trans1 design report 1

David Venhoek

December 30, 2017

1 Design goals

The goal for Trans1 is to build a computer entirely out of discrete components (transistors, resistors, capacitors, inductors, switches and LED's). Within this primary constraint, the project has the following goals:

1. Have a relatively small footprint and component count.
2. Be capable of running interesting programs

1.1 Component count and footprint

One of the primary concerns for this project is practical viability. I'm currently limited both in available space and money to spend on this project.

Minimising the number of components used and footprint needed for those components will both decrease cost and space needed for construction and storage.

Furthermore, decreasing the number of components will make any debugging of the design during construction easier, as there are fewer places for problems to occur.

1.2 Software goals

In terms of software, I would like it to have at least some software that speaks to the imagination. For this purpose, simple games seem fairly effective.

Given this, the goal is to have the machine capable of running simple games such as snake or pong.

2 Gate construction and logic type

For the construction of gates, a survey was conducted on various method of constructing logic gates from transistors. CMOS, TTL, ECL (Emitter-coupled logic) and ILL (Integrated injection logic) require too many components per gate to be of practical use. DTL (Diode-transistor logic) was deemed to complicated since it requires the presence of both positive and negative power supply lines

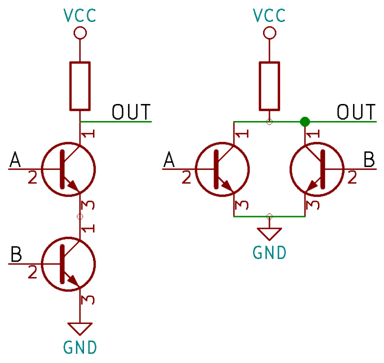

Figure 1: RTL NAND and NOR gates

on most gates. Finally, single transistor RTL (Resistor-transistor logic) gates were rejected because of their low between-input impedance.

This left the following candidates for gate construction:

1. Multiple transistor RTL based on BJT transistors
2. Multiple transistor RTL based on MOSFET transistors.

Based on common parts availability, the BC548B transistor was chosen as BJT candidate transistor, and the 2N7000 as the MOSFET candidate transistor.

2.1 RTL Gate design

Both logic families considered are multiple transistor RTL logics. The easily constructable gates are N-input NAND and NOR gates. The NOR gate is constructed by having the CE/DS pins connected in parallel, whilst the NAND gate is constructed with the CE/DS pins in series (see also figure 1). Note that a 1-input NAND or NOR gate is simply a NOT gate.

Beyond these basic gates, compound gates like in figure 2 are also possible, and could provide some component savings.

2.2 Transistor performance analysis

Both MOSFET and BJT based transistors were characterised whilst in a NOT gate configuration. Output voltage was measured as a function of input voltage. The data-sheets for both transistors (from ON semiconductors) were studied further to determine variation beyond what can be measured. Collector resistance was standardised to 1kΩ for both transistor types.

2.2.1 BJT analysis

For BJT transistors, and additional base transistor needs to be introduced to limit base current flow. A value of 10kΩ was chosen for this resistor.

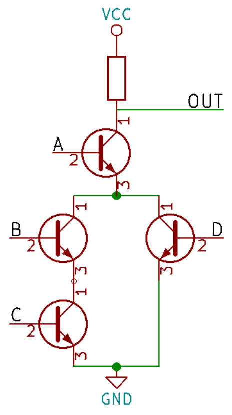

Figure 2: RTL compound gate calculating $OUT = \neg(A \wedge ((B \wedge C) \vee D))$.

As seen in figure 3, the switching point for the BC547 transistor (used as drop in replacement for the BC548) is observed at about 680mV. From about 800mV, the graph flattens, at an output of about 100mV, although some improvement in output value continuous beyond this point.

From the data-sheet, it is found that the switching point is at at least 580mV, and at most at 720mV. Logic levels of $<$ 500mV being considered as 0, and $>$ 1500mV for 1 seem reasonable given this information and the output improvement for larger input voltages.

2.2.2 MOSFET analysis

As seen in figure 4, the switching point of the BS170 transistor (used as drop in replacement for the BC548) is observed at about 2.5V, with the graph flattening at near 0V from about 3V on.

From the data-sheet, it can be calculated that the switching point of the transistor will be between 1V and 3V, with complete turn-on at 3.3V. Logic levels of $<$ 800mV for 0, and $>$ 3500mV for 1 seem achievable.

2.3 Logic type choice

Given the above results, I prefer the MOSFET based transistors for gate construction. The higher switch-on point and stronger switching behaviour compared to BJT transistors gives additional noise immunity, and the elimination of base resistors is a significant reduction in component count. Although the 2N7000 and BS170 transistors are reported as being more electrostatic discharge (ESD) sensitive, I have not yet observed any ESD induced component failure or degradation.

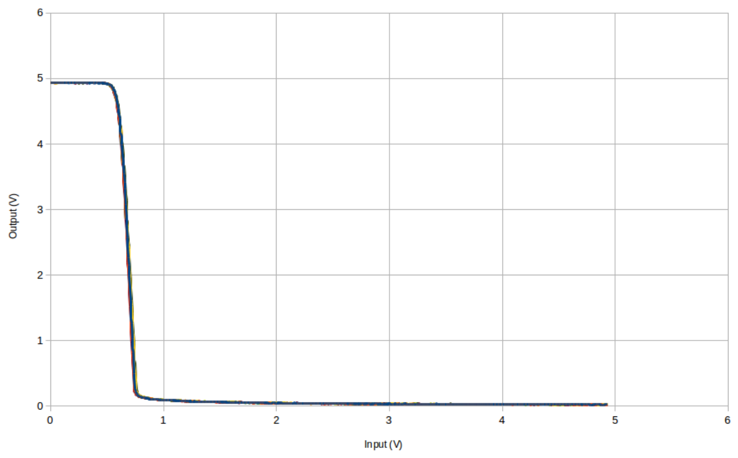

Figure 3: Behaviour of a sample of 10 BC547 BJT transistors.

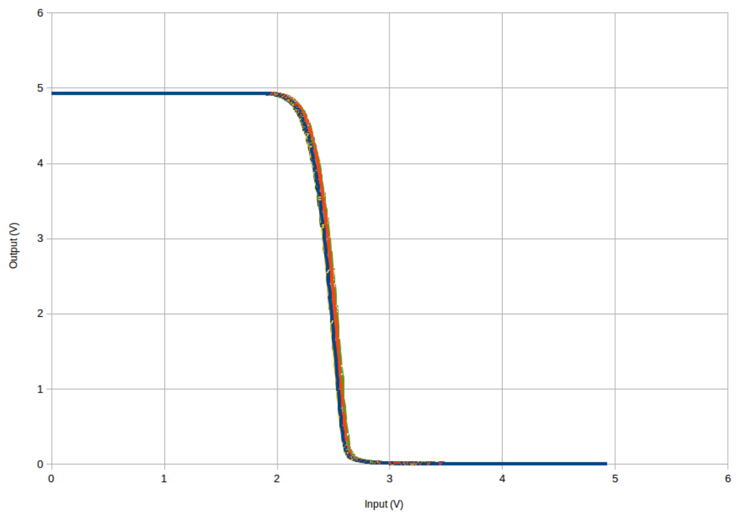

Figure 4: Behaviour of a sample of 10 BS170 MOSFET transistors.

3 Clock design considerations

Edge triggers made from discrete components will create problems in precisely controlling both arrival time and with of edge trigger pulses. In order to prevent racing in processor sub-circuits, I will enforce the principle that at no point a register or other data-storing element can depend on it's own input for output.

Beyond this, to ensure that control signals have had sufficient time to settle, all registers and related will clock on the rising edges, whilst the control logic will clock on the falling edge.

4 Basic processor architecture

For simplicity, it would be ideal to keep the data and address busses of the same size. Given goal 2, an 8 bit data word and address size was chosen. Furthermore, to keep complexity down, it was chosen to do I/O via memory mapping of devices at fixed addresses.

For the race avoidance goal, it is then necessary to have at least 3 register that function as temp or general purpose registers, as otherwise 2 input 1 output ALU instructions would not be possible. Given these restrictions, I decided to have one temp (T) register, and two general purpose registers (A and B). The A and B registers will be tied directly to the inputs of the ALU.

Beyond general purpose registers, at least a program counter (PC) and instruction register (IR) seem necessary. To keep complexity down, only the T and PC registers are connected to the address bus, and no direct connection is possible between the data bus and the address bus. Furthermore, with the exception of the T register, all registers only take input from the bus. The data bus can only be driven by the ALU, T register, control logic or memory controller.

To keep part count down on individual memory boards, the ram, ROM and peripherals will be split up into 32 boards, each containing 8 addressable locations. The memory controller will have all the logic needed to read/write memory locations, generates board select for each of the 32 boards, and passes through the remaining 3 address bits.

A complete block diagram of the machine is given in figure 5

5 Instruction set considerations

Given goal 1, the machine will use 8 bit data and address sizes. Given this, program size is at most 256 bytes, if all of the address space is used for instructions. As such, having protection features are not useful, and not including them will save significant complexity.

Hence, the instruction set will focus on only 3 actions: flow control, memory-register manipulation and register-register calculations. The decision was made to use 8-bit instructions, optionally with one 8 bit argument. This allows sig-

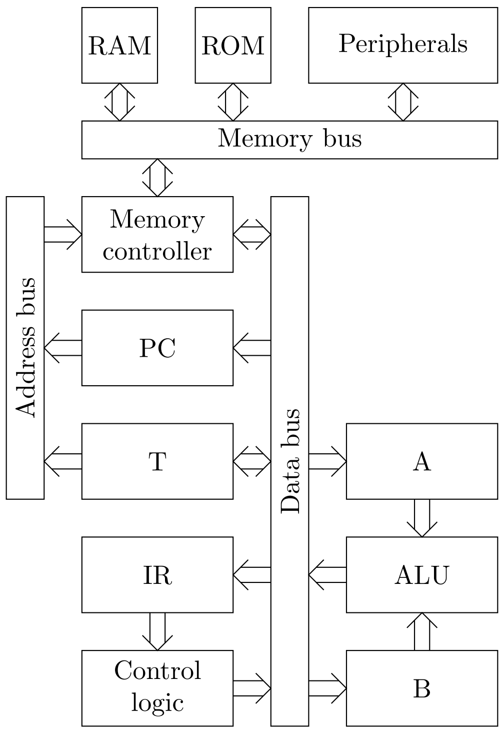

Figure 5: Computer block diagram.

nificant simplification of the control logic by tying some of the control signals of the ALU directly to the instruction register.

The following opcodes are currently considered:

- ALU opcodes ($\times$128)
- Load immediate into register ($\times$24) (with 8 bit argument)
- Load fixed memory location into register ($\times$2) (with 8 bit argument)
- Load memory location specified by register into register ($\times$4)
- Store register to fixed memory location ($\times$2) (with 8 bit argument)
- Store register to memory location specified by register ($\times$4)
- Halt machine ($\times$1)
- Sleep until timer interrupt ($\times$1)
- (Conditional) jump ($\times$16, of which 15 for conditional jumps) (with 8 bit argument)
- Load small immediate (4bit) into register ($\times$32) (4 bit argument encoded into instruction)
- Load low memory location (< 16) into register ($\times$32) (4 bit argument encoded into instruction)

Note that each of the load/store operations allows for access to only two registers. Only the A and B registers are planned to be visible to programs. This leaves the T register free to be used as temporary value register in instructions. The last three instructions are included as there is room for them, and their existence will allow significant reductions in program size.

For ALU operations, it is desired that the ALU provide at least the following:

- Addition A + B.
- Subtraction A - B.
- A and B register passthrough
- NOT A and NOT B
- Increment and decrement (at least on A).
- Shift left and right by 1.
- Logic OR, AND.

For conditional jumping, it would be useful to be able to jump based on the carry out of the last ALU operation, whether that operation was zero, whether the A or B register is zero, and the inverse of all these.

5.1 Instruction clock cycles

The instructions listed above are all possible to execute in 4 clock cycles. Doing this results in the following cadence:

The first two cycles are the same for all instructions, consisting of a fetch on cycle 1, and an increment PC on cycle 2. Cycles two and three are then used to execute the instruction.

For immediate loads, cycle 3 is used to transfer the data to the appropriate register, using clock cycle 4 to increment the PC if not using a small load. Alternatively (if this reduces transistor count in the control logic), the data can be transferred to the T register in cycle 3, and then moved to the target register in cycle 4 whilst simultaneously incrementing the program counter.

For memory loads, cycle 3 is used to transfer the address of the data to the T register, and on clock 4 the data is moved from memory to target register whilst PC is incremented if needed.

Stores are similar, on cycle 3 the destination address is loaded into the T register, with the write to memory occurring on the 4th cycle, simultaneously with any needed extra PC incrementation.

Jumps again follow a familiar pattern. On cycle 3 the destination address is loaded into the T register, and then in cycle 4, depending on the condition code, the program counter is either incremented or loaded from the T register.

Halt and sleep simply stop the machine clock on cycle 3, with sleep allowing a resuming of the clock by an external source. Cycle 4 is not needed on these instructions, and acts as a no-operation.

This leaves the ALU operations. In order to not read and write from the same register in a clock cycle, on cycle 3 the result of the ALU operation is computed, and stored in the T register. Any flags are also updated in this cycle. Then in cycle 4, the result is moved from T to the destination register.

6 Component design estimations

This report is not intended to give the full gate level design of the machine. However, for estimating the amount of materials needed to build the machine, it is useful to discuss the design of some essential elements.

6.1 ALU

The most important component is the ALU. Given the required functionality, let us start by considering the adding portion. Using NAND and NOR gates, a standard ripple carry full adder can be composed as shown in figure 6.

The logic function AND can be extracted by tapping of the first carry generation NAND after its corresponding NOT gate. For OR, a set of not gates is needed, but the NOR gate can be reused. Given that the full adder contains 3 NAND gates, 4 NOR gates and 2 not gates, the cost per bit of the adding,

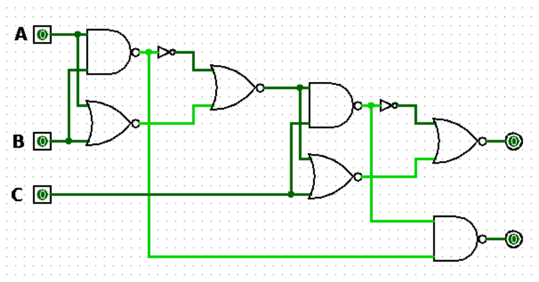

Figure 6: Full adder using NAND, NOR and NOT gates.

NAND, and NOR comes at 16 transistors and 9 resistors. Furthermore, by anding the carry input with an enable line (using 3 transistors per bit), the adder itself functions as an XOR gate.

The idea for the entire adder is to have 3 configurable options: The A input, the B input, and the output.

For the A input, for B passthrough it is nice to have the option to choose between A and 0.

For the B input, A passthrough needs an 0 as option. Furthermore, if we provide NOT B as input, and control the carry in of the adder, this enables subtraction. Finally, adding all ones as the final input option gives decrement capability on A, and rounds up the number of B inputs to 4.

The A input selection takes one bit, and the B input selection takes 2. This leaves 3 bits of the ALU opcodes to select output (the final bit selects which register to write the result to). The following options give all of the required gate options, whilst requiring very little gates to implement:

- Output=adder, carry=0.
- Output=adder, carry=1.
- Output=adder, carry from stored carry out bit.
- Output=adder, carry propagation disabled
- Output=adder shifted left once, carry=0
- Output=adder shifted right once, carry=0
- Output=AND gates
- Output=OR gates

It is useful if the ALU generates a number of flags on each operation. At a minimum, a carry flag and zero flag seem desirable. Since the carry flag is also used as input, two data flip-flops need to be used for it, one to buffer the carry, clocked on the control signal edge of the clock.

Based on the design outline above, it is estimated that between 200 and 250 transistors, and 100 to 150 resistors, are needed for the adder.

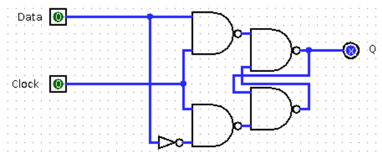

Figure 7: D Latch using NAND and NOT gates.

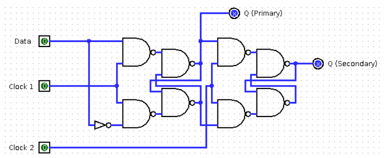

Figure 8: Master-Slave D latch using NAND and NOT gates.

6.2 Registers

The registers are most easily constructed from D latches. The most efficient way of constructing these I found was using 4 NAND gates and a single not gate (see figure 7).

Using 8 of these latches with clocks tied together, a full 8 bit register can be constructed.

Furthermore, to increase stability, it is useful to latch on the rising edge, instead of the high level. A simple edge detector can be constructed using an RC differentiator with $R = 10\text{k}\Omega$ and $C = 10\text{pF}$. This can easily be stabilised using 2 not gates on either side of the differentiator.

Including write enable logic, the A, B and IR registers will take about 80 transistors, 50 resistors and a capacitor each to construct. On the T register, the output to the data bus will use about another 20 transistors.

6.3 Program counter

In the previous section, we ignored the program counter. This is because we need to be able to increment the program counter.

During an increment, the program counter would need to act as both input to the increment circuitry, as well as store the resulting value. Using just a single register for this would violate the clock principles stated above, and could easily cause trouble.

The solution I have chosen is to use two latches in a master-slave configuration, the primary latch clocked on the rising edge with the normal logic, and the secondary one clocked on the falling edge, together with the control logic. The secondary register can then be used to drive the increment circuit. The combined master-slave latches is shown in figure 8

The increment logic can be implemented using XOR gates, using AND gates to determine which bits need to be flipped and which need to pass through

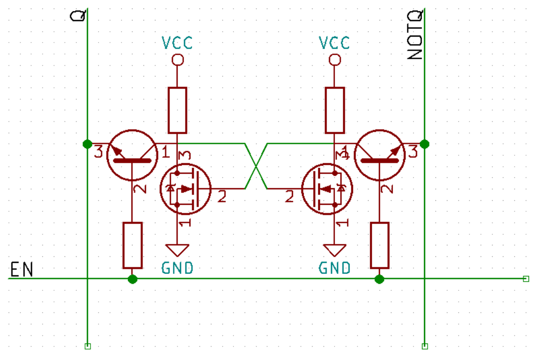

Figure 9: SRAM cell using 2 2N7000 and 2 BC548 transistors.

unchanged. Using combined logic gates, the XOR can be implemented using 6 transistors and 3 resistors.

For the entire PC register, including the increment and load from bus logic, I estimate between 250 and 300 transistors will be needed to build the program counter, together with approximately 150 resistors.

6.4 Control logic

The control logic will be highly dependent on the exact opcode assignments to instructions. As such, it will not be relevant to design this part in greater detail at this time. As most control signals should pass through relatively directly to their target components, my current estimates are a need for between 200 and 300 transistors, and between 100 and 150 resistors in the control logic.

6.5 Memory

The biggest challenge is going to be memory. Historically, there have been a number of options for storing data. Options like delay lines are too difficult to implement reliably, and for core memory the magnetic cores are hard to find.

The primary candidates thus considered were dynamic and static RAM. Dynamic ram requires only two components (one transistor and one capacitor) per data bit, but driving it is significantly more complicated, requiring refreshing of the memory to make sure data is not lost.

For this reason, my choice is to use SRAM. This has some disadvantages, specifically that it takes 4 transistors per cell, but the lower complexity in usage is worth that. The SRAM cell is shown in figure 9.

In the SRAM cell, the enable gates need to act as gates in both ways. Because of its generally poor enable gate performance (the GS threshold reduces the limit on passing voltage to about 2 V), and the body diode, the 2N7000 transistor is not suited to this role. Building a gate circuit performing a similar set of functions takes too many gates, hence I decided to use BC548 transistors here for that particular purpose.

Each cell of memory requires 8 parts, 4 transistors and 4 resistors. To keep part count memory, I decided to group the memory into 32 8 byte boards, each connected to the memory controller. To further limit the number of transistors required, most of the control logic, with the exception of the address decoding of the final 3 bits of the address, will be on the memory controller board. Each memory board then has a minimum of 22 connections to the memory controller: VCC, GND, 8 data bits, 8 inverse data bits (for writing), a chip select, and the lowest 3 bits of memory.

Per SRAM board, approximately 300 transistors and 300 resistors are needed. For the system, I am currently planning on somewhere between 32 and 64 bytes of memory, corresponding to between 4 and 8 boards. For estimating component counts, the upper bound of this range will be used.

The memory controller would probably require a further 300 transistors and 150 resistors.

6.6 ROM

In addition to RAM boards, ROM boards are also an option. For this, only address and chip select logic is necessary. This means that only about 50 transistors and 25 resistors are needed, in addition to (at most) 64 diodes, one for each data bit. Using the upper half of the memory as ROM, 16 of these boards will be needed at a minimum.

Multiple sets of ROM boards could be used to house multiple programs.

6.7 IO boards

Given an SRAM board, it is not that difficult to construct a display version, driving an array of 8x8 LED's. In addition to the parts for the SRAM itself, 64 LED's are needed. Also, given the voltage drop across LED's, it is necessary to use separate transistors to drive them, leading to a requirement of a further 64 resistors and transistors.

An input board would be more similar to a ROM board, only with switches in line with the diodes. I am currently planning to construct a 4 button controller interface.

7 Estimated bill of materials

Given the estimates from each of the parts listed above, it is possible to calculate an estimate for the number of parts needed for the entire computer.

2N7000 transistors	3500
BC548 transistors	1100
1kΩ resistors	2500
10kΩ resistors	1100
1N4148 diode	1000
LED's	64
220Ω resistors	64
Memory board connectors	32
Other connectors	100
15A 5V power supply	1

Given component prices of a few cents for most of these at these volumes, the entire machine will most likely cost something around the 500 euro mark for components and circuit boards.