

E

What's where in the source tree

Table E.1: code/ Implementation directory root.

directory	description
asm/	assembler and virtual machine
consts/	constants that represent opcodes and operations
firmware/	modules to compute SRAM firmware
logic-solver/	synthesize optimal SN74AUC-series glue logic
misc/	helper functions, constants, and macros
netlist/	“electrical source code” of the minicomputer and conversion software
netsim/	“electrical object code” of the minicomputer and simulation software
vm/	tool to test assembler and ALU firmware in virtual machine

This directory also contains the only `makefile` for the implementation.

Table E.2: code/asm/ Assembler and virtual machine.

file	contents
alu.c	ALU for the virtual machine, including colored trace output
asm.c	outermost routines for assembler and virtual machine
asm.h	API for assembler and virtual machine
context.h	non-shared struct for the API's opaque pointer
errors.c	de-duplicates and reports errors found by assembler
lex.c	assembler terminals and low-level parsing
no-bu	list of files that GNU Tar need not save
symc.c	symbol table implementation for assembler
syntax.c	assembler high-level parsing and instruction encoding

Table E.3: code/consts/ Constants that represent opcodes and operations.

file	contents
opcode.h	assigned numbers and source code names for opcodes
slots.h	assigned numbers and source code names for ALU operations

Table E.4: code/firmware/ Modules to compute SRAM firmware.

file	contents
alpha.c	ALU α layer
audited.c	short functions exempted from separately-written test cases
beta.c	ALU β layer
decoder.c	non-ALU portion of control decoder RAMs D0 and D1
firmware.c	firmware memory allocation and loading for simulations
firmware.h	local header file for firmware modules
gamma.c	ALU γ layer
gen-sbox.py	compute S-boxes derived from $\sqrt{2} \div 2$
instruct.c	ALU portion of control decoder RAMs D0 and D1
no-bu	list of files that GNU Tar need not save
pieces.c	firmware-computing functions that don't fit neatly elsewhere
sbox.c	automatically generated by gen-sbox.py
theta.c	ALU θ RAM
unary.c	stacked unary operations (simple unary are in audited.c)
uninit.c	obsolete stubs for "initializing" non-firmware RAMs
zeta.c	ALU ζ RAM

Table E.5: code/logic-solver/ Synthesize optimal SN74AUC-series glue logic.

file	contents
alpha-lock-permutations	what-if scenarios for various control signal meanings
freebie demonstration.odt	optimal boolean functions of 3 variables for SRAM enable inputs
logic solver demonstration.odt	optimal boolean functions of 3 variables
no-bu	list of files that GNU Tar need not save
size	database of size-optimized boolean functions of 4 variables
size-freebies	“size” recomputed for use as SRAM enable inputs
solve.c	solver program for fastest/smallest SN74AUC circuits
speed	database of speed-optimized boolean functions of 4 variables
speed-freebies	“speed” recomputed for use as SRAM enable inputs
tables.py	Python batch job to compute glue logic for node lockouts

Table E.6: code/misc/ Helper functions, constants, and macros.

file	contents
misc.c	general support functions, mostly not architecture-specific
misc.h	general and architecture-specific definitions
unused.c	unused code emulating the β layer transposition

Table E.7: code/netlist/ “Electrical source code” of the minicomputer and software to process it.

file	contents
2021 topology.ods	27 spreadsheets of circuit details; possibly out of sync with timing study.ods
feb.netlist	first-attempt CPU netlist started February 2, 2021 (human-written)
floorplan.svg	latest component floorplan generated by run-netlist.py
frozen.netlist	version of feb.netlist that generated netsim/frozen.ns
generated.bom	bill of materials, component list, and run statistics from run-netlist.py
generated.net	run-netlist.py output in a format that KiCad 5.0.2 can load
net-everything.svg	net connectivity markings on top of floorplan.svg
no-bu	list of files that GNU Tar need not save
run.netlist.py	main netlist processing program; its inputs are feb.netlist and tiles
tiles	circuit board locations for all components (human-written)
timing study.ods	possible unintentional fork of 2021 topology.ods; needs check for merge

Table E.8: `code/netsim/` “Electrical object code” of the minicomputer and software to simulate it.

C source code	description
<code>compo.h</code>	structs that represent components and pins
<code>comps/c_1g74.c</code>	simulated D flip-flop with preset and clear
<code>comps/c_244.c</code>	simulated quad 4-bit buffer with output enable
<code>comps/c_2gxx.c</code>	simulated AND, BUF, INV, NAND, NOR, OR, XOR
<code>comps/c_374.c</code>	simulated dual 8-bit D flip-flop
<code>comps/c_io.c</code>	simulated component for generic test hook
<code>comps/c_osc.c</code>	simulated crystal oscillator
<code>comps/c_pull.c</code>	simulated pull resistor
<code>comps/c_reset.c</code>	simulated voltage monitor (power-on reset)
<code>comps/c_sram.c</code>	simulated synchronous SRAM
<code>comps/c_switch.c</code>	simulated DIP switch
<code>connect.c</code>	deals with components, connections, short circuits, etc.
<code>load.c</code>	parser for simulation scripts
<code>ns.c</code>	root source file and <code>main()</code> for the simulator
<code>pin.c</code>	simulates component input, output, and clock input pins
<code>queue.c</code>	heap-based generic priority queue (wrapped by <code>quser.c</code>)
<code>quser.c</code>	time-based priority queue for netlist simulation
<code>util.c</code>	minor tidbit functions not used elsewhere
other file	description
<code>connectivity.ns</code>	connections and timings computed by <code>run-netlist.py</code>
<code>fib.a</code>	assembly: infinite loop of overflowing Fibonacci numbers
<code>fib.ns</code>	script: load <code>frozen.ns</code> and run <code>fib.a</code>
<code>fibclean.a</code>	assembly: compute xth Fibonacci number, $0 \leq x \leq 53$
<code>fibclean.ns</code>	script: load <code>frozen.ns</code> and run <code>fibclean.a</code>
<code>fire.report</code>	overcurrent/short-circuit measurements from last <code>ns</code> run
<code>frozen.ns</code>	<code>connectivity.ns</code> as computed using <code>frozen.netlist</code>
<code>last-test.ns</code>	symbolic link to last script run from <code>ns</code>
<code>no-bu</code>	list of files that GNU Tar need not save
<code>nop.a</code>	assembly: 8 NOPs and a HALT for system power-up tests
<code>nop.ns</code>	script: load <code>frozen.ns</code> and run <code>nop.a</code>
<code>ns</code>	binary executable for electrical simulation
<code>regress/</code>	directory of forgotten regression tests (may still work)
<code>writeprotect.ns</code>	script: disable SRAM writes for purpose of testing
<code>zero-pc.ns</code>	script: force program counters to zero for purpose of testing

Table E.9: code/vm/

file	contents
asm-tests.c	asm-tests/ folder converted to static C strings
asm-tests/abs.at	assembly: absolute value regression test
asm-tests/dist.test	assembly: mixed-signage arithmetic example
asm-tests/ham2.at	two-cycle popcount regression test
asm-tests/lmul.at	assembly: 64-bit multiplication regression test
asm-tests/rpop.at	assembly: R(ange) flag save and restore regression test
asm-tests/smul.at	assembly: absolute value regression test
gen-asm-tests.py	tool to convert asm-tests/*.at to asm-tests.c
no-bu	list of files that GNU Tar need not save
perms.c	regression test code for permutations
stats.c	statistical tests for MIX and XIM instructions
tests.c	regression tests for 109 ALU opcodes
vm	tool to test assembler and ALU firmware in virtual machine
vm.c	main() routine for vm