Block I
Apollo Guidance Computer (AGC)

How to build one in your basement

Part 2: Control (CTL) Module

John Pultorak

December, 2004

Abstract

This report describes my successful project to build a working reproduction of the 1964 prototype for the Block I Apollo Guidance Computer. The AGC is the flight computer for the Apollo moon landings, and is the world's first integrated circuit computer.

I built it in my basement. It took me 4 years.

If you like, you can build one too. It will take you less time, and yours will be better than mine.

I documented my project in 9 separate .pdf files:

Part 1 Overview: Introduces the project.

Part 2 CTL Module: Design and construction of the control module.

Part 3 PROC Module: Design and construction of the processing (CPU) module.

Part 4 MEM Module: Design and construction of the memory module.

Part 5 IO Module: Design and construction of the display/keyboard (DSKY) module.

Part 6 Assembler: A cross-assembler for AGC software development.

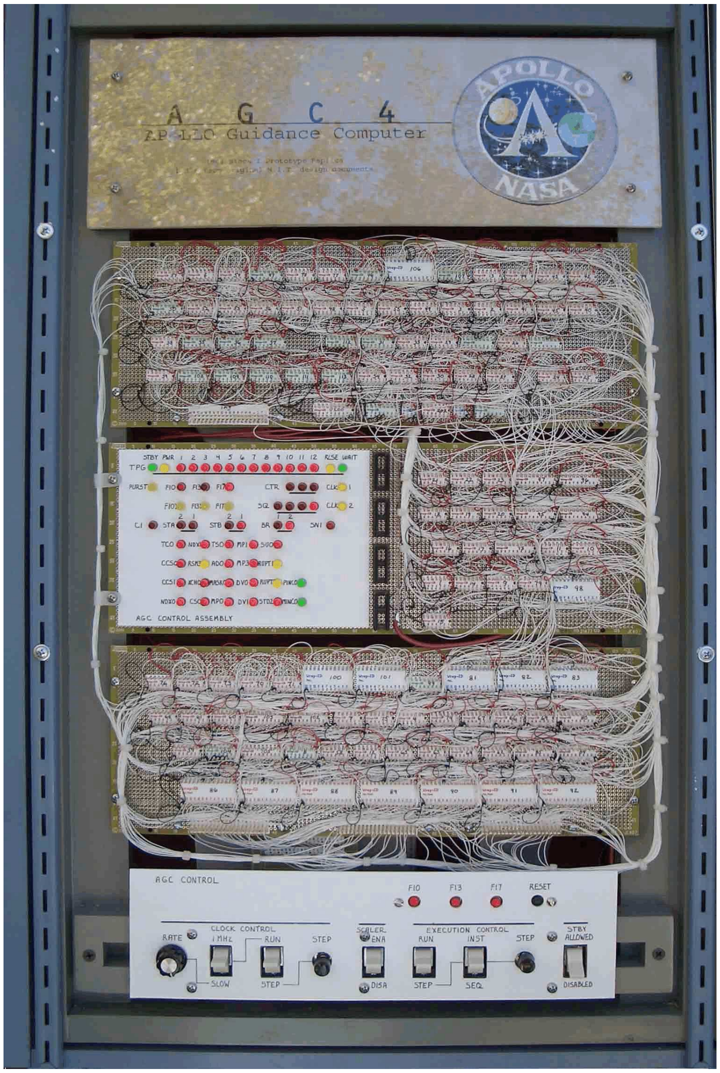
Part 7 C++ Simulator: A low-level simulator that runs assembled AGC code.

Part 8 Flight Software: My translation of portions of the COLOSSUS 249 flight software.

Part 9 Test & Checkout: A suite of test programs in AGC assembly language.

Overview

The Control Module (CTL) has 9 subsystems: CMI, MON, CLK, SCL, TPG, SEQ, CPM-A, CPM-B, CPM-C



CMI (Control Module external Interface)
The CMI interfaces the other control module subsystems (described below) to external AGC modules. 40-pin IDE connectors interface to the PROC, MEM, and IO modules. Inputs from those modules are buffered to 1 LSTTL load.

MON (AGC Monitor)
The monitor subsystem has the front-panel switches that control AGC operation, and also implements the power-up reset circuit.

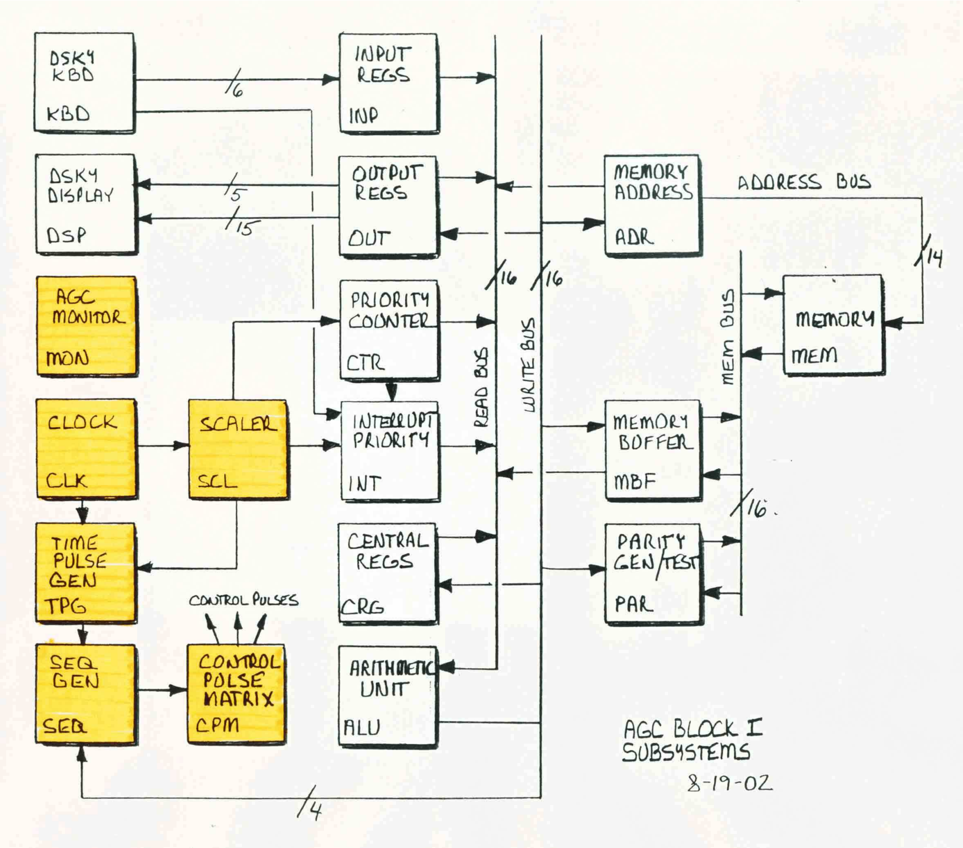
CLK (Clock)
The AGC is controlled by a 2.048 MHz crystal clock. The clock is divided to produce a 2-phase, non-overlapping 1.024 MHz AGC system clock for nominal operation. For test purposes, a low-frequency RC clock can also be selected, or the clock can be single-stepped.

SCL (Scaler)
The 1.024 MHz AGC clock is divided by two to produce a 512 kHz signal called the MASTER FREQUENCY; this signal is further divided through a SCALER, first by five to produce a 102.4 kHz signal. This is then divided by two through 17 successive stages called F1 (51.2 kHz) through F17 (0.78125 Hz). The F10 stage (100 Hz) is fed back into the AGC to increment the real-time clock and other priority counters in the PROC module. The F17 stage is used to intermittently run the AGC when it operates in the STANDBY mode.

TPG (Time Pulse Generator)
AGC instructions are implemented in groups of 12 steps, called timing pulses. The timing pulses, named TP1 through TP12, are produced by the Time Pulse Generator (TPG). Each set of 12 timing pulses is called an instruction subsequence. Simple instructions, such as TC, execute in a single subsequence of 12 pulses. More complex instructions require several subsequences.

SEQ (Sequence Generator)
The sequence generator has the SQ register, which holds the next op-code, and the CTR

register--a counter used to count instruction subsequences during multiplication. The sequence generator also has the branch register (BR) which controls conditional processing during instruction execution, and the STAGE registers (STA and STB) which select the instruction subsequences for complex instructions that have more than one subsequence.



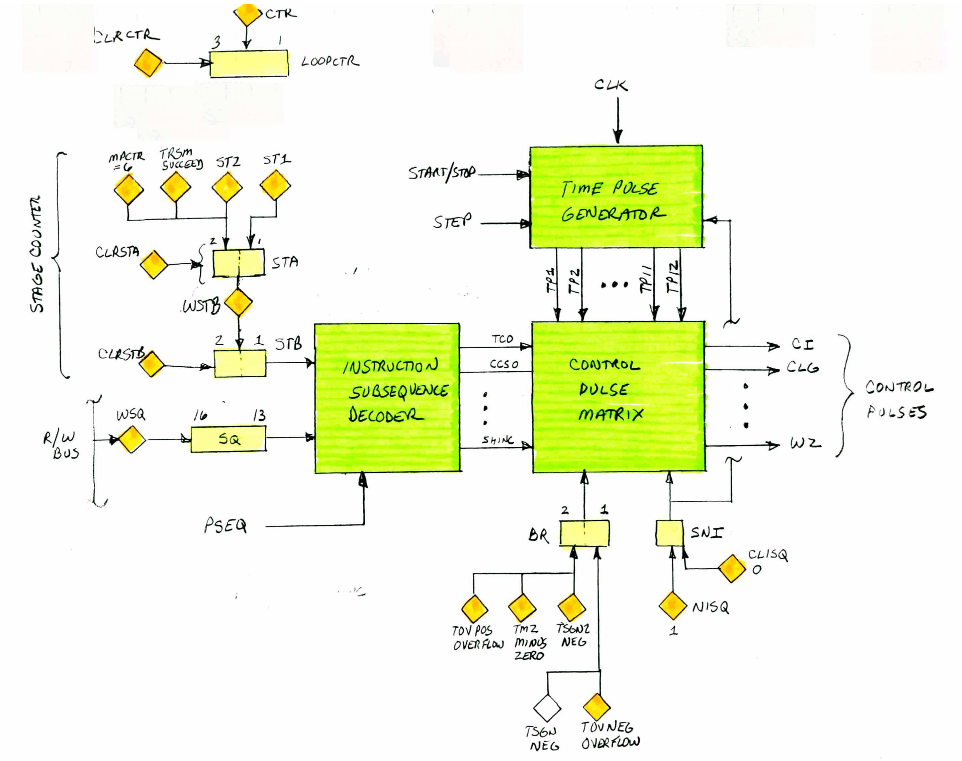
CPM-A (Control Pulse Matrix A)
The CPM-A is the combinational logic matrix that implements most of the control logic. It is driven by inputs from the SQ register (which selects the instruction), the STB stage register (which selects the instruction subsequence), and the BR branch register (which selects conditional steps in a subsequence).

CPM-B (Control Pulse Matrix B)
The CPM-B decodes read and write control signals for special memory location associated with input/output registers, central registers (A, Q, Z, and LP), and the editing registers used for rotation and shifting.

CPM-C (Control Pulse Matrix C)
The CPM-C decodes control signals primarily associated with interrupts, the memory cycle, and the selection of new instructions and instruction subsequences.

This is a functional diagram showing the interrelationships between the Time Pulse Generator (TPG), the Control Pulse Matrix (CPM-A, B, and C), and the registers that are in the Sequence Generator (SEQ).

The diagram is mine, but the style is borrowed from original AGC documentation: control signals are represented by diamonds. The arrows show the direction of data flow. When a control signal is asserted, data is allowed to flow through the diamond. For example, when WSQ is asserted, the opcode is written from the Read/Write bus into the SQ register.



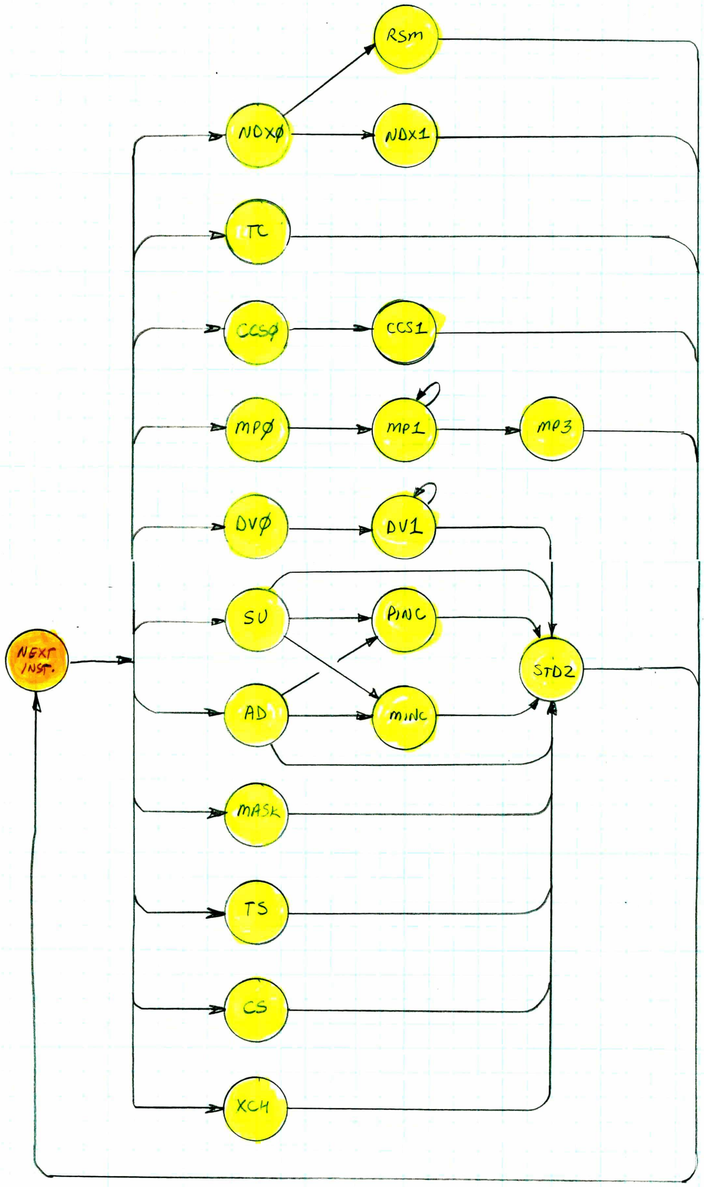
Registers (LOOPCTR, STA, STB, SQ, BR, and SNI) are represented by rectangles. The lower bit of the 2-bit STA register is set by ST1. The upper bit is set by any one of the 3 control signals flowing into it. The STA register is cleared by CLRSTA.

When WSTB is asserted, the STA register is copied to STB. STB and the SQ register select the instruction subsequence.

The instruction subsequence, time pulse generator, BR register, and SNI all feed into the Control Pulse Matrix to select the active control pulses.

The diagram was developed by analyzing the R-393 document. It was one of my first diagrams; a sort-of conceptual breakthrough that became my gateway for understanding the AGC control module.

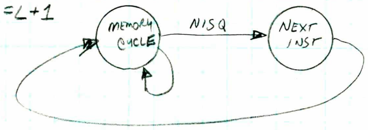
The instruction subsequences executed by the AGC are shown in this diagram. Each yellow circle is a subsequence; a set of 12 steps, with each step generating 0-5 control pulses. Eleven steps (TP1-TP11) are in the yellow circle; the 12th step, which selects the next subsequence (TP12), is in the orange circle. This is discussed in more detail in the TPG, SEQ, and CPM-A subsystems.

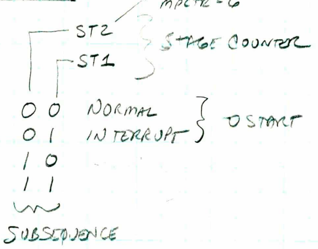


(3-1)

INST IN L

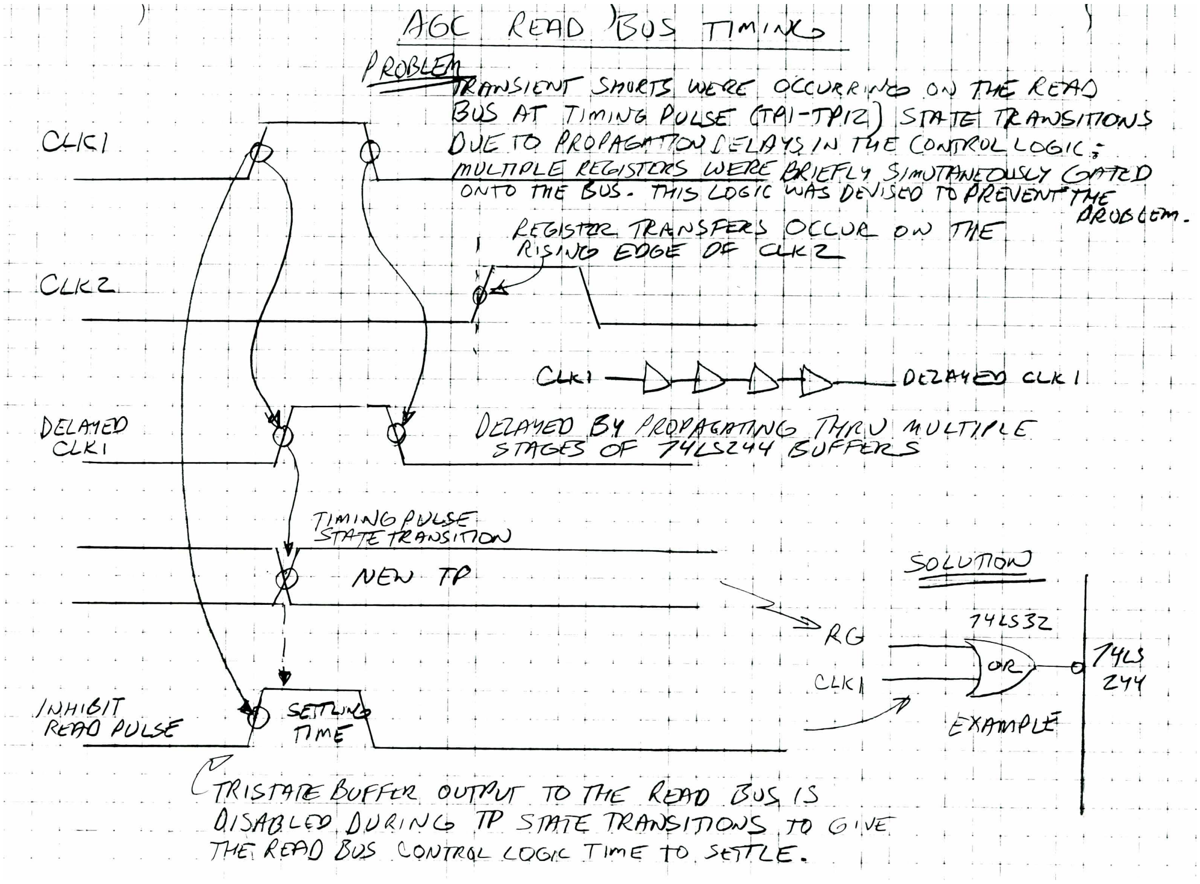
C(Z) = L+1





The counter is incremented during the memory cycle and transferred to "B" for output in T12. (2-9)

A late addition to the CTL design fixed a problem with the read bus logic. Due to propagation delays (and some poor design on my part), the tri-state buffers from multiple registers were simultaneously enabled for brief periods causing some transients. I “kludged” in a design change that suppressed output to the read bus during CLK1, thereby giving the bus control logic time to settle.



AGC READ BUS TIMING

PROBLEM

TRANSIENT SHORTS WERE OCCURRING ON THE READ BUS AT TIMING PULSE (TP1-TP12) STATE TRANSITIONS DUE TO PROPAGATION DELAYS IN THE CONTROL LOGIC; MULTIPLE REGISTERS WERE BRIEFLY SIMUTANEOUSLY GATED ONTO THE BUS. THIS LOGIC WAS DEVISED TO PREVENT THE PROBLEM.

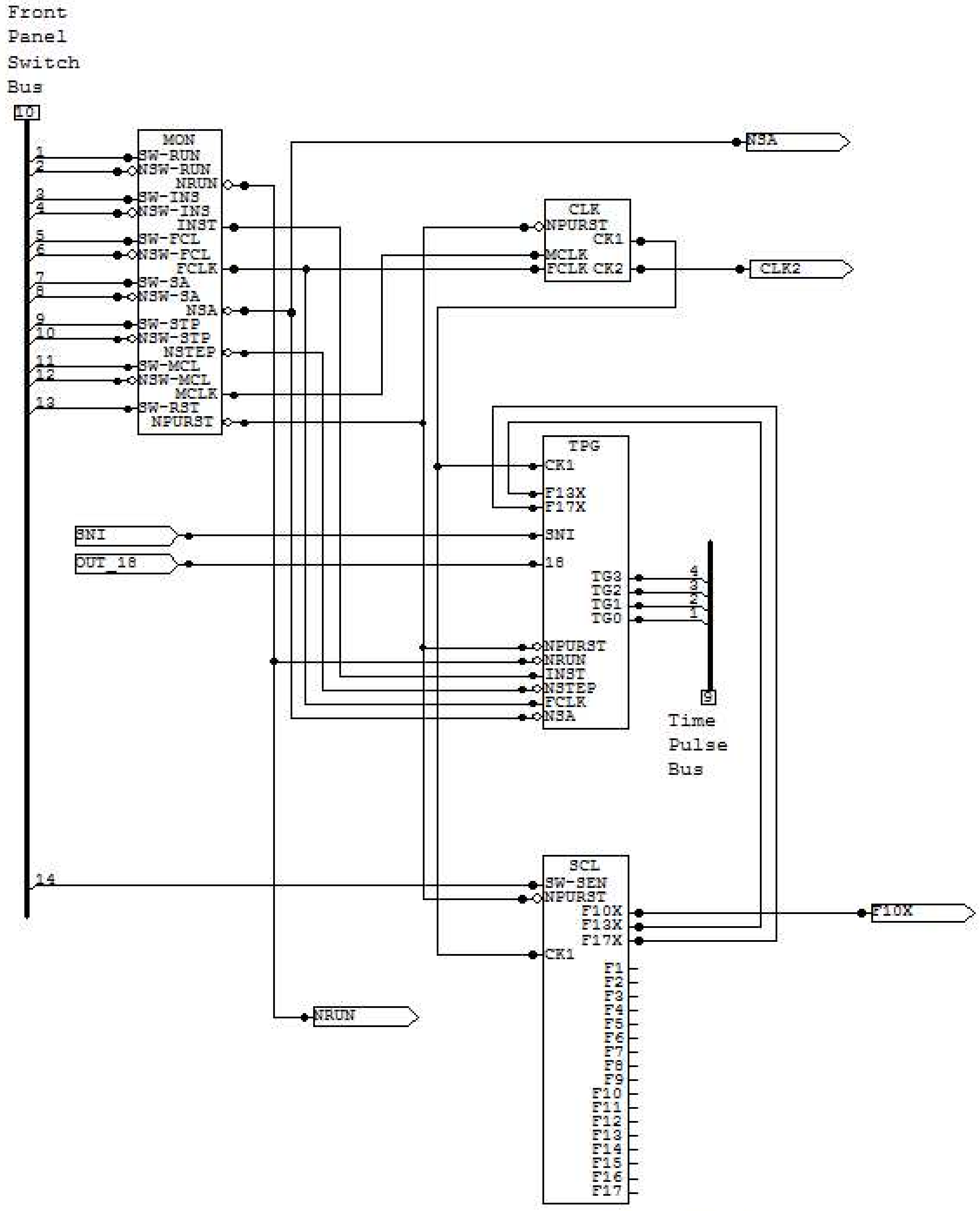
REGISTER TRANSFERS OCCUR ON THE RISING EDGE OF CLK2

DELAYED BY PROPAGATING THRU MULTIPLE STAGES OF 74LS244 BUFFERS

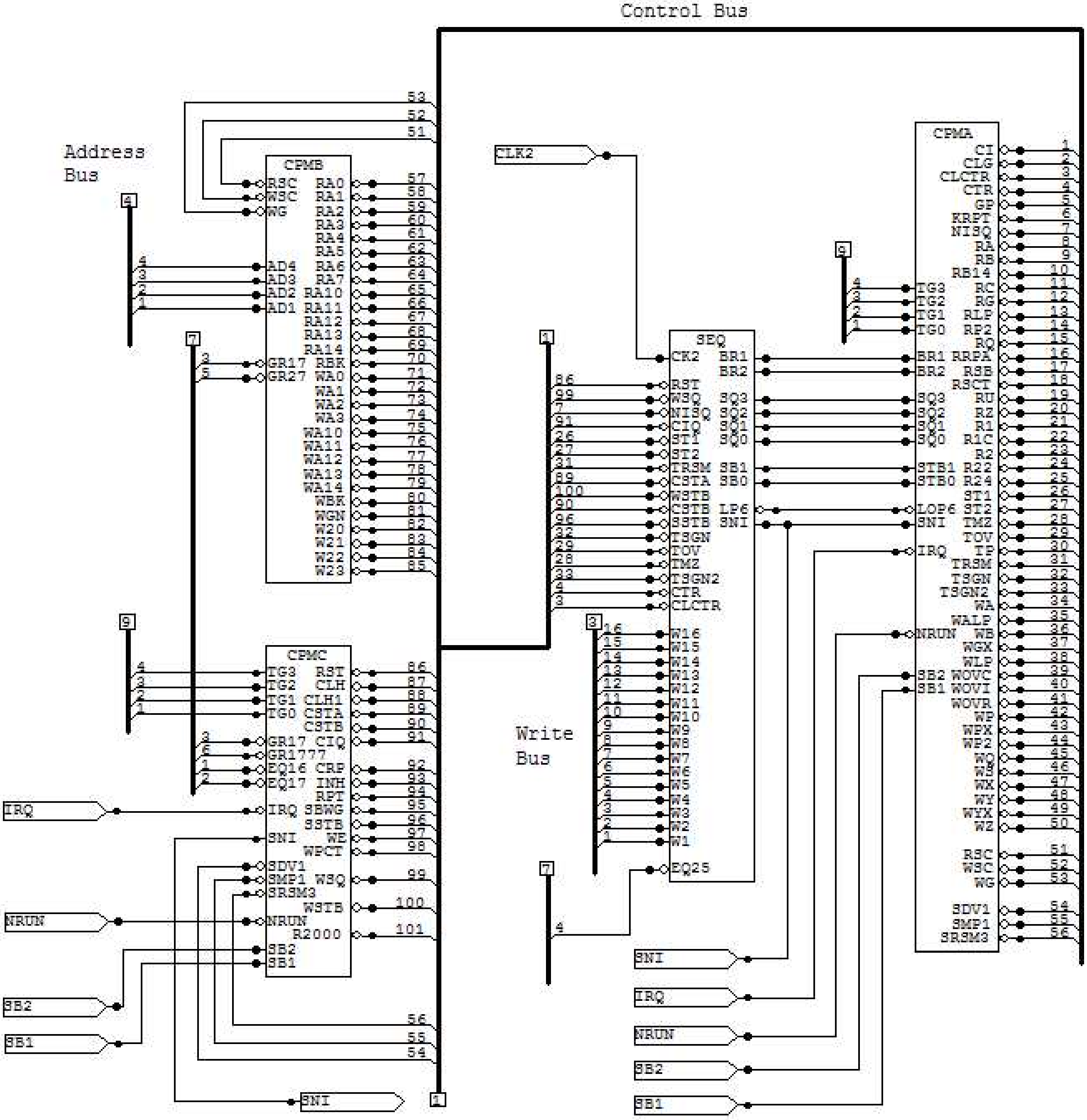
TRISTATE BUFFER OUTPUT TO THE READ BUS IS DISABLED DURING TP STATE TRANSITIONS TO GIVE THE READ BUS CONTROL LOGIC TIME TO SETTLE.

CTL Internal Subsystem Interconnections

This diagram shows internal interconnections for the subsystems in the CTL module.



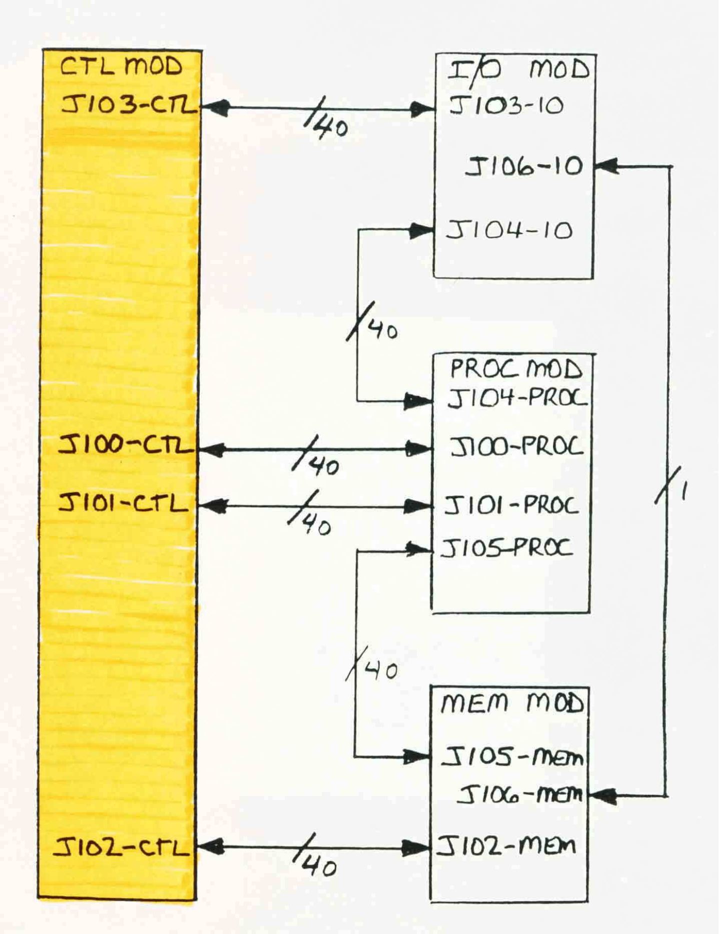
APOLLO GUIDANCE COMPUTER
AGC4 REPLICA
CONTROL MODULE (CTL)
9/8/2003


Control Bus
Address Bus
Write Bus
CPMB
RSC WSC WG AD4 AD3 AD2 AD1 GR17 GR27
RA0 RA1 RA2 RA3 RA4 RA5 RA6 RA7 RA10 RA11 RA12 RA13 RA14 RBK WA0 WA1 WA2 WA3 WA10 WA11 WA12 WA13 WA14 WBK WGN W20 W21 W22 W23
CPMC
TG3 TG2 TG1 TG0 GR17 GR1777 EQ16 EQ17 IRQ SNI SDV1 SMP1 SRSM3 NRUN SB2 SB1
RST CLH CLH1 CSTA CSTB CIQ CRP INH RPT SBWG SSTB WE WPCT WSQ WSTB R2000
SEQ
CK2 RST WSQ NISQ CIQ ST1 ST2 TRSM CSTA WSTB CSTB SSTB TSGN TOV TMZ TSGN2 CTR CLCTR W16 W15 W14 W13 W12 W11 W10 W9 W8 W7 W6 W5 W4 W3 W2 W1 EQ25
BR1 BR2 SQ3 SQ2 SQ1 SQ0 SB1 SB0 LP6 SNI
CPMA
TG3 TG2 TG1 TG0 BR1 BR2 SQ3 SQ2 SQ1 SQ0 STB1 STB0 LOP6 SNI IRQ NRUN SB2 SB1
CI CLG CLCTR CTR GP KRPT NISQ RA RB RB14 RC RG RLP RP2 RQ RRPA RSB RSCT RU RZ R1 R1C R2 R22 R24 ST1 ST2 TMZ TOV TP TRSM TSGN TSGN2 WA WALP WB WGX WLP WOVC WOVI WOVR WP WPX WP2 WQ WS WX WY WYX WZ RSC WSC WG SDV1 SMP1 SRSM3
CLK2
IRQ
NRUN
SB2
SB1
SNI

APOLLO GUIDANCE COMPUTER
AGC4 REPLICA
CONTROL MODULE (CTL)
9/8/2003

CTL Module External Interfaces

The CTL module interfaces to the PROC, MEM, and IO modules through 40-pin IDE ribbon cables.



AGC MODULE INTERCONNECTIONS

J100-CTL: CTL-to-PROC I/F
J100 is a 40-pin IDE ribbon cable that connects the CTL module to the PROC module.

OUTPUTS (from CTL):

PIN	signal	full name	state definition
1	WA3	WRITE ADDR 3 (74)	0=Write reg at address 3 (LP)
2	WA2	WRITE ADDR 2 (73)	0=Write reg at address 2 (Z)
3	WA1	WRITE ADDR 1 (72)	0=Write reg at address 1 (Q)
4	WA0	WRITE ADDR 0 (71)	0=Write reg at address 0 (A)
5	RA3	READ ADDR 3 (60)	0=Read reg at address 3 (LP)
6	RA2	READ ADDR 2 (59)	0=Read reg at address 2 (Z)
7	RA1	READ ADDR 1 (58)	0=Read reg at address 1 (Q)
8	RA0	READ ADDR 0 (57)	0=Read reg at address 0 (A)
9	WZ	WRITE Z (50)	0=Write Z
10	WYx	WRITE Y NO RESET (49)	0=Write Y (do not reset)
11	WY	WRITE Y (48)	0=Write Y
12	WX	WRITE X (47)	0=Write X
13	WQ	WRITE Q (45)	0=Write Q
14	WOVR	WRITE OVF (41)	0=Write overflow
15	WOVI	WRITE OVF RUPT INH (40)	0=Write overflow RUPT inhibit
16	WOVC	WRITE OVF CNTR (39)	0=Write overflow counter
17	WLP	WRITE LP (38)	0=Write LP
18	WB	WRITE B (36)	0=Write B
19	WALP	WRITE A/LP (35)	0=Write A and LP
20	WA	WRITE A (34)	0=Write A
21	F10X	F10 SCALER ONESHOT	1=timed out (100.0 Hz)
23	R24	READ 24 (25)	0=Read 24
24	R22	READ 22 (24)	0=Read 22
25	R2	READ 2 (23)	0=Read 2
26	R1C	READ 1 COMP (22)	0=Read 1 complimented
27	R1	READ 1 (21)	0=Read 1
28	RZ	READ Z (20)	0=Read Z
29	RU	READ U (19)	0=Read sum
30	RSCT	READ CNTR ADDR (18)	0=Read selected counter address
31	RSB	READ SIGN (17)	0=Read sign bit
32	RRPA	READ RUPT ADDR (16)	0=Read RUPT address
33	RQ	READ Q (15)	0=Read Q
34	RLP	READ LP (13)	0=Read LP
35	RC	READ C (11)	0=Read C
36	RB14	READ BIT 14 (10)	0=Read bit 14
37	RB	READ B (9)	0=Read B
38	RA	READ A (8)	0=Read A
39	KRPT	KNOCK DOWN RUPT (6)	0=Knock down Rupt priority
40	CI	SET CARRY IN (1)	0=Carry in

J101-CTL: CTL-to-PROC I/F
J101 is a 40-pin IDE ribbon cable that connects the CTL module to the PROC module.

INPUTS (to CTL):

PIN	signal	full name	state definition
21	SB_01	SUB SEL 01	SB_01 is LSB; SB_02 is MSB
22	SB_02	SUB SEL 02	00=no counter; 01=PINC; 10=MINC
23	IRQ	INT RQST	0=interrupt requested.
25	WB_01	WRITE BUS 01	(lsb)
26	WB_02	WRITE BUS 02	
27	WB_03	WRITE BUS 03	
28	WB_04	WRITE BUS 04	
29	WB_05	WRITE BUS 05	
30	WB_06	WRITE BUS 06	
31	WB_07	WRITE BUS 07	
32	WB_08	WRITE BUS 08	
33	WB_09	WRITE BUS 09	
34	WB_10	WRITE BUS 10	
35	WB_11	WRITE BUS 11	
36	WB_12	WRITE BUS 12	
37	WB_13	WRITE BUS 13	
38	WB_14	WRITE BUS 14	
39	WB_15	WRITE BUS 15	US (overflow) bit
40	WB_16	WRITE BUS 16	SG (sign) bit

OUTPUTS (from CTL):

PIN	signal	full name	state definition
1	R2000	READ 2000 (101)	0=Read 2000
2	WPCTR	WRITE PCTR (98)	0=Write PCTR (latch priority counter sequence)
3	RPT	READ RUPT (94)	0=Read RUPT opcode
4	INH	SET INHINT (93)	0=Set INHINT
5	CLRP	CLEAR RPCELL (92)	0=Clear RPCELL
6	CLINH1	CLEAR INHINT1 (88)	0=Clear INHINT1
7	CLINH	CLEAR INHINT (87)	0=Clear INHINT
8	GENRST	GENERAL RESET (86)	0=General Reset
19	CLK1	CLOCK1	1.024 MHz AGC clock 1 (normally low)
20	CLK2	CLOCK2	1.024 MHz AGC clock 2 (normally low)

J102-CTL: CTL-to-MEM I/F

J102 is a 40-pin IDE ribbon cable that connects the CTL module to the MEM module.

INPUTS (to CTL):

PIN	signal	full name	state definition
31	EQU_16	ADDRESS = 016 (1)	0=CADR in register S = 016
32	EQU_17	ADDRESS = 017 (2)	0=CADR in register S = 017
33	GTR_17	ADDRESS > 017 (3)	0=CADR in register S > 017
34	EQU_25	ADDRESS = 025 (4)	0=CADR in register S = 025
35	GTR_27	ADDRESS > 027 (5)	0=CADR in register S > 027
36	GTR_1777	ADDRESS > 01777 (6)	0=CADR in register S > 01777
37	AD_1	ADDRESS (1)	where AD_4 is MSB, AD_1 is LSB:
38	AD_2	ADDRESS (2)	(low-order bits of address)
38	AD_3	ADDRESS (3)	
40	AD_4	ADDRESS (4)	

OUTPUTS (from CTL):

PIN	signal	full name	state definition
1	WE	WRITE EMEM (97)	0=Write E-MEM from G
2	SBWG	WRITE G (95)	0=Write G from memory
3	GENRST	GENERAL RESET (86)	0=General Reset
4	W23	WRITE ADDR 23 (85)	0=Write into SL
5	W22	WRITE ADDR 22 (84)	0=Write into CYL
6	W21	WRITE ADDR 21 (83)	0=Write into SR
7	W20	WRITE ADDR 20 (82)	0=Write into CYR
8	WGn	WRITE G NORMAL (81)	0=Write G (normal gates)
9	WBK	WRITE BNK (80)	0=Write BNK reg
10	RBK	READ BNK (70)	0=Read BNK reg
11	WS	WRITE S (46)	0=Write S
12	WP2	WRITE P2 (44)	0=Write P2
13	WPx	WRITE P NO RESET (43)	0=Write P (do not reset)
14	WP	WRITE P (42)	0=Write P
15	WGx	WRITE G NO RESET (37)	0=Write G (do not reset)
16	TP	TEST PARITY (30)	0=Test parity
17	RP2	READ PARITY 2 (14)	0=Read parity 2
18	RG	READ G (12)	0=Read G
19	GP	GEN PARITY (5)	0=Generate Parity
20	CLG	CLR G (2)	0=Clear G
21	CLK2	CLOCK2	1.024 MHz AGC clock 2 (normally low)
22	CLK1	CLOCK1	1.024 MHz AGC clock 1 (normally low)
23	NPURST	POWER UP RESET	0=reset, 1=normal operation.
24	SWCLK	DEBOUNCE CLOCK	low freq clk for switch debounce
25	FCLK	CLOCK MODE	1=free-running clk mode; 0=single clk mode

J103-CTL: CTL-to-I/O I/F

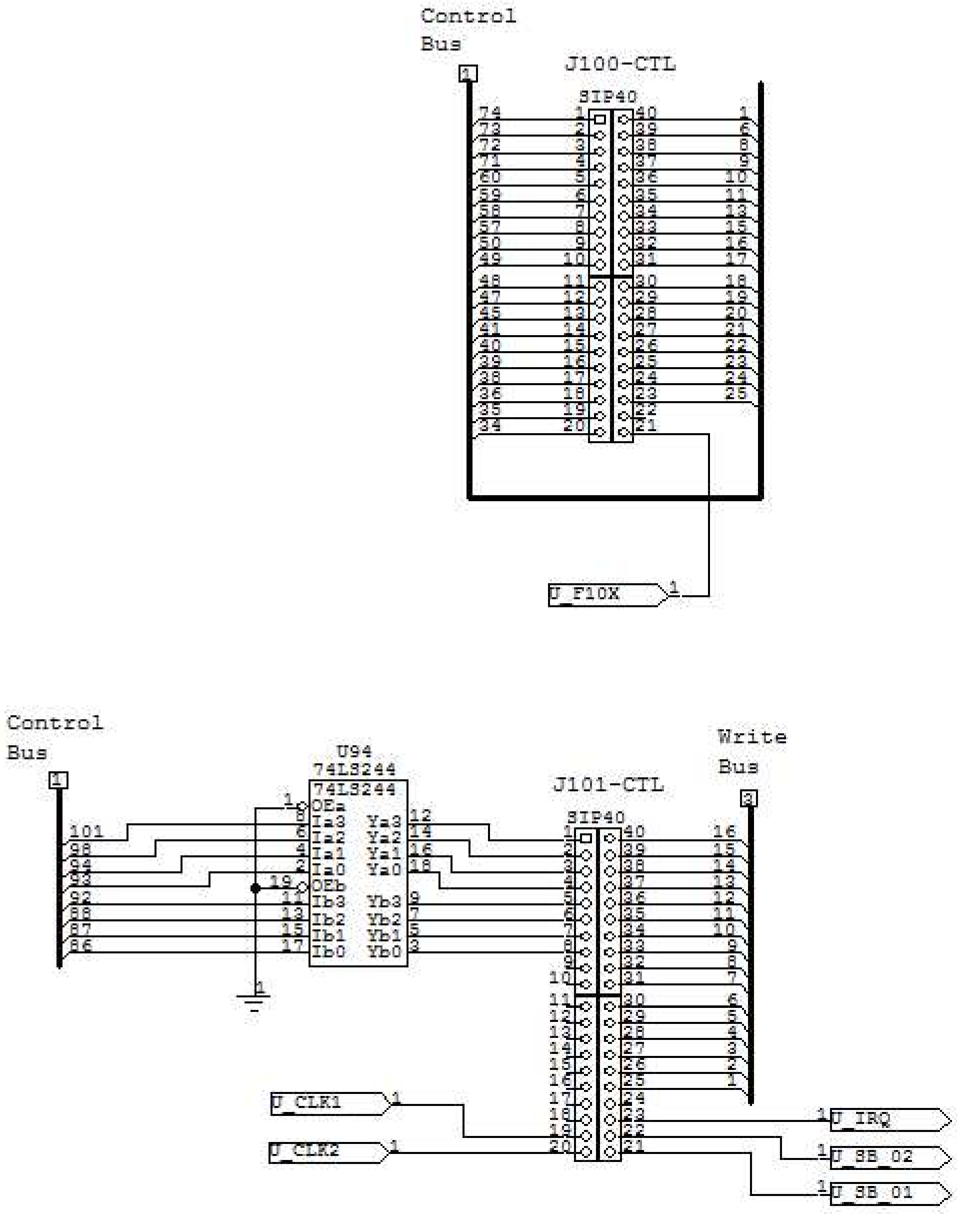
J100 is a 40-pin IDE ribbon cable that connects the CTL module to the I/O module.

INPUTS (to CTL):

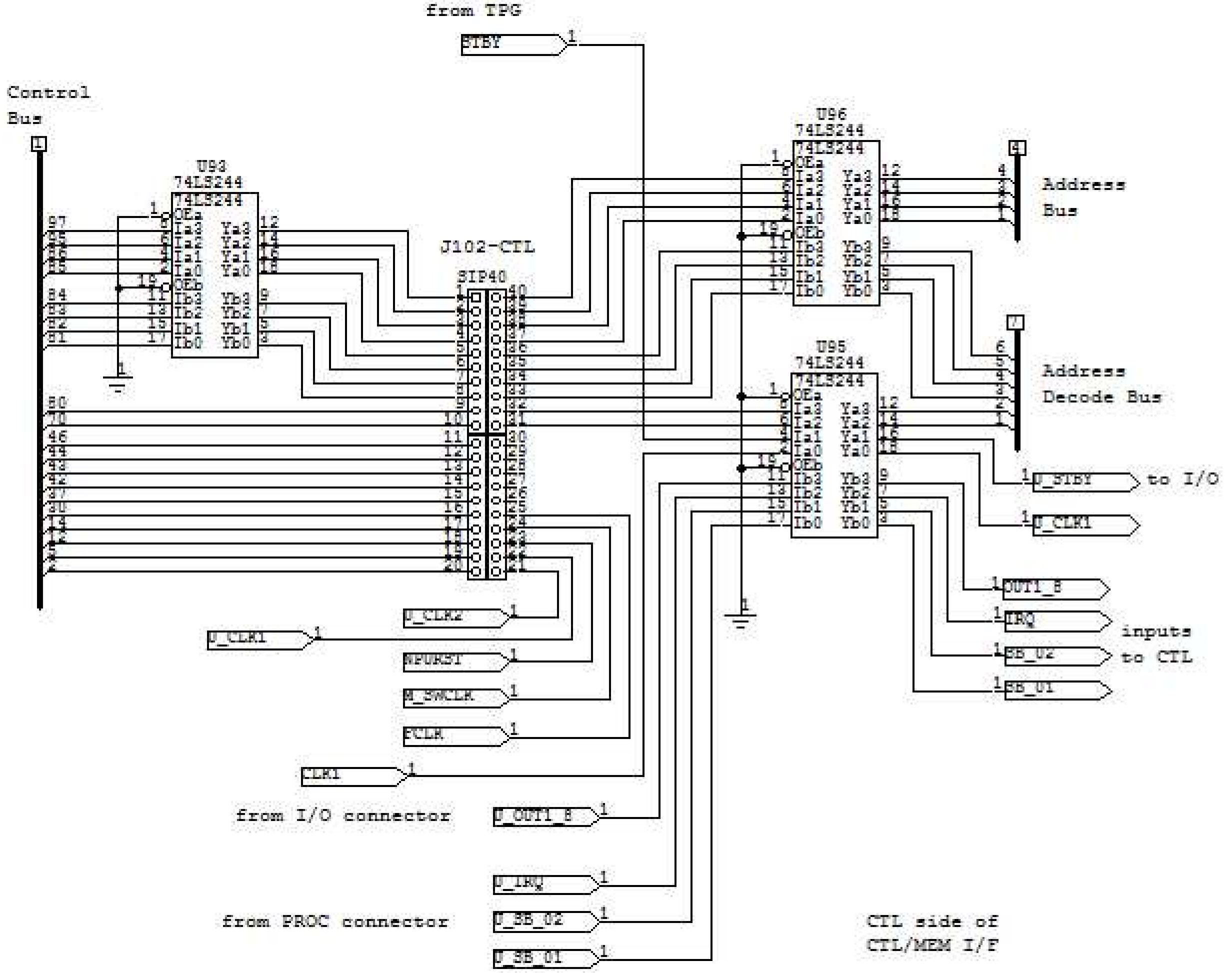
PIN	signal	full name	state definition
40	OUT1_8	STANDBY ENABLED	1=standby enabled; works with STANDBY ALLOWED switch

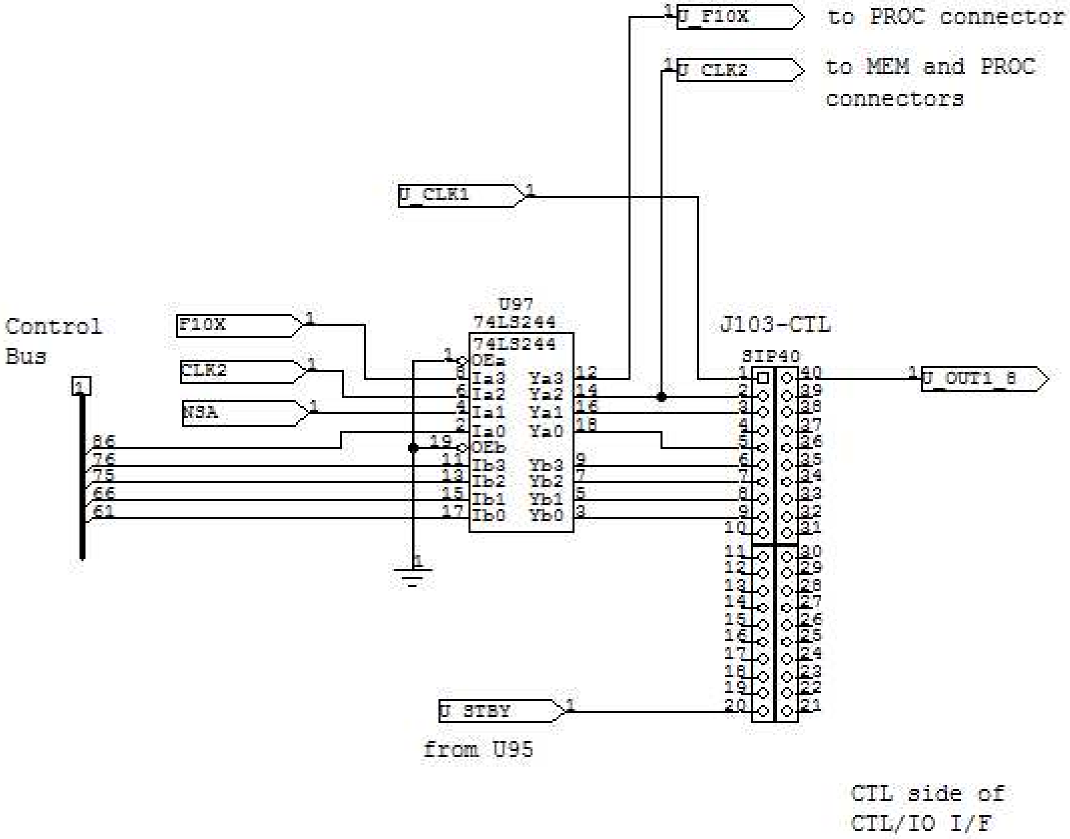
OUTPUTS (from CTL):

PIN	signal	full name	state definition
1	CLK1	CLOCK1	1.024 MHz AGC clock 1 (normally low)
2	CLK2	CLOCK2	1.024 MHz AGC clock 2 (normally low)
3	NSA	STANDBY ALLOWED	0=standby allowed
5	GENRST	GENERAL RESET (86)	0=clear the DSKY, OUT1, and OUT2.
6	WA11	WRITE OUT1 (76)	0=write into OUT1 from write bus
7	WA10	WRITE OUT0 (75)	0=write into OUT0 (DSKY) from write bus
8	RA11	READ OUT1 (66)	0=output OUT1 register to read bus
9	RA4	READ IN0 (61)	0=output IN0 register to read bus
20	STBY	STANDBY	0=AGC is in the standby state



CTL side of
CTL/PROC I/F

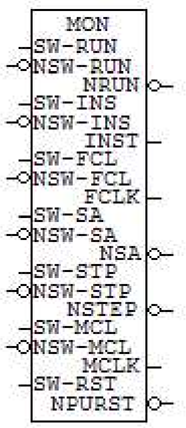

from TPG
STBY
Control
Bus
1
U93
74LS244
OEa
Ia3
Ia2
Ia1
Ia0
OEb
Ib3
Ib2
Ib1
Ib0
Ya3
Ya2
Ya1
Ya0
Yb3
Yb2
Yb1
Yb0
97
95
86
85
84
83
82
81
80
70
46
44
43
42
37
30
14
12
J102-CTL
SIP40
U96
74LS244
U95
74LS244
4
Address
Bus
7
Address
Decode Bus
U_STBY
to I/O
U_CLK1
OUT1_8
IRQ
SB_02
SB_01
inputs
to CTL
U_CLK2
U_CLK1
NPURST
N_SWCLR
FCLR
CLK1
from I/O connector
U_OUT1_8
U_IRQ
from PROC connector
U_SB_02
U_SB_01
CTL side of
CTL/MEM I/F

U_F10X
to PROC connector
U_CLK2
to MEM and PROC connectors
U_CLK1
Control Bus
F10X
CLK2
NSA
U97
74LS244
OEa
Ia3
Ia2
Ia1
Ia0
OEb
Ib3
Ib2
Ib1
Ib0
Ya3
Ya2
Ya1
Ya0
Yb3
Yb2
Yb1
Yb0
86
76
75
66
61
J103-CTL
SIP40
U_OUT1_8
U_STBY
from U95
CTL side of CTL/IO I/F

MON (AGC Monitor)

MON INPUTS:

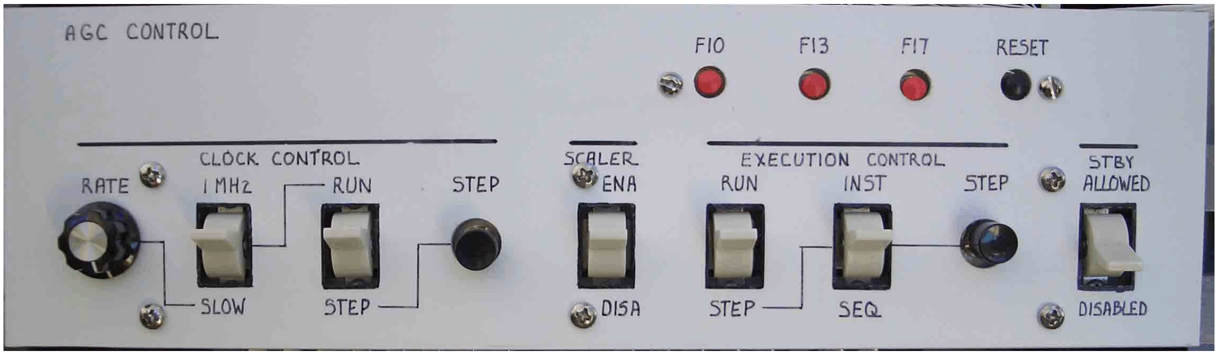
I/F	signal	full name	state definition
	SW-RUN	RUN/STEP SELECT SW	
	NSW-RUN		
	SW-INS	STEP MODE SW	
	NSW-INS		
	SW-FCL	CLOCK MODE SW	
	NSW-FCL		
	SW-SA	STANDBY ALLOWED SW	
	NSW-SA		
	SW-STP	INST STEP SW	
	NSW-STP		
	SW-MCL	CLOCK STEP SW	
	NSW-MCL		
	SW-RST	MASTER RESET SW	



MON OUTPUTS:

signal	full name	state definition
NRUN	RUN/HALT	0=run, 1=step
INST	INST STEP MODE	1=instruction step, 0=sequence step
NSTEP	SINGLE STEP	0=step (momentary)
NSA	STANDBY ALLOWED	0=standby allowed
MCLK	CLOCK STEP	1=step (momentary); triggers a single clock pulse. Ignored if FCLK is 1.
FCLK	CLOCK MODE	1=continuous clock output at 1.024 MHz, 0=single clock
NPURST	POWER UP RESET	0=reset, 1=normal operation.
SENAB	SCALER ENABLE	1=enable counting; 0=hold

CTL CONTROL PANEL SWITCHES



Clock Control:

RATE Controls the slow clock rate when the 1MHZ/SLOW switch is in the SLOW position.

1MHZ/SLOW Selects the free-running clock rate when the RUN/STEP switch is in the RUN position. The 1MHZ position gates a 2MHz signal into the 2-phased clock which produces a 1MHz 2-phased clock rate. The SLOW position gates a low frequency clock; the frequency is controlled by RATE.

RUN/STEP Selects a free-running (RUN) or a single-stepped (STEP) clock. In the RUN position, the rate is controlled by the 1MHZ/SLOW switch. In the STEP position, the clock steps each time the STEP button is depressed.

STEP Manually steps the clock when the RUN/STEP switch is in the STEP position. The clock is 2-phased, so each press steps an alternate phase.

Scaler:

ENAB/DISAB Enables or disables the SCALER.
F10 Manually triggers the F10 stage of the SCALER.
F13 Manually triggers the F13 stage of the SCALER.
F17 Manually triggers the F17 stage of the SCALER.

Execution Control:

RUN/STEP In the RUN position, the AGC free-runs at a rate determined by the clock controls. In the STEP position, the AGC single-steps to the next instruction or next sequence when the STEP button is depressed; the rate is determined by the clock controls.

INST/SEQ Selects whether the AGC single-steps all the way to the next instruction (INST) or just to the next sequence (SEQ). Some instructions, such as TC, have a single sequence; on these instructions, the switch has the same effect in either position.

STEP Single-steps the AGC when the RUN/STEP switch is in the STEP position.

RESET Reset the entire AGC. Puts the TPG into the standby state.

Standby:

ALLOW/DISA The ALLOWED position authorizes the AGC software to put the AGC in standby mode. The AGC is released from standby mode by a signal from F17 in the SCALER. If the scaler switch is in the DISAB position, the scaler is disabled, and the AGC will remain in the standby state. When the standby switch is in the DISAB position, the standby mode is inhibited.

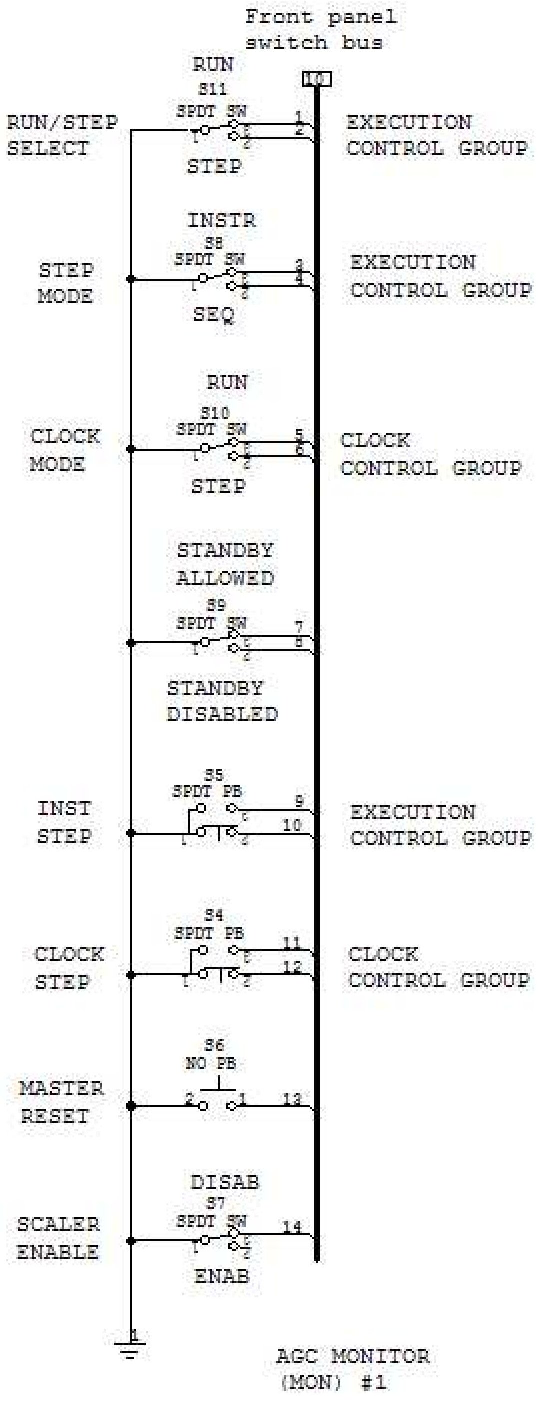
Normal Operation:

Set the following switch positions for nominal operation:

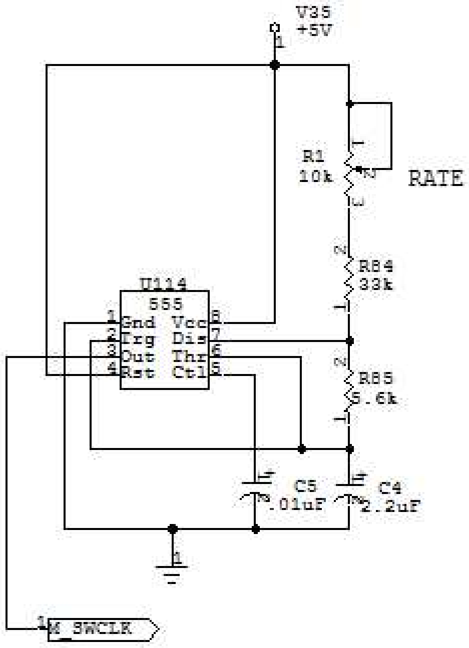
switch	position
1MHZ/SLOW	1MHZ
RUN/STEP	RUN (clock control)
ENAB/DISAB	ENAB (scaler)
RUN/STEP	RUN (execution control)
ALLOW/DISA	ALLOWED (standby)

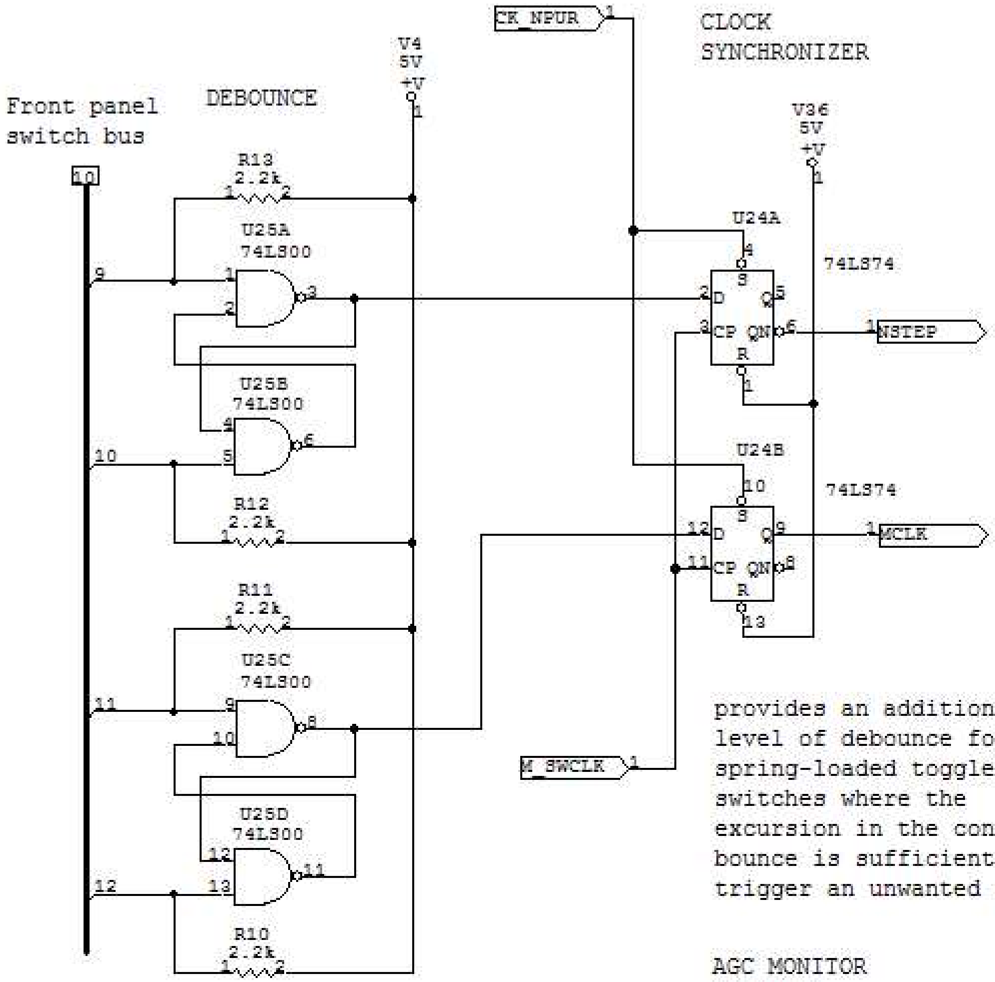
CTL CONTROL SWITCH CONNECTIONS

PIN	signal	state definition
1	BUS#10, line 1	Execution control: RUN/STEP
2	BUS#10, line 2	
3	BUS#10, line 3	Execution control: INST/SEQ
4	BUS#10, line 4	
5	BUS#10, line 5	Clock control: RUN/STEP
6	BUS#10, line 6	
7	BUS#10, line 7	STANDBY ALLOWED/DISABLED
8	BUS#10, line 8	
9	BUS#10, line 9	Execution control: STEP
10	BUS#10, line 10	
11	BUS#10, line 11	Clock control: STEP
12	BUS#10, line 12	
13	BUS#10, line 13	RESET
14	BUS#10, line 14	SCALER DISAB
15	RATE	(SLOW) CLOCK CONTROL RATE
16	1MHZ	Clock control: 1MHZ/SLOW
17	SLOW	
18	F10	MANUAL TRIGGER F10
19	F13	MANUAL TRIGGER F13
20	F17	MANUAL TRIGGER F17


Front panel
switch bus
10
RUN
S11
SPDT SW
RUN/STEP
SELECT
1
2
EXECUTION
CONTROL GROUP
STEP
INSTR
S8
SPDT SW
STEP
MODE
3
4
EXECUTION
CONTROL GROUP
SEQ
RUN
S10
SPDT SW
CLOCK
MODE
5
6
CLOCK
CONTROL GROUP
STEP
STANDBY
ALLOWED
S9
SPDT SW
7
8
STANDBY
DISABLED
S5
SPDT PB
INST
STEP
9
10
EXECUTION
CONTROL GROUP
S4
SPDT PB
CLOCK
STEP
11
12
CLOCK
CONTROL GROUP
S6
NO PB
MASTER
RESET
2
1
13
DISAB
S7
SPDT SW
SCALER
ENABLE
14
ENAB
AGC MONITOR
(MON) #1

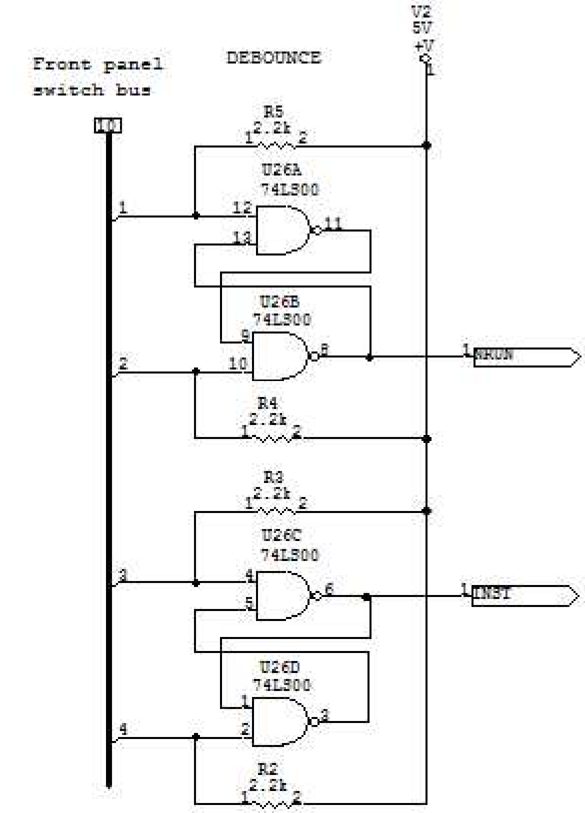
AGC MONITOR
SWITCH DEBOUNCE CLOCK

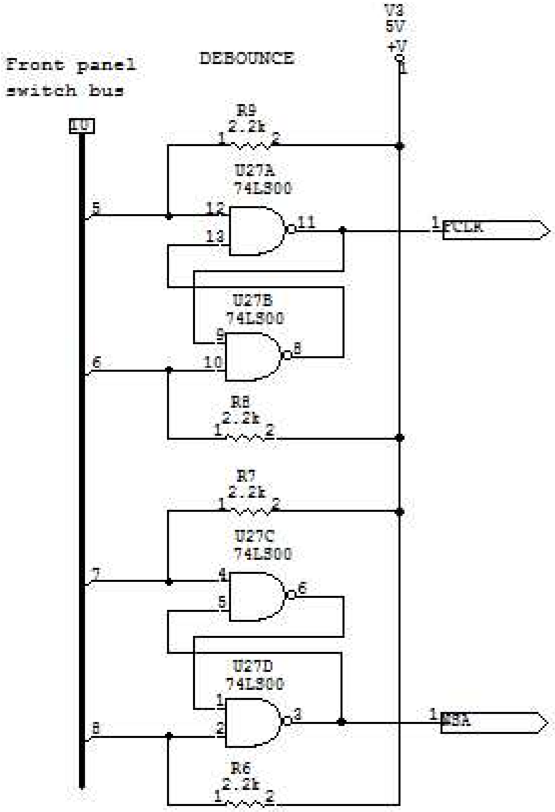


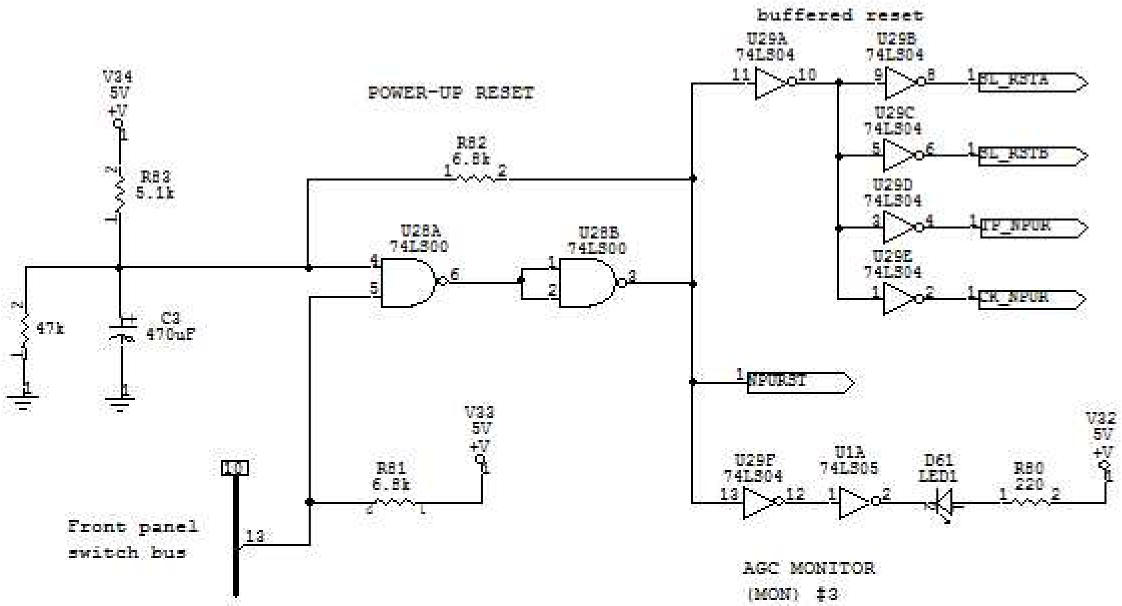


provides an additional level of debounce for spring-loaded toggle switches where the excursion in the contact bounce is sufficient to trigger an unwanted pulse

AGC MONITOR
(MON) #2



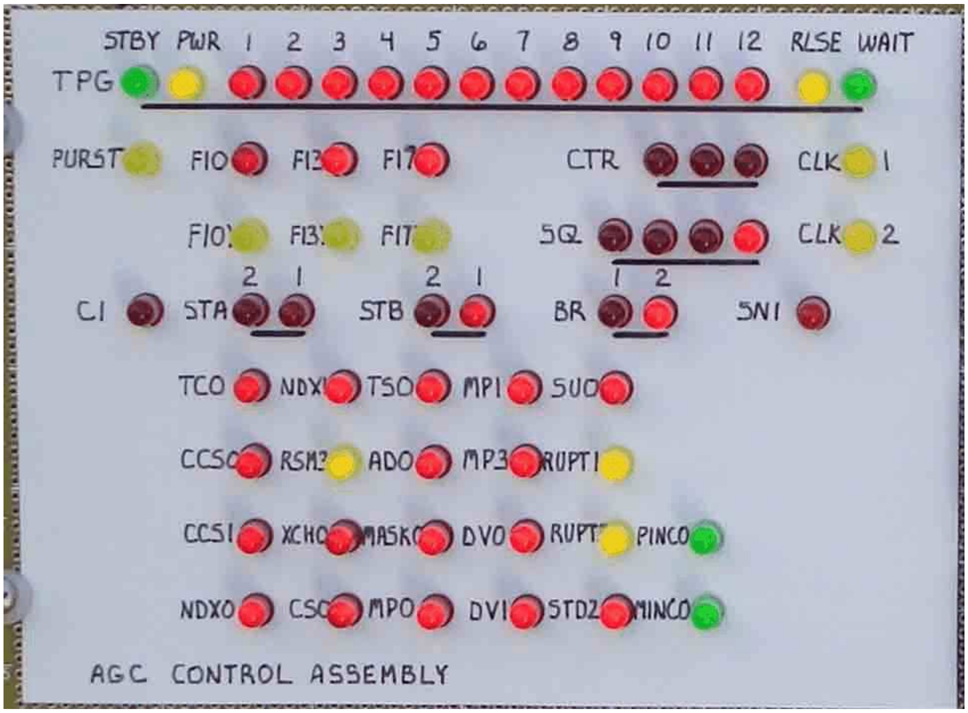




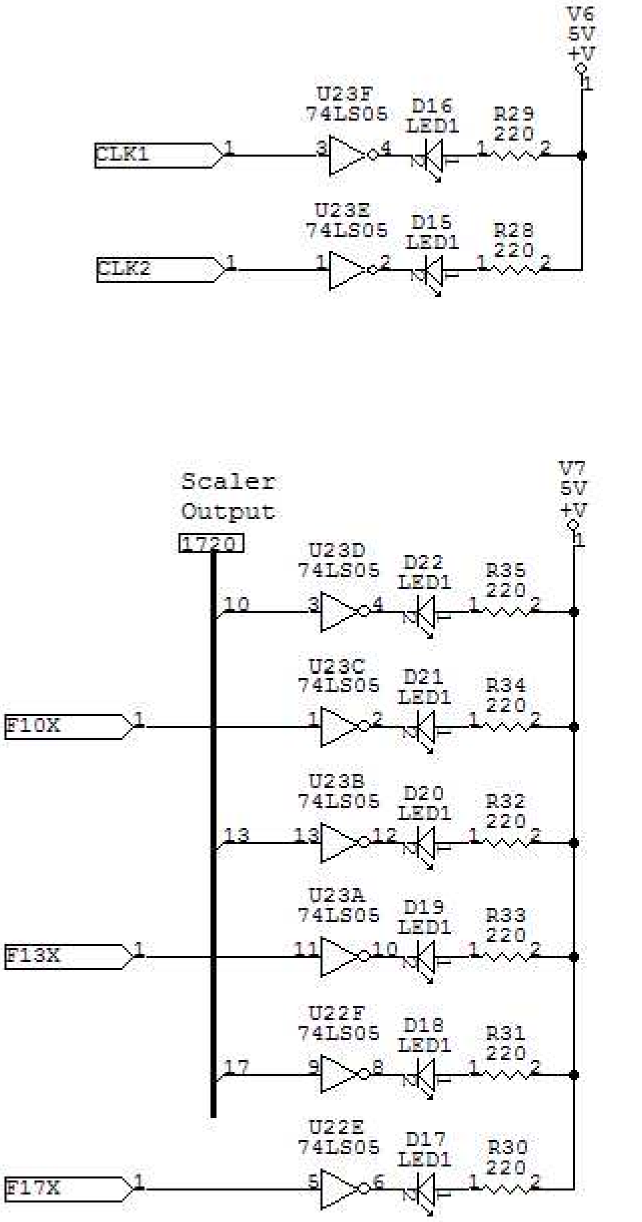
AGC MONITOR
(MON) #3

CTL INDICATORS

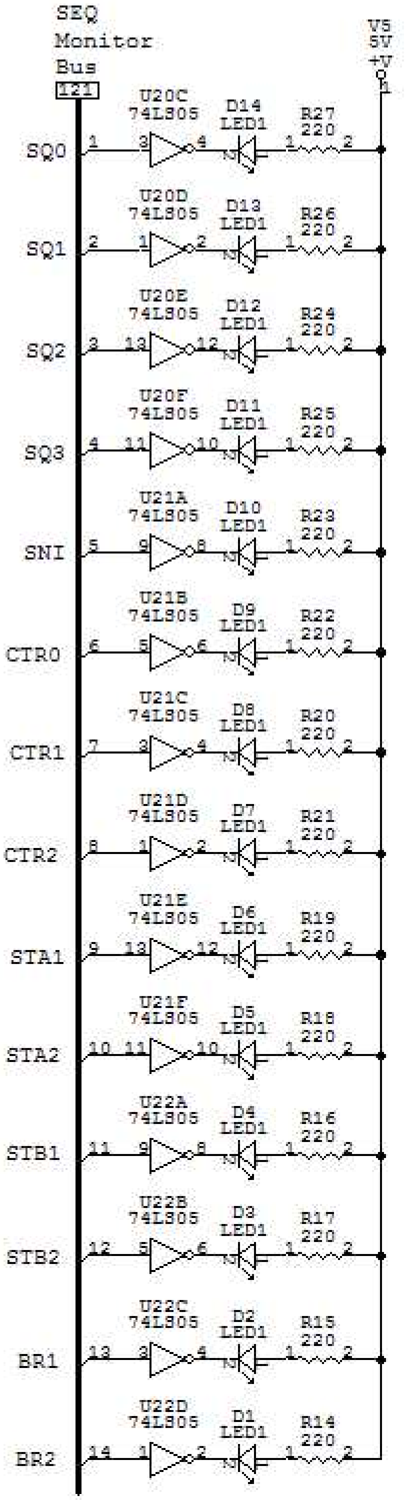
The CTL module has a panel of indicator lamps (LEDs) to show the state of CTL registers and critical logic signals.


STBY PWR 1 2 3 4 5 6 7 8 9 10 11 12 RLSE WAIT
TPG
PURST F10 F13 F17 CTR CLK 1
F10 F13 F17 SQ CLK 2
2 1 2 1 1 2
CI STA STB BR SNI
TCO NDX TSO MP1 SUO
CCSO RSM3 ADO MP3 RUPT1
CCS1 XCHO MASKO DVO RUPT PINCO
NDXO CSO MPO DV1 STD2 MINCO
AGC CONTROL ASSEMBLY

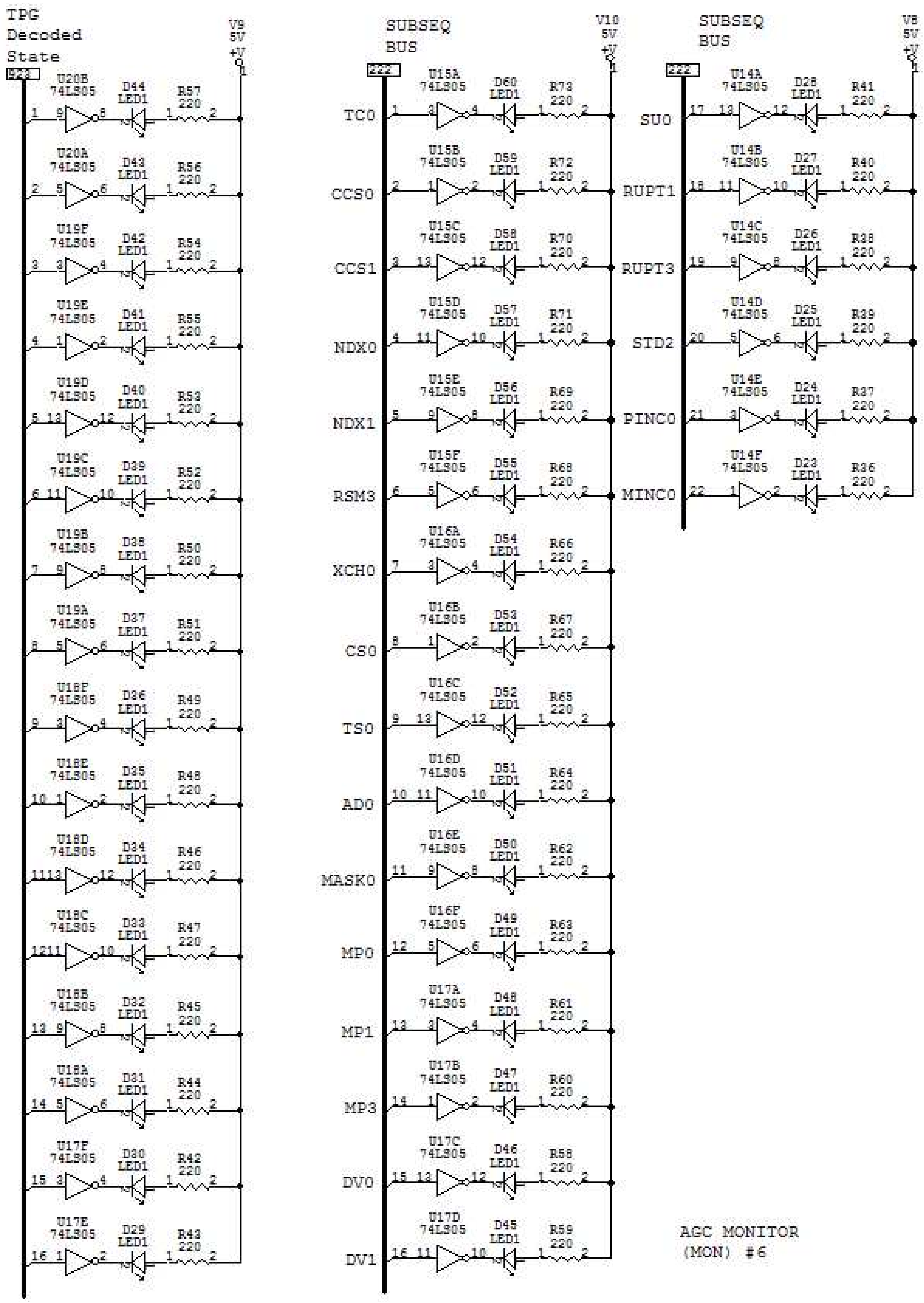
These indicator lamps show the current state of all registers and some additional, important logic signals produced by the CTL module. The matrix of lamps in the lower portion show active subsequences. Much of the control logic is negative (active low) where an illuminated lamp means that the signal is NOT asserted. At the time the photo was taken the AGC was running the COLOSSUS 249 flight software load, executing Verb 16, Noun 36: a monitor verb which displays the AGC real time clock.



AGC MONITOR
(MON) #4


SEQ
Monitor
Bus
121
V5
5V
+V
1
SQ0 1 U20C 74LS05 3 4 D14 LED1 R27 220 1 2
SQ1 2 U20D 74LS05 1 2 D13 LED1 R26 220 1 2
SQ2 3 U20E 74LS05 13 12 D12 LED1 R24 220 1 2
SQ3 4 U20F 74LS05 11 10 D11 LED1 R25 220 1 2
SNI 5 U21A 74LS05 9 8 D10 LED1 R23 220 1 2
CTR0 6 U21B 74LS05 5 6 D9 LED1 R22 220 1 2
CTR1 7 U21C 74LS05 3 4 D8 LED1 R20 220 1 2
CTR2 8 U21D 74LS05 1 2 D7 LED1 R21 220 1 2
STA1 9 U21E 74LS05 13 12 D6 LED1 R19 220 1 2
STA2 10 U21F 74LS05 11 10 D5 LED1 R18 220 1 2
STB1 11 U22A 74LS05 9 8 D4 LED1 R16 220 1 2
STB2 12 U22B 74LS05 5 6 D3 LED1 R17 220 1 2
BR1 13 U22C 74LS05 3 4 D2 LED1 R15 220 1 2
BR2 14 U22D 74LS05 1 2 D1 LED1 R14 220 1 2

AGC MONITOR
(MON) #5


TPG
Decoded
State
V9
5V
+V
U20B 74LS05 1 9 8 D44 LED1 R57 220 1 2
U20A 74LS05 2 5 6 D43 LED1 R56 220 1 2
U19F 74LS05 3 3 4 D42 LED1 R54 220 1 2
U19E 74LS05 4 1 2 D41 LED1 R55 220 1 2
U19D 74LS05 5 13 12 D40 LED1 R53 220 1 2
U19C 74LS05 6 11 10 D39 LED1 R52 220 1 2
U19B 74LS05 7 9 8 D38 LED1 R50 220 1 2
U19A 74LS05 8 5 6 D37 LED1 R51 220 1 2
U18F 74LS05 9 3 4 D36 LED1 R49 220 1 2
U18E 74LS05 10 1 2 D35 LED1 R48 220 1 2
U18D 74LS05 11 13 12 D34 LED1 R46 220 1 2
U18C 74LS05 12 11 10 D33 LED1 R47 220 1 2
U18B 74LS05 13 9 8 D32 LED1 R45 220 1 2
U18A 74LS05 14 5 6 D31 LED1 R44 220 1 2
U17F 74LS05 15 3 4 D30 LED1 R42 220 1 2
U17E 74LS05 16 1 2 D29 LED1 R43 220 1 2
SUBSEQ
BUS
222
V10
5V
+V
TC0 1 U15A 74LS05 3 4 D60 LED1 R73 220 1 2
CCS0 2 U15B 74LS05 1 2 D59 LED1 R72 220 1 2
CCS1 3 U15C 74LS05 13 12 D58 LED1 R70 220 1 2
NDX0 4 U15D 74LS05 11 10 D57 LED1 R71 220 1 2
NDX1 5 U15E 74LS05 9 8 D56 LED1 R69 220 1 2
RSM3 6 U15F 74LS05 5 6 D55 LED1 R68 220 1 2
XCH0 7 U16A 74LS05 3 4 D54 LED1 R66 220 1 2
CS0 8 U16B 74LS05 1 2 D53 LED1 R67 220 1 2
TS0 9 U16C 74LS05 13 12 D52 LED1 R65 220 1 2
AD0 10 U16D 74LS05 11 10 D51 LED1 R64 220 1 2
MASK0 11 U16E 74LS05 9 8 D50 LED1 R62 220 1 2
MP0 12 U16F 74LS05 5 6 D49 LED1 R63 220 1 2
MP1 13 U17A 74LS05 3 4 D48 LED1 R61 220 1 2
MP3 14 U17B 74LS05 1 2 D47 LED1 R60 220 1 2
DV0 15 U17C 74LS05 13 12 D46 LED1 R58 220 1 2
DV1 16 U17D 74LS05 11 10 D45 LED1 R59 220 1 2
SUBSEQ
BUS
222
V8
5V
+V
SU0 17 U14A 74LS05 13 12 D28 LED1 R41 220 1 2
RUPT1 18 U14B 74LS05 11 10 D27 LED1 R40 220 1 2
RUPT3 19 U14C 74LS05 9 8 D26 LED1 R38 220 1 2
STD2 20 U14D 74LS05 5 6 D25 LED1 R39 220 1 2
PINC0 21 U14E 74LS05 3 4 D24 LED1 R37 220 1 2
MINC0 22 U14F 74LS05 1 2 D23 LED1 R36 220 1 2

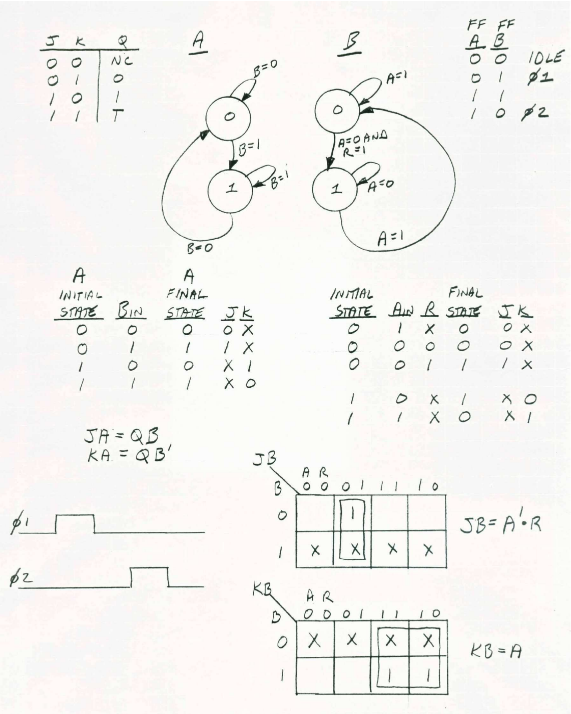
AGC MONITOR
(MON) #6

CLK (Clock)

The original AGC used asynchronous logic driven by a 4-phase clock. This recreation uses synchronous logic driven by a 2-phase non-overlapping clock. The synchronous clock logic was designed to produce the following state transitions:

FFA	FFB	
0	0	idle state
0	1	decoded for phase 1
1	1	
1	0	decoded for phase 2

The outputs of FFA and FFB are decoded by combinational logic to produce the non-overlapping phase 1 and phase 2 clock signals. The sequence is arranged so there is a single logic level transition for each state transition to prevent transients.



J	K	Q
0	0	NC
0	1	0
1	0	1
1	1	T

FF A	FF B	
0	0	IDLE
0	1	ø1
1	1	
1	0	ø2

A INITIAL STATE	B IN	A FINAL STATE	J	K
0	0	0	0	X
0	1	1	1	X
1	0	0	X	1
1	1	1	X	0

INITIAL STATE	A IN	R	FINAL STATE	J	K
0	1	X	0	0	X
0	0	0	0	0	X
0	0	1	1	1	X
1	0	X	1	X	0
1	1	X	0	X	1

JA = QB
KA = QB'

JB

B \ A R	00	01	11	10
0		1		
1	X	X	X	X

$JB = A' \cdot R$

KB

B \ A R	00	01	11	10
0	X	X	X	X
1			1	1

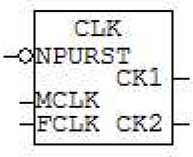
KB = A

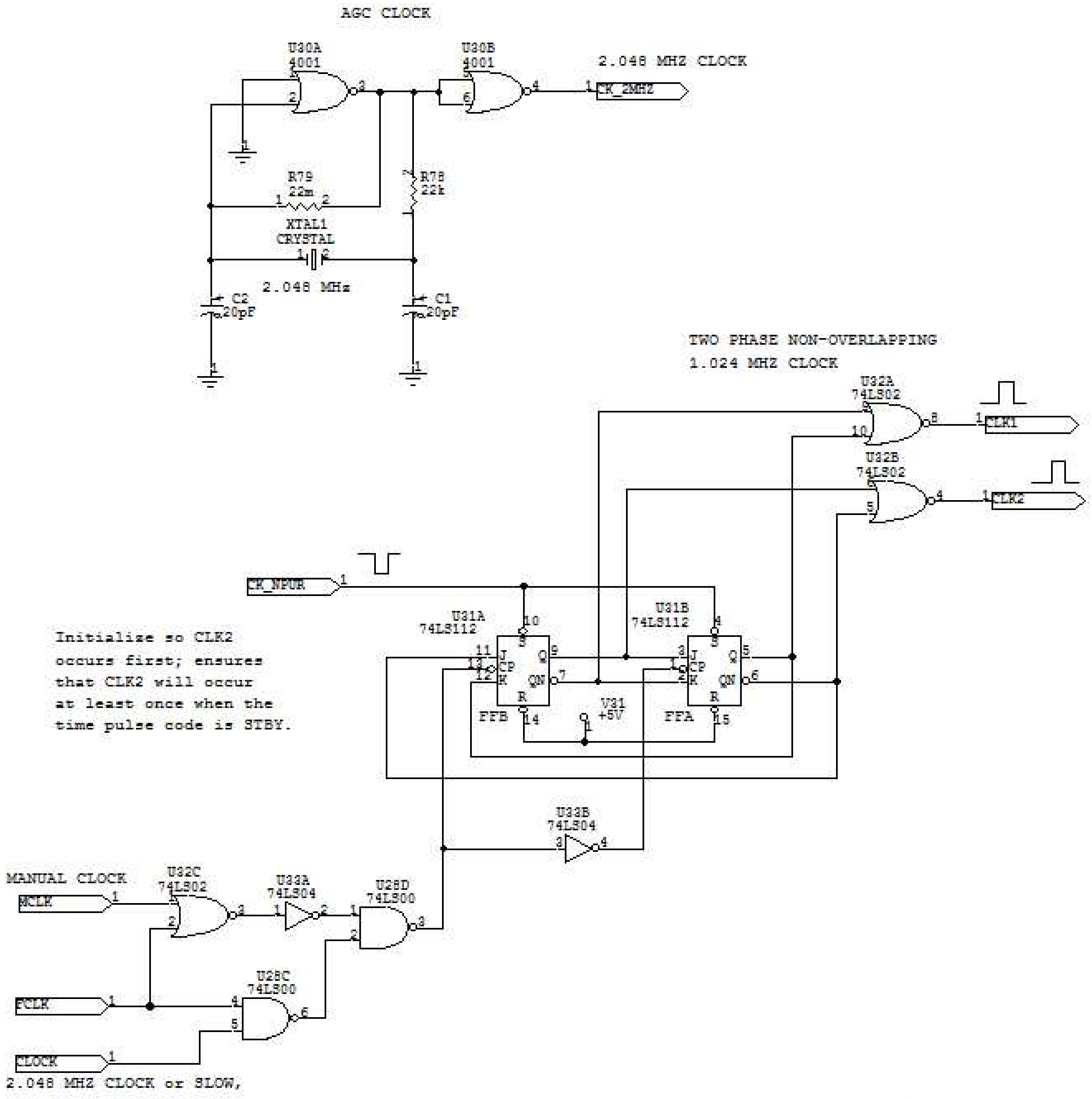
CLK INPUTS:

I/F	signal	full name	state definition
MON:			
	MCLK	CLOCK STEP	1=step (momentary); triggers a single clock pulse. Ignored if FCLK is 1.
	FCLK	CLOCK MODE	1=continuous clock output at 1.024 MHz, 0=single clock whenever MCLK is 1.
	NPURST	POWER UP RESET	0=power up reset

OUTPUTS:

signal	full name	state definition
CLK1	CLOCK1	1.024 MHz AGC clock 1 (normally low)
CLK2	CLOCK2	1.024 MHz AGC clock 2 (normally low)



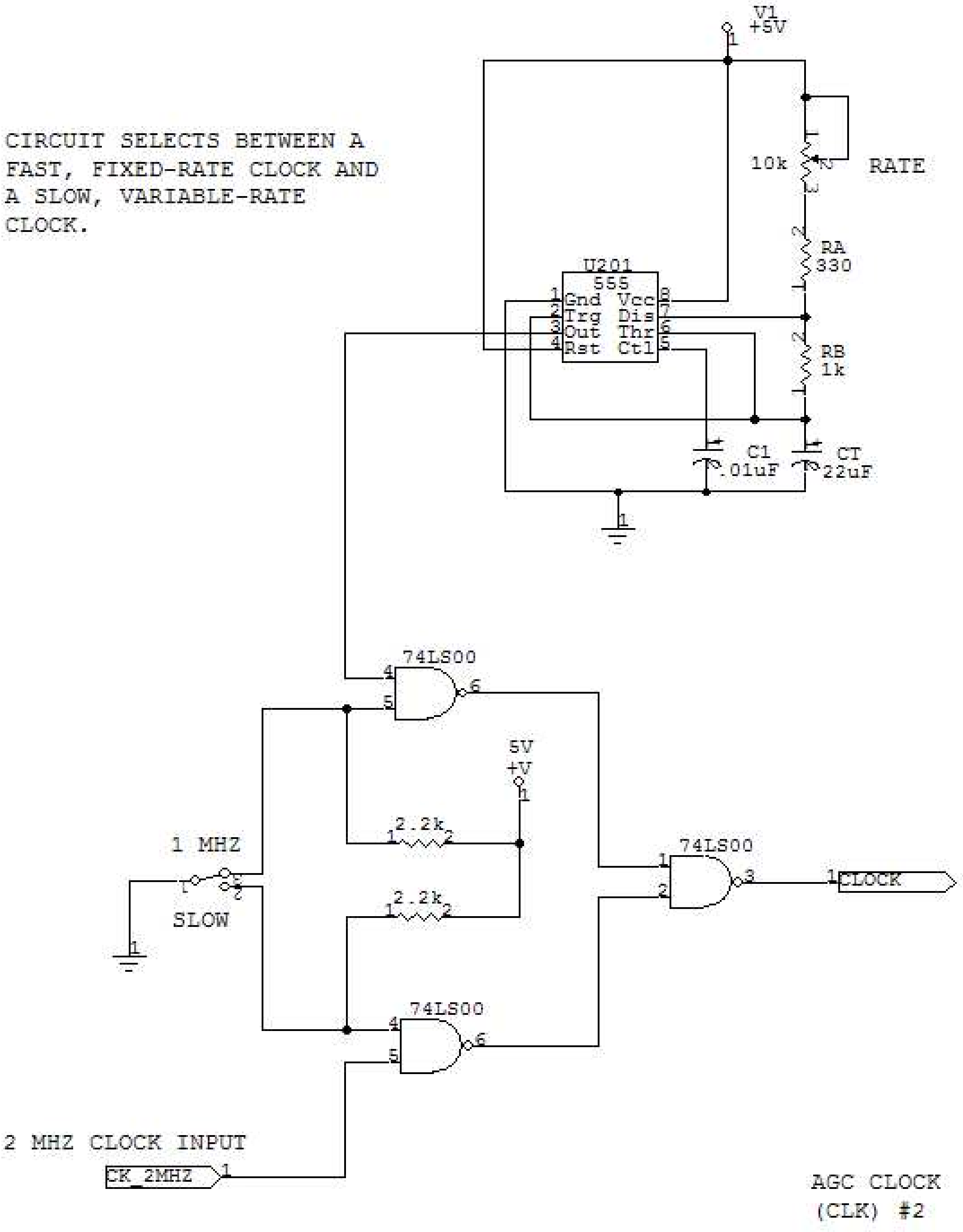


Initialize so CLK2
occurs first; ensures
that CLK2 will occur
at least once when the
time pulse code is STBY.

2.048 MHZ CLOCK or SLOW,
VARIABLE RATE CLOCK

FCLK=1: free-running
clock;
FCLK=0: manually-
stepped clock

AGC CLOCK
(CLK) $1



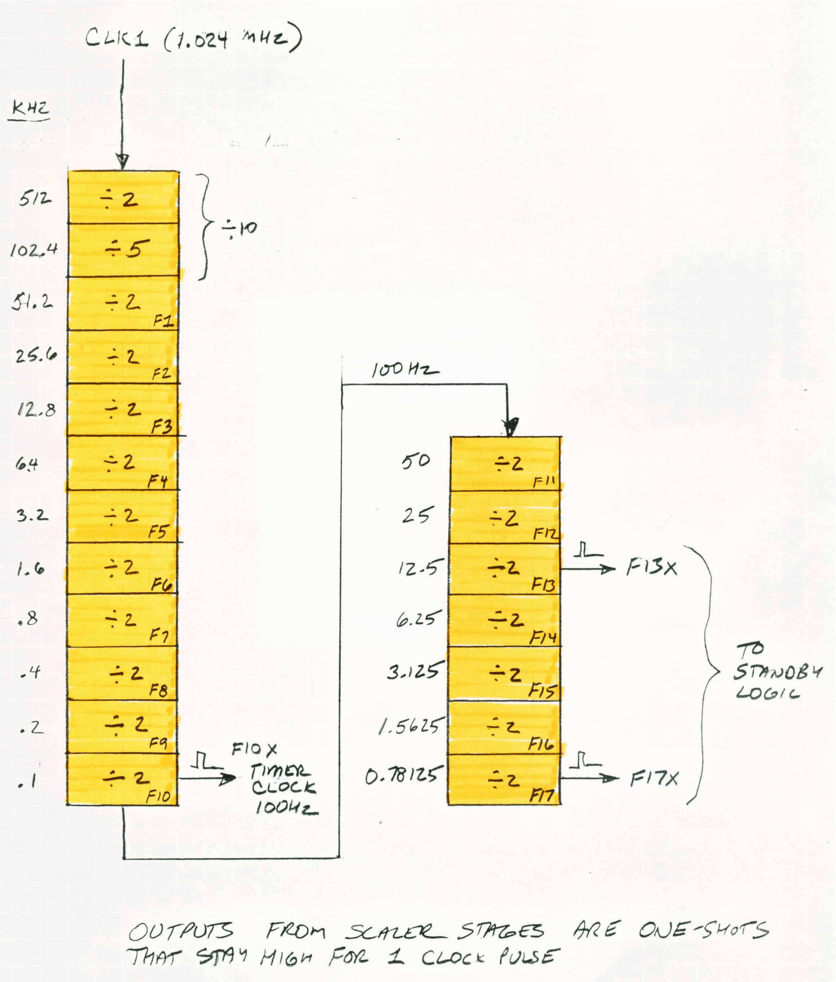
CIRCUIT SELECTS BETWEEN A FAST, FIXED-RATE CLOCK AND A SLOW, VARIABLE-RATE CLOCK.

AGC CLOCK
(CLK) #2

SCL (Scaler)

The 1.024 MHz AGC clock is divided by two to produce a 512 kHz signal called the MASTER FREQUENCY; this signal is further divided through a SCALER, first by five to produce a 102.4 kHz signal. This is then divided by two through 17 successive stages called F1 (51.2 kHz) through F17 (0.78125 Hz). The F10 stage (100 Hz) is fed back into the AGC to increment the real-time clock and other priority counters in the PROC module. The F17 stage is used to intermittently run the AGC when it operates in the STANDBY mode.

The F10, F13, and F17 outputs of the SCALER feed into a synchronous one-shot that produces a short output pulse on the rising edge of the input.



OUTPUTS FROM SCALER STAGES ARE ONE-SHOTS
THAT STAY HIGH FOR 1 CLOCK PULSE

SCALER (SCL) ONE-SHOT LOGIC DESIGN

Boolean operators:
* (AND), + (OR), ' (NOT)

Scaler Divide-by-10 Excitation Table:
The divide-by-10 state machine is implemented with a 74161 parallel counter. Control Mode for 74161 Parallel Counter:

NPE	CET	
0	x	LOAD
1	1	COUNT
1	0	HOLD

	Current	Next			Par In
State	D C B A	D C B A	NPE	CET	D C B A
0	0 0 0 0	0 0 0 1	1	1	
1	0 0 0 1	0 0 1 0	1	1	
2	0 0 1 0	0 0 1 1	1	1	
3	0 0 1 1	0 1 0 0	1	1	
4	0 1 0 0	0 1 0 1	1	1	
5	0 1 0 1	0 1 1 0	1	1	
6	0 1 1 0	0 1 1 1	1	1	
7	0 1 1 1	1 0 0 0	1	1	
8	1 0 0 0	1 0 0 1	1	1	
9	1 0 0 1	0 0 0 0	0	x	0 0 0 0

NPE = (D * A)'
CET = 1

One-shot Excitation Table:
The one-shot state machine is implemented with two J-K FFs.

JK flip-flop excitation table:

J	K	
0	0	Qn-1 (no change)
0	1	0
1	0	1
1	1	Qn-1' (toggle)

FN is the one-shot trigger; Q(A) is the one-shot output.

	Cur	Inp	Nxt	ffB	ffA
State	B A	FN	B A	J K	J K
IDLE	0 0	0	0 0	0 x	0 x
	0 0	1	0 1	0 x	1 x

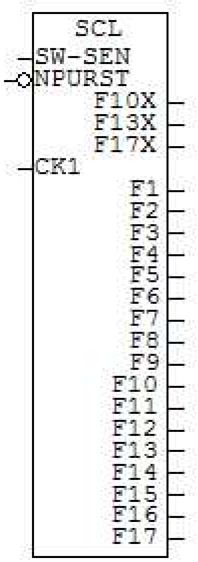
HIGH	0 1	0	1 0	1 x	x 1
	0 1	1	1 0	1 x	x 1
LOW	1 0	0	0 0	x 1	0 x
	1 0	1	1 0	x 0	0 x
UNUSED	1 1	0	x x	x x	x x
	1 1	1	x x	x x	x x

The excitation table and logic equations were derived from the CLK subsystem state machine which also performs a one-shot function. The CLK subsystems's FCLK signal was factored out by setting it to zero. The MCLK input is the one-shot trigger; it was renamed FN.

ff(B) J = QA
ff(B) K = FN'

ff(A) J = QB' * FN
ff(A) K = QA

SCL INPUTS:

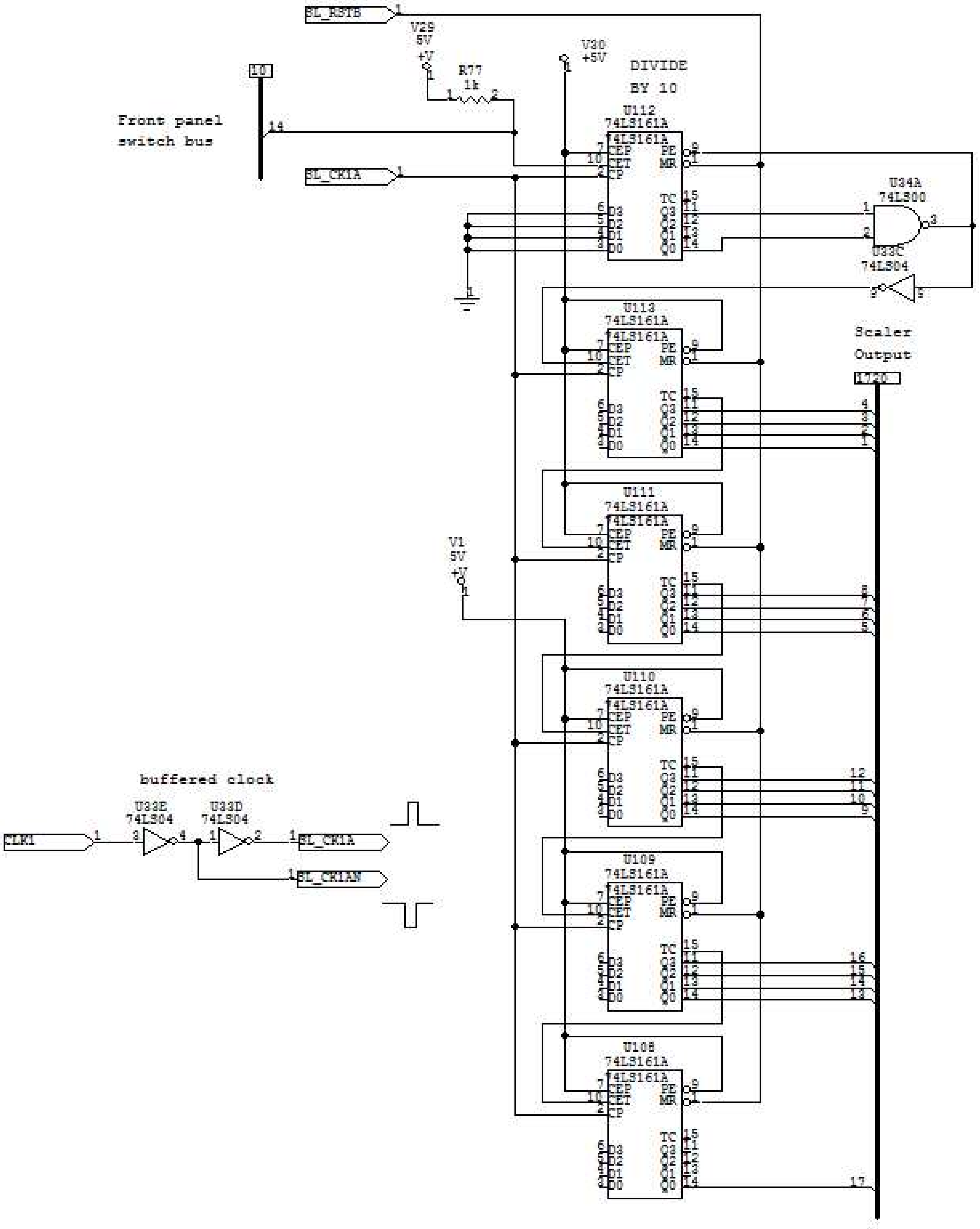


I/F	signal	full name	state definition
CLK:			
	CLK1	CLOCK1	1.024 MHz AGC clock
MON:			
	SW-SEN	SCALER ENABLE	1=enable counting; 0=hold
	NPURST	POWER UP RESET	0=power up reset

SCL OUTPUTS:

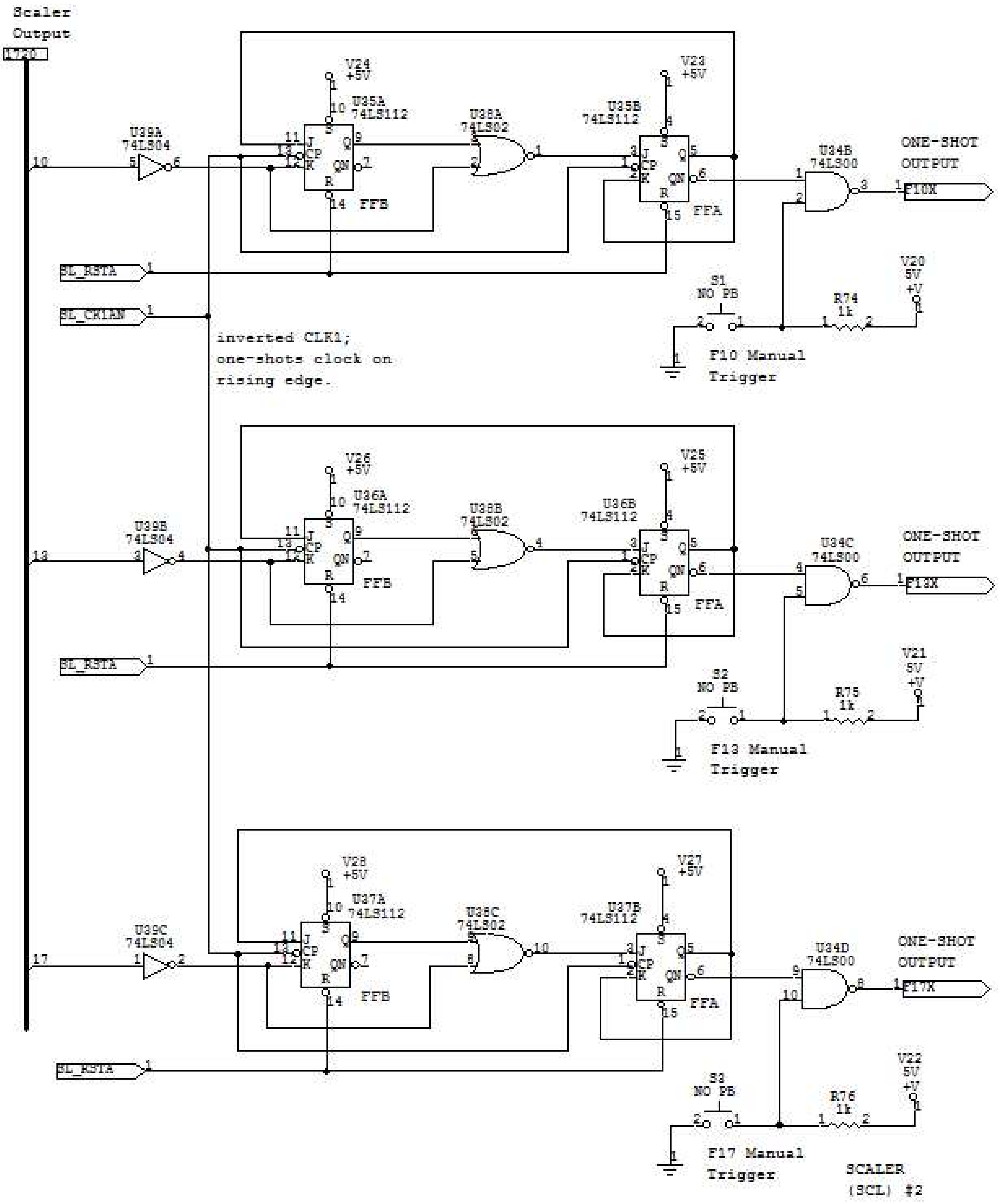
signal	full name	state definition
F10X	F10 SCALER ONESHOT	1=timed out (100.0 Hz)
F13X	F13 SCALER ONESHOT	1=timed out (12.5 Hz)
F17X	F17 SCALER ONESHOT	1=timed out (0.78125 Hz)
F1	SCALER OUT F1	51.2 KHz square wave
F2	SCALER OUT F2	25.6 KHz square wave
F3	SCALER OUT F3	12.8 KHz square wave
F4	SCALER OUT F4	6,4 KHz square wave
F5	SCALER OUT F5	3.2 KHz square wave
F6	SCALER OUT F6	1.6 KHz square wave
F7	SCALER OUT F7	0.8 KHz square wave
F8	SCALER OUT F8	0.4 KHz square wave
F9	SCALER OUT F9	0.2 KHz square wave
F10	SCALER OUT F10	0.1 KHz square wave (100 Hz)
F11	SCALER OUT F11	50.0 Hz square wave
F12	SCALER OUT F12	25.0 Hz square wave
F13	SCALER OUT F13	12.5 Hz square wave
F14	SCALER OUT F14	6.25 Hz square wave
F15	SCALER OUT F15	3.125 Hz square wave
F16	SCALER OUT F16	1.5625 Hz square wave
F17	SCALER OUT F17	0.78125 Hz square wave

Note: One shot outputs are active for one clock pulse.
State transitions occur on the rising edge of CLK1

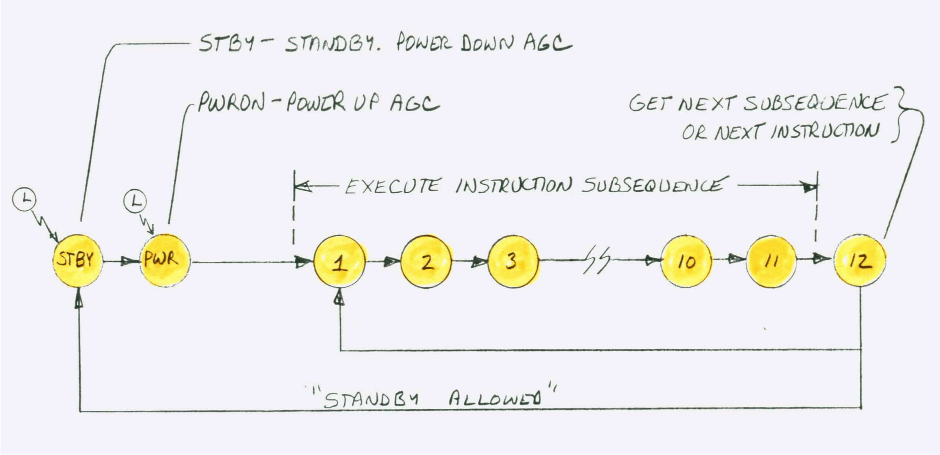

SL_RSTB
V29
5V
+V
R77
1k
V30
+5V
DIVIDE
BY 10
U112
74LS161A
Front panel
switch bus
10
14
SL_CK1A
CEP
CET
CP
D3
D2
D1
D0
PE
MR
TC
Q3
Q2
Q1
Q0
U34A
74LS00
U33C
74LS04
U113
74LS161A
Scaler
Output
1720
U111
74LS161A
V1
5V
+V
U110
74LS161A
buffered clock
CLK1
U33E
74LS04
U33D
74LS04
SL_CK1A
SL_CK1AN
U109
74LS161A
U108
74LS161A

SCALER

(SCL) #1


Scaler
Output
1720
10
13
17
U39A
74LS04
U39B
74LS04
U39C
74LS04
V24
+5V
U35A
74LS112
U38A
74LS02
V23
+5V
U35B
74LS112
FFB
FFA
U34B
74LS00
ONE-SHOT
OUTPUT
F10X
SL_RSTA
SL_CK1AN
S1
NO PB
V20
5V
+V
R74
1k
F10 Manual
Trigger
inverted CLK1;
one-shots clock on
rising edge.
V26
+5V
U36A
74LS112
U38B
74LS02
V25
+5V
U36B
74LS112
U34C
74LS00
ONE-SHOT
OUTPUT
F13X
S2
NO PB
V21
5V
+V
R75
1k
F13 Manual
Trigger
V28
+5V
U37A
74LS112
U38C
74LS02
V27
+5V
U37B
74LS112
U34D
74LS00
ONE-SHOT
OUTPUT
F17X
S3
NO PB
V22
5V
+V
R76
1k
F17 Manual
Trigger
SCALER
(SCL) #2

TPG (Time Pulse Generator)



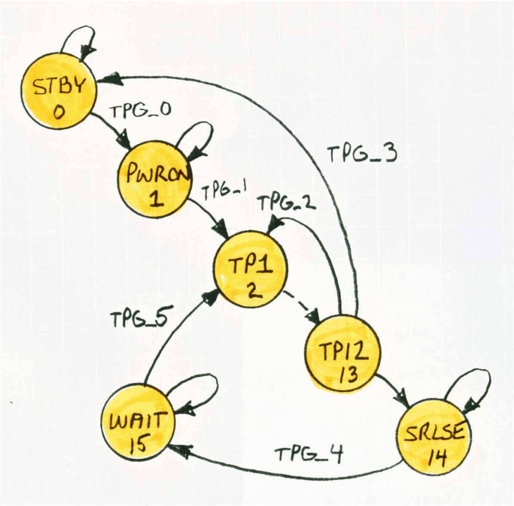
TIME PULSE GENERATOR (TPG) LOGIC DESIGN

Boolean operators:

* (AND), + (OR), ' (NOT)

<u>TPG Internal Control Signals</u>:

These are local to the TPG; if the signal is asserted, TPG makes a state transition. These decoded signals are the inputs to the excitation table.



TPG_0 = PURST' * (FCLK' + F17X)

TPG_1 = FCLK' + F13X

TPG_2 = RUN + (SNI' * INST)

TPG_3 = SNI * OUT1_8 * SA

TPG_4 = STEP'

TPG_5 = STEP + RUN

<u>TPG Excitation Table</u>:

Control Mode for 74161 Parallel Counter:

NPE	CET	
0	x	LOAD
1	1	COUNT
1	0	HOLD

*denotes active low

	Current	*Current	TPG_x	Next			Par In
State	D C B A	Decoder	0 1 2 3 4 5	D C B A	*NPE	CET	D C B A
STBY	0 0 0 0	NSTBY	0 x x x x x	0 0 0 0	1	0	
			1 x x x x x	0 0 0 1	1	1	
PWRON	0 0 0 1	NPWRON	x 0 x x x x	0 0 0 1	1	0	
			x 1 x x x x	0 0 1 0	1	1	
TP1	0 0 1 0	NTP1	x x x x x x	0 0 1 1	1	1	
TP2	0 0 1 1	NTP2	x x x x x x	0 1 0 0	1	1	
TP3	0 1 0 0	NTP3	x x x x x x	0 1 0 1	1	1	
TP4	0 1 0 1	NTP4	x x x x x x	0 1 1 0	1	1	
TP5	0 1 1 0	NTP5	x x x x x x	0 1 1 1	1	1	
TP6	0 1 1 1	NTP6	x x x x x x	1 0 0 0	1	1	
TP7	1 0 0 0	NTP7	x x x x x x	1 0 0 1	1	1	
TP8	1 0 0 1	NTP8	x x x x x x	1 0 1 0	1	1	
TP9	1 0 1 0	NTP9	x x x x x x	1 0 1 1	1	1	
TP10	1 0 1 1	NTP10	x x x x x x	1 1 0 0	1	1	
TP11	1 1 0 0	NTP11	x x x x x x	1 1 0 1	1	1	

TP12	1 1 0 1	NTP12	x x x 1 x x	0 0 0 0	0	x	0 0 0 0
			x x 1 0 x x	0 0 1 0	0	x	0 0 1 0
			x x 0 0 x x	1 1 1 0	1	1	
SRLSE	1 1 1 0	NSRLSE	x x x x 0 x	1 1 1 0	1	0	
			x x x x 1 x	1 1 1 1	1	1	
WAIT	1 1 1 1	NWAIT	x x x x x 1	0 0 1 0	0	x	0 0 1 0
			x x x x x 0	1 1 1 1	1	0	

Maxterms:

NPE = (NTP12 + TPG_3') * (NTP12 + TPG_2' + TPG_3) * (NWAIT + TPG_5')

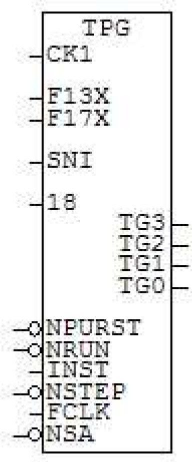
CET = (NSTBY + TPG_0) * (NPWRON + TPG_1) * (NSRLSE + TPG_4) * (NWAIT + TPG_5)

Par In A,B,C = 0

Par In B = (NTP12' * TPG_3)'

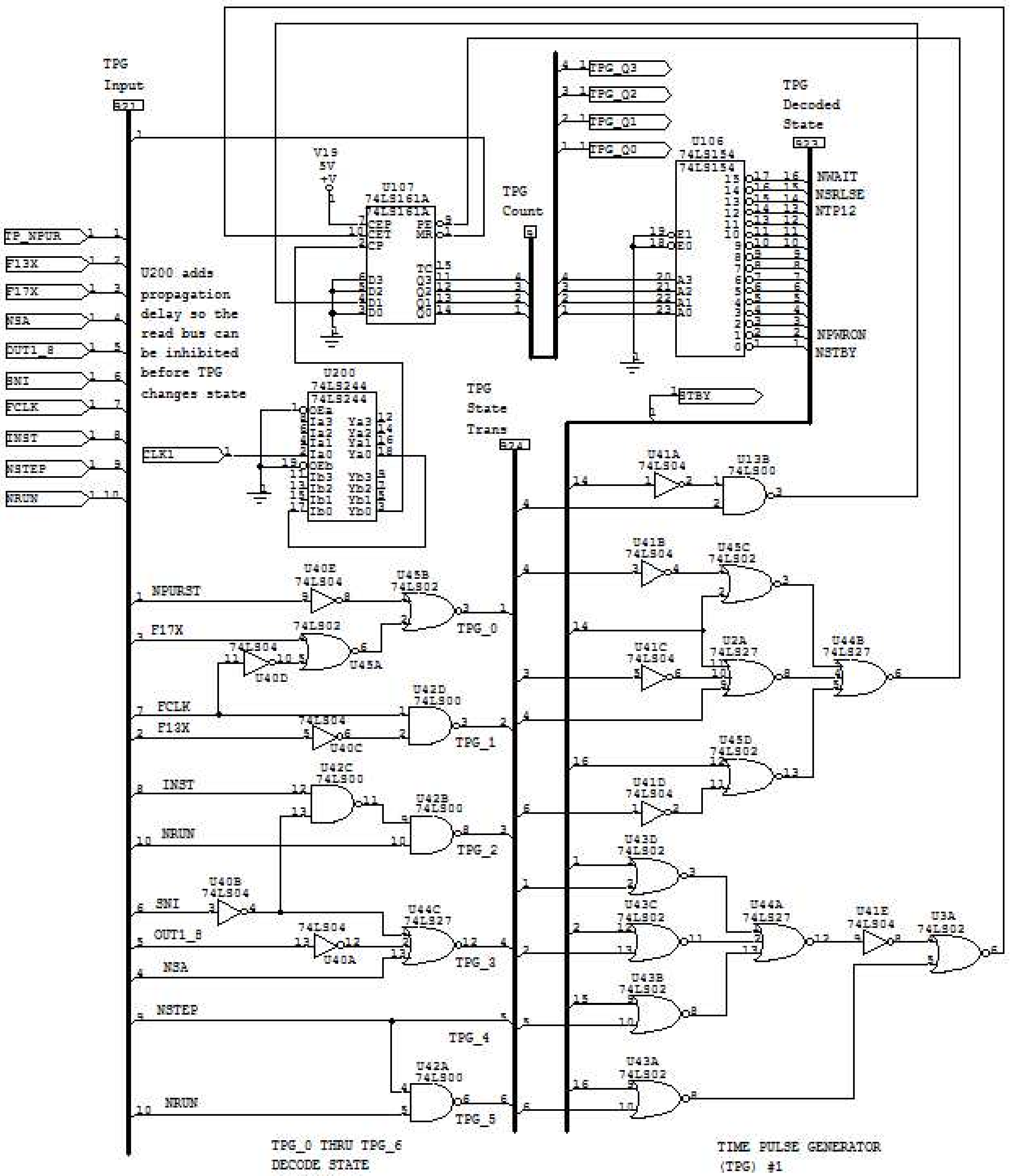
TPG INPUTS:

I/F	signal	full name	state definition
MON:			
	NRUN	RUN/HALT	0=run, 1=step
	INST	INST STEP MODE	1=instruction step, 0=sequence step
	NSTEP	SINGLE STEP	0=step (momentary)
	FCLK	CLOCK MODE	1=continuous, 0=single clock
	NSA	STANDBY ALLOWED	0=standby allowed
	NPURST	POWER UP RESET	0=power up reset
CLK:			
	CLK1	CLOCK1	1024 MHz AGC clock 1
SEQ:			
	SNI	SELECT NEXT INST	1=select next instruction
SCL:			
	F17X	F17 SCALER ONESHOT	1=timed out
	F13X	F13 SCALER ONESHOT	1=timed out
OUT:			
	OUT1_8	STANDBY ENABLED	1=standby enabled; works with STANDBY ALLOWED switch



TPG OUTPUTS:

signal	full name	state definition
TPG_Q3	TPG STATE	where Q0 is LSB, Q3 is MSB:
TPG_Q2		00 = STBY
TPG_Q1		01 = PWRON
TPG_Q0		02 = TP1
		03 = TP2
		04 = TP3
		05 = TP4
		06 = TP5
		07 = TP6
		08 = TP7
		09 = TP8
		10 = TP9
		11 = TP10
		12 = TP11
		13 = TP12
		14 = SRLSE
		15 = WAIT



STATE CHANGES OCCUR ON RISING EDGE OF CLK1

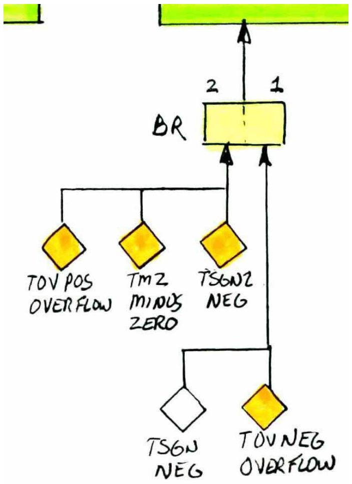
U200 adds propagation delay so the read bus can be inhibited before TPG changes state

TPG_0 THRU TPG_6 DECODE STATE TRANSITIONS

TIME PULSE GENERATOR (TPG) #1

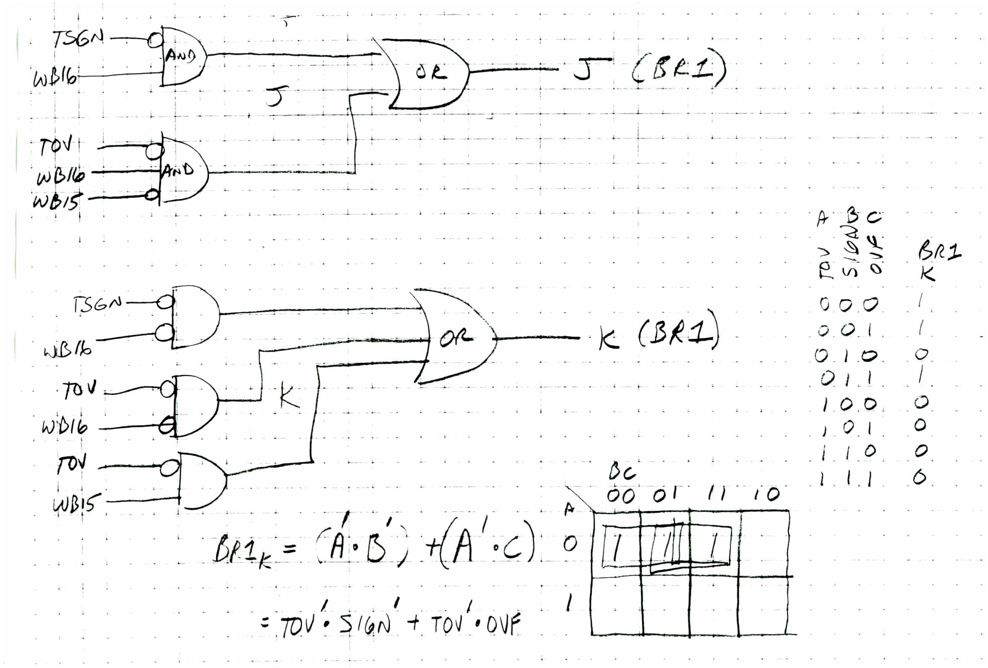
SEQ (Sequence Generator)

The sequence generator contains the stage registers and branch registers that (along with the time pulse generator) control execution of the microinstruction sequence.



Some back-of-the-envelope design that went into the branch register logic is shown in the next few charts. This is the conceptual design for the branch register. It has 2 flip-flops named BR2 and BR1. The flip-flops are set by the control signals shown as diamonds in this diagram.

The J and K inputs to the BR1 flip-flop are developed in this chart.



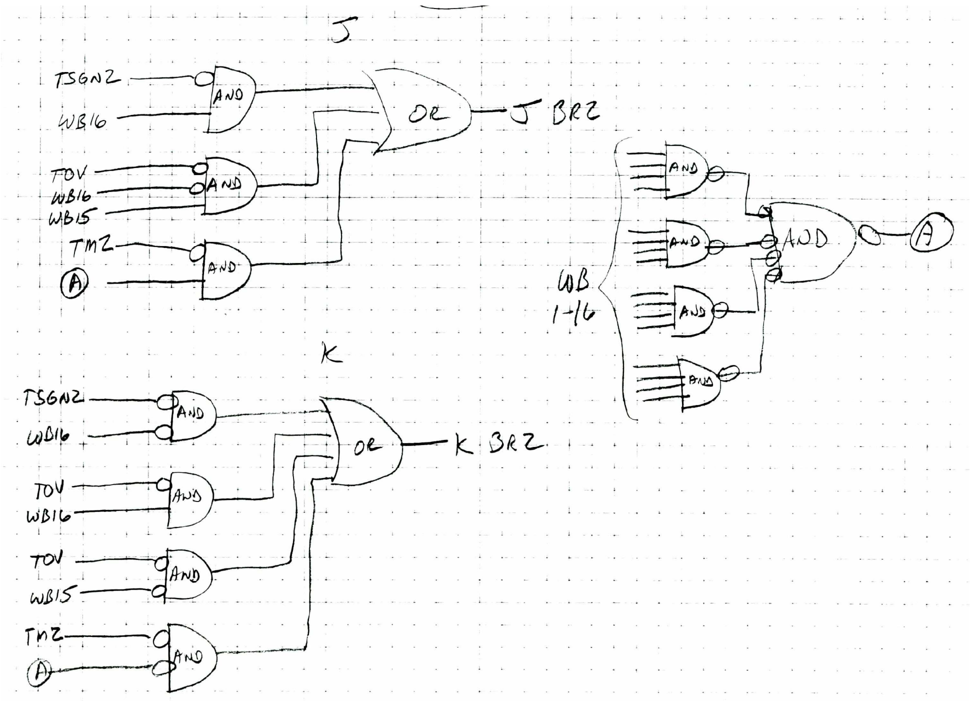
A TOV	B SIGN	C OVF	BR1 K
0	0	0	1
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

A \ BC	00	01	11	10
0	1	1	1	
1				

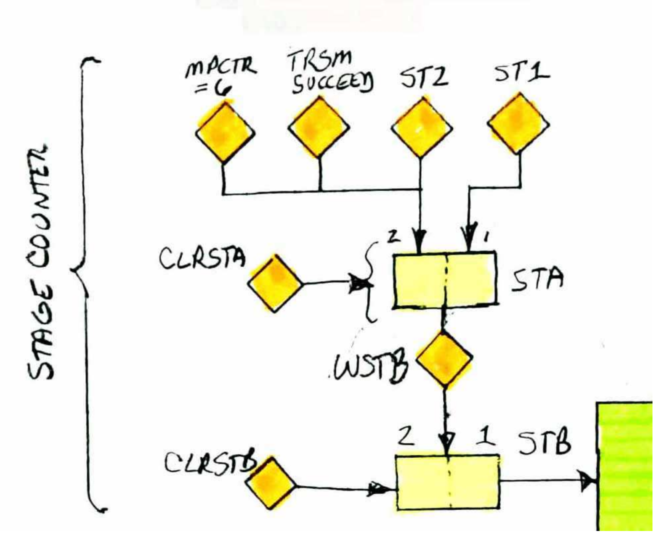
$$BR1_K = (A' \cdot B') + (A' \cdot C)$$

$$= TOV' \cdot SIGN' + TOV' \cdot OVF$$

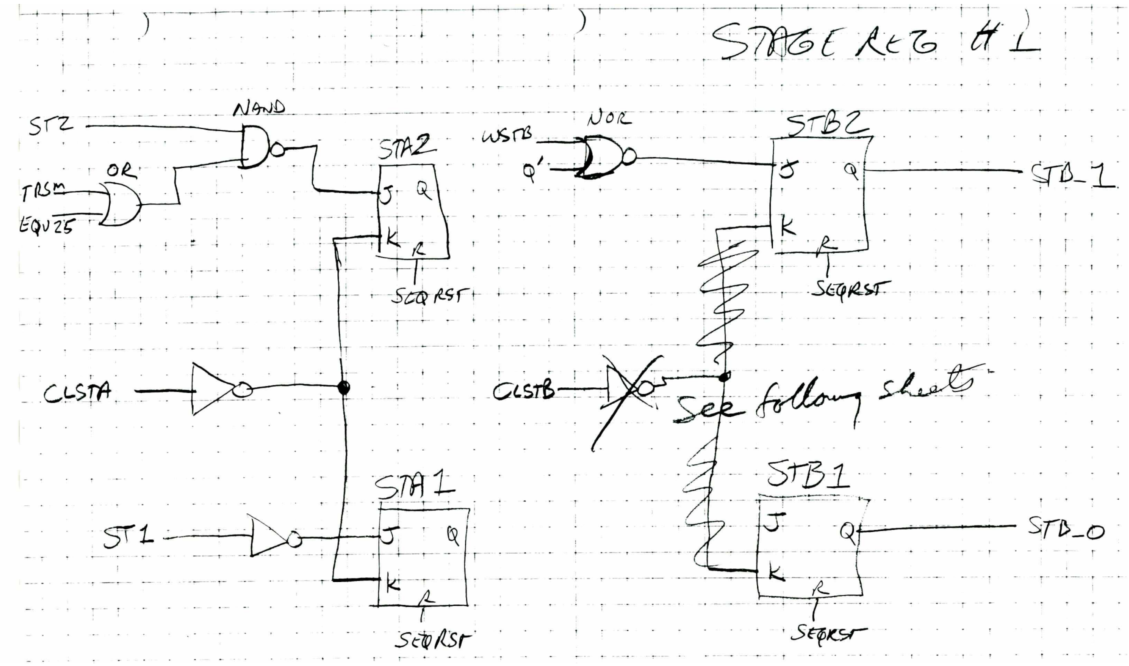
This chart shows the J and K inputs to the BR2 flip-flop. Logic to sense a 1's compliment minus zero from the write bus is also developed.



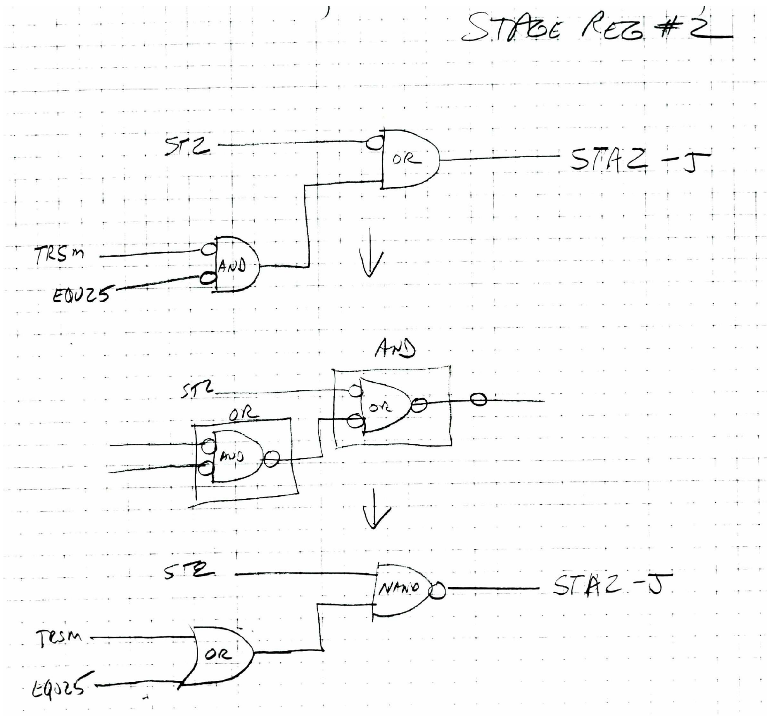
Here are some charts showing the stage register design. This is the conceptual design. There are (2) 2-bit stage registers: STA and STB.

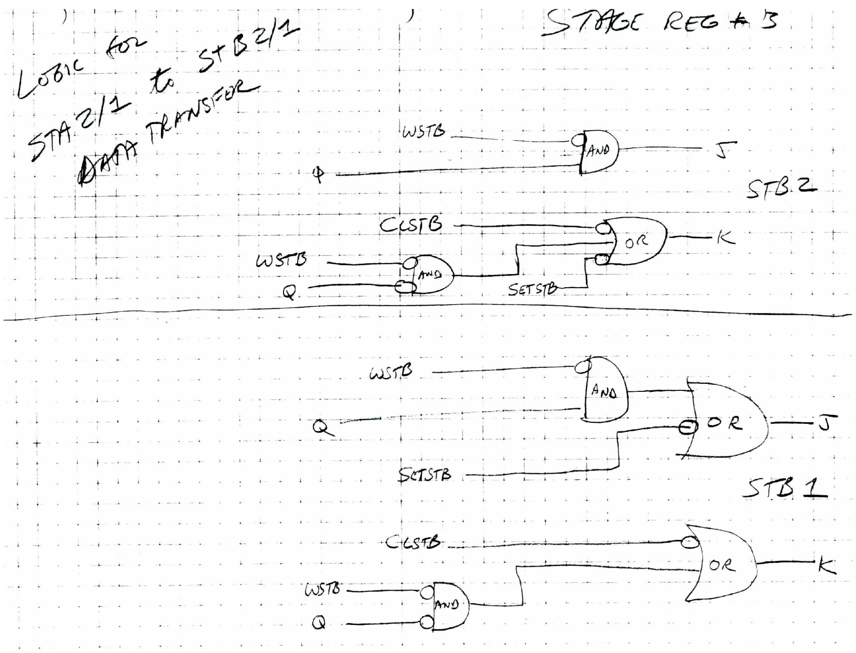


This is my initial cut at STA and STB. My initial attempt at CLSTB is scratched out and then developed into the correct solution on the later charts.



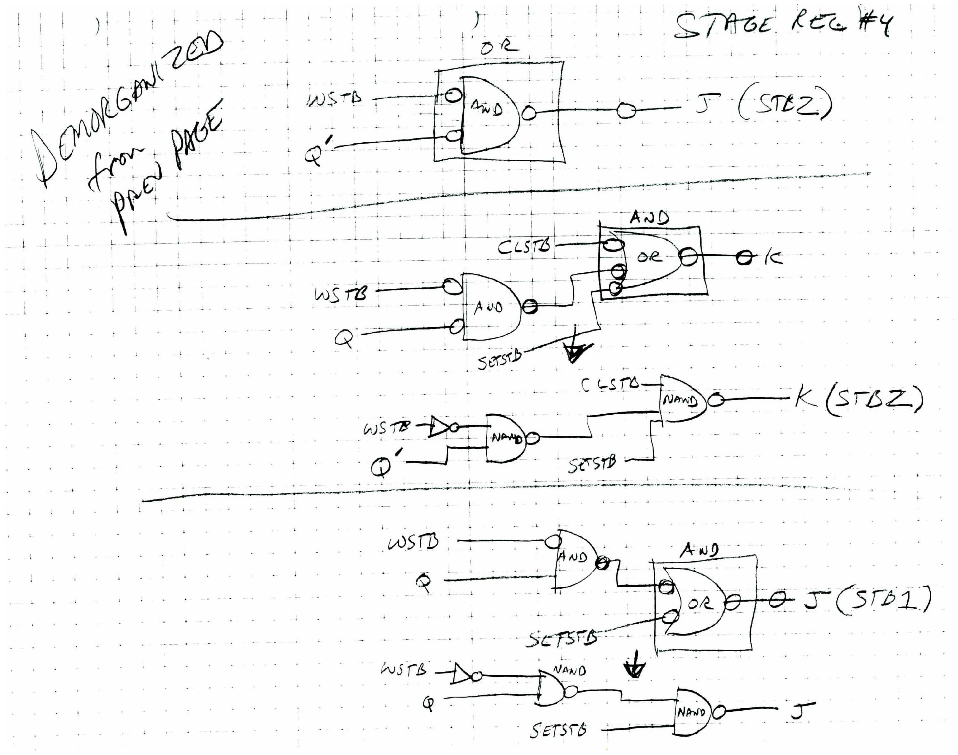
This is the J input to the STA2 flip-flop. The initial design is at the top. In the middle, I “DeMorganize” it using some bubble-pushing to get the final implementation at the bottom.

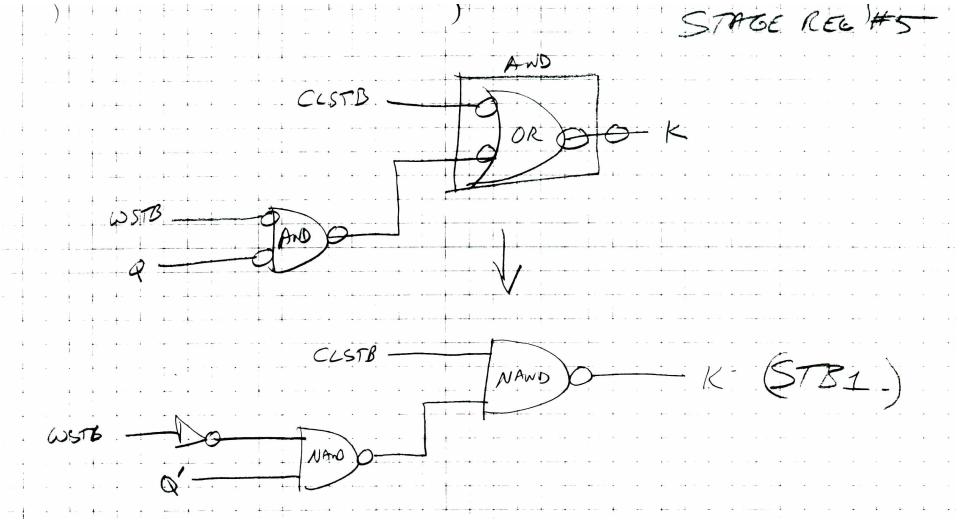




Here’s the J and K inputs to STB1 and STB2. This is the logic that transfers the contents of the STA flip-flops to STB.

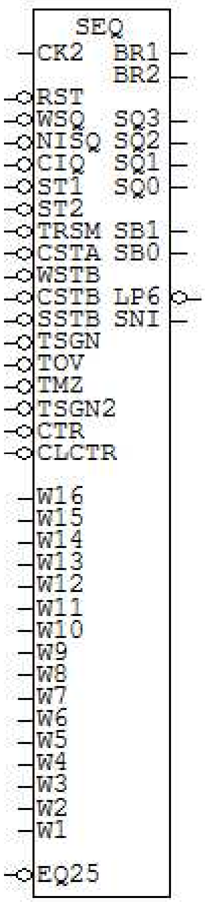
The J and K inputs to STB1 and STB2 move through a multi-step DeMorganizing process in the next 2 charts to reach their final state:





SEQ INPUTS:

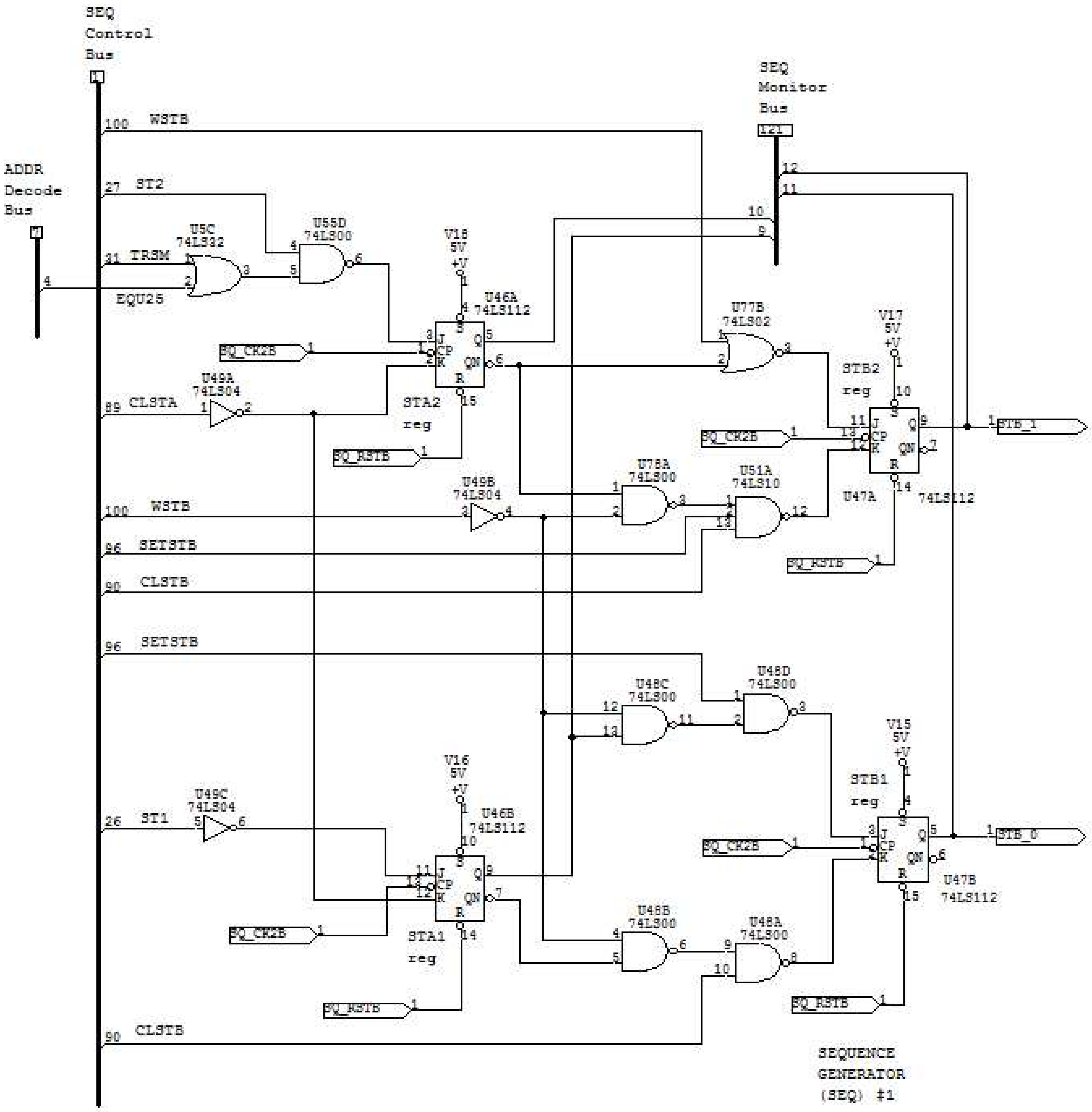
I/F	signal	full name	state definition
CLK:			
	CLK2	CLOCK 2	data transfer occurs on falling edge
CPM-C:			
	GENRST	GENERAL RESET	0=General Reset
	WSQ	WRITE SQ	0=Write SQ
	CLISQ	CLEAR SNI	0=Clear SNI
	CLSTA	CLEAR STA	0=Clear state counter A (STA)
	WSTB	WRITE STB	0=Write stage counter B (STB)
	CLSTB	CLEAR STB	0=Clear state counter B (STB)
	SETSTB	SET ST1	0=Set the ST1 bit of STB
CPM-A:			
	TRSM	TEST RESUME	0=Test for resume
	TSGN	TEST SIGN	0=Test sign
	TSGN2	TEST SIGN 2	0=Test sign 2
	ST1	SET STAGE 1	0=Stage 1
	ST2	SET STAGE 2	0=Stage 2
	CLCTR	CLR LOOP CTR	0=Clear loop counter
	CTR	INCR LOOP CTR	0=Loop counter
	TMZ	TEST MINUS ZERO	0=Test for minus zero
	TOV	TEST OVF	0=Test for overflow
	NISQ	NEW INSTRUCT SQ reg	0=New instruction to the
WBUS:			
	WB_01	WRITE BUS 01	
	...	...	
	WB_14	WRITE BUS 14	
	WB_15	WRITE BUS 15	US (overflow) bit for write bus
	WB_16	WRITE BUS 16	SG (sign) bit for write bus
ADR:			
	EQU_25	ADDRESS = 025	0=CADR in register S evaluates to = 025



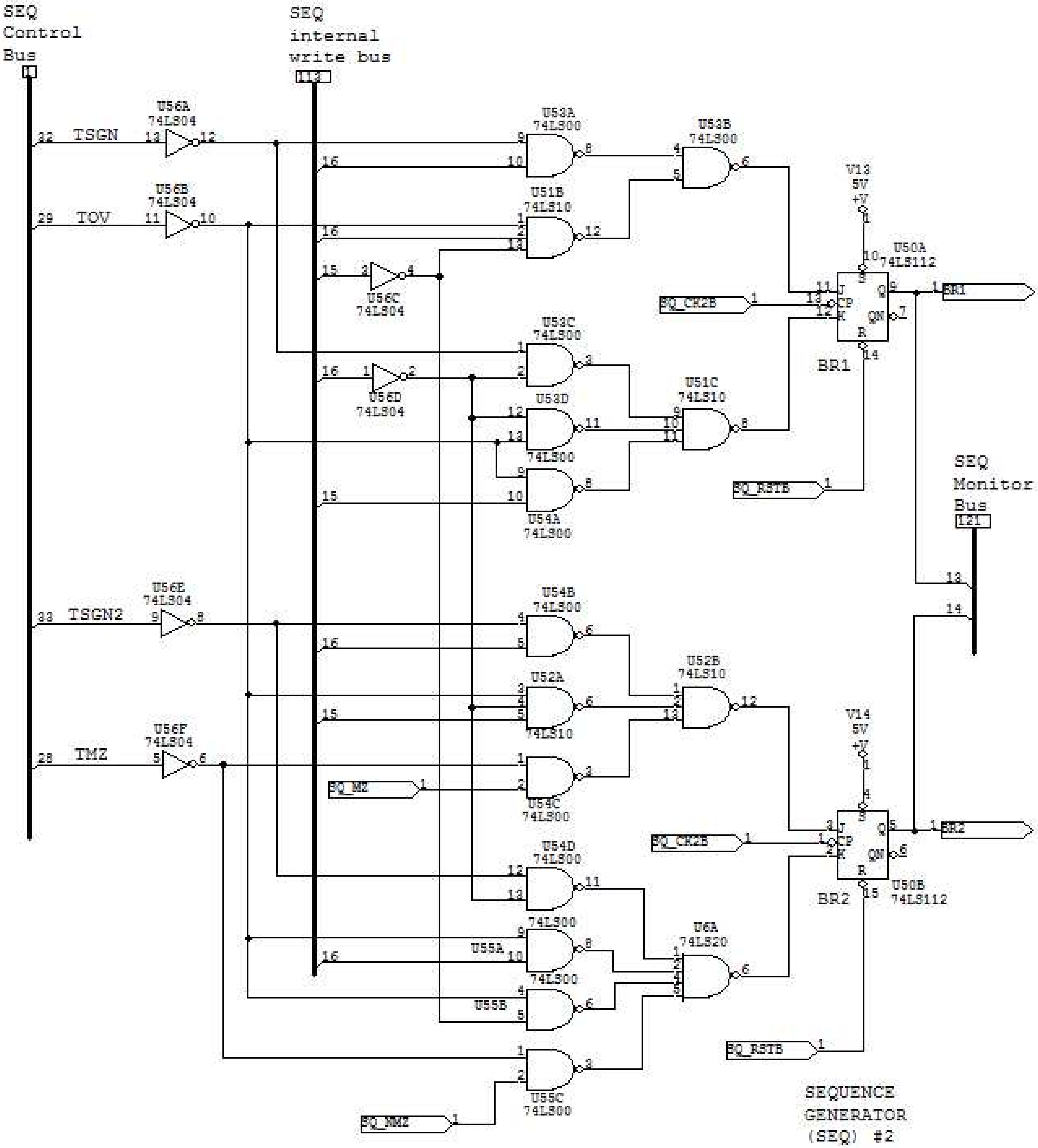
SEQ OUTPUTS:

I/F	signal	full name	state definition
CPM-A:			
	BR1	BRANCH REG 1	where BR1 is MSB, BR2 is LSB
	BR2	BRANCH REG 2	BR00 =0 BR1=0, BR2=0
			BR01 =1 BR1=0, BR2=1
			BR10 =2 BR1=1, BR2=0
			BR11 =3 BR1=1, BR2=1

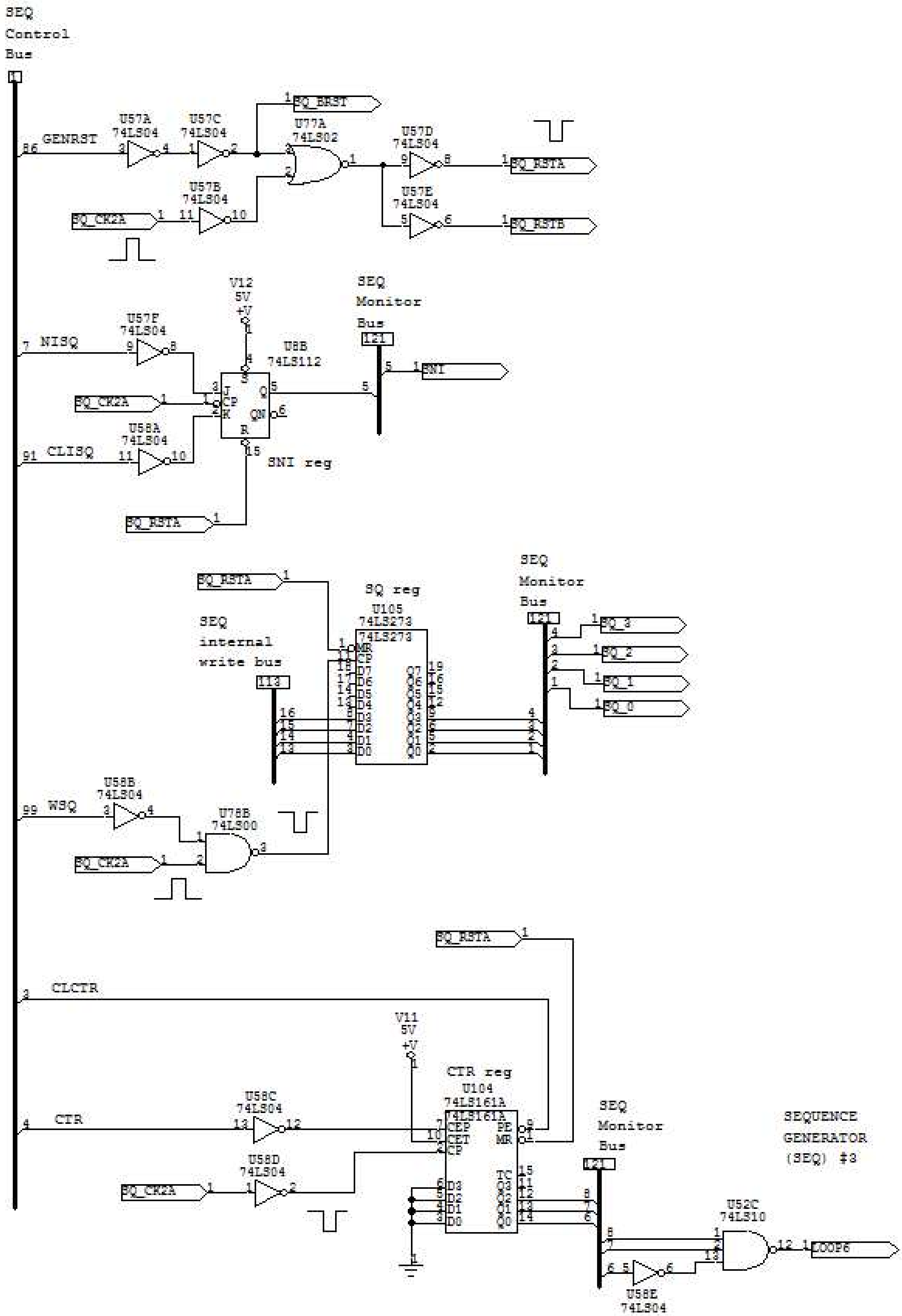
SQ_3	INST REG	where SQ_3 is MSB, SQ_0 is LSB
SQ_2		
SQ_1		
SQ_0		
STB_1	STAGE REG	where STB_1 is MSB, STB_0 is LSB
STB_0		
LOOP6	LOOPCNTR EQ 6	0=LOOPCNTR is holding the number 6.
SNI	SELECT NEXT INST	1=select next instruction (SNI register)

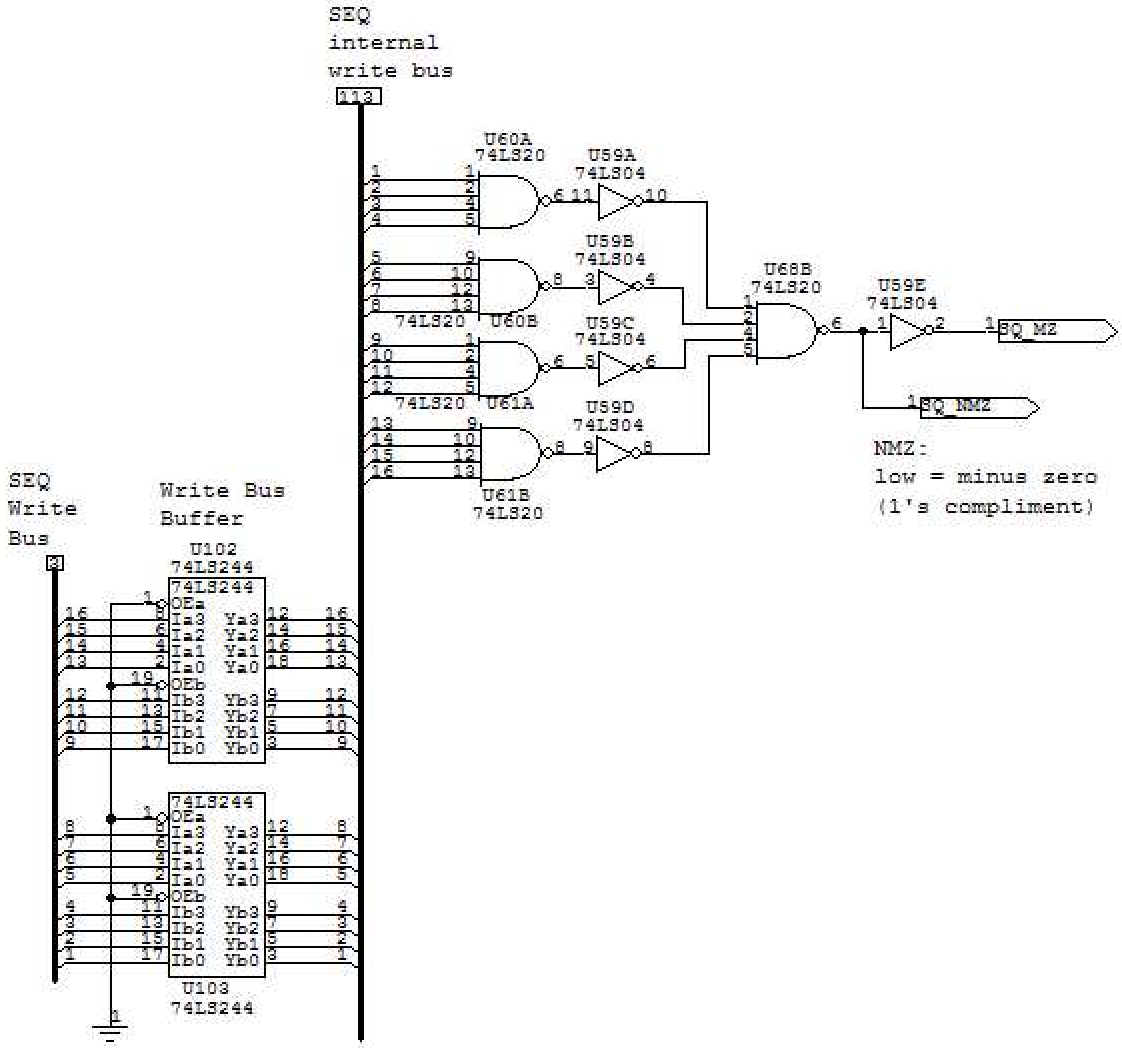


SEQUENCE
GENERATOR
(SEQ) #1

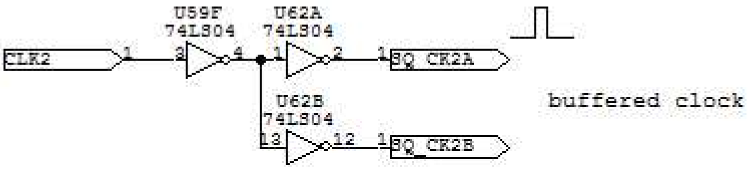


SEQUENCE GENERATOR (SEQ) #2


SEQ
Control
Bus
GENRST
U57A
74LS04
U57C
74LS04
SQ_BRST
U77A
74LS02
U57D
74LS04
SQ_RSTA
U57E
74LS04
SQ_RSTB
U57B
74LS04
SQ_CK2A
V12
5V
+V
SEQ
Monitor
Bus
NISQ
U57F
74LS04
U8B
74LS112
SNI
J
CP
K
S
R
Q
QN
CLISQ
U58A
74LS04
SNI reg
SQ_RSTA
SQ reg
U105
74LS273
SEQ
internal
write bus
MR
CP
D7
D6
D5
D4
D3
D2
D1
D0
Q7
Q6
Q5
Q4
Q3
Q2
Q1
Q0
SQ_3
SQ_2
SQ_1
SQ_0
WSQ
U58B
74LS04
U78B
74LS00
CLCTR
CTR
U58C
74LS04
V11
5V
+V
CTR reg
U104
74LS161A
CEP
CET
CP
PE
MR
TC
Q3
Q2
Q1
Q0
U58D
74LS04
U52C
74LS10
LOOP6
U58E
74LS04
SEQUENCE
GENERATOR
(SEQ) #3



NMZ:
low = minus zero
(1's compliment)



buffered clock

SEQUENCE
GENERATOR
(SEQ) #4

CPM-A (Control Pulse Matrix A)

In this AGC replica, the CPM-A subsequences are implemented in EPROM (they were implemented as a diode matrix in the original). The address into the EPROM is constructed as follows (bit 1 is the LSB):

bit	
13,14	CTR subsequence (2)
9-12:	register SQ (4)
7,8:	register STB (2)
3-6:	register SG (4)
2:	register BR1 (1)
1:	register BR2 (1)

Bits 7-14 (STB, SQ, and CTR) select the instruction subsequence. Bits 1-6 select the control pulses (control logic signals) that are asserted from that subsequence.

SELECTING THE INSTRUCTION SUBSEQUENCE

The 11 AGC instructions, priority counter operations, and interrupt operations are implemented in 22 instruction subsequences. Some instructions (TC) have a single subsequence; others have several subsequences.

The instruction subsequence is choosen by CTR, SQ, and STB. These form bits 7-14 of the CPM-A EPROM address.

CTR (EPROM address bits 13-14)

The CTR signal has 2 lines: SB_02 is the MSB; SB_01 is the LSB. The signal comes from the CTR subsystem in the PROC module. It indicates whether processing needs to be briefly interrupted to insert a PINC or MINC subsequence to increment or decrement a priority counter.

CTR00:	SB_02=0, SB_01=0	no counter; do the next subsequence
CTR01:	SB_02=0, SB_01=1	PINC
CTR10:	SB_02=1, SB_01=0	MINC
CTR11:	SB_02=1, SB_01=1	both; they cancel out, so do the next subsequence

Register SQ (EPROM address bits 9-12)

The 4-bit SQ register holds the currently executing instruction. The code in the SQ register is the same as the op code for these four instructions.

NMEM	SQ REG	OPCODE	USAGE	DESCRIPTION	CYCLES
TC	00	00	TC K	Transfer Control	1 MCT
CCS	01	01	CCS K	Count, Compare, Skip	2 MCT
INDEX	02	02	INDEX K	Index	2 MCT
XCH	03	03	XCH K	Exchange	2 MCT

The SQ register code for these four instructions is the op code + 010 (octal). This happens because all of these instructions have bit 15 set (the sign (SG) bit) while in memory. When the instruction is copied from memory to the memory buffer register (G) to register B, the SG bit moves from bit 15 to bit 16 and the sign is copied back into bit 15 (US). Therefore, the CS op code (04) becomes (14), and so on.

NMEM	SQ REG	OPCODE	USAGE	DESCRIPTION	CYCLES
CS	014	04	CS K	Clear and Subtract	2 MCT
TS	015	05	TS K	Transfer to Storage	2 MCT
AD	016	06	AD K	Add	2 or 3 MCT
MASK	017	07	MASK K	Bitwise AND	2 MCT

These are the three extended instructions. They are accessed by executing an INDEX 5777 before each instruction. By convention, address 5777 contains 47777. The INDEX instruction adds 47777 to the extended instruction to form the SQ op code. For example, the INDEX adds 4 to the 4 op code for MP to produce the 11 (octal; the addition generates an end-around-carry). SQ register code (the 7777 part is a negative zero).

NMEM	SQ REG	OPCODE	USAGE	DESCRIPTION	CYCLES
MP	011	04	MP K	Multiply	10 MCT
DV	012	05	DV K	Divide	18 MCT
SU	013	06	SU K	Subtract	4 or 5 MCT

STB (EPROM address bits 7-8)

The stage register B (STB) selects the subsequence for a given instruction. Some instructions have multiple subsequences; others (TC) have only one.

The stage register has 2 bits: STB2 is the MSB; STB1 is the LSB. All instructions initiallly begin with the stage register set to zero. If an instruction needs more than one subsequence, the stage register is incremented to select the next subsequence.

STB00:	STB2=0, STB1=0
STB01:	STB2=0, STB1=1
STB10:	STB2=1, STB1=0
STB11:	STB2=1, STB1=1

INSTRUCTION SUBSEQUENCES

There are 22 instruction subsequences:

TC0	0
CCS0	1
CCS1	2
NDX0	3
NDX1	4
RSM3	5
XCH0	6
CS0	7
TS0	8
AD0	9
MASK0	10
MP0	11
MP1	12

MP3	13
DV0	14
DV1	15
SU0	16
RUPT1	17
RUPT3	18
STD2	19
PINC0	20
MINC0	21

If the CTR signals are 01 (SB_02=0, SB_01=1) the PINC subsequence is selected. If the CTR signals are 10 (SB_02=1, SB_01=0) the MINC subsequence is selected. Otherwise, subsequences for each instruction are selected using the 4-bit SQ register and the 2-bit stage register (STB2, STB1, where STB2 is the MSB). Some instructions (TC) have only one subsequence. At the start of each instruction, the stage counter is initially set to zero. The interrupt call and resturn sequences are also stored at SQ=00.

	SQ	STB00	STB01	STB10	STB11
TC/RUPT	00:	TC0	RUPT1	STD2	RUPT3
CCS	01:	CCS0	CCS1	----	----
INDEX	02:	NDX0	NDX1	----	RSM3
XCH	03:	XCH0	----	STD2	----
	04:	----	----	----	----
	05:	----	----	----	----
	06:	----	----	----	----
	07:	----	----	----	----
	10:	----	----	----	----
MP	11:	MP0	MP1	----	MP3
DV	12:	DV0	DV1	STD2	----
SU	13:	SU0	----	STD2	----
CS	14:	CS0	----	STD2	----
TS	15:	TS0	----	STD2	----
AD	16:	AD0	----	STD2	----
MASK	17:	MASK0	----	STD2	----

SELECTING THE CONTROL PULSES

Each subsequence consists of 12 steps (TP1 - TP12), with each step asserting up to 5 control pulses (control logic signals). Steps TP1-TP11 are unique to each subsequence; step TP12 is common to all subsequences. Control pulses for TP12 are discussed in CPM-C.

For some steps, selection of the control pulses is also conditional on the state of the 2-bit branch register (BR1 and BR2: BR1 is the MSB and BR2 is the LSB):

BR00	BR1=0, BR2=0
BR01	BR1=0, BR2=1
BR10	BR1=1, BR2=0
BR11	BR1=1, BR2=1

Bits 1-6 of the CPM-A EPROM address is formed as follows:

3-6:	register SG (4)
2:	register BR1 (1)
1:	register BR2 (1)

The column on the far left is the step; columns to the right specify the control pulses that are asserted for that step. Some steps have no control pulses.

subsequence TC0:

TP1		RB	WY	WS	CI	----
TP2						
TP3		WG	----	----	----	----
TP4		RA	WOVI	----	----	----
TP5						
TP6						
TP7		RG	RSC	WB	WP	----
TP8		RZ	WQ	GP	TP	----
TP9		RB	WSC	WG	----	----
TP10		RU	WZ	----	----	----
TP11		NISQ	----	----	----	----

subsequence CCS0:

TP1		RB	WS	----	----	----
TP2		RZ	WY	----	----	----
TP3		WG	----	----	----	----
TP4						
TP5						
TP6		RG	RSC	WB	TSGN	WP
TP7	BR00,	RC	TMZ	----	----	----
	BR01,	RC	TMZ	----	----	----
	BR10,	RB	TMZ	----	----	----
	BR11,	RB	TMZ	----	----	----
TP8	BR00,	GP	TP	----	----	----
	BR01,	R1	WX	GP	TP	----
	BR10,	R2	WX	GP	TP	----
	BR11,	R1	R2	WX	GP	TP,
TP9		RB	WSC	WG	----	----
TP10	BR00,	RC	WA	----	----	----
	BR01,	WA	R1C	----	----	----
	BR10,	RB	WA	----	----	----
	BR11,	WA	R1C	----	----	----
TP11		RU	ST1	WZ	----	----

subsequence CCS1:

TP1		RZ	WY	WS	CI	----
TP2						
TP3		WG	----	----	----	----
TP4		RU	WZ	----	----	----
TP5		RA	WY	CI	----	----
TP6						
TP7		RG	RSC	WB	WP	----
TP8		RU	WB	GP	TP	----

TP9					
TP10	RC	WA	WOVI	----	----
TP11	RG	RSC	WB	NISQ	----

subsequence NDX0:

TP1	RB	WS	----	----	----
TP2					
TP3	WG	----	----	----	----
TP4	RA	WOVI	----	----	----
TP5					
TP6					
TP7	RG	RSC	WB	WP	----
TP8	GP	TP	----	----	----
TP9	RB	WSC	WG	----	----
TP10	TRSM	----	----	----	----
TP11	ST1	----	----	----	----

subsequence NDX1:

TP1	RZ	WY	WS	CI	----
TP2					
TP3	WG	----	----	----	----
TP4	RU	WZ	----	----	----
TP5					
TP6	RB	WY	----	----	----
TP7	RG	RSC	WB	WP	----
TP8	RB	WX	GP	TP	----
TP9	RB	WSC	WG	----	----
TP10					
TP11	RU	WB	WOVI	NISQ	----

subsequence RSM3:

TP1	R24	WS	----	----	----
TP2					
TP3	WG	----	----	----	----
TP4					
TP5					
TP6					
TP7	RG	WZ	----	----	----
TP8					
TP9					
TP10					
TP11	NISQ	----	----	----	----

subsequence XCH0:

TP1	RB	WS	----	----	----
TP2	RA	WP	----	----	----
TP3	WG	----	----	----	----
TP4	WP2	----	----	----	----
TP5					
TP6					

TP7		RG	RSC	WB	WP,	----	
TP8		GP	TP	----	----	----	
TP9		RA	WSC	WG	RP2	----	
TP10		RB	WA	WOVI	----	----	
TP11		ST2	----	----	----	----	

subsequence CS0:

TP1		RB	WS	----	----	----	
TP2							
TP3		WG	----	----	----	----	
TP4							
TP5							
TP6							
TP7		RG	RSC	WB	WP,	----	
TP8		GP	TP	----	----	----	
TP9		RB	WSC	WG	----	----	
TP10		RC	WA	WOVI	----	----	
TP11		ST2	----	----	----	----	

subsequence TS0:

TP1		RB	WS	----	----	----	
TP2		RA	WB	TOV	WP	----	
TP3		WG	----	----	----	----	
TP4	BR00,	----	----	----	----	----	
	BR01,	RZ	WY	CI	----	----	(overflow)
	BR10,	RZ	WY	CI	----	----	(underflow)
	BR11,	----	----	----	----	----	
TP5	BR00,	----	----	----	----	----	
	BR01,	R1	WA	----	----	----	
	BR10,	WA	R1C	----	----	----	
	BR11,	----	----	----	----	----	
TP6							
TP7	BR00,	----	----	----	----	----	
	BR01,	RU	WZ	----	----	----	
	BR10,	RU	WZ	----	----	----	
	BR11,	----	----	----	----	----	
TP8		GP	----	----	----	----	
TP9		RB	WSC	WG	----	----	
TP10		RA	WOVI	----	----	----	
TP11		ST2	----	----	----	----	

subsequence AD0:

TP1		RB	WS	----	----	----	
TP2		RA	WY	----	----	----	
TP3		WG	----	----	----	----	
TP4							
TP5							
TP6							
TP7		RG	RSC	WB	WP	----	
TP8		RB	WX	GP	TP	----	
TP9		RB	WSC	WG	----	----	
TP10							

TP11	RU	WA	WOVC	ST2	WOVI

SUB_MASK0 performs a logical AND using DeMorgan's Theorem: the inputs are inverted, a logical OR is performed, and the result is inverted. The implementation of the OR (at TP8) is somewhat unorthodox: the inverted inputs are in registers U and C. The OR is achieved by gating both registers onto the read/write bus simultaneously. (The bus only transfers logical 1's; register-to-register transfers are performed by clearing the destination register and then transferring the 1's from the source register to the destination). When the 1's from both registers are simultaneously gated onto the bus, the word on the bus is a logical OR of both registers.

subsequence MASK0:

TP1	RB	WS	----	----	----
TP2	RA	WB	----	----	----
TP3	WG	----	----	----	----
TP4	RC	WY	----	----	----
TP5					
TP6					
TP7	RG	RSC	WB	WP	----
TP8	RU	RC	WA	GP	TP
TP9					
TP10	RA	WB	----	----	----
TP11	RC	WA	ST2	WOVI	----

subsequence MP0:

TP1		RB	WS	----	----	----
TP2		RA	WB	TSGN	----	----
TP3		RSC	WG	----	----	----
TP4	BR00,	RB	WLP	----	----	----
	BR01,	RB	WLP	----	----	----
	BR10,	RC	WLP	----	----	----
	BR11,	RC	WLP	----	----	----
TP5		RLP	WA	----	----	----
TP6						
TP7	BR00,	RG	WY	WP	----	----
	BR01,	RG	WY	WP	----	----
	BR10,	RG	WB	WP	----	----
	BR11,	RG	WB	WP	----	----
TP8	BR00,	GP	TP	----	----	----
	BR01,	GP	TP	----	----	----
	BR10,	RC	WY	GP	TP	----
	BR11,	RC	WY	GP	TP	----
TP9		RU	WB	TSGN2	----	----
TP10	BR00,	RA	WLP	TSGN	----	----
	BR01,	RA	RB14	WLP	TSGN	----
	BR10,	RA	WLP	TSGN	----	----
	BR11,	RA	RB14	WLP	TSGN	----
TP11	BR00,	ST1	WALP	----	----	----
	BR01,	R1	ST1	WALP	R1C	----
	BR10,	RU	ST1	WALP	----	----
	BR11,	RU	ST1	WALP	----	----

subsequence MP1:

```
TP1                 RA                  WY                  ----                ----                ----
TP2                 RLP                 WA                  TSGN                ----                ----
TP3       BR00,     ----                ----                ----                ----                ----
          BR01,     ----                ----                ----                ----                ----
          BR10,     RB                  WX                  ----                ----                ----
          BR11,     RB                  WX                  ----                ----                ----
TP4                 RA                  WLP                 ----                ----                ----
TP5                 RLP                 TSGN                ----                ----                ----
TP6                 RU                  WALP                ----                ----                ----
TP7                 RA                  WY                  ----                ----                ----
TP8       BR00,     ----                ----                ----                ----                ----
          BR01,     ----                ----                ----                ----                ----
          BR10,     RB                  WX                  ----                ----                ----
          BR11,     RB                  WX                  ----                ----                ----
TP9                 RLP                 WA                  ----                ----                ----
TP10                RA                  WLP                 CTR                 ----                ----
TP11                RU                  ST1                 WALP                ----                ----
```

subsequence MP3:

```
TP1                 RZ                  WY                  WS                  CI                  ----
TP2                 RLP                 TSGN                ----                ----                ----
TP3                 WG                  ----                ----                ----                ----
TP4                 RU                  WZ                  ----                ----                ----
TP5                 RA                  WY                  ----                ----                ----
TP6       BR00,     ----                ----                ----                ----                ----
          BR01,     ----                ----                ----                ----                ----
          BR10,     RB                  WX                  ----                ----                ----
          BR11,     RB                  WX                  ----                ----                ----
TP7                 RG                  RSC                 WB                  WP                  ----
TP8                 RLP                 WA                  GP                  TP                  ----
TP9                 RB                  WSC                 WG                  ----                ----
TP10                RA                  WLP                 ----                ----                ----
TP11                RU                  WALP                NISQ                ----                ----
```

subsequence DV0:

```
TP1                 RB                  WS                  ----                ----                ----
TP2                 RA                  WB                  TSGN                ----                ----
TP3                 RSC                 WG                  ----                ----                ----
TP4       BR00,     RC                  WA                  ----                ----                ----
          BR01,     RC                  WA                  ----                ----                ----
          BR10,     ----                ----                ----                ----                ----
          BR11,     ----                ----                ----                ----                ----
TP5       BR00,     R1                  WLP                 ----                ----                ----
          BR01,     R1                  WLP                 ----                ----                ----
          BR10,     R2                  WLP                 ----                ----                ----
          BR11,     R2                  WLP                 ----                ----                ----
TP6                 RA                  WQ                  ----                ----                ----
TP7                 RG                  WB                  TSGN                WP                  ----
TP8                 RB                  WA                  GP                  TP                  ----
TP9       BR00,     RLP                 R2                  WB                  ----                ----
```

```
        BR01, RLP             R2              WB              ----            ----
        BR10, ----            ----            ----            ----            ----
        BR11, ----            ----            ----            ----            ----
TP10    BR00, RB              WLP             ----            ----            ----
        BR01, RB              WLP             ----            ----            ----
        BR10, RC              WA              ----            ----            ----
        BR11, RC              WA              ----            ----            ----
TP11          R1              ST1             WB              ----            ----
```

subsequence DV1:

```
TP1           R22             WS              ----            ----            ----
TP2           RQ              WG              ----            ----            ----
TP3           RG              WQ              WY              RSB             ----
TP4           RA              WX              ----            ----            ----
TP5           RLP             TSGN2                   ----            ----            ----
TP6
TP7           RU              TSGN            ----            ----            ----
TP8     BR00, ----            ----            ----            ----            ----
        BR01, ----            ----            ----            ----            ----
        BR10, RU              WQ              ----            ----            ----
        BR11, RU              WQ              ----            ----            ----
TP9     BR00, RB              RSB             WG              ----            ----
        BR01, RB              RSB             WG              ----            ----
        BR10, RB              WG              ----            ----            ----
        BR11, RB              WG              ----            ----            ----
TP10          RG              WB              TSGN            ----            ----
TP11    BR00, ST1             ----            ----            ----            ----
        BR01, ST1             ----            ----            ----            ----
        BR10, RC              WA              ST2             ----            ----
        BR11, RB              WA              ST2             ----            ----
```

subsequence SU0:

```
TP1           RB              WS              ----            ----            ----
TP2           RA              WY              ----            ----            ----
TP3           WG              ----            ----            ----            ----
TP4
TP5
TP6
TP7           RG              RSC             WB              WP              ----
TP8           RC              WX              GP              TP              ----
TP9           RB              WSC             WG              ----            ----
TP10
TP11          RU              WA              WOVC            ST2             WOVI
```

subsequence RUPT1:

```
TP1           R24             WY              WS              CI,             ----
TP2
TP3           WG              ----            ----            ----            ----
TP4
TP5
TP6
```

TP7					
TP8					
TP9	RZ	WG	----	----	----
TP10	RU	WZ	----	----	----
TP11	ST1	ST2	----	----	----

subsequence RUPT3:

TP1	RZ	WS	----	----	----
TP2	RRPA	WZ	----	----	----
TP3	RZ	KRPT	WG	----	----
TP4					
TP5					
TP6					
TP7					
TP8					
TP9	RB	WSC	WG	----	----
TP10					
TP11	ST2	----	----	----	----

subsequence STD2:

TP1	RZ	WY	WS	CI,	----
TP2					
TP3	WG	----	----	----	----
TP4	RU	WZ	----	----	----
TP5					
TP6					
TP7	RG	RSC	WB	WP	----
TP8	GP	TP	----	----	----
TP9	RB	WSC	WG	----	----
TP10					
TP11	NISQ	----	----	----	----

subsequence PINC:

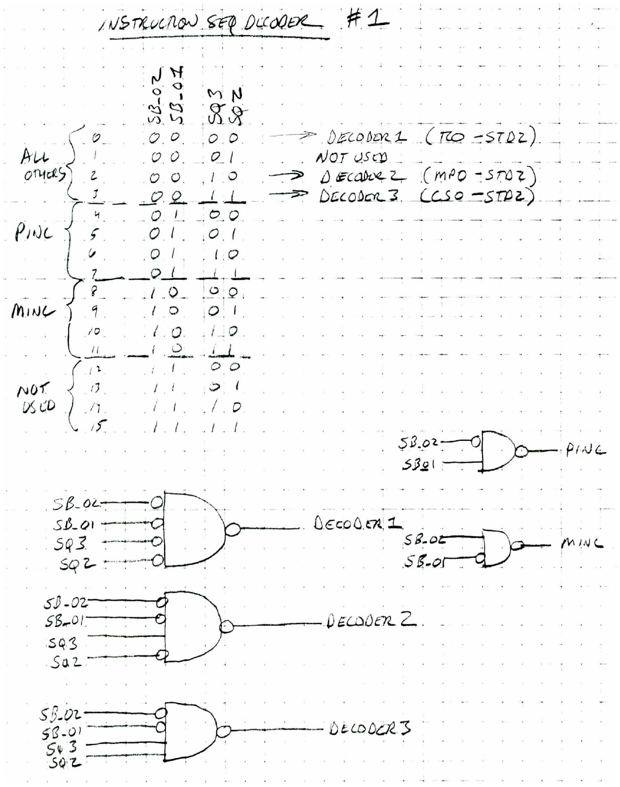
TP1	WS	RSCT	----	----	----
TP2					
TP3	WG	----	----	----	----
TP4	R1	WY	----	----	----
TP5					
TP6	RG	WX	WP	----	----
TP7	TP	----	----	----	----
TP8	WP	----	----	----	----
TP9	RU	CLG	WPx	----	----
TP10	RU	WGx	WOVR	----	----
TP11					

subsequence MINC:

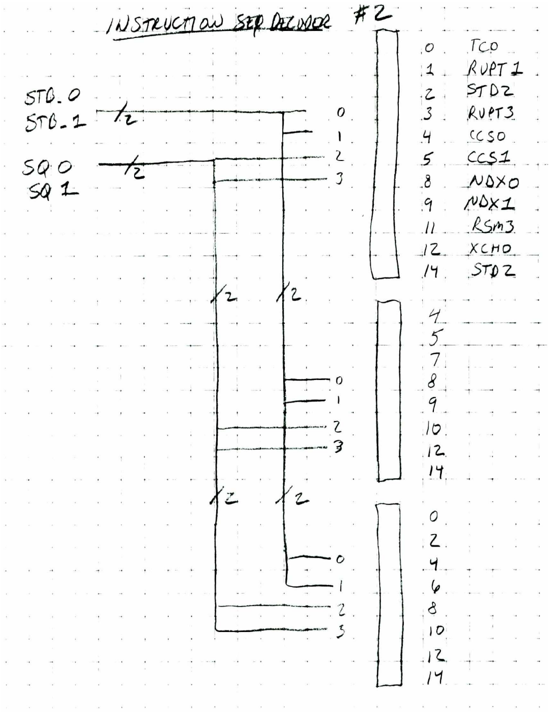
TP1	WS,	RSCT	----	----	----
TP2					
TP3	WG	----	----	----	----
TP4	WY	R1C	----	----	----

TP5					
TP6	RG	WX	WP	----	----
TP7	TP	----	----	----	----
TP8	WP	----	----	----	----
TP9	RU	CLG	WPx	----	----
TP10	RU	WGx	WOVR	----	----
TP11					

Here's some of the analysis used to develop the instruction sequence decoder. This decoder takes in the STB and SQ inputs and decodes the instruction sequence for display to the operator.



		SB_02	SB_01	SQ3	SQ2	
ALL OTHERS	0	0	0	0	0	→ DECODER 1 (TCO - STD2)
	1	0	0	0	1	NOT USED
	2	0	0	1	0	→ DECODER 2 (MPO - STD2)
	3	0	0	1	1	→ DECODER 3 (CCSO - STD2)
PINC	4	0	1	0	0	
	5	0	1	0	1	
	6	0	1	1	0	
	7	0	1	1	1	
MINC	8	1	0	0	0	
	9	1	0	0	1	
	10	1	0	1	0	
	11	1	0	1	1	
NOT USED	12	1	1	0	0	
	13	1	1	0	1	
	14	1	1	1	0	
	15	1	1	1	1	



CONTROL SIGNAL DEFINITIONS

SIGNAL	#	DESCRIPTION
CI	1	Carry in
CLG	2	Clear G
CLCTR	3	Clear loop counter
CTR	4	Loop counter
GP	5	Generate Parity
KRPT	6	Knock down Rupt priority
NISQ	7	New instruction to the SQ register
RA	8	Read A
RB	9	Read B
RB14	10	Read bit 14
RC	11	Read C
RG	12	Read G
RLP	13	Read LP
RP2	14	Read parity 2
RQ	15	Read Q
RRPA	16	Read RUPT address
RSB	17	Read sign bit
RSCT	18	Read selected counter address
RU	19	Read sum
RZ	20	Read Z
R1	21	Read 1
R1C	22	Read 1 complimented
R2	23	Read 2
R22	24	Read 22
R24	25	Read 24
ST1	26	Stage 1
ST2	27	Stage 2
TMZ	28	Test for minus zero
TOV	29	Test for overflow
TP	30	Test parity
TRSM	31	Test for resume
TSGN	32	Test sign
TSGN2	33	Test sign 2
WA	34	Write A
WALP	35	Write A and LP
WB	36	Write B
Wgx	37	Write G (do not reset)
WLP	38	Write LP
WOVC	39	Write overflow counter
WOVI	40	Write overflow RUPT inhibit
WOVR	41	Write overflow
WP	42	Write P
Wpx	43	Write P (do not reset)
WP2	44	Write P2
WQ	45	Write Q
WS	46	Write S
WX	47	Write X
WY	48	Write Y
Wyx	49	Write Y (do not reset)

WZ	50	Write Z

Control signal outputs from CPM-A used as inputs to CPM-B only (not used outside of CPM):

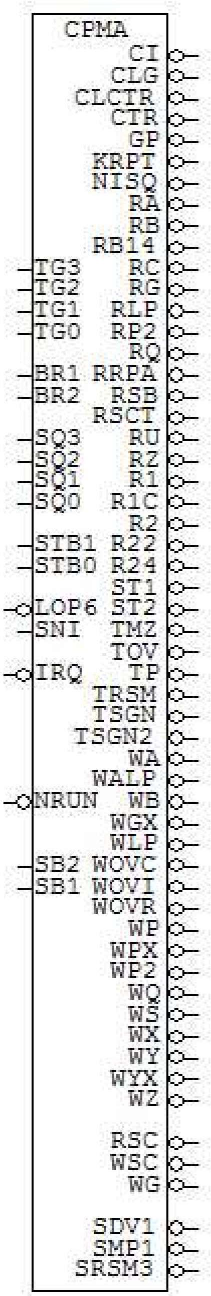
SIGNAL	#	DESCRIPTION
RSC	51	Read special and central (output to B only, not outside CPM)
WSC	52	Write special and central (output to B only, not outside CPM)
WG	53	Write G (output to B only, not outside CPM)

Control signal outputs from CPM-A used as inputs to CPM-C only (not used outside of CPM):

SIGNAL	#	DESCRIPTION
SDV1	54	Subsequence DV1 is currently active
SMP1	55	Subsequence MP1 is currently active
SRSM3	56	Subsequence RSM3 is currently active

CPM-A INPUTS:

I/F	signal	full name	state definition
TPG:			
	TPG_Q3	TPG STATE	where Q3 is MSB, Q0 is LSB:
	TPG_Q2		00 = STBY
	TPG_Q1		01 = PWRON
	TPG_Q0		02 = TP1
			03 = TP2
			04 = TP3
			05 = TP4
			06 = TP5
			07 = TP6
			08 = TP7
			09 = TP8
			10 = TP9
			11 = TP10
			12 = TP11
			13 = TP12
			14 = SRLSE
			15 = WAIT
SEQ:			
	BR1	BRANCH REG 1	BR1 is MSB, BR2 is LSB
	BR2	BRANCH REG 2	BR00=0:BR1=0, BR2=0
			BR01=1:BR1=0, BR2=1
			BR10=2:BR1=1, BR2=0
			BR11=3:BR1=1, BR2=1
	SQ_3	INST REG	where SQ_3 is MSB, SQ_0 is LSB
	SQ_2		
	SQ_1		
	SQ_0		
	STB_1	STAGE REG	where STB_1 is MSB, STB_0 is LSB
	STB_0		
	LOOP6	LOOPCNTR EQ 6	0=LOOPCNTR is holding the number 6.
	SNI	SELECT NEXT INST	1=select next instruction (SNI register)
CTR:			
	SB_01	SUB SEL 01	SB_01 is LSB; SB_02 is MSB
	SB_02	SUB SEL 02	00=no counter; 01=PINC; 10=MINC
INT:			



IRQ	INT RQST	0=interrupt requested.

MON:

NRUN	RUN/HALT	0=run, 1=step

CPM-A OUTPUTS:

signal	full name	state definition
CI	SET CARRY IN	0=Carry in
CLG	CLR G	0=Clear G
CLCTR	CLR LOOP CTR	0=Clear loop counter
CTR	INCR LOOP CTR	0=Loop counter
GP	GEN PARITY	0=Generate Parity
KRPT	KNOCK DOWN RUPT	0=Knock down Rupt priority
NISQ	NEW INSTRUCT	0=New instruction to the SQ reg
RA	READ A	0=Read A
RB	READ B	0=Read B
RB14	READ BIT 14	0=Read bit 14
RC	READ C	0=Read C
RG	READ G	0=Read G
RLP	READ LP	0=Read LP
RP2	READ PARITY 2	0=Read parity 2
RQ	READ Q	0=Read Q
RRPA	READ RUPT ADDR	0=Read RUPT address
RSB	READ SIGN	0=Read sign bit
RSCT	READ CNTR ADDR	0=Read selected counter address
RU	READ U	0=Read sum
RZ	READ Z	0=Read Z
R1	READ 1	0=Read 1
R1C	READ 1 COMP	0=Read 1 complimented
R2	READ 2	0=Read 2
R22	READ 22	0=Read 22
R24	READ 24	0=Read 24
ST1	SET STAGE 1	0=Stage 1
ST2	SET STAGE 2	0=Stage 2
TMZ	TEST MINUS ZERO	0=Test for minus zero
TOV	TEST OVF	0=Test for overflow
TP	TEST PARITY	0=Test parity
TRSM	TEST RESUME	0=Test for resume
TSGN	TEST SIGN	0=Test sign
TSGN2	TEST SIGN 2	0=Test sign 2
WA	WRITE A	0=Write A
WALP	WRITE A/LP	0=Write A and LP
WB	WRITE B	0=Write B
WGx	WRITE G NO RESET	0=Write G (do not reset)
WLP	WRITE LP	0=Write LP
WOVC	WRITE OVF CNTR	0=Write overflow counter
WOVI	WRITE OVF RUPT INH	0=Write overflow RUPT inhibit
WOVR	WRITE OVF	0=Write overflow
WP	WRITE P	0=Write P
WPx	WRITE P NO RESET	0=Write P (do not reset)
WP2	WRITE P2	0=Write P2

WQ	WRITE Q	0=Write Q
WS	WRITE S	0=Write S
WX	WRITE X	0=Write X
WY	WRITE Y	0=Write Y
WYx	WRITE Y NO RESET	0=Write Y (do not reset)
WZ	WRITE Z	0=Write Z

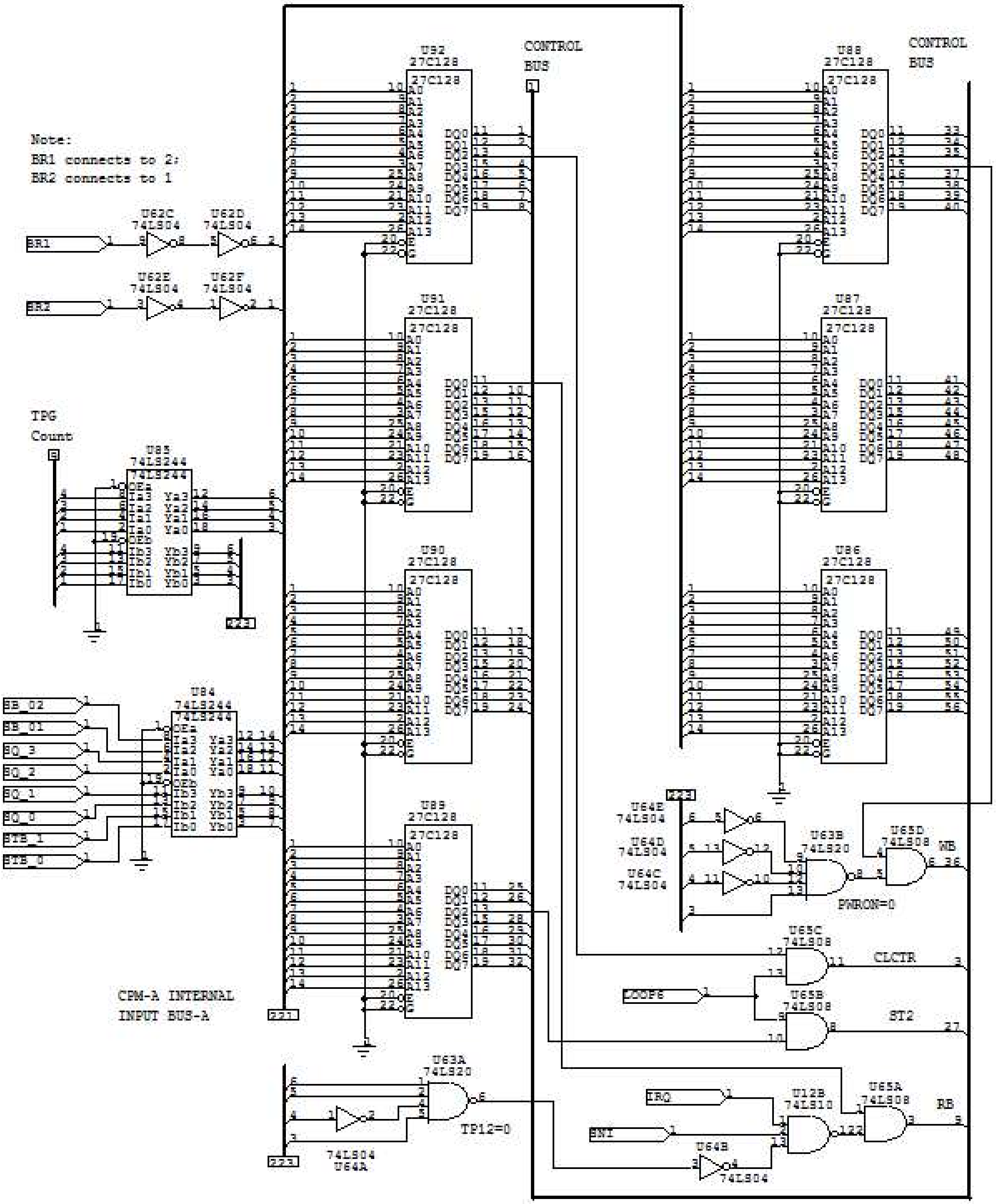
OUTPUTS TO CPM-B ONLY; NOT USED OUTSIDE CPM

RSC	READ SPECIAL REG	0=Read special and central
WSC	WRITE SPECIAL REG	0=Write special and central
WG	WRITE G	0=Write G

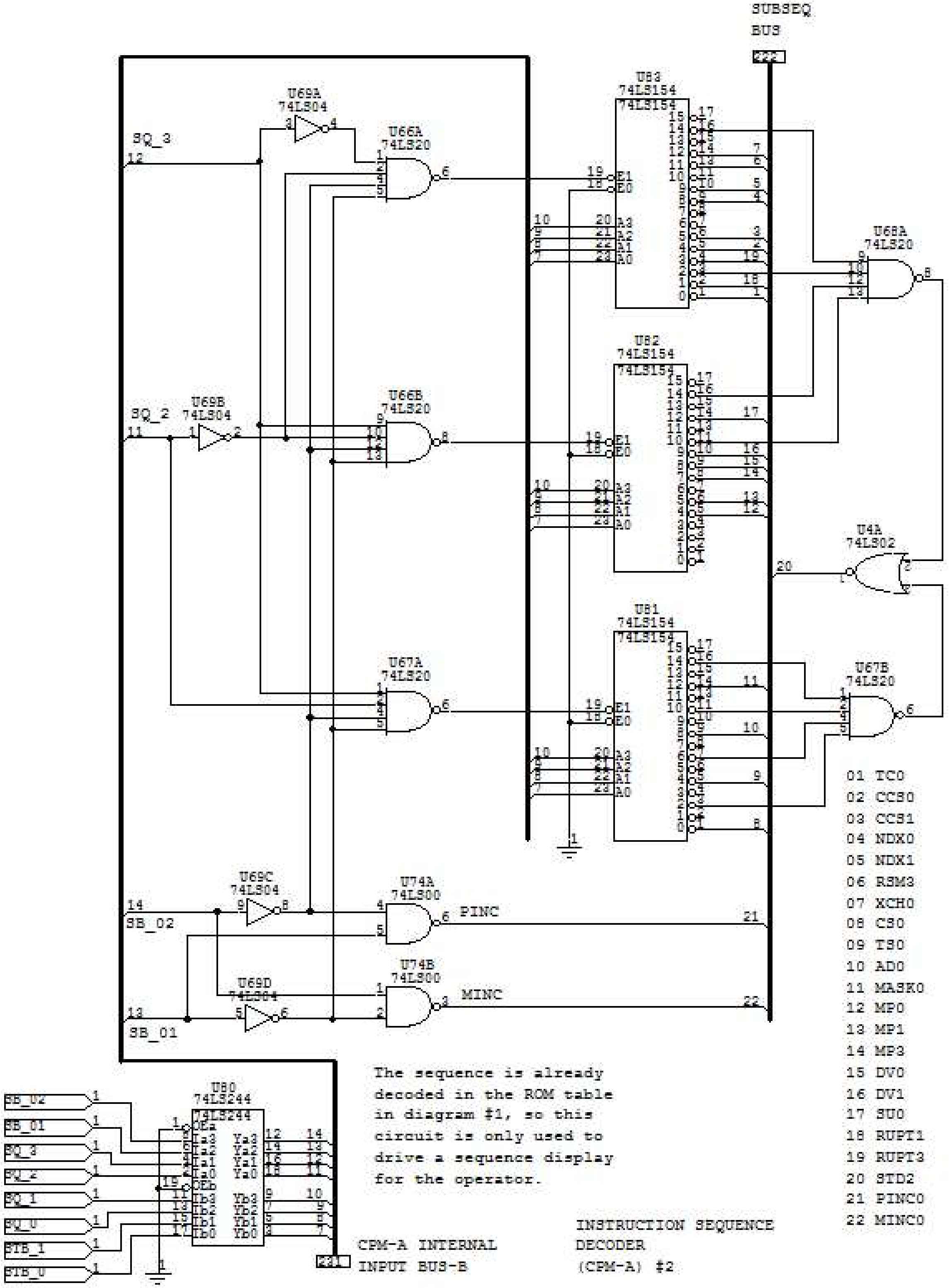
OUTPUTS TO CPM-C ONLY; NOT USED OUTSIDE CPM

SDV1	SUBSEQ DV1	0=Subsequence DV1 is selected
SMP1	SUBSEQ MP1	0=Subsequence MP1 is selected
SRSM3	SUBSEQ RSM3	0=Subsequence RSM3 is selected

Note:
BR1 connects to 2;
BR2 connects to 1



CONTROL PULSE MATRIX A
(CPM-A) #1



The sequence is already
decoded in the ROM table
in diagram #1, so this
circuit is only used to
drive a sequence display
for the operator.

01	TC0
02	CCS0
03	CCS1
04	NDX0
05	NDX1
06	RSM3
07	XCH0
08	CS0
09	TS0
10	AD0
11	MASK0
12	MP0
13	MP1
14	MP3
15	DV0
16	DV1
17	SU0
18	RUPT1
19	RUPT3
20	STD2
21	PINC0
22	MINC0

INSTRUCTION SEQUENCE
DECODER
(CPM-A) #2

CPM-B (Control Pulse Matrix B)

Some AGC registers are mapped onto low memory addresses (00 - 17 octal), so reading or writing to those addresses causes data to be read from, or written into, flip-flop registers instead of memory. These addresses include the central registers (A, Q, Z, LP), the bank register (BNK), and I/O registers.

Addresses 16 and 17 are used in conjunction with the INDEX instruction to inhibit or enable interrupts; this is a trick used to extend the instruction set; to cram more instructions into a 3-bit op code.

The addresses from 20-23 are in eraseable memory, but any data written into those addresses is rotated or shifted. This is implemented through the G register in the MEM module.

Addresses 24-27 are reserved for saving the central register before servicing an interrupt. Registers Z and B are automatically saved by the interrupt subsequence. Registers A and Q must be saved by the interrupt service routine.

SPECIAL REGISTERS

These addresses called special registers. All numbers are in octal.

addr	Flip-Flop registers	
00	A	register (accumulator)
01	Q	register
02	Z	register (program counter)
03	LP	register
04	IN0	input register 0
05	IN1	input register 1
06	IN2	input register 2
07	IN3	input register 3
10	OUT0	output register 0
11	OUT1	output register 1
12	OUT2	output register 2
13	OUT3	output register 3
14	OUT4	output register 4
15	BANK	bank register
16	RELINT	
17	INHINT	
	Eraseable memory registers	
20	CYR	cycle right
21	SR	shift right
22	CYL	cycle left
23	SL	shift left
24	ZRUPT	save register Z
25	BRUPT	save register B
26	ARUPT	save register A
27	QRUPT	save register Q

CPM-B translates the WG, RSC, and WSC signals generated by SUBSYSTEM A into signals that read from or write to these registers. The logic is given below (all numbers are in octal):

if WG is asserted from CPM-A,

...and the address bus = 020:	assert:W20
...and the address bus = 021:	assert:W21
...and the address bus = 022:	assert:W22
...and the address bus = 023:	assert:W23

...otherwise, if the address bus > 17: assert:WGn (not a central register)

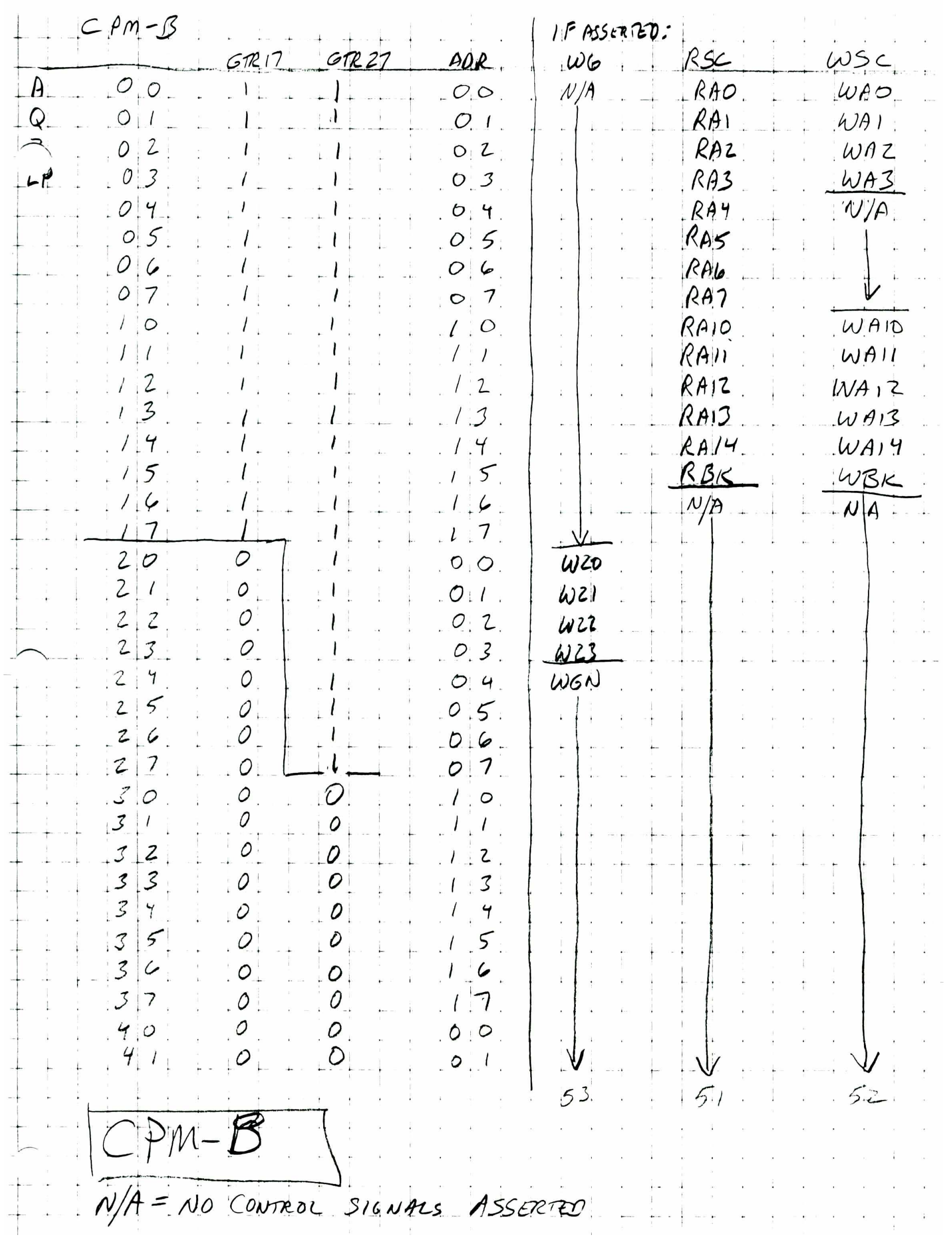
if RSC is asserted from CPM-A

...and the address bus = 00:	assert:RA0
...and the address bus = 01:	assert:RA1
...and the address bus = 02:	assert:RA2
...and the address bus = 03:	assert:RA3
...and the address bus = 04:	assert:RA4
...and the address bus = 05:	assert:RA5
...and the address bus = 06:	assert:RA6
...and the address bus = 07:	assert:RA7
...and the address bus = 010:	assert:RA10
...and the address bus = 011:	assert:RA11
...and the address bus = 012:	assert:RA12
...and the address bus = 013:	assert:RA13
...and the address bus = 014:	assert:RA14
...and the address bus = 015:	assert:RBK
...and the address bus = 016:	do nothing
...and the address bus = 017:	do nothing

if WSC is asserted from CPM-A,

...and the address bus = 00:	assert:WA0
...and the address bus = 01:	assert:WA1
...and the address bus = 02:	assert:WA2
...and the address bus = 03:	assert:WA3
...and the address bus = 010:	assert:WA10
...and the address bus = 011:	assert:WA11
...and the address bus = 012:	assert:WA12
...and the address bus = 013:	assert:WA13
...and the address bus = 014:	assert:WA14
...and the address bus = 015:	assert:WBK
...and the address bus = 016:	do nothing
...and the address bus = 017:	do nothing

Here's a table I developed to work out the relationships between addresses and CPM-B logic signals:



	CPM-B				IF ASSERTED:		
		GTR17	GTR27	ADR	W6	RSC	WSC
A	0 0	1	1	0 0	N/A	RA0	WA0
Q	0 1	1	1	0 1		RA1	WA1
	0 2	1	1	0 2		RA2	WA2
LP	0 3	1	1	0 3		RA3	WA3
	0 4	1	1	0 4		RA4	N/A
	0 5	1	1	0 5		RA5	
	0 6	1	1	0 6		RA6	
	0 7	1	1	0 7		RA7	
	1 0	1	1	1 0		RA10	WA10
	1 1	1	1	1 1		RA11	WA11
	1 2	1	1	1 2		RA12	WA12
	1 3	1	1	1 3		RA13	WA13
	1 4	1	1	1 4		RA14	WA14
	1 5	1	1	1 5		RBK	WBK
	1 6	1	1	1 6		N/A	N/A
	1 7	1	1	1 7			
	2 0	0	1	0 0	W20		
	2 1	0	1	0 1	W21		
	2 2	0	1	0 2	W22		
	2 3	0	1	0 3	W23		
	2 4	0	1	0 4	WGN		
	2 5	0	1	0 5			
	2 6	0	1	0 6			
	2 7	0	1	0 7			
	3 0	0	0	1 0			
	3 1	0	0	1 1			
	3 2	0	0	1 2			
	3 3	0	0	1 3			
	3 4	0	0	1 4			
	3 5	0	0	1 5			
	3 6	0	0	1 6			
	3 7	0	0	1 7			
	4 0	0	0	0 0			
	4 1	0	0	0 1			
					53	51	52

CPM-B

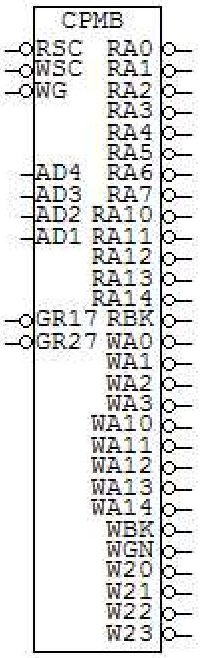
N/A = NO CONTROL SIGNALS ASSERTED

CPM-B CONTROL SIGNALS

SIGNAL	#	DESCRIPTION
RA0	57	Read register at address 0 (A)
RA1	58	Read register at address 1 (Q)
RA2	59	Read register at address 2 (Z)
RA3	60	Read register at address 3 (LP)
RA4	61	Read register at address 4
RA5	62	Read register at address 5
RA6	63	Read register at address 6
RA7	64	Read register at address 7
RA10	65	Read register at address 10 (octal)
RA11	66	Read register at address 11 (octal)
RA12	67	Read register at address 12 (octal)
RA13	68	Read register at address 13 (octal)
RA14	69	Read register at address 14 (octal)
RBK	70	Read BNK
WA0	71	Write register at address 0 (A)
WA1	72	Write register at address 1 (Q)
WA2	73	Write register at address 2 (Z)
WA3	74	Write register at address 3 (LP)
WA10	75	Write register at address 10 (octal)
WA11	76	Write register at address 11 (octal)
WA12	77	Write register at address 12 (octal)
WA13	78	Write register at address 13 (octal)
WA14	79	Write register at address 14 (octal)
WBK	80	Write BNK
WGn	81	Write G (normal gates)
W20	82	Write into CYR
W21	83	Write into SR
W22	84	Write into CYL
W23	85	Write into SL

CPM-B INPUTS:

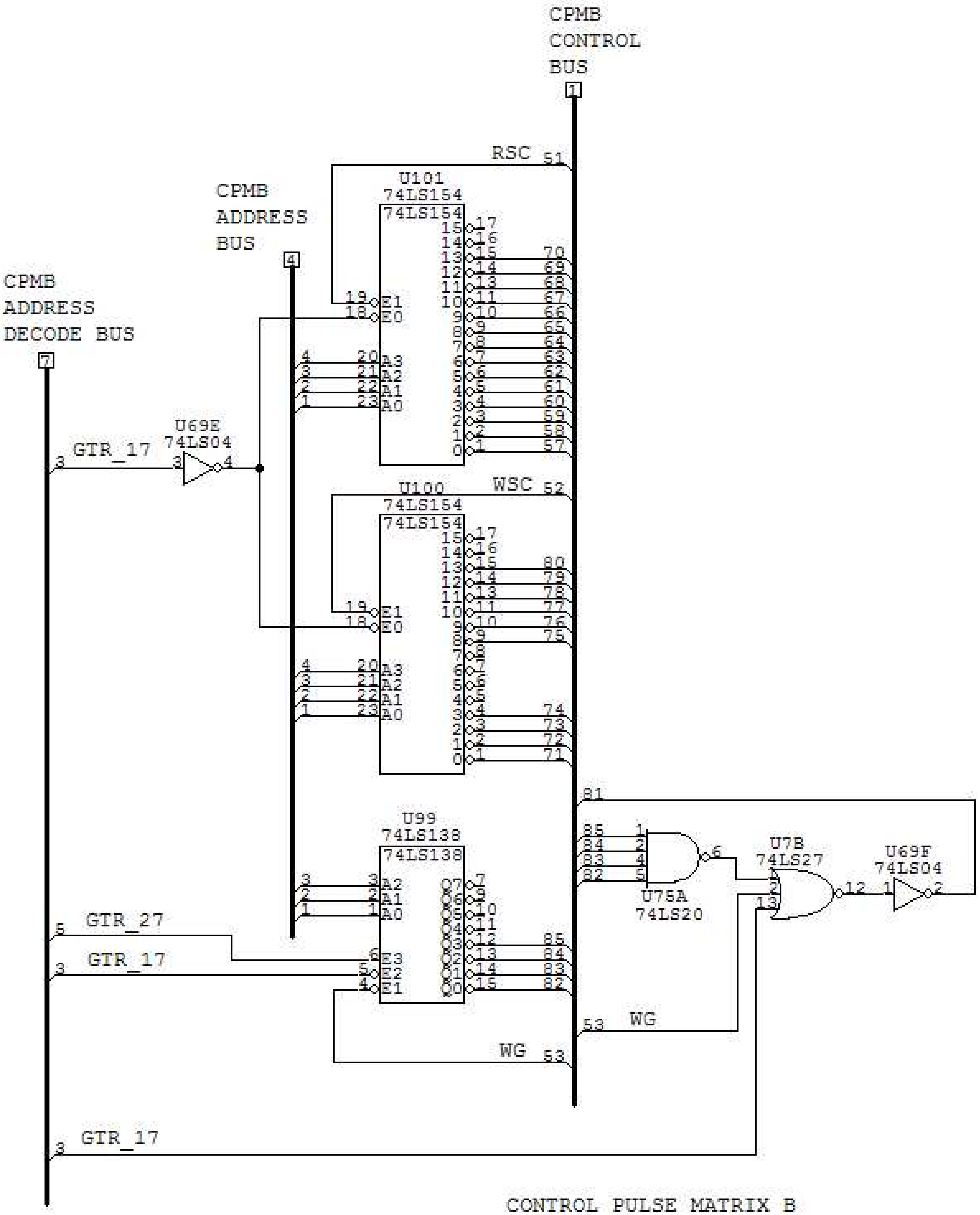
I/F	signal	full name	state definition
CPM-A			
	RSC	READ SPECIAL REG	0=Read special and central
	WSC	WRITE SPECIAL REG	0=Write special and central
	WG	WRITE G	0=Write G
ADR:			
	AD_4	ADDRESS	AD_4=MSB, AD_1=LSB: (low-order bits of 14-bit address)
	AD_3		
	AD_2		
	AD_1		
	GTR_17	ADDRESS > 017	0=CADR in register S > 017
	GTR_27	ADDRESS > 027	0=CADR in register S > 027



CPM-B OUTPUTS:

I/F	signal	full name	state definition
	RA0	READ ADDR 0 (A)	0=Read reg at address 0
	RA1	READ ADDR 1	0=Read reg at address 1 (Q)
	RA2	READ ADDR 2	0=Read reg at address 2 (Z)
	RA3	READ ADDR 3	0=Read reg at address 3 (LP)
	RA4	READ ADDR 4	0=Read reg at address 4
	RA5	READ ADDR 5	0=Read reg at address 5
	RA6	READ ADDR 6	0=Read reg at address 6
	RA7	READ ADDR 7	0=Read reg at address 7
	RA10	READ ADDR 10	0=Read reg at address 10 (octal)
	RA11	READ ADDR 11	0=Read reg at address 11 (octal)
	RA12	READ ADDR 12	0=Read reg at address 12 (octal)
	RA13	READ ADDR 13	0=Read reg at address 13 (octal)
	RA14	READ ADDR 14	0=Read reg at address 14 (octal)
	RBK	READ BNK	0=Read BNK reg
	WA0	WRITE ADDR 0	0=Write reg at address 0 (A)
	WA1	WRITE ADDR 1	0=Write reg at address 1 (Q)
	WA2	WRITE ADDR 2	0=Write reg at address 2 (Z)
	WA3	WRITE ADDR 3	0=Write reg at address 3 (LP)
	WA10	WRITE ADDR 10	0=Write reg at address 10 (octal)
	WA11	WRITE ADDR 11	0=Write reg at address 11 (octal)
	WA12	WRITE ADDR 12	0=Write reg at address 12 (octal)
	WA13	WRITE ADDR 13	0=Write reg at address 13 (octal)
	WA14	WRITE ADDR 14	0=Write reg at address 14 (octal)
	WBK	WRITE BNK	0=Write BNK reg

WGn	WRITE G NORMAL	0=Write G (normal gates)
W20	WRITE ADDR 20	0=Write into CYR
W21	WRITE ADDR 21	0=Write into SR
W22	WRITE ADDR 22	0=Write into CYL
W23	WRITE ADDR 23	0=Write into SL



CONTROL PULSE MATRIX B
(CPM-B) #1

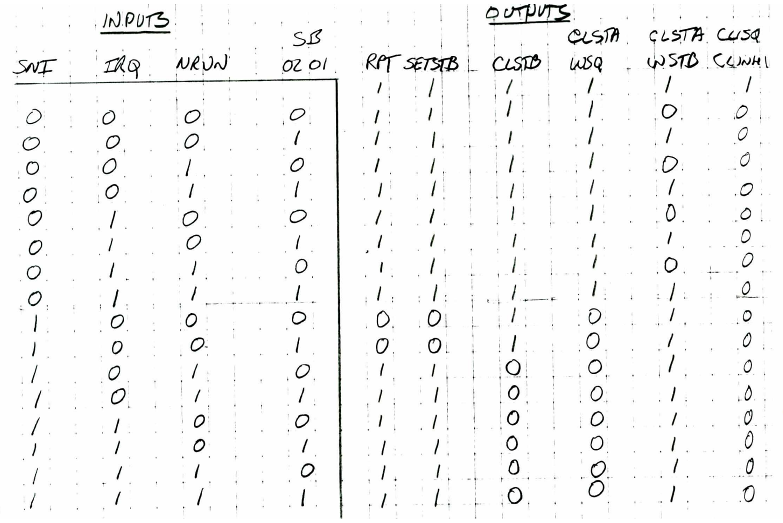
CPM-C (Control Pulse Matrix C)

The CPM-C subsystem issues control signals for the memory cycle, selecting the next instruction, and performing priority counter subsequences. These signals are issued at specific points in the 12-step cycle of the time pulse generator (TPG).

STBY:		assert:GENRST	Resets various AGC registers.
PWRON:		assert:R2000	Put the starting address on the read bus. CPM-A asserts:WB, which copies the data into register B, which is the prefetch register for the next instruction. Since the opcode for a branch is 0, the instruction in B becomes TC 2000, which is the first instruction always executed by the AGC.
TP1:		assert:CLISQ	SNI <- 0. Moved from TP12 to TP1 because CLISQ was getting cleared in this hardware AGC replica before TPG was clocked; therefore TPG was not seeing the SNI indication.
TP5:	if:	the address bus > 17	(not a central register)
	and	the address bus < 2000	(not fixed memory; must be erasable)
	and	SDV1 or SMP1 are not asserted	(not a loop counter subsequence)
	then:	assert:SBWG	read erasable memory into G by TP6
	if:	the address bus = 17	
	then:	assert:INH	INHINT instruction (INDEX 017)
	if:	the address bus = 16	
	then:	assert:CLINH	RELINT instruction (INDEX 016)
TP6:	if:	the address bus > 1777	(not eraseable memory)
	and	SDV1 or SMP1 are not asserted	(not a loop counter subsequence)
	then:	assert:SBWG	read fixed memory into G register by TP7
TP11:	if:	the address bus > 17	(not a central register)
	and	the address bus < 2000	(not fixed memory; must be erasable)
	and	SDV1 or SMP1 are not asserted	(not a loop counter subsequence)
	then:	assert:WE	G register written to memory beginning at TP11; Memory updates are in G by TP10 for all normal and extracode instructions, but the PINC and MINC sequences write to G in TP10 because they need to update the parity bit.
	if:	SRSM3 is asserted	
	then:	assert:CLRP	Additional interrupts are inhibited during servicing of an interrupt; Remove the inhibition when RESUME is executed (INDEX 025)

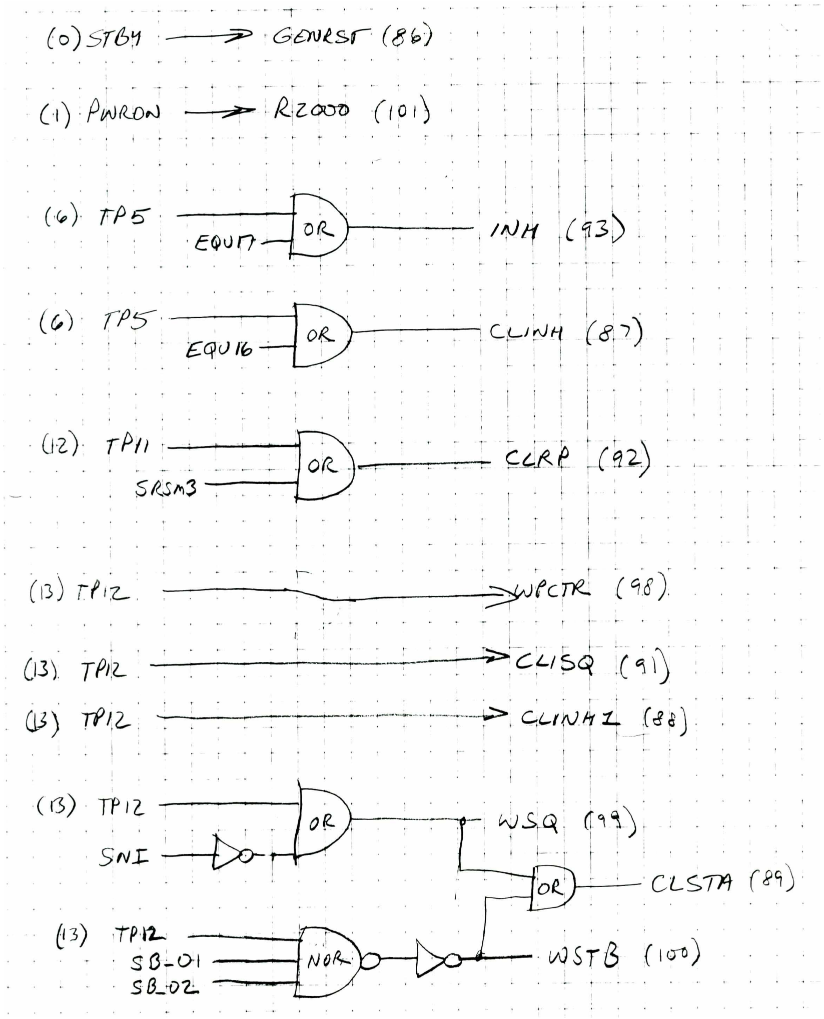
```
TP12:   assert:WPCTR                            Check the priority counters; service any
                                                waiting inputs on the next memory cycle.

        if:     the SNI register = 1            (if SNI is set, get next instruction)
        then:
                if:     IRQ is asserted         (if interrupt requested (see CPM-A for
                                                similar assertion))
                then:
                        assert:RPT              Read the interrupt vector.
                        assert:SETSTB           STB <- 1. Will cause the RUPT1
                                                subsequence to execute.
                else:                           (not an interrupt; a normal instruction)
                        assert:CLSTB            STB <- 0 . The CPM-A will assert RB here,
                                                which, when accompanied by WSQ
                                                (below), will read the next instruction from
                                                register B onto the bus. WSQ will write it
                                                into SQ.
                endif
                assert:WSQ                      Write the next instruction (on the write
                                                bus) into the SQ register.
                assert:CLSTA                    Clear STA register.


                assert:CLINH1                   Clear INHINT1. Removes inhibition of
                                                interrupts (if they were) AFTER the next
                                                instruction
        else:                                   (not a new instruction)
                if:     CTR00 or CTR11          if previous sequence was not a PINC or
                                                MINC, get next subsequence for same
                                                instruction. if the previous sequence was
                                                PINC or MINC, we already have the
                                                subsequence, but it was interrupted by the
                                                counter.
                then:
                        assert:WSTB             Copy STB <- STA. Gets next sequence for
                                                same instruction.
                        assert:CLSTA            STA <- 0
```

This truth table shows the logic needed at TP12 to produce the RPT, SETSTB, CLSTB, CLSTA, WSQ, WSTB, CLISQ, and CLINH1 signals, given the SNI, IRQ, NRUN, and SB inputs.

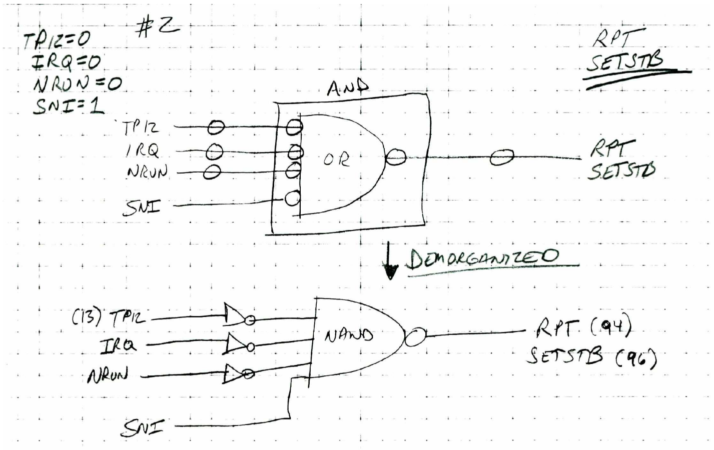


INPUTS				OUTPUTS					
			SB				CLSTA	CLSTA	CLISQ
SNI	IRQ	NRUN	02 01	RPT	SETSTB	CLSTB	WSQ	WSTB	CLINH1
				1	1	1	1	1	1
0	0	0	0	1	1	1	1	0	0
0	0	0	1	1	1	1	1	1	0
0	0	1	0	1	1	1	1	0	0
0	0	1	1	1	1	1	1	1	0
0	1	0	0	1	1	1	1	0	0
0	1	0	1	1	1	1	1	1	0
0	1	1	0	1	1	1	1	0	0
0	1	1	1	1	1	1	1	1	0
1	0	0	0	0	0	1	0	1	0
1	0	0	1	0	0	1	0	1	0
1	0	1	0	1	1	0	0	1	0
1	0	1	1	1	1	0	0	1	0
1	1	0	0	1	1	0	0	1	0
1	1	0	1	1	1	0	0	1	0
1	1	1	0	1	1	0	0	1	0
1	1	1	1	1	1	0	0	1	0

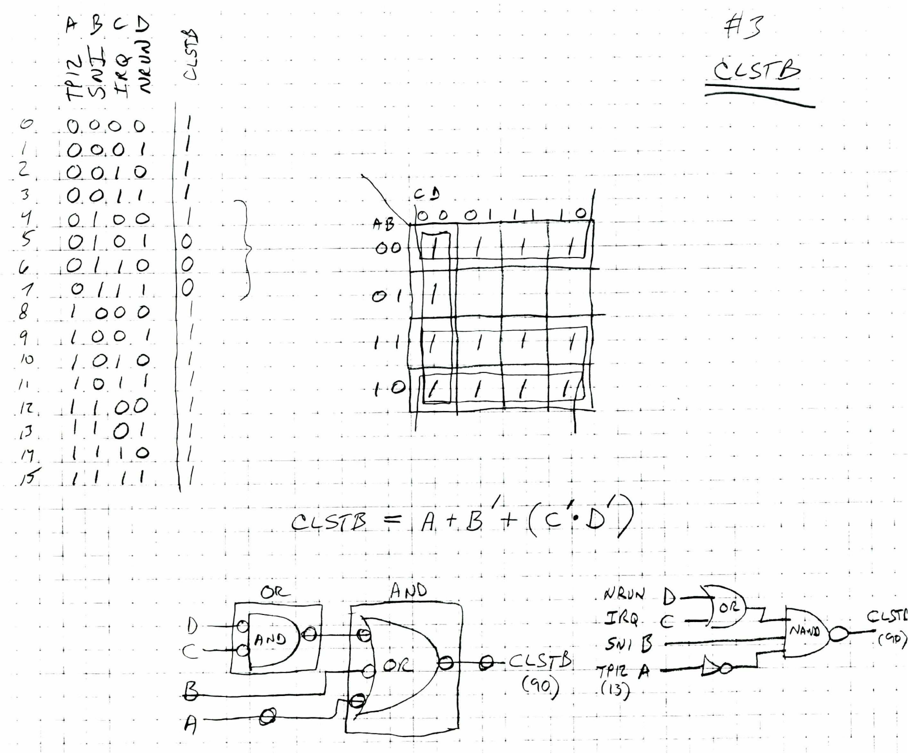
This chart shows the logic for some of the more simple control pulses generated by CMP-C. Some pulses reduce to states from the TPG, such as CLISQ, WPCTR, and CLINH1, which reduce to TP12.



This is the logic for RPT and SETSTB.



The design for CLSTB is a little more complex. The truth table to the left is reduced through the Karnaugh map to the Minterm equation below the map. A little bubble pushing DeMorganizes the logic to the final solution at the bottom right.

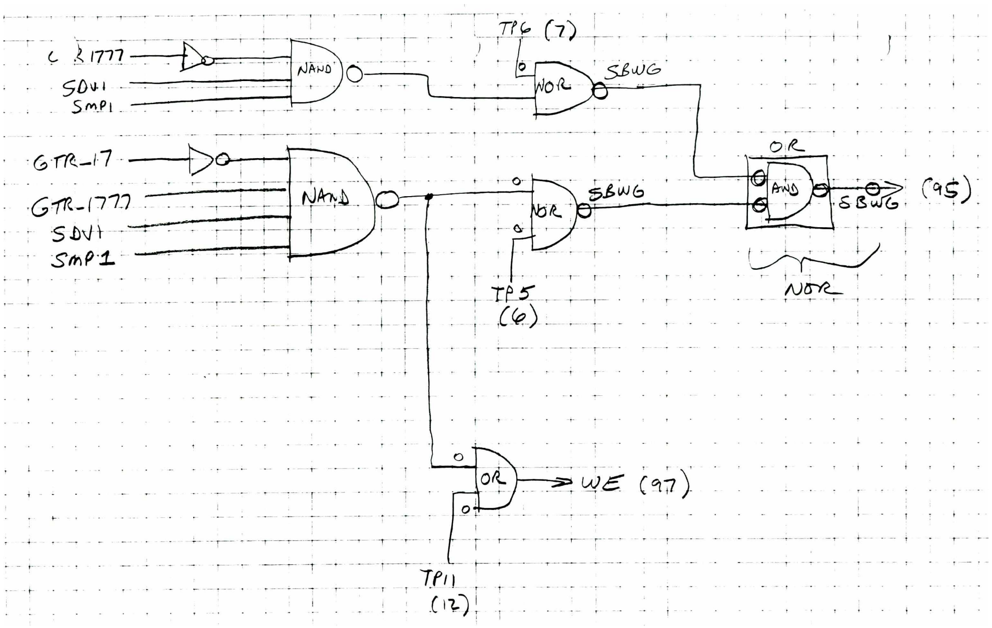


	A TP12	B SNI	C IRQ	D NRUN	CLSTB
0	0	0	0	0	1
1	0	0	0	1	1
2	0	0	1	0	1
3	0	0	1	1	1
4	0	1	0	0	1
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	1
9	1	0	0	1	1
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

AB \ CD	00	01	11	10
00	1	1	1	1
01	1			
11	1	1	1	1
10	1	1	1	1

$$CLSTB = A + B' + (C' \cdot D')$$

This is the logic for WE and SBWG, two signals that control memory access.

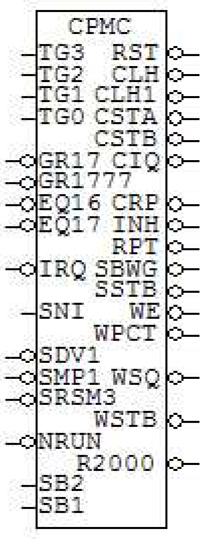


CPM-C CONTROL SIGNALS

SIGNAL	#	DESCRIPTION
GENRST	86	General Reset
CLINH	87	Clear INHINT
CLINH1	88	Clear INHINT1
CLSTA	89	Clear state counter A (STA)
CLSTB	90	Clear state counter B (STB)
CLISQ	91	Clear SNI
CLRP	92	Clear RPCELL
INH	93	Set INHINT
RPT	94	Read RUPT opcode
SBWG	95	Write G from memory
SETSTB	96	Set the ST1 bit of STB
WE	97	Write E-MEM from G
WPCTR	98	Write PCTR (latch priority counter sequence)
WSQ	99	Write SQ
WSTB	100	Write stage counter B (STB)
R2000	101	Read 2000

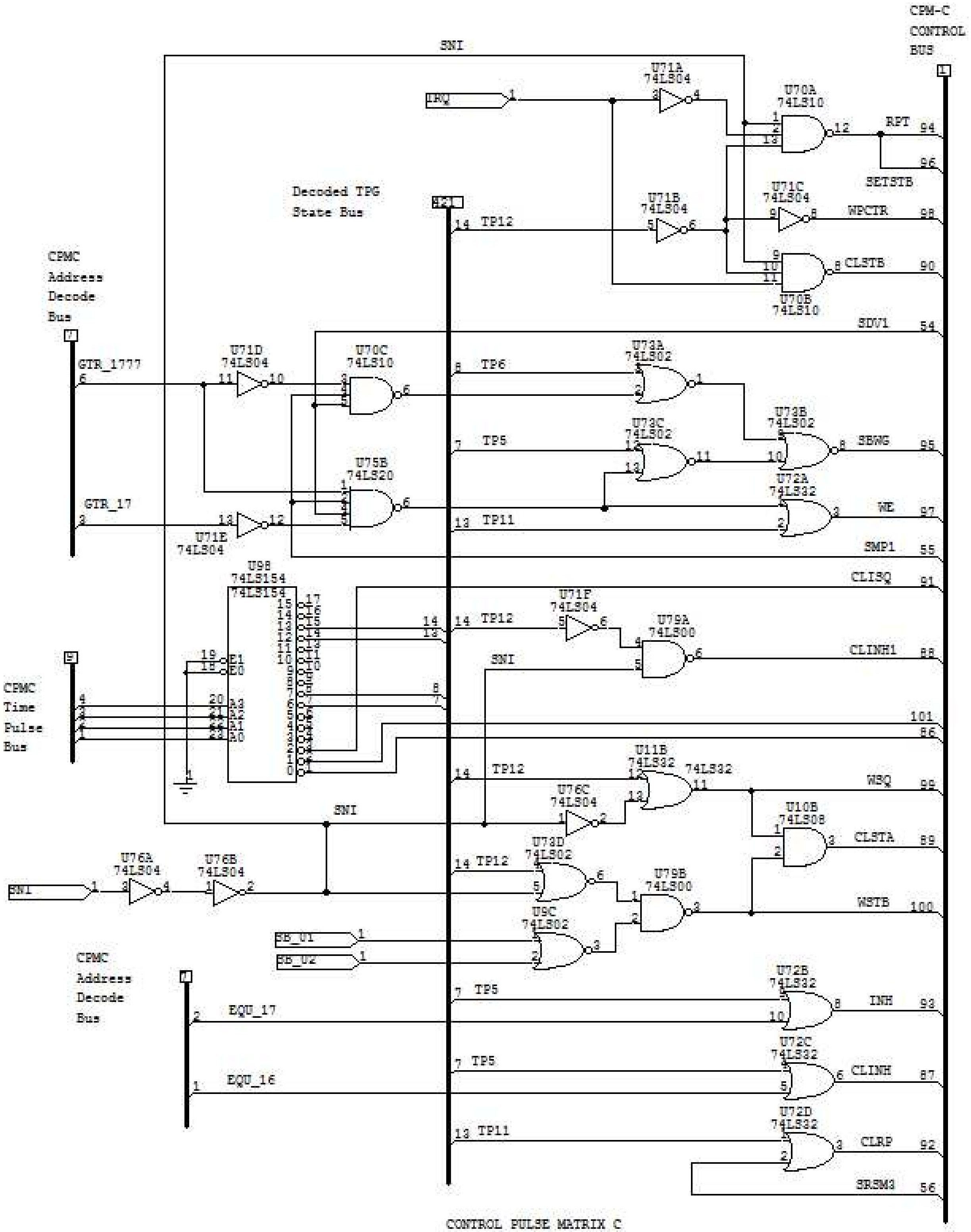
CPM INPUTS:

I/F	signal	full name	state definition
TPG:			
	TPG_Q3	TPG STATE	where Q3 is MSB, Q0 is LSB:
	TPG_Q2		00 = STBY
	TPG_Q1		01 = PWRON
	TPG_Q0		02 = TP1
			03 = TP2
			04 = TP3
			05 = TP4
			06 = TP5
			07 = TP6
			08 = TP7
			09 = TP8
			10 = TP9
			11 = TP10
			12 = TP11
			13 = TP12
			14 = SRLSE
			15 = WAIT
ADR:			
	EQU_16	ADDRESS = 016	0=CADR in register S = 016
	EQU_17	ADDRESS = 017	0=CADR in register S = 017
	GTR_17	ADDRESS > 017	0=CADR in register S > 017
	GTR_1777	ADDRESS > 01777	0=CADR in register S > 01777
CPM-A:			
	SDV1	SUBSEQ DV1	0=Subsequence DV1 is selected
	SMP1	SUBSEQ MP1	0=Subsequence MP1 is selected
	SRSM3	SUBSEQ RSM3	0=Subsequence RSM3 is selected
CTR:			
	SB_01	SUB SEL 01	SB_01 is LSB; SB_02 is MSB
	SB_02	SUB SEL 02	00=no counter; 01=PINC; 10=MINC
SEQ:			
	SNI	SELECT NEXT INST	1=select next instruction (SNI register)
INT:			
	IRQ	INT RQST	0=interrupt requested.
MON:			
	NRUN	RUN/HALT	0=run, 1=step



CPM OUTPUTS:

I/F	signal	full name	state definition
	GENRST	GENERAL RESET	0=General Reset
	CLINH	CLEAR INHINT	0=Clear INHINT
	CLINH1	CLEAR INHINT1	0=Clear INHINT1
	CLSTA	CLEAR STA	0=Clear state counter A (STA)
	CLSTB	CLEAR STB	0=Clear state counter B (STB)
	CLISQ	CLEAR SNI	0=Clear SNI
	CLRP	CLEAR RPCELL	0=Clear RPCELL
	INH	SET INHINT	0=Set INHINT
	RPT	READ RUPT	0=Read RUPT opcode
	SBWG	WRITE G	0=Write G from memory
	SETSTB	SET ST1	0=Set the ST1 bit of STB
	WE	WRITE EMEM	0=Write E-MEM from G
	WPCTR	WRITE PCTR	0=Write PCTR (latch priority counter sequence)
	WSQ	WRITE SQ	0=Write SQ
	WSTB	WRITE STB	0=Write stage counter B (STB)
	R2000	READ 2000	0=Read 2000



CONTROL PULSE MATRIX C
(CPM-C) #1

Fabrication

The CTL module is (3) 13"x5" circuit boards, and 1 control panel.

Module Rack

The module framework is designed to resemble a relay rack, but scaled to fit the circuit board dimensions. It is constructed out of 1"x2" pine and spray-painted semi-gloss gray.

Circuit boards are mounted to the rack by 2 phillips screws at either end. Nylon spacers (1/4") are used as standoffs to hold the board edges above the rack. The boards are mounted so the chips are in the back and the pins are wiring are visible from the front.

Power is distributed by 2 heavy aluminum bus bars mounted vertically, one per side, on the back of the module. Machine screws are mounted through the bus bars at evenly-spaced intervals to provide connection points for the boards.

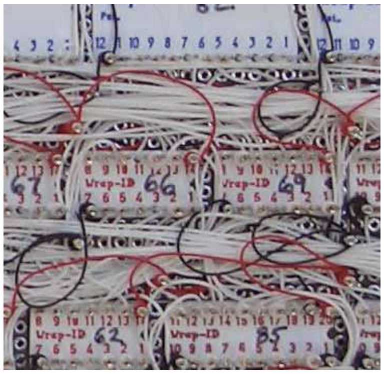
Solid copper wire (24 gauge) connects the boards to the bus bars. Ring terminals are used on the bus bar side of the connection. On the circuit board size, the wires are soldered directly to the supply rails.

Materials were purchased from Home Depot, ACE Hardware, and Radio Shack.

Circuit Boards

The circuit boards are 13"x5" general purpose prototyping boards, epoxy glass with double-side plated through pads on 0.1" centers (JAMECO 21477CL).

ICs are mounted in level 3 machine tooled wire-wrap sockets: 8, 14, 16, 20, 24, and 28 pin (JAMECO). Each socket has the pin-out labeled with a wire-wrap socket ID marker, which slips onto the socket before wrapping (JAMECO). The part number is written onto the ID marker.



Sockets are arranged in 4 horizontal rows on each board, with about 10 sockets per row.

Power is distributed on the back-side of each board by bare 24-gauge solid copper wire supply rails soldered at equal intervals to Klipwrap terminals: 3-prong terminals with a square tail for wire-wrapping (JAMECO 34163CL). A +5V rail runs above each row of sockets and a ground rail runs below. Each rail connects directly to the aluminum module power bus using a ring tail connector.

On the pin side of the board, all connections are made with 30 AWG Kynar wire-wrap wire (JAMECO). Red wire is used for direct connections to the +5V supply rail. Black wire is used for direct connections to ground. White wire is used for everything else.

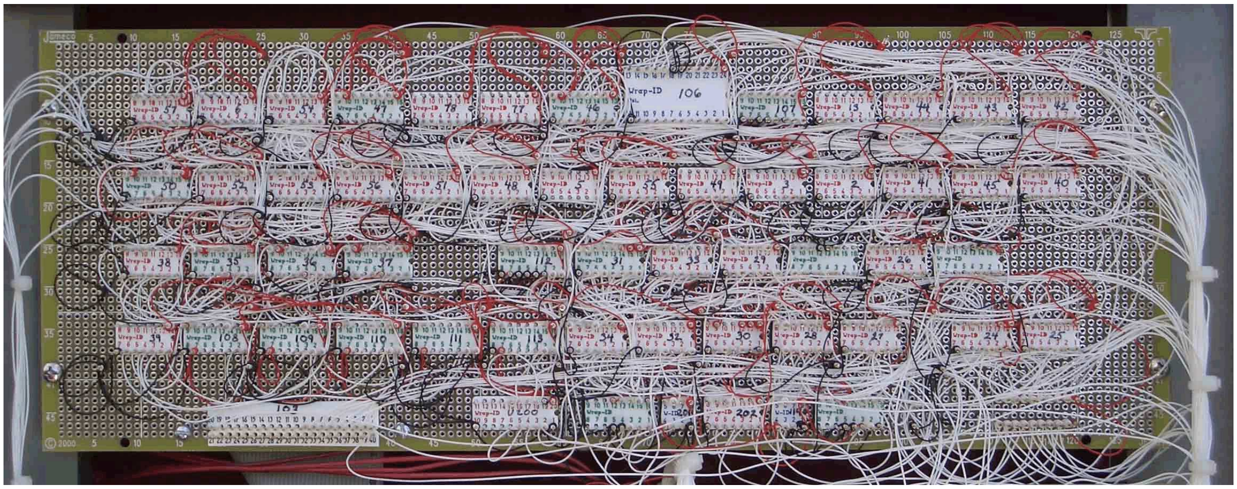
Power connections from the supply rails to each ICs are double-wrapped. Bypassing capacitors (.1 uf disc) are soldered across the supply rails at the Klipwrap terminals; about 1 capacitor for every 2 IC packages.

All connections were stripped and hand-wrapped using a Radio Shack hand-wrap tool. As each connection was made, the corresponding line on the schematic was marked with a colored highlighter.

DIP resistor networks (JAMECO) plugged into 20-pin wire-wrap sockets were used as current limiting resistors for the panel indicators.

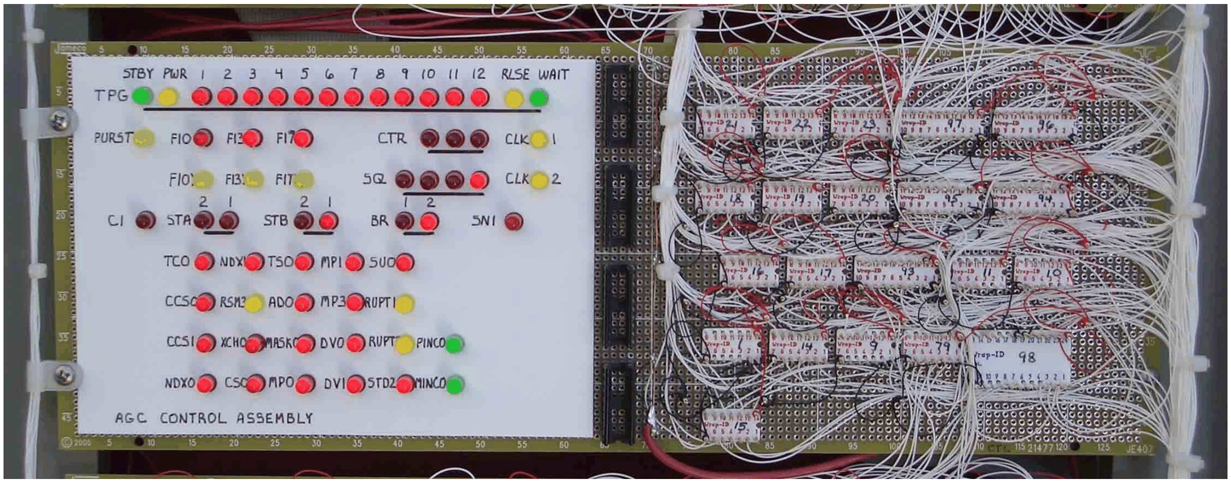
CTL Printed Circuit Board (PCB) A

The A board contains the clock (CLK), the scaler (SCL), and the time pulse generator (TPG).



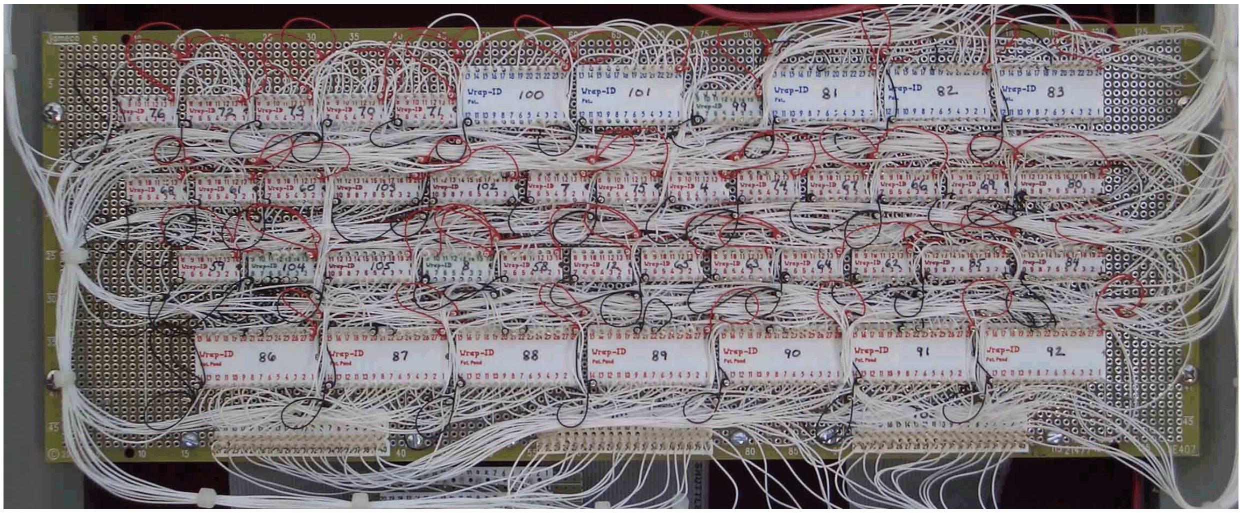
CTL Printed Circuit Board (PCB) B

The B board contains the display indicators, their current-limiting resistor networks, and the open collector drivers. The display panel is a sheet of white styrene plastic. A push pin was used to make holes through the plastic, and the LEDs were inserted in rows. The panel was hand-lettered with an indelible marker. A few chips near the bottom right of the B board are associated with subsystems on the C board.



CTL Printed Circuit Board (PCB) C

The C board contains the sequence generator (SEQ) and control pulse matrixes (CPM-A, CPM-B, and CPM-C). The big ICs at the bottom are the EPROMs that hold the control pulse matrix table for CPM-A. Each EPROM holds the tables for 8 control signals. The large ICs at the top are decoders for the CPM-B and CPM-C logic.



Parts (ICs)

74LS00	(14)	U74,U26,U27,U25,U79,U78,U55,U54,U53,U48,U13,U42,U34,U28
74LS02	(9)	U4,U77,U9,U73,U45,U43,U3,U38,U32
74LS04	(15)	U62,U29,U76,U69,U64,U57,U71,U59,U58,U56,U49,U41,U40,U39,U33
74LS06	(11)	U22,U21,U20,U23,U14,U17,U18,U19,U16,U15,U1
74LS08	(2)	U65,U10
74LS10	(4)	U12,U70,U52,U51
74LS20	(8)	U67,U68,U66,U63,U75,U61,U60,U6
74LS27	(3)	U7,U44,U2
74LS32	(3)	U72,U11,U5
74LS74	(1)	U24
74LS112	(8)	U8,U50,U47,U46,U35,U36,U37,U31
74LS138	(1)	U99
74LS154	(7)	U81,U82,U83,U98,U100,U101,U106
74LS161	(8)	U104,U107,U108,U109,U110,U111,U112,U113
74LS244	(10)	U80,U84,U85,U93,U94,U95,U96,U97,U102,U103
74LS273	(1)	U105
27C128	(7)	U86,U87,U88,U89,U90,U91,U92
4001	(1)	U30
555	(1)	U114

IC's, sockets, PCB's, resistors, capacitors, wire-wrap wire were purchased from JAMECO. The 2.048 MHz crystal and the IDE wire-wrap were from DigiKey. Wire ties, wire-wrap tools, and copper wire were purchased from Radio Shack. IDE ribbon cables were purchased from an online computer supplier.

Power Budget

	qty	mA (ea)		mA (tot)
74LS00		14	2.4	33.6
74LS02		9	2.4	21.6
74LS04		15	3.6	54.0
74LS06		11	3.6	39.6
74LS08		2	4.4	8.8
74LS10		4	1.8	7.2
74LS20		8	1.2	9.6
74LS27		3	3.4	10.2
74LS32		3	4.9	14.7
74LS74		1	4.0	4.0
74LS112		8	4.0	32.0
74LS138		1	6.3	6.3
74LS154		7	6.2	43.4
74LS161		8	19.0	152.0
74LS244		10	32.0	320.0
74LS273		1	17.0	17.0
27C128		7	25.0	175.0
4001	1	0.4	0.4	
555	1	3.0	3.0	
LED	61	20.0	1220.0	

2.2	Amps total
1.0	Amps (excluding LEDs)

EPROM generator program

This C++ program generates all files needed to program the CPM-A EPROMs. The files are generated in Motorola S-Record format.

```
/*
 ****************************************************************
 *
 * CPM-A EPROM GENERATOR
 *
 *9/14/01
 *
 *****************************************************************

  Versions:
        Derived from AGC C++ simulator 1.15.


Operation:
        Generates all of the CPM-A EPROM files in Motorola s2f S-Record format
        suitable for EPROM programmers.

*/

#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <iostream.h>
#include <stdio.h>

#define MAXPULSES 15
#define MAX_IPULSES 5 // no more than 5 instruction-generated pulses active at any time


enum cpType { // **inferred; not defined in orignal R393 AGC4 spec.
        NO_PULSE=0,


        // OUTPUTS FROM SUBSYSTEM A
        CI              =1,     // Carry in
        CLG             =2,     // Clear G
        CLCTR           =3,     // Clear loop counter
        CTR             =4,     // Loop counter
        GP              =5,     // Generate Parity
        KRPT            =6,     // Knock down Rupt priority
        NISQ            =7,     // New instruction to the SQ register
        RA              =8,     // Read A
        RB              =9,     // Read B
        RB14            =10,    // Read bit 14
        RC              =11,    // Read C
        RG              =12,    // Read G
        RLP             =13,    // Read LP
        RP2             =14,    // Read parity 2
        RQ              =15,    // Read Q
        RRPA            =16,    // Read RUPT address
        RSB             =17,    // Read sign bit
        RSCT            =18,    // Read selected counter address
        RU              =19,    // Read sum
        RZ              =20,    // Read Z
        R1              =21,    // Read 1
        R1C             =22,    // Read 1 complimented
        R2              =23,    // Read 2
        R22             =24,    // Read 22
        R24             =25,    // Read 24
        ST1             =26,    // Stage 1
        ST2             =27,    // Stage 2
        TMZ             =28,    // Test for minus zero
        TOV             =29,    // Test for overflow
        TP              =30,    // Test parity
        TRSM            =31,    // Test for resume
        TSGN            =32,    // Test sign
        TSGN2           =33,    // Test sign 2
        WA              =34,    // Write A
```

```
WALP                =35,      // Write A and LP
WB                  =36,      // Write B
WGx                 =37,      // Write G (do not reset)
WLP                 =38,      // Write LP
WOVC                =39,      // Write overflow counter
WOVI                =40,      // Write overflow RUPT inhibit
WOVR                =41,      // Write overflow
WP                  =42,      // Write P
WPx                 =43,      // Write P (do not reset)
WP2                 =44,      // Write P2
WQ                  =45,      // Write Q
WS                  =46,      // Write S
WX                  =47,      // Write X
WY                  =48,      // Write Y
WYx                 =49,      // Write Y (do not reset)
WZ                  =50,      // Write Z


// OUTPUTS FROM SUBSYSTEM A; USED AS INPUTS TO SUBSYSTEM B ONLY;
// NOT USED OUTSIDE CPM
RSC                 =51,      // Read special and central (output to B only, not outside CPM)
WSC                 =52,      // Write special and central (output to B only, not outside CPM)
WG                  =53,      // Write G (output to B only, not outside CPM)

// OUTPUTS FROM SUBSYSTEM A; USED AS INPUTS TO SUBSYSTEM C ONLY;
// NOT USED OUTSIDE CPM
SDV1                =54,      // Subsequence DV1 is currently active
SMP1                =55,      // Subsequence MP1 is currently active
SRSM3               =56,      // Subsequence RSM3 is currently active

// EXTERNAL OUTPUTS FROM SUBSYSTEM B
//
RA0                 =57,      // Read register at address 0 (A)
RA1                 =58,      // Read register at address 1 (Q)
RA2                 =59,      // Read register at address 2 (Z)
RA3                 =60,      // Read register at address 3 (LP)
RA4                 =61,      // Read register at address 4
RA5                 =62,      // Read register at address 5
RA6                 =63,      // Read register at address 6
RA7                 =64,      // Read register at address 7
RA10                =65,      // Read register at address 10 (octal)
RA11                =66,      // Read register at address 11 (octal)
RA12                =67,      // Read register at address 12 (octal)
RA13                =68,      // Read register at address 13 (octal)
RA14                =69,      // Read register at address 14 (octal)
RBK                 =70,      // Read BNK
WA0                 =71,      // Write register at address 0 (A)
WA1                 =72,      // Write register at address 1 (Q)
WA2                 =73,      // Write register at address 2 (Z)
WA3                 =74,      // Write register at address 3 (LP)
WA10                =75,      // Write register at address 10 (octal)
WA11                =76,      // Write register at address 11 (octal)
WA12                =77,      // Write register at address 12 (octal)
WA13                =78,      // Write register at address 13 (octal)
WA14                =79,      // Write register at address 14 (octal)
WBK                 =80,      // Write BNK
WGn                 =81,      // Write G (normal gates)**
W20                 =82,      // Write into CYR
W21                 =83,      // Write into SR
W22                 =84,      // Write into CYL
W23                 =85,      // Write into SL


// THESE ARE THE LEFTOVERS -- THEY'RE PROBABLY USED IN SUBSYSTEM C
//
GENRST              =86,      // General Reset**
CLINH               =87,      // Clear INHINT**
CLINH1              =88,      // Clear INHINT1**
CLSTA               =89,      // Clear state counter A (STA)**
CLSTB               =90,      // Clear state counter B (STB)**
CLISQ               =91,      // Clear SNI**
CLRP                =92,      // Clear RPCELL**
INH                 =93,      // Set INHINT**
RPT                 =94,      // Read RUPT opcode **
SBWG                =95,      // Write G from memory
SETSTB              =96,      // Set the ST1 bit of STB
```

```
        WE              =97,    // Write E-MEM from G
        WPCTR           =98,    // Write PCTR (latch priority counter sequence)**
        WSQ             =99,    // Write SQ
        WSTB            =100,   // Write stage counter B (STB)**
        R2000           =101,   // Read 2000 **
};

static cpType glbl_cp[MAXPULSES]; // current set of asserted control pulses (MAXPULSES)

enum scType { // identifies subsequence for a given instruction
        SUB0=0,         // ST2=0, ST1=0
        SUB1=1,         // ST2=0, ST1=1
        SUB2=2,         // ST2=1, ST1=0
        SUB3=3          // ST2=1, ST1=1
};

enum brType {
        BR00    =0,     // BR1=0, BR2=0
        BR01    =1,     // BR1=0, BR2=1
        BR10    =2,     // BR1=1, BR2=0
        BR11    =3,     // BR1=1, BR2=1
        NO_BR   =4      // NO BRANCH
};

struct controlSubStep {
        brType br; // normally no branch (NO_BR)
        cpType pulse[MAX_IPULSES]; // contains 0 - MAXPULSES control pulses
};

struct controlStep {
        controlSubStep substep[4]; // indexed by brType (BR00, BR01, BR10, BR11)
};

struct subsequence {
        controlStep tp[11]; // indexed by tpType (TP1-TP11)
};

struct sequence {
        subsequence* subseq[4]; // indexed by scType
};

#define STEP_INACTIVE \
        NO_BR,  {NO_PULSE, NO_PULSE, NO_PULSE, NO_PULSE, NO_PULSE}, \
        NO_BR,  {NO_PULSE, NO_PULSE, NO_PULSE, NO_PULSE, NO_PULSE}, \
        NO_BR,  {NO_PULSE, NO_PULSE, NO_PULSE, NO_PULSE, NO_PULSE}, \
        NO_BR,  {NO_PULSE, NO_PULSE, NO_PULSE, NO_PULSE, NO_PULSE}

#define STEP(p1, p2, p3, p4, p5) \
        NO_BR,  { p1, p2, p3, p4, p5}, \
        NO_BR,  {NO_PULSE, NO_PULSE, NO_PULSE, NO_PULSE, NO_PULSE}, \
        NO_BR,  {NO_PULSE, NO_PULSE, NO_PULSE, NO_PULSE, NO_PULSE}, \
        NO_BR,  {NO_PULSE, NO_PULSE, NO_PULSE, NO_PULSE, NO_PULSE}

subsequence SUB_TC0 = {
        STEP (  RB,             WY,             WS,             CI,             NO_PULSE        ), // TP 1
        STEP_INACTIVE, // TP 2
        STEP (  WG,             NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 3
        STEP (  RA,             WOVI,           NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 4
        STEP_INACTIVE, // TP 5
        STEP_INACTIVE, // TP 6
        STEP (  RG,             RSC,            WB,             WP,             NO_PULSE        ), // TP 7
        STEP (  RZ,             WQ,             GP,             TP,             NO_PULSE        ), // TP 8
        STEP (  RB,             WSC,            WG,             NO_PULSE,       NO_PULSE        ), // TP 9
        STEP (  RU,             WZ,             NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 10
        STEP (  NISQ,           NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 11
};

subsequence SUB_CCS0 = {
        STEP (  RB,             WS,             NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 1
        STEP (  RZ,             WY,             NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 2
        STEP (  WG,             NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 3
        STEP_INACTIVE, // TP 4
        STEP_INACTIVE, // TP 5
        STEP (  RG,             RSC,            WB,             TSGN,           WP              ), // TP 6
        BR00,   RC,             TMZ,            NO_PULSE,       NO_PULSE,       NO_PULSE,          // TP 7
        BR01,   RC,             TMZ,            NO_PULSE,       NO_PULSE,       NO_PULSE,
        BR10,   RB,             TMZ,            NO_PULSE,       NO_PULSE,       NO_PULSE,
        BR11,   RB,             TMZ,            NO_PULSE,       NO_PULSE,       NO_PULSE,
        BR00,   GP,             TP,             NO_PULSE,       NO_PULSE,       NO_PULSE,         // TP 8
        BR01,   R1,             WX,             GP,             TP,             NO_PULSE,
        BR10,   R2,             WX,             GP,             TP,             NO_PULSE,
        BR11,   R1,             R2,             WX,             GP,             TP,
        STEP (  RB,             WSC,            WG,             NO_PULSE,       NO_PULSE        ), // TP 9
        BR00,   RC,             WA,             NO_PULSE,       NO_PULSE,       NO_PULSE,         // TP 10
        BR01,   WA,             R1C,            NO_PULSE,       NO_PULSE,       NO_PULSE,
        BR10,   RB,             WA,             NO_PULSE,       NO_PULSE,       NO_PULSE,
        BR11,   WA,             R1C,            NO_PULSE,       NO_PULSE,       NO_PULSE,
        STEP (  RU,             ST1,            WZ,             NO_PULSE,       NO_PULSE        ) // TP 11
};

subsequence SUB_CCS1 = {
        STEP (  RZ,             WY,             WS,             CI,             NO_PULSE        ), // TP 1
        STEP_INACTIVE, // TP 2
        STEP (  WG,             NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 3
        STEP (  RU,             WZ,             NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 4
        STEP (  RA,             WY,             CI,             NO_PULSE,       NO_PULSE        ), // TP 5
        STEP_INACTIVE, // TP 6
        STEP (  RG,             RSC,            WB,             WP,             NO_PULSE        ), // TP 7
        STEP (  RU,             WB,             GP,             TP,             NO_PULSE        ), // TP 8
        STEP_INACTIVE, // TP 9
        STEP (  RC,             WA,             WOVI,           NO_PULSE,       NO_PULSE        ), // TP 10
        STEP (  RG,             RSC,            WB,             NISQ,           NO_PULSE        )  // TP 11
};

subsequence SUB_NDX0 = {
        STEP (  RB,             WS,             NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 1
        STEP_INACTIVE, // TP 2
        STEP (  WG,             NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 3
        STEP (  RA,             WOVI,           NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 4
        STEP_INACTIVE, // TP 5
        STEP_INACTIVE, // TP 6
        STEP (  RG,             RSC,            WB,             WP,             NO_PULSE        ), // TP 7
        STEP (  GP,             TP,             NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 8
        STEP (  RB,             WSC,            WG,             NO_PULSE,       NO_PULSE        ), // TP 9
        STEP (  TRSM,           NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 10
        STEP (  ST1,            NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 11
};

subsequence SUB_NDX1 = {
        STEP (  RZ,             WY,             WS,             CI,             NO_PULSE        ), // TP 1
        STEP_INACTIVE, // TP 2
        STEP (  WG,             NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 3
        STEP (  RU,             WZ,             NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 4
        STEP_INACTIVE, // TP 5
        STEP (  RB,             WY,             NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 6
        STEP (  RG,             RSC,            WB,             WP,             NO_PULSE        ), // TP 7
        STEP (  RB,             WX,             GP,             TP,             NO_PULSE        ), // TP 8
        STEP (  RB,             WSC,            WG,             NO_PULSE,       NO_PULSE        ), // TP 9
        STEP_INACTIVE, // TP 10
        STEP (  RU,             WB,             WOVI,           NISQ,           NO_PULSE        ), // TP 11
};

subsequence SUB_RSM3 = {
        STEP (  R24,            WS,             NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 1
        STEP_INACTIVE, // TP 2
        STEP (  WG,             NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 3
        STEP_INACTIVE, // TP 4
        STEP_INACTIVE, // TP 5
        STEP_INACTIVE, // TP 6
        STEP (  RG,             WZ,             NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 7
        STEP_INACTIVE, // TP 8
        STEP_INACTIVE, // TP 9
        STEP_INACTIVE, // TP 10
        STEP (  NISQ,           NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 11
};

subsequence SUB_XCH0 = {
        STEP (  RB,             WS,             NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 1
        STEP (  RA,             WP,             NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 2
        STEP (  WG,             NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 3
        STEP (  WP2,            NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 4
        STEP_INACTIVE, // TP 5
        STEP_INACTIVE, // TP 6
        STEP (  RG,             RSC,            WB,             WP,             NO_PULSE        ), // TP 7
        STEP (  GP,             TP,             NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 8
        STEP (  RA,             WSC,            WG,             RP2,            NO_PULSE        ), // TP 9
        STEP (  RB,             WA,             WOVI,           NO_PULSE,       NO_PULSE        ), // TP 10
        STEP (  ST2,            NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 11
};

subsequence SUB_CS0 = {
        STEP (  RB,             WS,             NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 1
        STEP_INACTIVE, // TP 2
        STEP (  WG,             NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 3
        STEP_INACTIVE, // TP 4
        STEP_INACTIVE, // TP 5
        STEP_INACTIVE, // TP 6
        STEP (  RG,             RSC,            WB,             WP,             NO_PULSE        ), // TP 7
        STEP (  GP,             TP,             NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 8
        STEP (  RB,             WSC,            WG,             NO_PULSE,       NO_PULSE        ), // TP 9
        STEP (  RC,             WA,             WOVI,           NO_PULSE,       NO_PULSE        ), // TP 10
        STEP (  ST2,            NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 11
};

subsequence SUB_TS0 = {
        STEP (  RB,             WS,             NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 1
        STEP (  RA,             WB,             TOV,            WP,                     NO_PULSE
), // TP 2
        STEP (  WG,             NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 3
        BR00,   NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE,         // TP 4
        BR01,   RZ,             WY,             CI,             NO_PULSE,       NO_PULSE,         //
overflow
        BR10,   RZ,             WY,             CI,             NO_PULSE,       NO_PULSE,         //
underflow
        BR11,   NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE,
        BR00,   NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE,         // TP 5
        BR01,   R1,             WA,             NO_PULSE,       NO_PULSE,       NO_PULSE,
        BR10,   WA,             R1C,            NO_PULSE,       NO_PULSE,       NO_PULSE,
        BR11,   NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE,
        STEP_INACTIVE, // TP 6
        BR00,   NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE,         // TP 7
        BR01,   RU,             WZ,             NO_PULSE,       NO_PULSE,       NO_PULSE,
        BR10,   RU,             WZ,             NO_PULSE,       NO_PULSE,       NO_PULSE,
        BR11,   NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE,
        STEP (  GP,             NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 8
        STEP (  RB,             WSC,            WG,                     NO_PULSE,       NO_PULSE
), // TP 9
        STEP (  RA,             WOVI,           NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 10
        STEP (  ST2,            NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 11
};

subsequence SUB_AD0 = {
        STEP (  RB,             WS,             NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 1
        STEP (  RA,             WY,             NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 2
        STEP (  WG,             NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 3
        STEP_INACTIVE, // TP 4
        STEP_INACTIVE, // TP 5
        STEP_INACTIVE, // TP 6
        STEP (  RG,             RSC,            WB,             WP,             NO_PULSE        ), // TP 7
        STEP (  RB,             WX,             GP,             TP,             NO_PULSE        ), // TP 8
        STEP (  RB,             WSC,            WG,             NO_PULSE,       NO_PULSE        ), // TP 9
        STEP_INACTIVE, // TP 10
        STEP (  RU,             WA,             WOVC,           ST2,            WOVI            ), // TP 11
};

// Note: AND is performed using DeMorgan's Theorem: the inputs are inverted, a
// logical OR is performed, and the result is inverted. The implementation of the
// OR (at TP8) is somewhat unorthodox: the inverted inputs are in registers U
// and C. The OR is achieved by gating both registers onto the read/write bus
// simultaneously. (The bus only transfers logical 1's; register-to-register transfers
// are performed by clearing the destination register and then transferring the
// 1's from the source register to the destination). When the 1's from both
// registers are simultaneously gated onto the bus, the word on the bus is a logical
// OR of both registers.
subsequence SUB_MASK0 = {
        STEP (  RB,             WS,             NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 1
        STEP (  RA,             WB,             NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 2
        STEP (  WG,             NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 3
        STEP (  RC,             WY,             NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 4
        STEP_INACTIVE, // TP 5
        STEP_INACTIVE, // TP 6
        STEP (  RG,             RSC,            WB,             WP,             NO_PULSE        ), // TP 7
        STEP (  RU,             RC,             WA,             GP,             TP              ), // TP 8
(CHANGED)
        STEP_INACTIVE, // TP 9
        STEP (  RA,             WB,             NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 10
(CHANGED)
        STEP (  RC,             WA,             ST2,            WOVI,           NO_PULSE        ), // TP 11
};

subsequence SUB_MP0 = {
        STEP (  RB,             WS,             NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 1
        STEP (  RA,             WB,             TSGN,           NO_PULSE,       NO_PULSE        ), // TP 2
        STEP (  RSC,            WG,             NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 3
        BR00,   RB,             WLP,            NO_PULSE,       NO_PULSE,       NO_PULSE,         // TP 4
        BR01,   RB,             WLP,            NO_PULSE,       NO_PULSE,       NO_PULSE,
        BR10,   RC,             WLP,            NO_PULSE,       NO_PULSE,       NO_PULSE,
        BR11,   RC,             WLP,            NO_PULSE,       NO_PULSE,       NO_PULSE,
        STEP (  RLP,            WA,             NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 5
        STEP_INACTIVE, // TP 6
        BR00,   RG,             WY,             WP,             NO_PULSE,       NO_PULSE,         // TP 7
        BR01,   RG,             WY,             WP,             NO_PULSE,       NO_PULSE,
        BR10,   RG,             WB,             WP,             NO_PULSE,       NO_PULSE,
        BR11,   RG,             WB,             WP,             NO_PULSE,       NO_PULSE,
        BR00,   GP,             TP,             NO_PULSE,       NO_PULSE,       NO_PULSE,         // TP 8
        BR01,   GP,             TP,             NO_PULSE,       NO_PULSE,       NO_PULSE,
        BR10,   RC,             WY,             GP,             TP,             NO_PULSE,
        BR11,   RC,             WY,             GP,             TP,             NO_PULSE,
        STEP (  RU,             WB,             TSGN2,          NO_PULSE,       NO_PULSE        ), // TP 9
        BR00,   RA,             WLP,            TSGN,           NO_PULSE,       NO_PULSE,         // TP 10
        BR01,   RA,             RB14,           WLP,            TSGN,           NO_PULSE,
        BR10,   RA,             WLP,            TSGN,           NO_PULSE,       NO_PULSE,
        BR11,   RA,             RB14,           WLP,            TSGN,           NO_PULSE,
        BR00,   ST1,            WALP,           NO_PULSE,       NO_PULSE,       NO_PULSE,         // TP 11
        BR01,   R1,             ST1,            WALP,           R1C,            NO_PULSE,
        BR10,   RU,             ST1,            WALP,           NO_PULSE,       NO_PULSE,
        BR11,   RU,             ST1,            WALP,           NO_PULSE,       NO_PULSE,
};

subsequence SUB_MP1 = {
        STEP (  RA,             WY,             NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 1
        STEP (  RLP,            WA,             TSGN,           NO_PULSE,       NO_PULSE        ), // TP 2
        BR00,   NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE,         // TP 3
        BR01,   NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE,
        BR10,   RB,             WX,             NO_PULSE,       NO_PULSE,       NO_PULSE,
        BR11,   RB,             WX,             NO_PULSE,       NO_PULSE,       NO_PULSE,
        STEP (  RA,             WLP,            NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 4
        STEP (  RLP,            TSGN,           NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 5
        STEP (  RU,             WALP,           NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 6
        STEP (  RA,             WY,             NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 7
        BR00,   NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE,         // TP 8
        BR01,   NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE,
        BR10,   RB,             WX,             NO_PULSE,       NO_PULSE,       NO_PULSE,
        BR11,   RB,             WX,             NO_PULSE,       NO_PULSE,       NO_PULSE,
        STEP (  RLP,            WA,             NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 9
        STEP (  RA,             WLP,            CTR,            NO_PULSE,       NO_PULSE        ), // TP 10
        STEP (  RU,             ST1,            WALP,           NO_PULSE,       NO_PULSE        ), // TP 11
};

subsequence SUB_MP3 = {
        STEP (  RZ,             WY,             WS,             CI,             NO_PULSE        ), // TP 1
        STEP (  RLP,            TSGN,           NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 2
        STEP (  WG,             NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 3
        STEP (  RU,             WZ,             NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 4
        STEP (  RA,             WY,             NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 5
        BR00,   NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE,         // TP 6
        BR01,   NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE,       NO_PULSE,
        BR10,   RB,             WX,             NO_PULSE,       NO_PULSE,       NO_PULSE,
        BR11,   RB,             WX,             NO_PULSE,       NO_PULSE,       NO_PULSE,
        STEP (  RG,             RSC,            WB,             WP,             NO_PULSE        ), // TP 7
        STEP (  RLP,            WA,             GP,             TP,             NO_PULSE        ), // TP 8
        STEP (  RB,             WSC,            WG,             NO_PULSE,       NO_PULSE        ), // TP 9
        STEP (  RA,             WLP,            NO_PULSE,       NO_PULSE,       NO_PULSE        ), // TP 10
        STEP (  RU,             WALP,           NISQ,           NO_PULSE,       NO_PULSE        ), // TP 11
```

```
};

subsequence SUB_DV0 = {
        STEP (  RB,                 WS,                 NO_PULSE,           NO_PULSE,           NO_PULSE            ), // TP 1
        STEP (  RA,                 WB,                 TSGN,               NO_PULSE,           NO_PULSE            ), // TP 2
        STEP (  RSC,                WG,                 NO_PULSE,           NO_PULSE,           NO_PULSE            ), // TP 3
        BR00,   RC,                 WA,                 NO_PULSE,           NO_PULSE,           NO_PULSE,             // TP 4
        BR01,   RC,                 WA,                 NO_PULSE,           NO_PULSE,           NO_PULSE,
        BR10,   NO_PULSE,           NO_PULSE,           NO_PULSE,           NO_PULSE,           NO_PULSE,
        BR11,   NO_PULSE,           NO_PULSE,           NO_PULSE,           NO_PULSE,           NO_PULSE,
        BR00,   R1,                 WLP,                NO_PULSE,           NO_PULSE,           NO_PULSE,             // TP 5
        BR01,   R1,                 WLP,                NO_PULSE,           NO_PULSE,           NO_PULSE,
        BR10,   R2,                 WLP,                NO_PULSE,           NO_PULSE,           NO_PULSE,
        BR11,   R2,                 WLP,                NO_PULSE,           NO_PULSE,           NO_PULSE,
        STEP (  RA,                 WQ,                 NO_PULSE,           NO_PULSE,           NO_PULSE            ), // TP 6
        STEP (  RG,                 WB,                 TSGN,               WP,                 NO_PULSE            ), // TP 7
        STEP (  RB,                 WA,                 GP,                 TP,                 NO_PULSE            ), // TP 8
        BR00,   RLP,                R2,                 WB,                 NO_PULSE,           NO_PULSE,             // TP 9
        BR01,   RLP,                R2,                 WB,                 NO_PULSE,           NO_PULSE,
        BR10,   NO_PULSE,           NO_PULSE,           NO_PULSE,           NO_PULSE,           NO_PULSE,
        BR11,   NO_PULSE,           NO_PULSE,           NO_PULSE,           NO_PULSE,           NO_PULSE,
        BR00,   RB,                 WLP,                NO_PULSE,           NO_PULSE,           NO_PULSE,             // TP 10
        BR01,   RB,                 WLP,                NO_PULSE,           NO_PULSE,           NO_PULSE,
        BR10,   RC,                 WA,                 NO_PULSE,           NO_PULSE,           NO_PULSE,
        BR11,   RC,                 WA,                 NO_PULSE,           NO_PULSE,           NO_PULSE,
        STEP (  R1,                 ST1,                WB,                 NO_PULSE,           NO_PULSE            ), // TP 11
};

subsequence SUB_DV1 = {
        STEP (  R22,                WS,                 NO_PULSE,           NO_PULSE,           NO_PULSE            ), // TP 1
        STEP (  RQ,                 WG,                 NO_PULSE,           NO_PULSE,           NO_PULSE            ), // TP 2
        STEP (  RG,                 WQ,                 WY,                 RSB,                NO_PULSE            ), // TP 3
        STEP (  RA,                 WX,                 NO_PULSE,           NO_PULSE,           NO_PULSE            ), // TP 4
        STEP (  RLP,                TSGN2,              NO_PULSE,           NO_PULSE,           NO_PULSE            ), // TP 5
        STEP_INACTIVE, // TP 6
        STEP (  RU,                 TSGN,               NO_PULSE,           NO_PULSE,           NO_PULSE            ), // TP 7
        BR00,   NO_PULSE,           NO_PULSE,           NO_PULSE,           NO_PULSE,           NO_PULSE,             // TP 8
        BR01,   NO_PULSE,           NO_PULSE,           NO_PULSE,           NO_PULSE,           NO_PULSE,
        BR10,   RU,                 WQ,                 NO_PULSE,           NO_PULSE,           NO_PULSE,
        BR11,   RU,                 WQ,                 NO_PULSE,           NO_PULSE,           NO_PULSE,
        BR00,   RB,                 RSB,                WG,                 NO_PULSE,           NO_PULSE,             // TP 9
        BR01,   RB,                 RSB,                WG,                 NO_PULSE,           NO_PULSE,
        BR10,   RB,                 WG,                 NO_PULSE,           NO_PULSE,           NO_PULSE,
        BR11,   RB,                 WG,                 NO_PULSE,           NO_PULSE,           NO_PULSE,
        STEP (  RG,                 WB,                 TSGN,               NO_PULSE,           NO_PULSE            ), // TP 10
        BR00,   ST1,                NO_PULSE,           NO_PULSE,           NO_PULSE,           NO_PULSE,             // TP 11
        BR01,   ST1,                NO_PULSE,           NO_PULSE,           NO_PULSE,           NO_PULSE,
        BR10,   RC,                 WA,                 ST2,                NO_PULSE,           NO_PULSE,
        BR11,   RB,                 WA,                 ST2,                NO_PULSE,           NO_PULSE,
};

subsequence SUB_SU0 = {
        STEP (  RB,                 WS,                 NO_PULSE,           NO_PULSE,           NO_PULSE            ), // TP 1
        STEP (  RA,                 WY,                 NO_PULSE,           NO_PULSE,           NO_PULSE            ), // TP 2
        STEP (  WG,                 NO_PULSE,           NO_PULSE,           NO_PULSE,           NO_PULSE            ), // TP 3
        STEP_INACTIVE, // TP 4
        STEP_INACTIVE, // TP 5
        STEP_INACTIVE, // TP 6
        STEP (  RG,                 RSC,                WB,                 WP,                 NO_PULSE            ), // TP 7
        STEP (  RC,                 WX,                 GP,                 TP,                 NO_PULSE            ), // TP 8
        STEP (  RB,                 WSC,                WG,                 NO_PULSE,           NO_PULSE            ), // TP 9
        STEP_INACTIVE, // TP 10
        STEP (  RU,                 WA,                 WOVC,               ST2,                WOVI                ), // TP 11
};

subsequence SUB_RUPT1 = {
        STEP (  R24,                WY,                 WS,                 CI,                 NO_PULSE            ), // TP 1
        STEP_INACTIVE, // TP 2
        STEP (  WG,                 NO_PULSE,           NO_PULSE,           NO_PULSE,           NO_PULSE            ), // TP 3
        STEP_INACTIVE, // TP 4
        STEP_INACTIVE, // TP 5
        STEP_INACTIVE, // TP 6
        STEP_INACTIVE, // TP 7
        STEP_INACTIVE, // TP 8
        STEP (  RZ,                 WG,                 NO_PULSE,           NO_PULSE,           NO_PULSE            ), // TP 9
        STEP (  RU,                 WZ,                 NO_PULSE,           NO_PULSE,           NO_PULSE            ), // TP 10
        STEP (  ST1,        ST2,        NO_PULSE,   NO_PULSE,   NO_PULSE    ), // TP 11
};

subsequence SUB_RUPT3 = {
        STEP (  RZ,         WS,         NO_PULSE,   NO_PULSE,   NO_PULSE    ), // TP 1
        STEP (  RRPA,       WZ,         NO_PULSE,   NO_PULSE,   NO_PULSE    ), // TP 2
        STEP (  RZ,         KRPT,       WG,         NO_PULSE,   NO_PULSE    ), // TP 3
        STEP_INACTIVE, // TP 4
        STEP_INACTIVE, // TP 5
        STEP_INACTIVE, // TP 6
        STEP_INACTIVE, // TP 7
        STEP_INACTIVE, // TP 8
        STEP (  RB,         WSC,        WG,         NO_PULSE,   NO_PULSE    ), // TP 9
        STEP_INACTIVE, // TP 10
        STEP (  ST2,        NO_PULSE,   NO_PULSE,   NO_PULSE,   NO_PULSE    ), // TP 11
};

subsequence SUB_STD2 = {
        STEP (  RZ,         WY,         WS,         CI,         NO_PULSE    ), // TP 1
        STEP_INACTIVE, // TP 2
        STEP (  WG,         NO_PULSE,   NO_PULSE,   NO_PULSE,   NO_PULSE    ), // TP 3
        STEP (  RU,         WZ,         NO_PULSE,   NO_PULSE,   NO_PULSE    ), // TP
        STEP_INACTIVE, // TP 5
        STEP_INACTIVE, // TP 6
        STEP (  RG,         RSC,        WB,         WP,         NO_PULSE    ), // TP 7
        STEP (  GP,         TP,         NO_PULSE,   NO_PULSE,   NO_PULSE    ), // TP 8
        STEP (  RB,         WSC,        WG,         NO_PULSE,   NO_PULSE    ), // TP 9
        STEP_INACTIVE, // TP 10
        STEP (  NISQ,       NO_PULSE,   NO_PULSE,   NO_PULSE,   NO_PULSE    ), // TP 11
};

subsequence SUB_PINC = {
        STEP (  WS,         RSCT,       NO_PULSE,   NO_PULSE,   NO_PULSE    ), // TP 1
        STEP_INACTIVE, // TP 2
        STEP (  WG,         NO_PULSE,   NO_PULSE,   NO_PULSE,   NO_PULSE    ), // TP 3
        STEP (  R1,         WY,         NO_PULSE,   NO_PULSE,   NO_PULSE    ), // TP 4
        STEP_INACTIVE, // TP 5
        STEP (  RG,         WX,         WP,         NO_PULSE,   NO_PULSE    ), // TP 6
        STEP (  TP,         NO_PULSE,   NO_PULSE,   NO_PULSE,   NO_PULSE    ), // TP 7
        STEP (  WP,         NO_PULSE,   NO_PULSE,   NO_PULSE,   NO_PULSE    ), // TP 8
        STEP (  RU,         CLG,        WPx,        NO_PULSE,   NO_PULSE    ), // TP 9
        STEP (  RU,         WGx,        WOVR,       NO_PULSE,   NO_PULSE    ), // TP 10
        STEP_INACTIVE, // TP 11
};

subsequence SUB_MINC = {
        STEP (  WS,         RSCT,       NO_PULSE,   NO_PULSE,   NO_PULSE    ), // TP 1
        STEP_INACTIVE, // TP 2
        STEP (  WG,         NO_PULSE,   NO_PULSE,   NO_PULSE,   NO_PULSE    ), // TP 3
        STEP (  WY,         R1C,        NO_PULSE,   NO_PULSE,   NO_PULSE    ), // TP 4
        STEP_INACTIVE, // TP 5
        STEP (  RG,         WX,         WP,         NO_PULSE,   NO_PULSE    ), // TP 6
        STEP (  TP,         NO_PULSE,   NO_PULSE,   NO_PULSE,   NO_PULSE    ), // TP 7
        STEP (  WP,         NO_PULSE,   NO_PULSE,   NO_PULSE,   NO_PULSE    ), // TP 8
        STEP (  RU,         CLG,        WPx,        NO_PULSE,   NO_PULSE    ), // TP 9
        STEP (  RU,         WGx,        WOVR,       NO_PULSE,   NO_PULSE    ), // TP 10
        STEP_INACTIVE, // TP 11
};

subsequence SUB_SHINC = {
        STEP (  WS,         RSCT,       NO_PULSE,   NO_PULSE,   NO_PULSE    ), // TP 1
        STEP_INACTIVE, // TP 2
        STEP (  WG,         NO_PULSE,   NO_PULSE,   NO_PULSE,   NO_PULSE    ), // TP 3
        STEP (  WY,         NO_PULSE,   NO_PULSE,   NO_PULSE,   NO_PULSE    ), // TP 4
        STEP_INACTIVE, // TP 5
        STEP (  RG,         WYx,        WX,         WP,         NO_PULSE    ), // TP 6
        STEP (  TP,         NO_PULSE,   NO_PULSE,   NO_PULSE,   NO_PULSE    ), // TP 7
        STEP (  WP,         NO_PULSE,   NO_PULSE,   NO_PULSE,   NO_PULSE    ), // TP 8
        STEP (  RU,         CLG,        WPx,        NO_PULSE,   NO_PULSE    ), // TP 9
        STEP (  RU,         WGx,        WOVR,       NO_PULSE,   NO_PULSE    ), // TP 10
        STEP_INACTIVE, // TP 11
};

char* subseqString[] =
{
        "TC0",
        "CCS0",
        "CCS1",
        "NDX0",
        "NDX1",
        "RSM3",
        "XCH0",
        "CS0",
        "TS0",
        "AD0",
        "MASK0",
        "MP0",
        "MP1",
        "MP3",
        "DV0",
        "DV1",
        "SU0",
        "RUPT1",
        "RUPT3",
        "STD2",
        "PINC0",
        "MINC0",
        "SHINC0",
        "NO_SEQ"
};

enum subseq {
        TC0     =0,
        CCS0    =1,
        CCS1    =2,
        NDX0    =3,
        NDX1    =4,
        RSM3    =5,
        XCH0    =6,
        CS0     =7,
        TS0     =8,
        AD0     =9,
        MASK0   =10,
        MP0     =11,
        MP1     =12,
        MP3     =13,
        DV0     =14,
        DV1     =15,
        SU0     =16,
        RUPT1   =17,
        RUPT3   =18,
        STD2    =19,
        PINC0   =20,
        MINC0   =21,
        SHINC0  =22,
        NO_SEQ  =23
};


enum tpType {
        STBY            =0,
        PWRON           =1,

        TP1             =2,     // TIME PULSE 1: start of memory cycle time (MCT)
        TP2             =3,
        TP3             =4,
        TP4             =5,
        TP5             =6,
        TP6             =7,     // EMEM is available in G register by TP6
        TP7             =8,     // FMEM is available in G register by TP7
        TP8             =9,
        TP9             =10,
        TP10            =11,    // G register written to memory beginning at TP10
        TP11            =12,    // TIME PULSE 11: end of memory cycle time (MCT)
        TP12            =13,    // select new subsequence/select new instruction

        SRLSE           =14,    // step switch release
        WAIT            =15
};

subseq instructionSubsequenceDecoder(
        int SB2_field, int SB1_field, int SQ_field, int STB_field)
```

```
{
        // Combinational logic decodes instruction and the stage count
        // to get the instruction subsequence.
    static subseq decode[16][4] = {
        {   TC0,        RUPT1,      STD2,       RUPT3     }, // 00
        {   CCS0,       CCS1,       NO_SEQ,     NO_SEQ    }, // 01
        {   NDX0,       NDX1,       NO_SEQ,     RSM3      }, // 02
        {   XCH0,       NO_SEQ,     STD2,       NO_SEQ    }, // 03

        {   NO_SEQ,     NO_SEQ,     NO_SEQ,     NO_SEQ    }, // 04
        {   NO_SEQ,     NO_SEQ,     NO_SEQ,     NO_SEQ    }, // 05
        {   NO_SEQ,     NO_SEQ,     NO_SEQ,     NO_SEQ    }, // 06
        {   NO_SEQ,     NO_SEQ,     NO_SEQ,     NO_SEQ    }, // 07
        {   NO_SEQ,     NO_SEQ,     NO_SEQ,     NO_SEQ    }, // 10

        {   MP0,        MP1,        NO_SEQ,     MP3       }, // 11
        {   DV0,        DV1,        STD2,       NO_SEQ    }, // 12
        {   SU0,        NO_SEQ,     STD2,       NO_SEQ    }, // 13

        {   CS0,        NO_SEQ,     STD2,       NO_SEQ    }, // 14
        {   TS0,        NO_SEQ,     STD2,       NO_SEQ    }, // 15
        {   AD0,        NO_SEQ,     STD2,       NO_SEQ    }, // 16
        {   MASK0,      NO_SEQ,     STD2,       NO_SEQ    }  // 17

    };

    if(SB2_field == 0 && SB1_field == 1)
        return PINC0;
    else if(SB2_field == 1 && SB1_field == 0)
        return MINC0;
    else
        return decode[SQ_field][STB_field];
}

void clearControlPulses()
{
    for(unsigned i=0; i<MAXPULSES; i++)
        glbl_cp[i] = NO_PULSE;
}

void assert(cpType* pulse)
{
    int j=0;
    for(unsigned i=0; i<MAXPULSES && j<MAX_IPULSES && pulse[j] != NO_PULSE; i++)
    {
        if(glbl_cp[i] == NO_PULSE)
        {
            glbl_cp[i] = pulse[j];
            j++;
        }
    }
}

void assert(cpType pulse)
{
    for(unsigned i=0; i<MAXPULSES; i++)
    {
        if(glbl_cp[i] == NO_PULSE)
        {
            glbl_cp[i] = pulse;
            break;
        }
    }
}

void get_CPM_A(int CPM_A_address)
{
        // EPROM address bits (bit 1 is LSB)
        //  1:          register BR2
        //  2:          register BR1
        //  3-6:    register SG (4)
        //  7,8:    register STB (2)
        //  9-12:   register SQ (4)
        //  13:         STB_01
        //      14:         STB_02
    //**********************************************************
    // EPROM emulator
    int SB2_field = (CPM_A_address >> 13) & 0x1;
    int SB1_field = (CPM_A_address >> 12) & 0x1;
    int SQ_field  = (CPM_A_address >> 8)  & 0xf;
    int STB_field = (CPM_A_address >> 6)  & 0x3;
    int SG_field  = (CPM_A_address >> 2)  & 0xf;
    int BR1_field = (CPM_A_address >> 1)  & 0x1;
    int BR2_field = (CPM_A_address     )  & 0x1;


        // Decode the current instruction subsequence (glbl_subseq).
    subseq glbl_subseq = instructionSubsequenceDecoder(SB2_field, SB1_field, SQ_field, STB_field);

    static subsequence* subsp[] =
    {
    &SUB_TC0,       &SUB_CCS0,      &SUB_CCS1,      &SUB_NDX0,      &SUB_NDX1,      &SUB_RSM3,
    &SUB_XCH0,      &SUB_CS0,       &SUB_TS0,       &SUB_AD0,       &SUB_MASK0,     &SUB_MP0,
    &SUB_MP1,       &SUB_MP3,       &SUB_DV0,       &SUB_DV1,       &SUB_SU0,       &SUB_RUPT1,
    &SUB_RUPT3,     &SUB_STD2,      &SUB_PINC,      &SUB_MINC,      &SUB_SHINC,0
    };

        // Clear old control pulses.
    clearControlPulses();

        // Get new control pulses for the current instruction subsequence.
    if(glbl_subseq != NO_SEQ && // THIS TESTS OUT OK
        SG_field >= TP1 &&
        SG_field <= TP11)
    {
        subsequence* subseqP = subsp[glbl_subseq];
        if(subseqP)
        {
                // index t-2 because TP1=2, but array is indexed from zero
            controlStep& csref = subseqP->tp[SG_field-2];

            brType b = (brType) ((BR1_field << 1) | BR2_field);
            controlSubStep& cssref = csref.substep[b];
            if(cssref.br == NO_BR)
                cssref = csref.substep[0];

            cpType* p = cssref.pulse;
            assert(p);
        }
    }

        // Implement these here, because the instruction sequence decoder
        // function is buried in the CPM-A ROM and so, identification of
        // the sequences is not available outside CPM-A. CPM-C needs info
        // on these 3 sequences.
    switch(glbl_subseq)
    {
    case DV1:       assert(SDV1); break;
    case MP1:       assert(SMP1); break;
    case RSM3:      assert(SRSM3); break;
    }

    //**********************************************************


}

char* cpTypeString[] =
{
    "NO_PULSE",

    // OUTPUTS FROM SUBSYSTEM A
    "CI", "CLG", "CLCTR", "CTR", "GP", "KRPT", "NISQ", "RA", "RB",
    "RB14", "RC", "RG", "RLP", "RP2", "RQ", "RRPA", "RSB", "RSCT",
    "RU", "RZ", "R1", "R1C", "R2", "R22", "R24", "ST1", "ST2", "TMZ",
    "TOV", "TP", "TRSM", "TSGN", "TSGN2", "WA", "WALP", "WB", "WGx",
    "WLP", "WOVC", "WOVI", "WOVR", "WP", "WPx", "WP2", "WQ", "WS",
    "WX", "WY", "WYx", "WZ",

    // OUTPUTS FROM SUBSYSTEM A; USED AS INPUTS TO SUBSYSTEM B ONLY;
    // NOT USED OUTSIDE CPM
    //
    "RSC", "WSC", "WG",

    // OUTPUTS FROM SUBSYSTEM A; USED AS INPUTS TO SUBSYSTEM C ONLY;
    // NOT USED OUTSIDE CPM
    //
    "SDV1", "SMP1", "SRSM3",

    // EXTERNAL OUTPUTS FROM SUBSYSTEM B
    //
    "RA0", "RA1", "RA2", "RA3", "RA4", "RA5", "RA6", "RA7", "RA10", "RA11",
    "RA12", "RA13", "RA14", "RBK", "WA0", "WA1", "WA2", "WA3", "WA10",
    "WA11", "WA12", "WA13", "WA14", "WBK", "WGn", "W20", "W21", "W22", "W23",

    // THESE ARE THE LEFTOVERS -- THEY'RE PROBABLY USED IN SUBSYSTEM C
    //
    "GENRST", "CLINH", "CLINH1", "CLSTA", "CLSTB", "CLISQ", "CLRP", "INH",
    "RPT", "SBWG", "SETSTB", "WE", "WPCTR", "WSQ", "WSTB", "R2000"
};

    // for debug purposes only
char* printControlPulses()
{
    static char buf[MAXPULSES*6];
    strcpy(buf,"");

    for(unsigned i=0; i<MAXPULSES && glbl_cp[i] != NO_PULSE; i++)
    {
        strcat(buf, cpTypeString[glbl_cp[i]]);
        strcat(buf," ");
    }
    //if(strcmp(buf,"") == 0) strcat(buf,"NONE");
    return buf;
}

    // return the EPROM word corresponding to the pulses
    // in glbl_cp.
unsigned writeEPROM(int lowBit)
{
    unsigned EPROMword = 0x00; // no pulses; default
    for(unsigned i=0; i<MAXPULSES && glbl_cp[i] != NO_PULSE; i++)
    {
        int pulse = glbl_cp[i] - lowBit;
        if(pulse < 0 || pulse > 7)
            continue; // pulse is not in this EPROM
        EPROMword |= 0x01 << pulse;
    }
//  printf("%02X\n",EPROMword);
    //return EPROMword;

    // The CPM-A control signals are negative logic, so we need to
    // bit-flip the word. No signal is a 1, and an asserted signal is
    // a 0:
    return ((~EPROMword) & 0xff);
}


const unsigned agcMemSize = 0x3fff+1; // # of cells in a 16-bit address range

void writeEPROM(FILE* fpObj, int lowBit)
{
        // Write an EPROM file using Motorola's S-Record format (s2f).

        // Some parameters that control file format. You can change maxBytes
        // without affecting anything else. 'addressBytes' is determined by
        // the choosen S-Record format.
    const int maxBytes = 20;  // set limit on record length
    const int addressBytes = 3; // 24-bit address range
    const int sumCheckBytes = 1;

    const int maxdata = maxBytes - addressBytes - sumCheckBytes;

    int i=0; // current EPROM address
    int sumCheck = 0;
    while (i < agcMemSize)
    {
```

```
                        // get dataByteCount; the number of bytes of EPROM data per record.
                int dataByteCount = maxdata;
                if(i + dataByteCount >= agcMemSize)
                {
                        dataByteCount = agcMemSize - i;
                }
                        // write record header (*** 3 byte address assumed ***)
                int totalByteCount = dataByteCount + addressBytes + sumCheckBytes;
                fprintf(fpObj, "S2%02X%06X", totalByteCount, i);
                sumCheck = totalByteCount & 0xff;
                sumCheck = (sumCheck + ((i & 0xff0000) >> 16)) % 256;
                sumCheck = (sumCheck + ((i & 0x00ff00) >>  8)) % 256;
                sumCheck = (sumCheck + ((i & 0x0000ff)      )) % 256;

                        // write data bytes into record
                for(int j=0; j<dataByteCount; j++)
                {
                        get_CPM_A(i+j); // get CPM-A pulses for address i+j
                        int data = writeEPROM(lowBit); // comvert pulses to EPROM format
                        fprintf(fpObj, "%02X", data);
                        sumCheck = (sumCheck + data) % 256;

                }
                        // terminate record by adding the checksum and a newline.
                fprintf(fpObj, "%02X\n", (~sumCheck) & 0xff);

                i += dataByteCount;
        }
                // write an end-of-file record here
        i = 0; // use address zero for last record
        sumCheck  = 0x04; // byte count
        sumCheck = (sumCheck + ((i & 0xff0000) >> 16)) % 256;
        sumCheck = (sumCheck + ((i & 0x00ff00) >>  8)) % 256;
        sumCheck = (sumCheck + ((i & 0x0000ff)      )) % 256;
        fprintf(fpObj, "S804%06X%02X", i, (~sumCheck) & 0xff);

}



void main(int argc, char* argv[])
{
        FILE* fpObj = 0;


        fpObj = fopen("CPM1_8.hex", "w");
        if(!fpObj)
        {
                perror("fopen failed for object file");
                exit(-1);
        }
        writeEPROM(fpObj, 1);  // pulses  1-8
        fclose(fpObj);


        fpObj = fopen("CPM9_16.hex", "w");
        if(!fpObj)
        {
                perror("fopen failed for object file");
                exit(-1);
        }
        writeEPROM(fpObj, 9);  // pulses  9-16
        fclose(fpObj);


        fpObj = fopen("CPM17_24.hex", "w");
        if(!fpObj)
        {
                perror("fopen failed for object file");
                exit(-1);
        }
        writeEPROM(fpObj, 17);  // pulses  17-24
        fclose(fpObj);


        fpObj = fopen("CPM25_32.hex", "w");
```

```
    if(!fpObj)
    {
        perror("fopen failed for object file");
        exit(-1);
    }
    writeEPROM(fpObj, 25);  // pulses  25-32
    fclose(fpObj);


    fpObj = fopen("CPM33_40.hex", "w");
    if(!fpObj)
    {
        perror("fopen failed for object file");
        exit(-1);
    }
    writeEPROM(fpObj, 33);  // pulses  33-40
    fclose(fpObj);


    fpObj = fopen("CPM41_48.hex", "w");
    if(!fpObj)
    {
        perror("fopen failed for object file");
        exit(-1);
    }
    writeEPROM(fpObj, 41);  // pulses  41-48
    fclose(fpObj);


    fpObj = fopen("CPM49_56.hex", "w");
    if(!fpObj)
    {
        perror("fopen failed for object file");
        exit(-1);
    }
    writeEPROM(fpObj, 49);  // pulses  49-56
    fclose(fpObj);

}
```