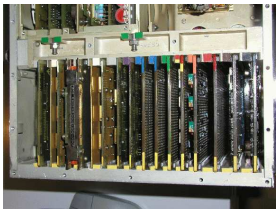
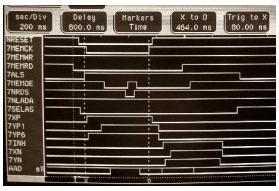
Programmer Electronic Control
- Command Set -

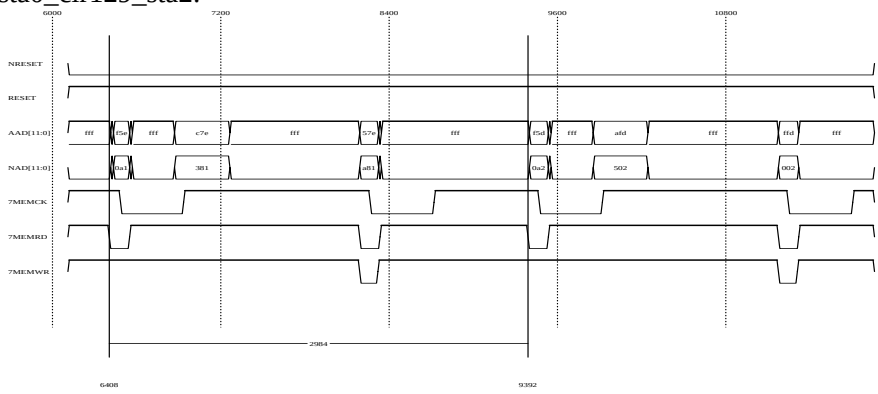
Reverse-Engineering and restoration Project

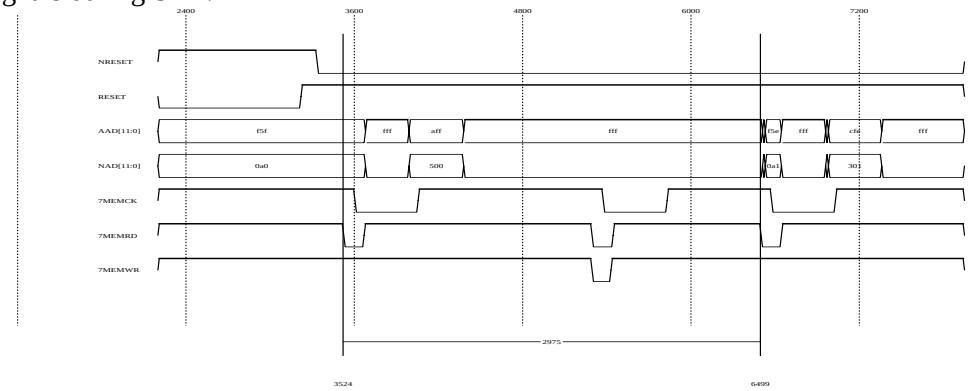
2005-2007 by Erik Baigar,

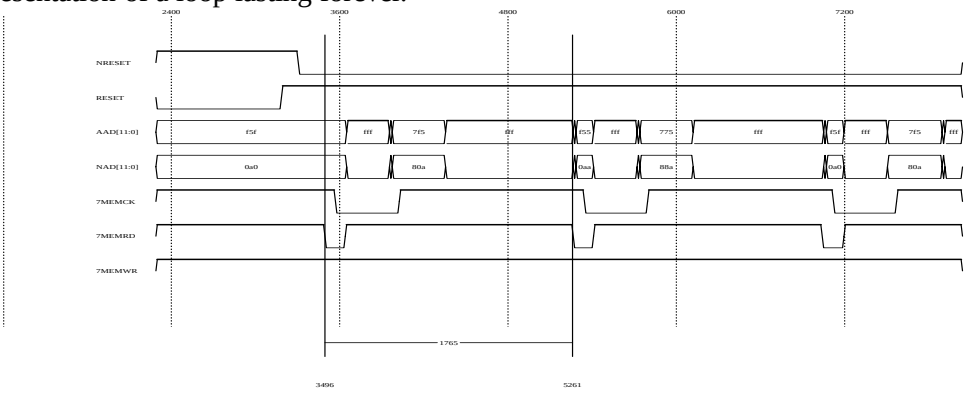
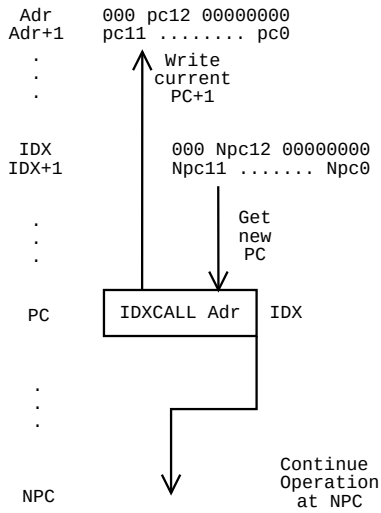
Revision 2007/03/18





x_{11}	x_{10}	x_9	x_8	$x_7 ... x_0$ **and Description**
0	0	0	0	**LDI-Group:** Data from memory position $[x_6 ... x_0]$ is transferred to index register IDX. Next Operation of STA, STS, INC, ADD and SUB occurs relative to this index register. Afterwards IDX is cleared by these instructiond. The index register is always loaded from the zero-page. If $x_7 = 1$ the unit freezes during memory access to the zero page of the upper memory bank ($x_{12} = 1$). Needs power cycle to recover. 0x000, 0, Function 0, $2.400 \mu s$
0	0	0	1	**ADD-Group:** Data from memory position $[IDX + x_6 ... x_0]$ is ADDed to accumulator and result is stored within accumulator. IDX is cleared durind adding and Programmer Electronic Control freezes if $x_7 = 1$ in an attempt to access the upper bank ($x_{12} = 1$). There seems to be no carry flag mechanism during addition. 0x100, 256, Function 1, $2.394 \mu s$
0	0	1	0	**SUB-Group:** Accumulator is subtracted from data in memory location $[IDX + x_6 ... x_0]$. Result is kept in the accumulator register, IDX set to zero after the instruction and the CPU freezes if $x_7 = 1$ on a memory acces to the upper bank ($x_{12} = 1$) of core memory. No carry mechanism discovered so far. 0x200, 512, Function 2, $3.554 \mu s$
0	0	1	1	**STS-Group:** Stores the extended shifter register to core memory $[IDX + x_6 ... x_0]$. IDX is used for relative memory access and cleared after this access. Accumulator remains unchanged. The register is implemented on the data boards SK8 and SK9 and is used in SHR and SHL commands - see below. Bit7 switches to write access to the upper half of the memory if $x_7 = 1$ and the unit freezes in doing so. Waveform-Example sta0_clr129_sta2:  0x300, 768, Function 3, $2.992 \mu s$

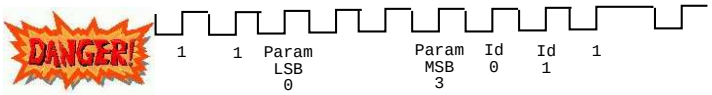
x_{11}	x_{10}	x_9	x_8	$x_7 ... x_0$ **and Description**
0	1	0	0	**LDA-Group:** The accumulator register is loaded from memory location $[IDX + x_6...x_0]$ and the extended shift register may be modified by this instruction. IDX is cleared by the LDA operation and PEC ignores a $x_7 = 1$ and accesses the equivalent $x_7 = 0$-location.  0x400, 1024, Function 4, $2.382\mu s$
0	1	0	1	**STA-Group:** The accumulator register is stored in memory location $[IDX + x_6...x_0]$ and the accumulator is not affected by the instruction. IDX is cleared by the STA operation and PEC freezes if $x_7 = 1$ in trying to access the upper bank ($x_{12} = 1$). Signals during STA:  0x500, 1280, Function 5, $2.965\mu s$
0	1	1	0	**AND-Group:** The accumulator register logically ANDes with memory location $[IDX + x_6...x_0]$. IDX is cleared by the AND operation and PEC maps $x_7 = 1$ to $x_7 = 0$ and accessing a lower word than intended.  0x600, 1536, Function 6, $2.397\mu s$
0	1	1	1	**RJAZ-Group:** Continues with next instruction immediately if accumulator is not zero. Otherwise the RJAZ instruction (**R**elative-**J**ump-if-**A**ccu-**Z**ero) performs a relative jump like RJMP. 0x700, 2047, Function 7, $1.776 - 2.921\mu s$

x_{11}	x_{10}	x_9	x_8	$x_7...x_0$ **and Description**
1	0	0	0	**RJMP-Group:** x_7 determines wether the jump is $x_6...x_0$ instructions forward (x_7=0) or backward (x_7=1). Especially 100000000000 and 100010000000 are the same representation of a loop lasting forever.  **Special care** has to be taken if IDX is not zero. In this case the jump width is calculated in the following way: $(-1) * x_7 * [IDX + x_6...x_0]$ and IDX is cleared by the jump. 0x800, 2048, Function 8, $1.768\mu s$
1	0	0	1	**RJAN-Group:** Continues with next instruction immediately if accumulator is positive, i.e. $a_{11} = 0$. Otherwise the RJAN instruction (**R**elative-**J**ump-if-**A**ccu-**N**egative) performs a relative jump like RJMP. 0x900, 2304, Function 9, $1.773\mu s$
1	0	1	0	**INC-Group:** The accumulator register is loaded from memory location $[IDX + x_6...x_0]$. Then the accumulator incremented and the result is stored back to $[IDX + x_6...x_0]$. IDX is cleared after the INC's write operation and PEC freezes if $x_7 = 1$ in trying to access the upper bank ($x_{12} = 1$) already on the read cycle. Especially remember, that INC modifies the accumulator! 0xa00, 2560, Function 10, $3.606\mu s$
1	0	1	1	**IDXCALL-Group:** First the memory location of the next instruction to execute (i.e. $PC + 1$) is saved to the memory adress specified in the lower 7 bits of the command: $PC + 1 \rightarrow [0...0x_7...x_0]$ (CAUTION: If adr>127 PEC will freeze). Afterwards the new program counter PC is loaded from the memory location where IDX points to, i.e. the execution is continued with an indirect jump ($[IDX] \rightarrow PC$). Adresses are stored in two successive words with MSW ($000a_{12}00000000$) first and than LSW ($a_{11}...a_0$). The accumulator register is not affected by this operation and IDX is cleared. This operation can jump the the upper half of the memory without freezing the unit - at least some times. But unfortunately the unit obviously can not read properly from upper half during program execution...  0xb00, 2816, Function 11, $6.607\mu s$

x_{11}	x_{10}	x_9	x_8	$x_7 ... x_0$ **and Description**
1	1	0	0	**UMUL-Group:** This command multiplies the accumulator with the value read from $[IDX + x_6 ... x_0]$. Box freezes for $x_7 = 1$. This multiplication is executed in microcode and incorporates 12 shift and add operations. Therefore the multiply lasts for more than $8.8\mu s$. IDX operates as usual. The result is stored in the special shift register (LSW, i.e. 0 $r_{10} ... r_0$) and the accumulator (MSW, 0 $r_{21} ... r_{11}$). Caution: Only positive numbers (i.e. with bit 11 zero) are multiplied correctly. Thus UMUL essentially is a 11bit * 11bit multiplication. IDX is cleared by the operation and the special shift register is cleared before the multiply process. 0xc00, 3072, Function 12, $9.360\mu s$
1	1	0	1	**UDIV-Group:** Reads value from $[IDX + x_6 ... x_0]$. Influences special shift register and accu. Purpose maybe divide.  0xd00, 3382, Function 13, $9.952\mu s$

<table>
<tr><th>x_{11}</th><th>x_{10}</th><th>x_9</th><th>x_8</th><th>$x_7...x_0$ and Description</th></tr>
<tr><td rowspan="6">1</td><td rowspan="6">1</td><td rowspan="6">1</td><td rowspan="6">0</td><td>Special-Group: Depending on $x_7...x_0$ special functionality is available:
<table>
<tr><th>x_7</th><th>x_6</th><th>x_5</th><th>$x_4...x_0$ and Description</th></tr>
<tr><td>0</td><td>0</td><td>0</td><td>SHL: Shifts accu left by $x_4...x_0$ bits. Instruction takes $(4 + x_4...x_0)$-MEMEN/AADEN2-Cycles to complete. In each shift cycle first accumulator is shifted left where the MSB is dropped. Afterwards on the right side of accumulator s_{10} of the extended shift register is inserted for a_0. Afterwards the extended shift register is shifted left as well where the LSB is cleared: $s_0 = 0$. The extended shift register an be accessed via the STS instruction.
0xe00, 3584, Function 14-0-0, $2.928\mu s + 0.576\mu s * N$</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>$x_4\ x_3\ x_2\ x_1\ 0$ - MSRTA: regardless of $x_4\ x_3\ x_2\ x_1$ Moves the extended Shift Register to the Accumulator. All 16 possible opcodes encode the MSRTA instruction. Additionally, if IDX has been set prior to MSRTA then the following actions occur:
(1) ShiftRegister:=$(IDX >> 1)$
(2) Akku:=$(IDX >> 1)$
0xe20, 3616, Function 14-0-32, $2.336\mu s$</td></tr>
<tr><td>0</td><td>0</td><td>1</td><td>$x_4\ x_3\ x_2\ x_1\ 1$ - RSIDXTA: R ight Shift IDX to Accumulator (16 possible bit patterns): With IDX set prior to RSIDXTA the following operations are executed:
(1) ShiftRegister:=$(IDX >> 1)$ and
(2) if IDX odd then Akku:=$(IDX >> 1)$ otherwise Akku unchanged.
0xe21, 3617, Function 14-0-33, $2.368\mu s$</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>$x_4\ x_3\ x_2\ x_1\ 0$ - Regardless of $x_4\ x_3\ x_2\ x_1$ accu (only lower byte) is read back from hidden register (see 111011000000). Extended Shift Register seems to be unimportant and is not changed by the Operation. Suspect purpose of this command group (occupies 16 bit patterns). 
0xe40, 3648, Function 14-0-64, $2.349\mu s$</td></tr>
<tr><td>0</td><td>1</td><td>0</td><td>$x_4\ x_3\ x_2\ x_1\ 1$ - MTA: This instruction makes a second core read cycle at the next program counter address and reads this to the accumulator register. Regardless of $x_4\ x_3\ x_2\ x_1$ this is done, i.e. 16 possible bit patterns exist for this instruction and this is the only two-cycle-instructionof the unit! (Move to Accumulator.)
0xe41, 3649, Function 14-0-65, $2.964\mu s$</td></tr>
<tr><td>0</td><td>1</td><td>1</td><td>SHR: Shifts accu right by -$x_4...x_0$ bits (i.e. $x_4...x_0 = 11111$ shifts right one bit). Instruction takes $(4+!(x_4...x_0) + 1)$-MEMEN/AADEN2-Cycles to complete. First the extended shift register which can be accessed by the STS instruction is shifted right one position where it's s_0 bit is lost. Afterwards the accu is shifted right and therein a_{11} is replicated to a_{10}. The bit a_0 which is shifted out of accumulator is inserted as s_{10} into the exteded shift register. Thus there exist 11 hidden bits in the shifter unit's extended shift register $s_{10}...s_0$.
0xe60, 3680, Function 14-0-96, $2.928\mu s + 0.576\mu s * N$</td></tr>
</table></td></tr>
</table>

x_{11}	x_{10}	x_9	x_8	$x_7 \ldots x_0$ **and Description**
1	1	1	0	**Special-Group:** Depending on $x_7 \ldots x_0$ special functionality is available: 1 0 0: Does an IOR instruction to address $10x_4 \ldots x_0$ (EMUX=EMUXA=EMUXB=1) where the accumulator is modified.  0xe80, 3712, Function 14-1-0, $4.078\mu s$ 1 0 1: **MATSR**: Completely independent of $x_4 \ldots x_0$ this instruction **M**oves the **A**ccumulator **t**o extended **S**hift **R**egister. All 32 possible combinations encode the MATSR instruction! 0xea0, 3744, Function 14-1-32, $2.344\mu s$ 1 1 0: $x_4 \ldots x_1 0$ is a **single** word instruction with unknown purpose. The accumulator is stored somewhere (at least the lower byte, see 111001000000). 0xec0, 3776, Function 14-1-64, $2.930\mu s$ 1 1 0: $x_4 \ldots x_1 1$ is a **dual** word instruction with unknown purpose. Neither operation asserts IOR or IOW or modifies the accumulator or the extended shift register.  0xec1, 3777, Function 14-1-65, $3.539\mu s$ 1 1 1: 0 0 0 0 0 Does a IOW instruction to address $111x_4 \ldots x_0$ and writes accu with EMUX=EMUXA=EMUXB=1 to this address. Does not alter the accumulator or the extended shift register.  0xee0, 3808, Function 14-1-96, $5.117\mu s$

x_{11}	x_{10}	x_9	x_8	$x_7 \ldots x_0$ **and Description**
1	1	1	1	**IO-Group:** Data is sent to the outputs (IOW) if $x_7 = 1$, otherwise data is read (IOR). $x_4 \ldots x_0$ is some kind of address which is applied to the bus in an intermediate state (DPL-number in x_5 and x_6 is visible here, too): 0 ? ?: IOR from DPL01 to DPL04 - Map identical to write (regarding x,y and z), but 0xf00 is waiting forever (can be canceled by reset!).  0xf00, 3840, Function 15-0, $2.339 - \infty\mu s$

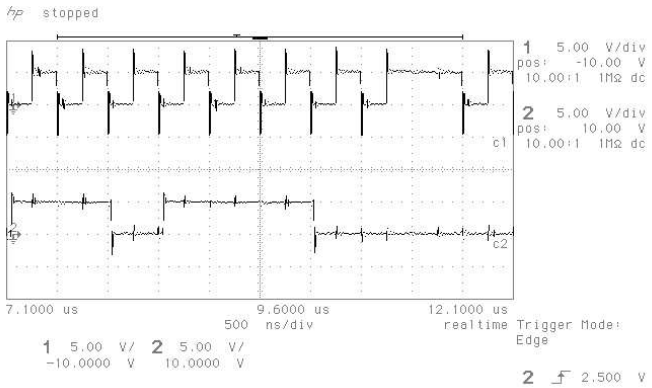
x_{11}	x_{10}	x_9	x_8	$x_7...x_0$ **and Description**
1	1	1	1	**IO-Group:** Data is sent to the outputs (IOW) if $x_7 = 1$, otherwise data is read (IOR). $x_4...x_0$ is some kind of address which is applied to the bus in an intermediate state (DPL-number in x_5 and x_6 is visible here, too). This information is transferred to the sender-register as well and no output is generated if $x_0 = x_1 = 0$:

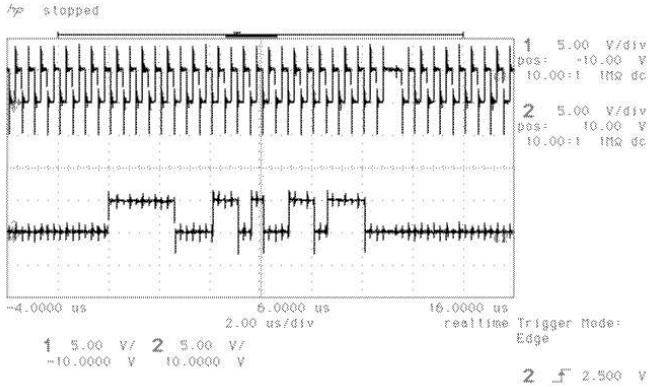
1 x y

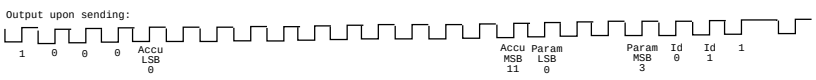
Applying word $zx_3...x_0$ as parameter launches an IOW instruction to address $xyzx_3...x_0$ and writes accu out to a Panavia-Link DPL01 to DPL04. $x_3...x_0$ are inserted as destination into the datagram, where x, y and z determine the link used for output and the identifier used. The following scheme applies:

Plug	x	y	z	Identifier	**Command**
DPL02	0	1	1	00	**WDPL02** 0, $x_3...x_0$
DPL03	1	1	1	00	**WDPL03** 0, $x_3...x_0$ $(\neq 1111)$
DPL02	0	0	1	01	**WDPL02** 1, $x_3...x_0$
DPL03	1	0	1	01	**WDPL03** 1, $x_3...x_0$
DPL02	0	1	0	10	**WDPL02** 2, $x_3...x_0$
DPL03	1	1	0	10	**WDPL03** 2, $x_3...x_0$
DPL01	0	0	0	11	**WDPL01** 3, $x_3...x_0$
DPL04	1	0	0	11	**WDPL04** 3, $x_3...x_0$

Monitoring the DPL01-DPL03 outputs in an negative regime, i.e. use CLK- on Pin7 for clock and DATA- on Pin10 for signal one gets the following diagrams:



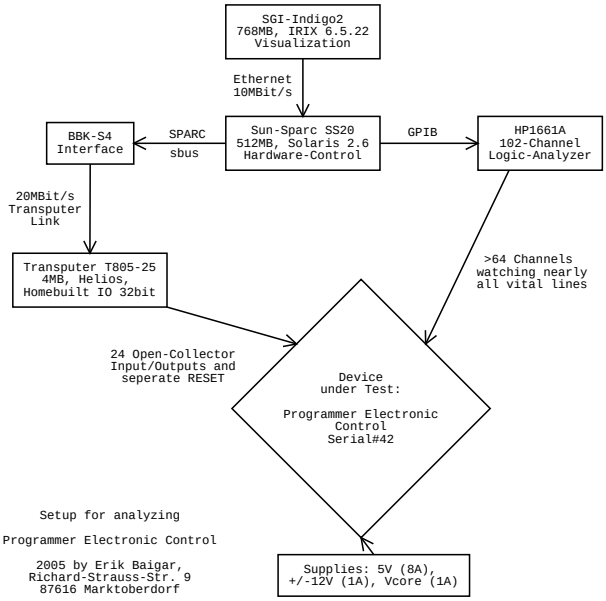


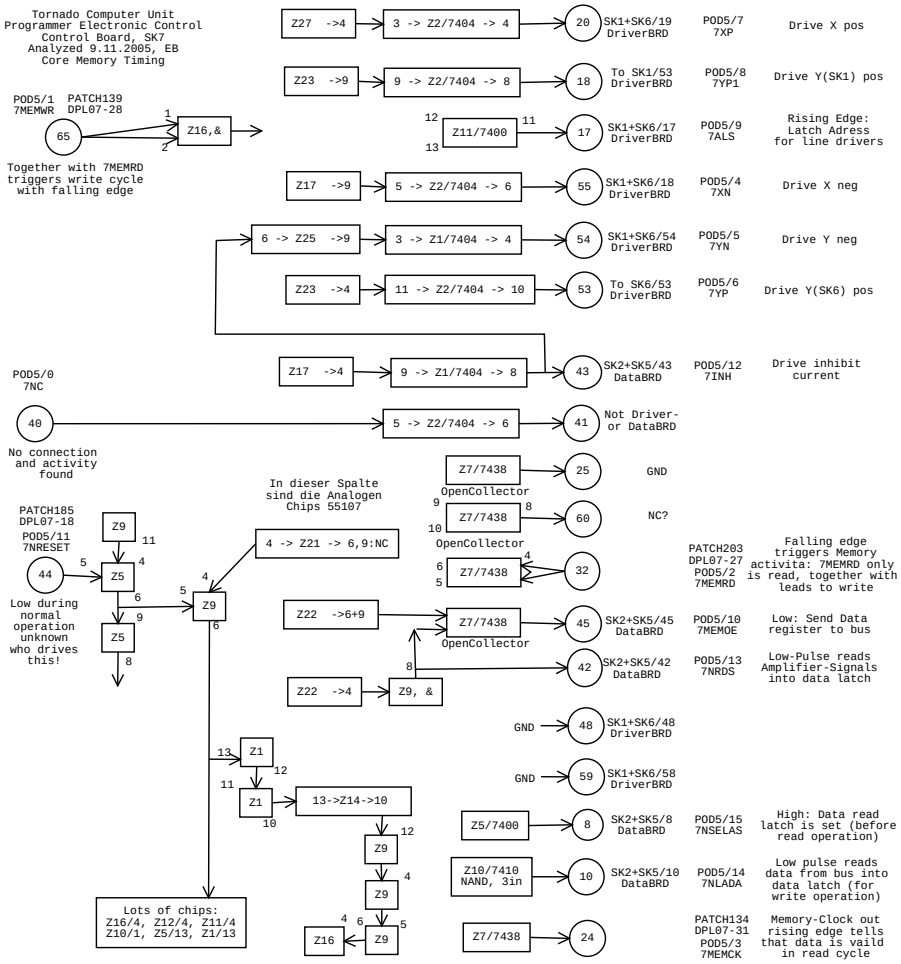


Issuing a second transmission while another transmission is running causes the program execution to be delayed until transmission has completed. The same applies if the second transmission is issued to a different DPL01-DPL04 output or a read operation is initiated.

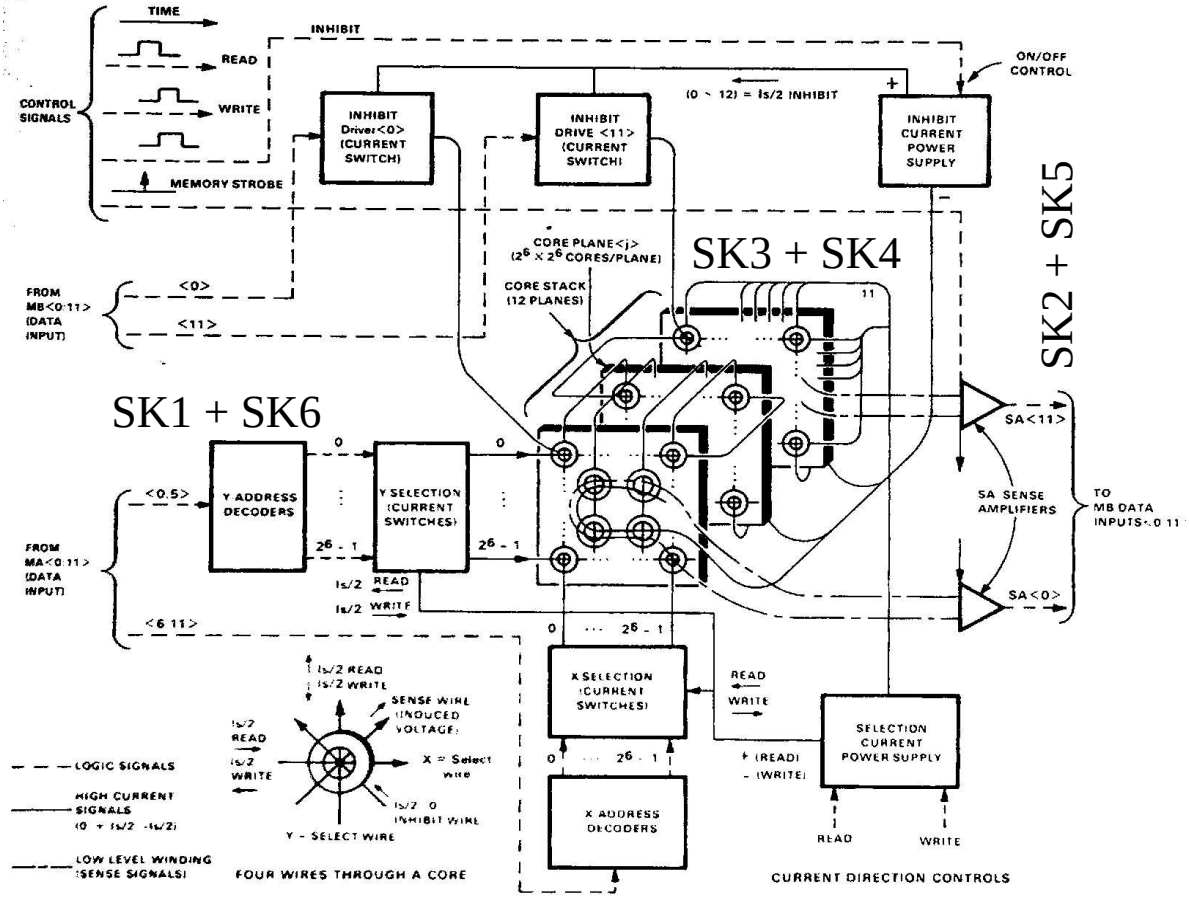
0xf80, 4095, Function 15-1, $4.866\mu s - 13.000\mu s$

x_{11}	x_{10}	x_9	x_8	$x_7 ... x_0$ **and Description**
1	1	1	1	**RETFI:** Return from Interrupt if $x_7 = ... = x_0 = 1$. 0xfff, 4095, Function 15-1-127, $7.120\mu s$

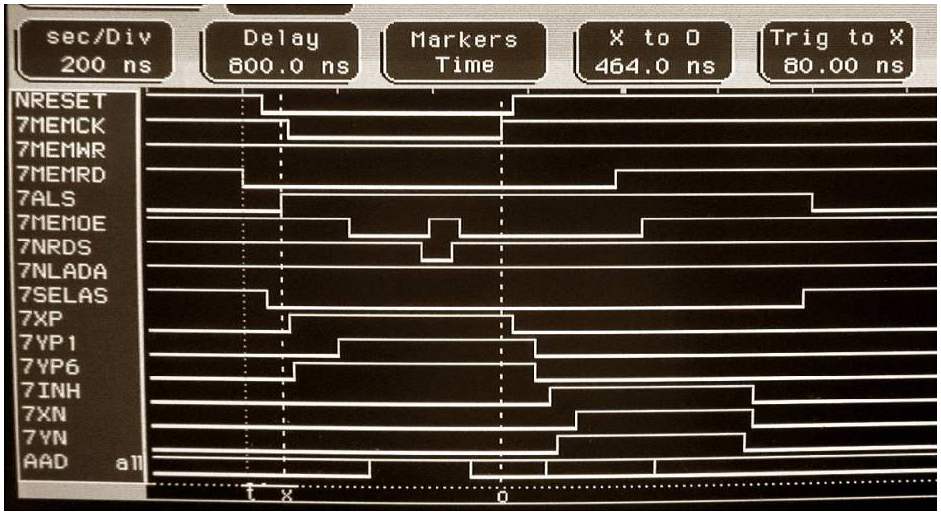


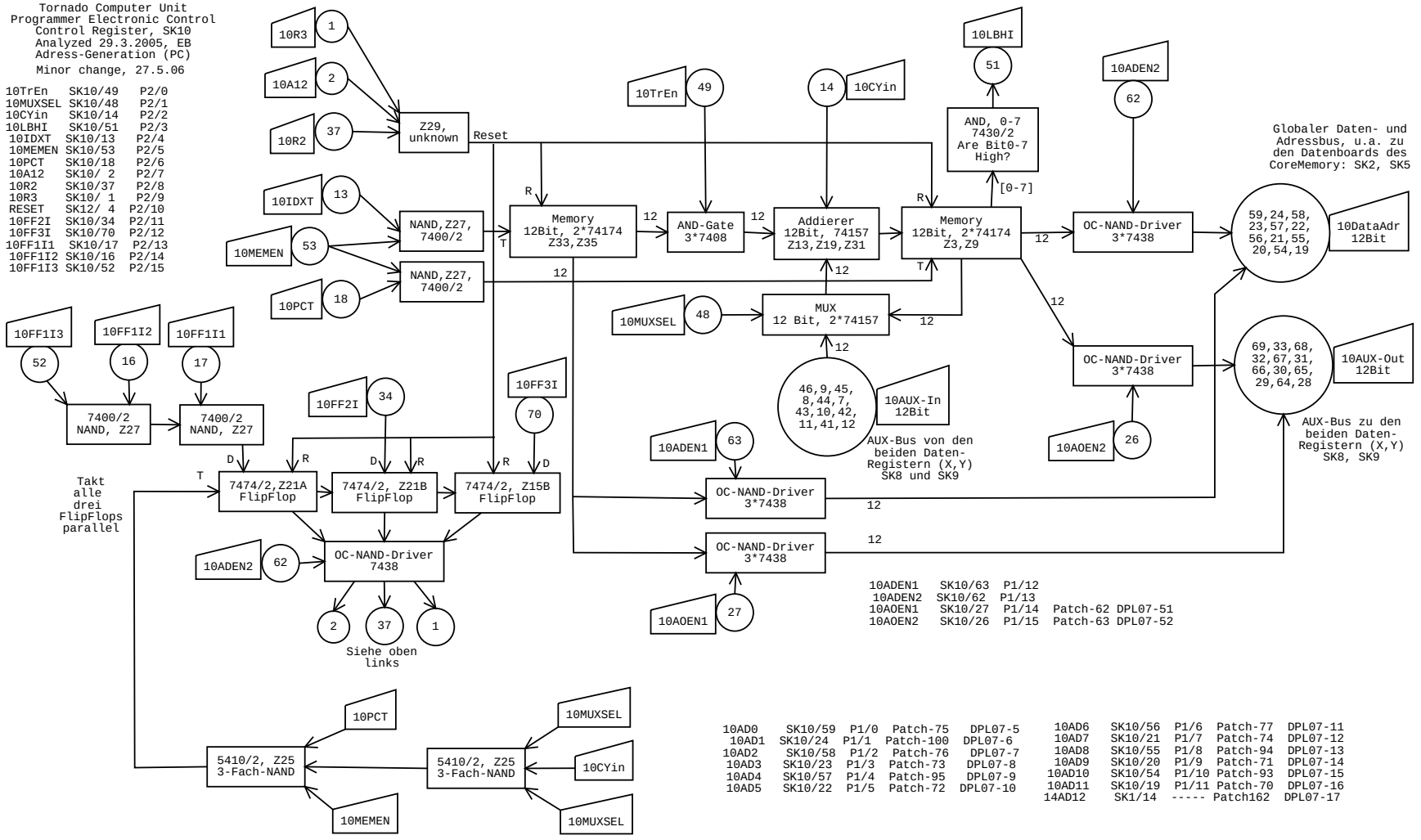


from SK7

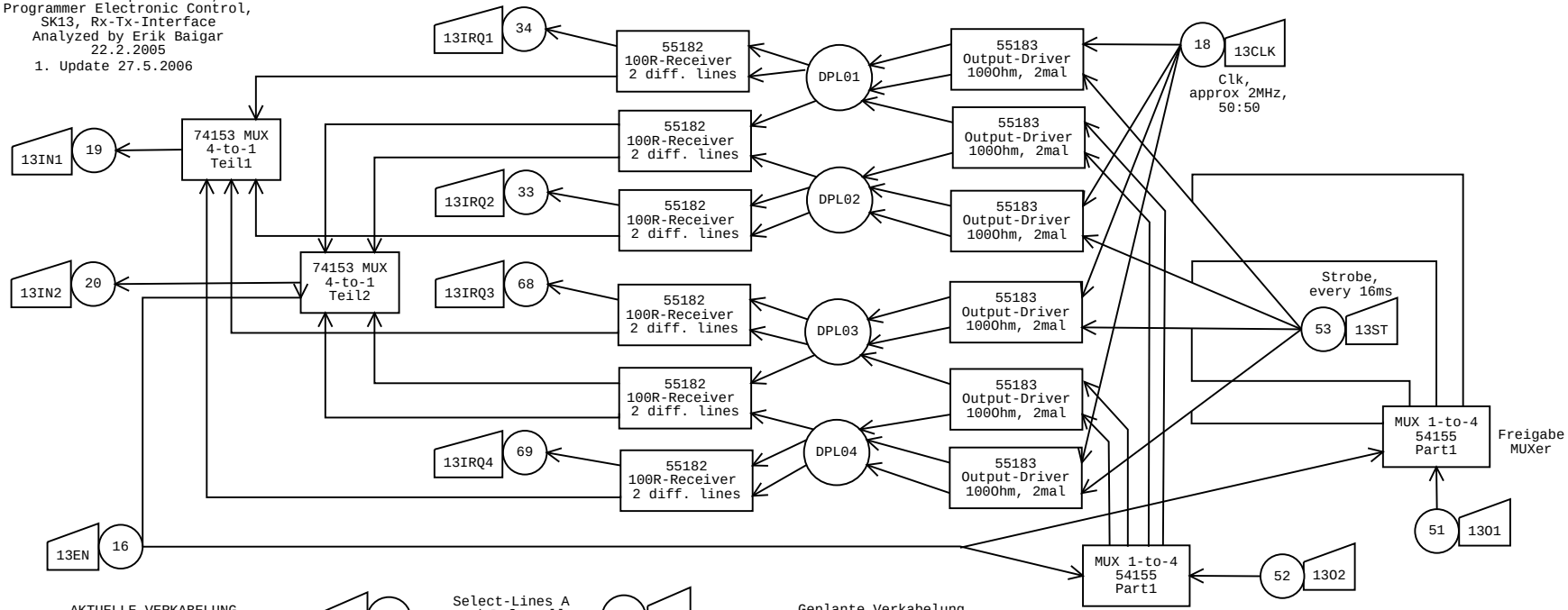


SK1 + SK6




Tornado Computer Unit
Programmer Electronic Control
Control Register, SK10
Analyzed 29.3.2005, EB
Adress-Generation (PC)
Minor change, 27.5.06
10TrEn SK10/49 P2/0
10MUXSEL SK10/48 P2/1
10CYin SK10/14 P2/2
10LBHI SK10/51 P2/3
10IDXT SK10/13 P2/4
10MEMEN SK10/53 P2/5
10PCT SK10/18 P2/6
10A12 SK10/ 2 P2/7
10R2 SK10/37 P2/8
10R3 SK10/ 1 P2/9
RESET SK12/ 4 P2/10
10FF2I SK10/34 P2/11
10FF3I SK10/70 P2/12
10FF1I1 SK10/17 P2/13
10FF1I2 SK10/16 P2/14
10FF1I3 SK10/52 P2/15
10R3
1
10A12
2
10R2
37
Z29, unknown
Reset
10IDXT
13
10MEMEN
53
10PCT
18
NAND,Z27, 7400/2
NAND,Z27, 7400/2
Memory 12Bit, 2*74174 Z33,Z35
10TrEn
49
AND-Gate 3*7408
14
10CYin
Addierer 12Bit, 74157 Z13,Z19,Z31
10LBHI
51
AND, 0-7 7430/2 Are Bit0-7 High?
[0-7]
Memory 12Bit, 2*74174 Z3,Z9
10ADEN2
62
OC-NAND-Driver 3*7438
Globaler Daten- und Adressbus, u.a. zu den Datenboards des CoreMemory: SK2, SK5
59,24,58, 23,57,22, 56,21,55, 20,54,19
10DataAdr 12Bit
10MUXSEL
48
MUX 12 Bit, 2*74157
OC-NAND-Driver 3*7438
69,33,68, 32,67,31, 66,30,65, 29,64,28
10AUX-Out 12Bit
46,9,45, 8,44,7, 43,10,42, 11,41,12
10AUX-In 12Bit
AUX-Bus von den beiden Daten-Registern (X,Y) SK8 und SK9
AUX-Bus zu den beiden Daten-Registern (X,Y) SK8, SK9
10AOEN2
26
10FF1I3
52
10FF1I2
16
10FF1I1
17
7400/2 NAND, Z27
7400/2 NAND, Z27
10FF2I
34
10FF3I
70
Takt alle drei FlipFlops parallel
7474/2,Z21A FlipFlop
7474/2, Z21B FlipFlop
7474/2, Z15B FlipFlop
10ADEN1
63
OC-NAND-Driver 3*7438
OC-NAND-Driver 3*7438
10ADEN2
62
OC-NAND-Driver 7438
2
37
1
Siehe oben links
10AOEN1
27
10ADEN1 SK10/63 P1/12
10ADEN2 SK10/62 P1/13
10AOEN1 SK10/27 P1/14 Patch-62 DPL07-51
10AOEN2 SK10/26 P1/15 Patch-63 DPL07-52
10PCT
10MUXSEL
5410/2, Z25 3-Fach-NAND
5410/2, Z25 3-Fach-NAND
10CYin
10MEMEN
10MUXSEL
10AD0 SK10/59 P1/0 Patch-75 DPL07-5
10AD1 SK10/24 P1/1 Patch-100 DPL07-6
10AD2 SK10/58 P1/2 Patch-76 DPL07-7
10AD3 SK10/23 P1/3 Patch-73 DPL07-8
10AD4 SK10/57 P1/4 Patch-95 DPL07-9
10AD5 SK10/22 P1/5 Patch-72 DPL07-10
10AD6 SK10/56 P1/6 Patch-77 DPL07-11
10AD7 SK10/21 P1/7 Patch-74 DPL07-12
10AD8 SK10/55 P1/8 Patch-94 DPL07-13
10AD9 SK10/20 P1/9 Patch-71 DPL07-14
10AD10 SK10/54 P1/10 Patch-93 DPL07-15
10AD11 SK10/19 P1/11 Patch-70 DPL07-16
14AD12 SK1/14 ----- Patch162 DPL07-17

Tornado Computer Unit,
Programmer Electronic Control,
SK13, Rx-Tx-Interface
Analyzed by Erik Baigar
22.2.2005
1. Update 27.5.2006



AKTUELLE VERKABELUNG

13A	SK13/17	POD6/13
13B	SK13/50	POD6/12
13EN	SK13/16	POD6/14
13CLK	SK13/18	POD6/15

13A 17

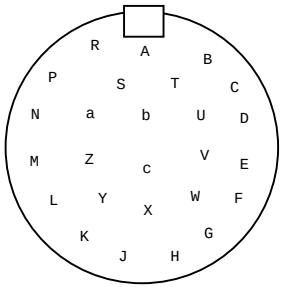
Select-Lines A
and B for all
MUXes connected

50 13B

Geplante Verkabelung,
NICHT UMGESETZT

13EN	SK13/19	P5/0	13IRQ1	SK13/34	P5/3	13A	SK13/17	P5/7
13IN1	SK13/20	P5/1	13IRQ2	SK13/33	P5/4	13B	SK13/50	P5/8
13IN2	SK13/16	P5/2	13IRQ3	SK13/68	P5/5			
			13IRQ4	SK13/69	P5/6			

13CLK	SK13/18	P5/9
13ST	SK13/53	P5/10
13O1	SK13/51	P5/11
13O2	SK13/52	P5/12



DPL01 - DPL04

A: CLK Out +
B: CLK Out -

C: Data Out +
D: Data Out -

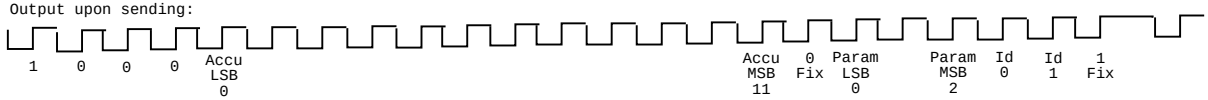
E: Data In +
F: Data In -

G: CLK In +
H: CLK In -

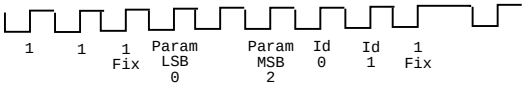
V: IRQ out +
W: IRQ out -

T: IRQ in +
U: IRQ in -

Output upon sending:

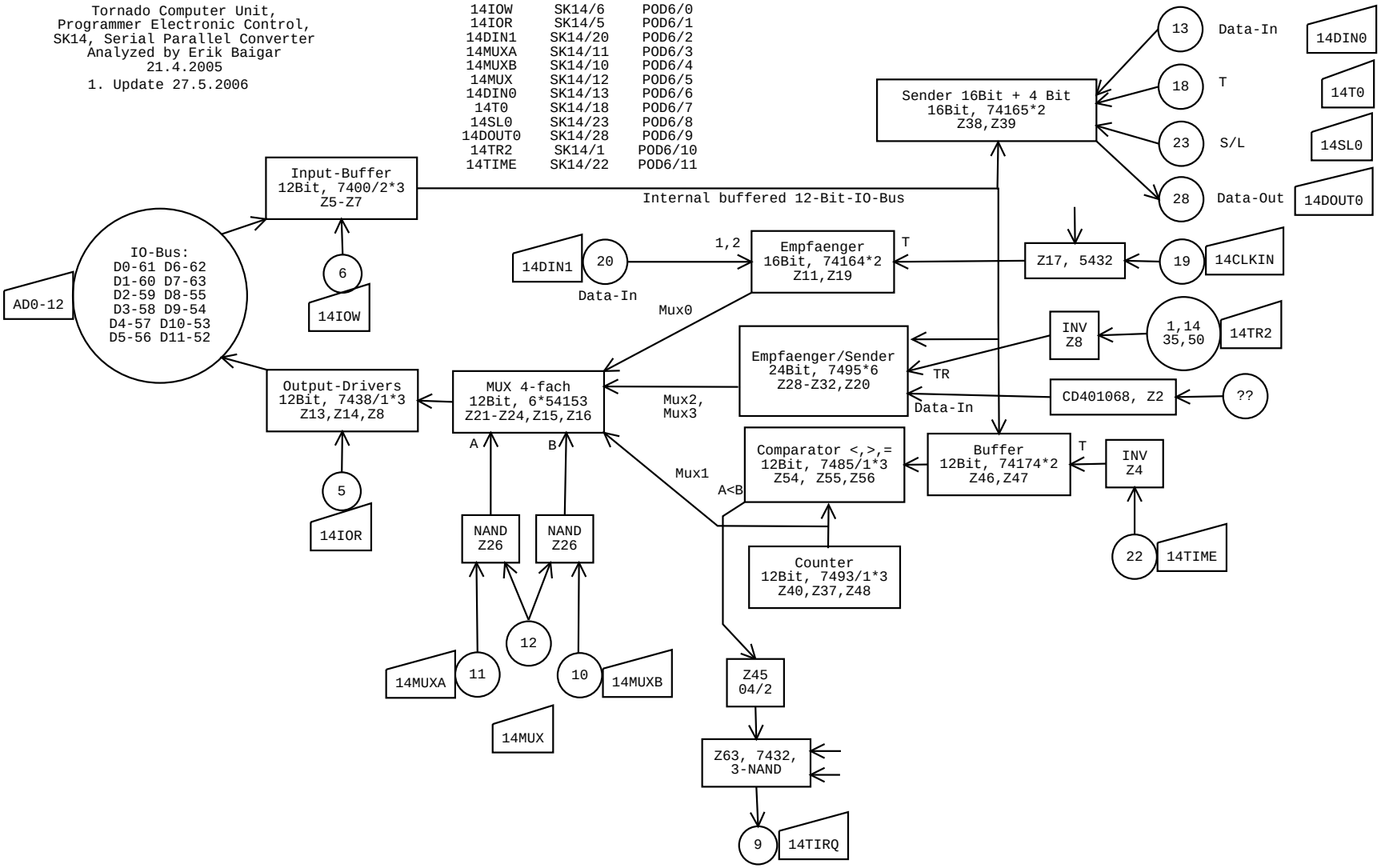


Output due to reading:

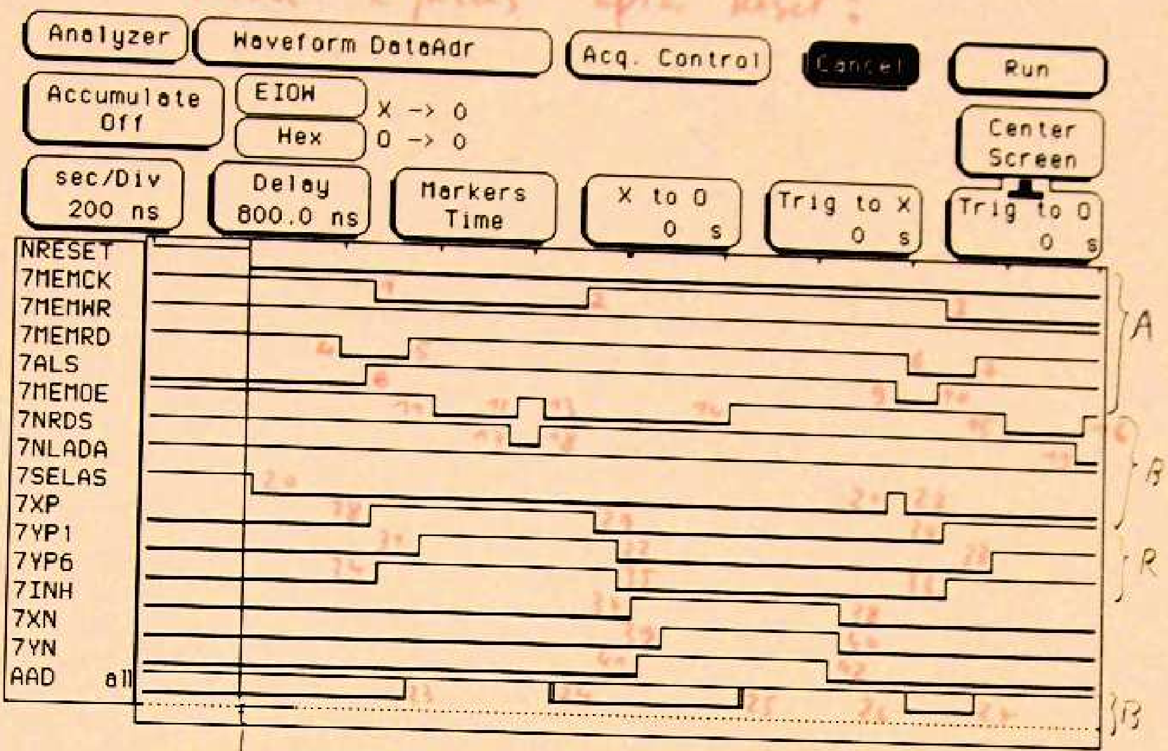


Tornado Computer Unit,
Programmer Electronic Control,
SK14, Serial Parallel Converter
Analyzed by Erik Baigar
21.4.2005
1. Update 27.5.2006

14IOW	SK14/6	POD6/0
14IOR	SK14/5	POD6/1
14DIN1	SK14/20	POD6/2
14MUXA	SK14/11	POD6/3
14MUXB	SK14/10	POD6/4
14MUX	SK14/12	POD6/5
14DIN0	SK14/13	POD6/6
14T0	SK14/18	POD6/7
14SL0	SK14/23	POD6/8
14DOUT0	SK14/28	POD6/9
14TR2	SK14/1	POD6/10
14TIME	SK14/22	POD6/11



Read-Zyklus after Reset:



t=0

↑ of Mem RD triggers read-Cycle

1	264ns	4	192ns	8	248ns	(A)
2	712ns	5	336ns	9	1368ns	
3	1472ns	6	1392ns	10	1456ns	
		7	1536ns			

11	392ns	17	552ns	20	16ns
12	568ns	18	616ns	21	1352ns
13	624ns	19	1152ns	22	1392ns
14	1016ns				
15	1600ns				
16	1168ns				

(B)

23	334ns	24	648-656ns
26	1400ns	25	1048-1056ns
27	1544ns		

24, 25 → Data valid

R:

28	264ns
29	736ns
30	1472ns
31	368ns
32	784ns
33	1576ns
34	280ns
35	784ns
36	1480ns

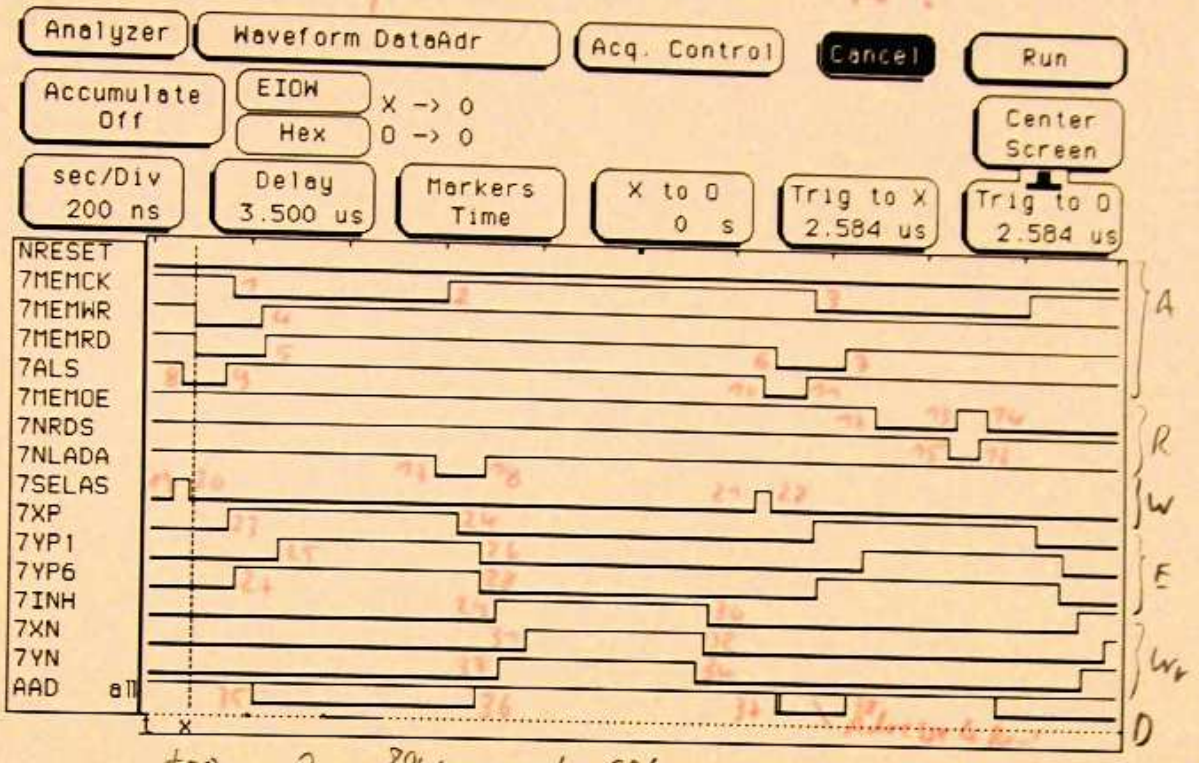
W:

37	816ns
38	1256ns
39	880ns
40	1256ns
41	832ns
42	1232ns

37, 38] INH

Cycle-Time 1200ns

Write-Cycle : 1010000000 3rd :



t=0

(A)

1	80ns	4	736ns	8	-24ns
2	520ns	5	744ns	9	64ns
3	1280ns	6	1200ns	10	1176ns
		7	1344ns	11	1264ns

(R):

12	1408ns	17	496ns
13	1576ns	18	600ns
14	1640ns	19	-40ns
15	1560ns	20	-8ns
16	1624ns	21	1160ns
		22	1792ns

(W)

(E) write:

23	72ns	29	624ns
24	544ns	30	1064ns
25	176ns	31	688ns
26	592ns	32	1056ns
27	88ns	33	672ns
28	592ns	34	1040ns

(Wr) write

Beide falling-edges zusammen triggern Write-Cycle

Data: (Wr)

35:	128ns	36	584ns

Addr & Rest:

37:	1208-1216	38	1352ns