MOTOROLA
SEMICONDUCTOR APPLICATION NOTE

Order this document
as AN492/D

AN492/D

A video display board for CD-i development

Colin MacDonald,
CD-i and Multimedia Group,
Motorola Ltd., East Kilbride,
Scotland.

INTRODUCTION

Compact Disc-Interactive (CD-i) is one of the latest developments in CD technology. Aimed primarily at the consumer market, CD-i allows users to interact with a high quality audio-visual presentation. The presentation or title can be designed for entertainment or education or business. CD-i's worldwide standard, known as the Green Book, defines the system hardware, software and data encoding methods to a degree that ensures worldwide compatibility without specifying a particular architectural implementation.

Presently, there are two video standards used in the field of CD-i: the Green Book-specified "Base Case" standard; and the Motion Pictures Experts Group (MPEG) ISO-11172 standard. The term "Base Case", as used in CD-i, defines the minimum functionality required of a player. This application note describes the design of a CD-i Base Case Video board, hereafter BCV, which forms part of Motorola's CD-i Development System (MCDS).

The MCDS, running either "extended OS-9" or "extended CD-RTOS", allows users to develop CD-i hardware and software solutions based on Motorola's chipset. Evaluation of key CD-i components is facilitated through their implementation in the system. Also, further hardware development can be achieved via the addition of boards or by use of the prototype areas on existing MCDS boards. The high performance MCDS is suitable for both resident and cross software development and can be used for CD-RTOS file manager/driver evaluation.

The MCDS is a multi-board, dual-bus system based on the VME double-extended Eurocard hardware specification. The P1 backplane is used for the VME, while P2 carries the MCDS "System Bus". This dual backplane approach allows the use of industry standard VME cards via P1, while P2 provides a highly-optimised interface between key components for performance evaluation. While general interboard connectivity is provided via P2, additional connectors are used for specific signals required by a subset of system boards. In the case of the BCV, the 32-bit Base Case Video Bus (V_BUS) is driven on connector P9 for use by MPEG-V boards.

DESIGN FEATURES

- Motorola CD-i Development System Compatible
- Based on the highly integrated MCD210.
- Double extended Eurocard board with 0–70 °C operation
- High speed system bus interface
- Multiple memory configurations using 256Kx4,1Mx4 or 256Kx16 DRAMs
- PAL / NTSC / NTSC MONITOR compatible
- PAL Channel 36 or NTSC Channel 34 RF modulated output
- CENELEC peritelevision / SCART interface
- PAL and NTSC RS-343A compatible singly-terminated RGB outputs with separate sync
- SUPER-VIDEO output
- 32-bit digital bus for off-board drive of Base Case video and control signals
- Support for Motorola's CD-i Development System MPEG board

7/94

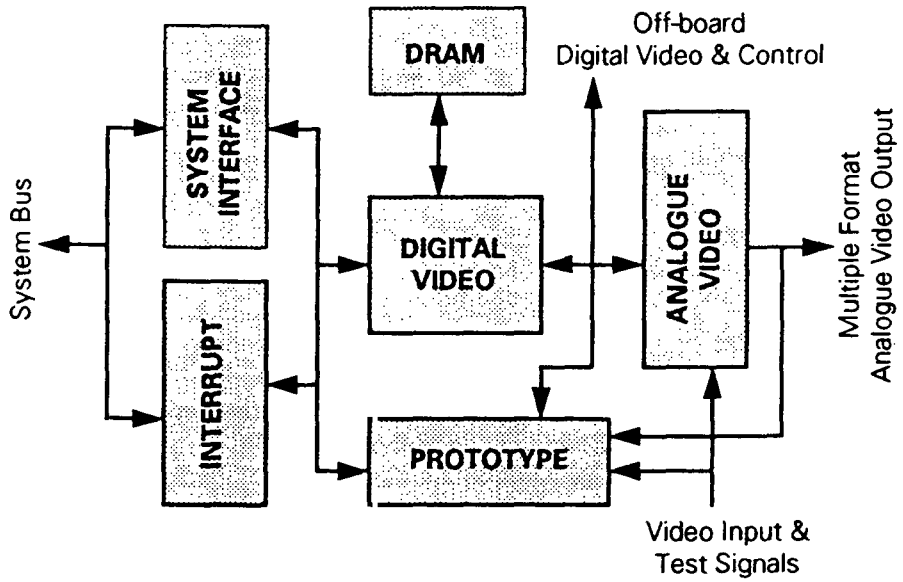


Figure 1. Block diagram of the Base Case Video (BCV) Board

DESIGN DESCRIPTION

TOP LEVEL BLOCK #1

The BCV design consists of six functional blocks, four digital and two mixed signal blocks. The SYSTEM INTERFACE block provides backplane bus buffering and termination in addition to bus conversion and memory map decode.The INTERRUPT module supports interrupt multiplexing and can generate interrupt requests to the system processor on 6 backplane levels. The DIGITAL VIDEO module, the heart of the BCV design, contains DRAM controller and video controller/decoder functions. The DRAM block provides system and video RAM with three possible implementations. Video and control outputs are driven both to a connector for off-board use and to the ANALOGUE VIDEO block that converts the BCV's digital video into industry-standard analogue formats.

SYSTEM INTERFACE

The SYSTEM INTERFACE block is a high-performance, flexible module providing address, data and control bus buffering, VDSC and control register decode, control bus conversion, address strobe and data strobe delay circuitry and manual reset. In detail, IC's 24 and 25 provide data bus buffering under the control of IC38, while three 74F244s (ICs 19, 20 and 26) provide simple address buffering with minimal propagation delays. Address bus lines are assigned to buffer inputs so that the low-order (usually higher frequency) signals are distributed evenly across the three buffers—thus reducing the opportunity for ground bounce.

The CD-i development system's P2 backplane features two sets of control buses, based on the MC68000 and MC6834x processors. The BCV can operate with either or both of these buses. MC6834x-style control signals are translated (by IC38) into the BCV board's M68000-style bus. On-board address and data validation can be delayed by one to three clock periods through IC's 37 and 42. Jumpers 21 through 23 set the address validation delay period, and jumpers 24, 25 and 17 set the data validation delay. These delays can be used to allow the board to work with higher speed processor cards or with noisy backplanes.

In addition to decoding the BCV's address space, IC27 also provides chip selects for the PROTOTYPE and INTERRUPT modules.

DIGITAL VIDEO

The functionality of the DIGITAL VIDEO module, and the BCV, is largely determined by one component, the MCD210. The MCD210 Video Decoder and System Controller (VDSC) is a highly integrated circuit designed specifically for CD-i. This device (IC29) incorporates DRAM control, video image decode, video timing generation and system interface functions. As video timing is derived from the VDSC clock, the oscillator OSC1 is socketed to allow the production of PAL, NTSC or NTSC monitor video formats. Jumper J7 can be configured so that a prototype area device supplies the VDSC clock.

Three 74F373 devices (IC's 31, 33, 30) are used to latch the 24-bit RGB output of the VDSC for driving to the MPEG-V board. Most video control signals are buffered by a PAL (IC21), thus providing some degree of flexibility in a crucial area. Series termination resistors (RN 6, 7, 10, 11, 15–18) located close to the driving chips, counteract transmission line reflections and improve signal integrity.

Resistor networks (RN 12–14, 19, 25–27)) are similarly used for VDSC signals going to on-board destinations. In this case the resistors are placed as close as possible to IC29.

Series termination resistors are again used with the DRAM address bus. In fact, the VDSC's memory address bus [MA(9:0)] is translated into two buses [MA1(9:0) and MA2(9:0)] via these resistors (RN 3-5, 8, 9). This dual bus arrangement is designed to improve the effectiveness of series termination with distributed loads such as the DRAM array.

A 74F244 (IC22) buffers DRAM control signals, while IC46 drives RUN and STOP LEDs, both of which reflect the status of the BCV, not the system. The green "RUN" LED, D2, is illuminated on DRAM accesses whereas the red "STOP" LED, D1 is illuminated when the VDSC's RSTOUT pin is low. Jumpers 1 and 2 are used to validate the VDSC's DRAM banks.

DRAM

In a CD-i player, MCD210 controlled DRAM is used for both system and video functions. As system RAM, it holds code or data and can provides buffer area for MPEG data in extended CD-i systems. As video RAM, it holds image and image control data. All accesses to the DRAM go through and are controlled by the VDSC.

The VDSC is designed to support 256Kx4, 1Mx4 and 256Kx16 fast page-mode DRAMs. These can be arranged in six configurations (listed in the VDSC specification), ranging in size from 0.5 to 4 Mbyte.The BCV board is compatible with all VDSC-supported DRAM types and configurations. IC's 4–7, 13–16, nominally MCM54400AZ devices (1Mx4), are in fact zig-zag sockets.These may also be populated with MCM514256AZ, 256Kx4 DRAMs. If 256Kx16 devices (IC's 11, 12) are used, IC's 4–7, 13–16 should not be inserted.

The BCV DRAM array is designed to minimise transmission line ringing. Address bus lines have series termination, whilst control lines have daisy-chained wiring and Thevenin terminations (RN1,RN2).

INTERRUPT

The BCV interrupt module has two functions. When the BCV is used in an IOP2[1] "Host Bus" based system, the interrupt module can be used to "multiplex" up to three interrupts onto one IOP2 interrupt line. When the BCV is used in the CD-i development system, "multiplexing" of interrupts is unnecessary and only the jumper circuitry is used to map the BCV's IRQ to a chosen backplane line. Operation in both these environments is described below.

1. The IOP2 CPU board is described in Motorola application note AN451 "An MC68340-based Input/ Output Processor Design".

IOP2-BASED OPERATION

As the IOP2 system bus contains only one interrupt line, IRQs from other boards are forced to occupy one level. This means that the processor is unable to selectively mask off-board interrupts, reducing the flexibility and performance of the system.The BCV can be used to solve this problem, by performing the interrupt masking function for the processor board.

In multiplexing mode, the BCV can route two IOP2 Host Bus backplane interrupts on P2 26B and 27B[2] (SYS_IRQE/F_N) through jumpers J9 and J8 and into IC8 as IRQ_MUX1_N and IRQ_MUX2_N respectively. These, along with the on-board VDSC interrupt (VC_IRQ_N), are prioritised in IC8 according to a code held in the BCV's "Interrupt Priority Level Register", IPL_REG (IC's 2, 9). [This prioritisation code reflects the absolute level of each interrupt source that, again, is held in the IPL_REG]. Then the output of IC8 corresponding to the highest priority active interrupt (Pins 17–19) is asserted, generating two events in IC1. Firstly, the level of the highest priority IRQ is output on IPL(2:0) and compared (in IC18) to a copy of the CPU's interrupt mask held in the "Duplicate Status Register", DUPL_SR (IC10). Secondly, if the comparison indicates (through an assertion of A_GT_B_O) that the priority level is higher than the masking level, then IC1 asserts BCV_IRQ_N.

Following BCV_IRQ_N's assertion, the IOP2 drives P2 25B (SYS_IRQD_N) low, indicating an IACK cycle. With jumper J5 set, this signal forms a chip select for the interrupt vector register, IOPIVR (IC3). This register provides an interrupt vector based on the contents of the IPL_REG and on the generating IRQ.

In order to prevent interrupts being asserted before the appropriate registers are initialised, IRQ generation is disabled until the memory-mapped "Interrupt Enable" line, IRQENA_SEL_N, is asserted. Also, to prevent the interrupt vector from being corrupted, interrupt prioritisation is frozen by the IOPIVR_SEL_N line during IACK cycles.

INTERRUPT REGISTERS

The BCV design includes four registers that are used to control or monitor interrupt logic for IOP2-based operation. These registers are described below.

INTERRUPT PRIORITY LEVEL REGISTER

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BIV3	BIV2	BIV1	BIV0	RIP2	RIP1	RIP0	L2_2	L2_1	L2_0	L1_2	L1_1	L1_0	LV2	LV1	LV0
RESET															
U	U	U	U	U	U	U	U	U	U	U	U	U	U	U	U

Table 1. Interrupt Priority Level Register

The interrupt priority level register is a 16-bit wide, write-only register. It controls the absolute and relative priorities of interrupt sources and provides the most significant bits of the interrupt vector.

BIV(3:0) – Base Case Interrupt Vector

This field provides the most significant 5 bits of the BCV's interrupt vector. IV(3:1) map directly to bits 7–5 of the vector whereas IV(0) maps to bits 4 and 3.

2. The MCDS System Bus is based on the IOP2 Host Bus, but is modified to suit CD-i development. In particular, lines P2 25B, 26B, 27B have changed function. All labels referenced here reflect MCDS usage.

RIP(2:0) – Relative Interrupt Priority

This field determines the relative priority of interrupt sources. It is used to decide if an active interrupt is propagated or blocked. This field should reflect the absolute priority assigned to an interrupt source. It should be programmed according to the following table.

RELATIVE PRIORITY OF INTERRUPT SOURCES	RIP(2:0)
IRQ_MUX2_N>IRQ_MUX1_N>VC_IRQ_N	000
IRQ_MUX2_N>VC_IRQ_N>IRQ_MUX1_N	001
RESERVED	010
VC_IRQ_N>IRQ_MUX2_N>IRQ_MUX1_N	011
IRQ_MUX1_N>IRQ_MUX2_N>VC_IRQ_N	100
RESERVED	101
IRQ_MUX1_N>VC_IRQ_N>IRQ_MUX2_N	110
VC_IRQ_N>IRQ_MUX1_N>IRQ_MUX2_N	111

Table 2. RIP Encodings

L2_(2:0) – Level of MUX2 Interrupt

This is the priority level assigned to IRQ_MUX2_N. It will be used to compare with the interrupt mask level held in the duplicate status register.

L1_(2:0) – Level of MUX1 Interrupt

This is the priority level assigned to IRQ_MUX1_N. It will be used to compare with the interrupt mask level held in the duplicate status register.

LV(2:0) – Level of VDSC Interrupt

This is the interrupt level assigned to VC_IRQ_N. It will be used to compare with the interrupt mask level held in the duplicate status register.

DUPLICATE STATUS REGISTER

The duplicate status register is an eight-bit write-only register that sits on the upper half of the data bus. It is used to control the propagation of interrupts to BCV_TGATE_N and BCV_IRQ_N.

7	6	5	4	3	2	1	0
DC4	DC3	DC2	DC1	DC0	IPM2	IPM1	IPM0
RESET							
U	U	U	U	U	U	U	U

Table 3. Duplicate Status Register

DC(4:0) – Don't Care

Any values can be written to these bits without effect.

IPM(2:0) – Interrupt Priority Mask

These bits should reflect the processor's current interrupt mask level. The value held in this field determines if an active interrupt is sent to the processor. Only if the interrupt level of the highest priority active interrupt is greater than I(2:0) will the interrupt be sent.

INTERRUPT STATUS REGISTER

7	6	5	4	3	2	1	0
M2OS	M1OS	VCOS	AIOS	0	IOL2	IOL1	IOL0
RESET							
1	1	1	0	0	0	0	0

Table 4. Interrupt Status Register

The interrupt status register is an 8-bit read only register mainly intended for debug purposes. It reflects the unlatched condition of the BCV's interrupt multiplexing circuitry outputs and can therefore change at any time.

M2OS – MUX2 Output Status

0 – IRQ_MUX2_N is the highest priority asserted IRQ source

1 – IRQ_MUX2_N is not the highest priority asserted IRQ source

M1OS – MUX1 Output Status

0 – IRQ_MUX1_N is the highest priority asserted IRQ source

1 – IRQ_MUX2_N is not the highest priority asserted IRQ source

VCOS – VDSC Interrupt OutputStatus

0 – IRQ_MUX1_N is the highest priority asserted IRQ source

1 – IRQ_MUX2_N is not the highest priority asserted IRQ source

AIOS – Any Interrupt Output Status

0 – The highest priority asserted interrupt has a lower level than the DUPL_SR mask

1– The highest priority asserted interrupt has a higher level than the DUPL_SR mask

IOL(2:0) – Interrupt Output Level

This field reflects the level of the highest priority asserted IRQ source

INTERRUPT VECTOR REGISTER

The Interrupt Vector Register is an 8-bit, non-memory-mapped register. It is used to drive an interrupt vector number onto the data bus in response to an IACK cycle. The interrupt vector reflects the values of the IPLREG BIV field and interrupt output status lines latched on the falling edge of IACK.

7	6	5	4	3	2	1	0
LBIV4	LBIV3	LBIV2	LBIV1	LBIV0	LMOS2	LMOS1	LVOS
RESET							
U	U	U	U	U	0	0	0

Table 5. Interrupt Svector Register

LBIV(4:0) – Latched Base Case Interrupt Vector

LBIV(4:2) reflects the latched value IPLREG BIV(3:1)

LBIV(1:0) reflects the latched value of IPLREG BIV(0)

LMOS2, LMOS1 and LVOS reflect the latched values of IRQ_MUX2_OUT, IRQ_MUX1_OUT and IRQ_VC_OUT respectively.

CD-i DEVELOPMENT SYSTEM

The CD-i Processor board (CDIP) and MCDS System Bus were both designed to support seven interrupt lines. The BCV board can map MCD210 generated interrupts to any of SYS_IRQA_N through SYS_IRQF_N using jumpers J3, J8-J11. As the MCD210 doesn't support vectored interrupts, the IRQ assigned to the BCV has to be autovectored.

ANALOGUE VIDEO

The Motorola CD-i Development System is intended to be a flexible platform and the design of this module clearly reflects this. Taking in 24-bit digital RGB signals, the ANALOGUE VIDEO block outputs PAL, NTSC or NTSC MONITOR formats on a variety of industry-standard connectors such as BNC, PERITEL and S-VIDEO.

In order to produce a high quality video signal, the BCV is manufactured as a multi-layer, split-plane PCB. Only components shown with an analogue ground connection are placed over the analogue plane, with digital devices and tracks restricted to the digital plane. The planes connect through 2.2µH inductors (L3, L4) and through the MC44200 (IC44), so these are placed together to minimise current loops.

The MC44200 Triple 8-bit Video DAC, with segregated analogue and digital signals, is designed for use on split-plane PCBs. The digital signals consist of Red, Green and Blue buses, pixel clock and digital power. Inductors and capacitors C128, C143 provide transient suppression and decoupling on the digital power inputs. Although each analogue video output (R, G or B) on the MC44200 has its own power supply pin, these are all tied to the analogue 5V rail and decoupled with a range of capacitors (C126, 132, 135, 145, 149). These RGB outputs are configured for high current mode via resistor R58, giving a full scale voltage

swing of 1.5V into 75 ohms. Resistor sockets R14, R16 and R23 are only populated during board test, leaving R15, R17 and R24 to provide the correct operational loading. The MC44200's RGB outputs drive MC14576A video amplifiers directly (IC40,41), but are AC-coupled into the MC13077 (IC39).

The MC14576A Dual Video Amplifiers (IC35, 40, 41, 45) provide a non-inverting voltage gain of 6db with 150 ohm loads and, provided input signal levels are less than 1.5V, will work from a single 5V supply. Since the Red, Green, Blue, Composite Video and CENELEC Composite Sync signals do not exceed 1.5V, IC's 40, 41 and 45 all use the analogue 5V rail. However, in order to generate a BNC Sync signal to industry standard levels (approximately 3V), it is necessary to provide a higher supply voltage for IC35. A steady +8Volt level is supplied using an LM317 (IC34) in the arrangement shown. This higher supply voltage also allows IC35 to drive the CENELEC Fast Blanking signal to 2V, the middle of its specified positive range. The actual output levels of CENELEC Fast Blanking, BNC Sync and CENELEC Sync are set by potentiometers R4, R5 and R22.

The MC13077 Advanced PAL/NTSC Encoder produces both composite video and Y/C outputs. The encoder can work in a variety of modes, but here it is configured for RGB inputs and an unlocked 4x subcarrier. The RGB inputs are AC-coupled (C7-9) and the subcarrier frequency is determined by the crystal (Y_TV_STD) and tuning capacitor (C6) values. In this particular mode, the colour difference inputs R-Y and B-Y require clamp storage capacitors (C118,123), whereas the LumaClmp input uses a storage capacitor (C116) in all circumstances.

The MC13077 input SyncIn_Sep connects to jumper J16, which can select one of two sources for this signal: the VDSC or BNC test connectors. The composite sync output of the VDSC is buffered (IC36) and is put through a 4k7 series resistor (R62) before input to the encoder. Reducing the signal current transients minimises cross coupling onto sensitive analogue signals. For board test, a composite sync voltage is developed across R3 and then AC-coupled (C84) into the encoder.

Switching between PAL and NTSC formats is performed by setting J14 appropriately. When switching formats, the subcarrier crystal (Y_TV_STD) and chroma bandpass filter (TOKO-BPF) also have to be changed, but the same luma delay line (LumaDLY) can be used for both TV standards.

The MC13077's composite video output (COMP_V) connects to both of IC45's non-inverting inputs. This means that both of the MC14576A's amplifiers output composite video. One amplifier drives to CENELEC /RS-343 levels, while the other provides an input to an RF-modulator (IC46). As the UM1233 RF-modulator requires a 0.64V peak-to-peak input around a 2.83V black level, the associated amplifier output signal requires some level conversion. Resistors R27, 28 reduce the peak-to-peak voltage and a 470µF capacitor (C10) provides AC-coupling. The LM317LZ regulator (IC43) and associated circuitry provides a DC-bias for the AC-coupled signal.

Video outputs on the MC13077 and MC14576A have all been designed to drive 150 ohm loads. With 75 ohm series resistors added (R 8, 9, 10–13, 18–20, 25, 27), reflections from line discontinuities are largely absorbed. The BCV has also been designed for high quality video. All video signals have short 75 ohm impedance tracks in shielded planes.

PROTOTYPE AREA

The prototype area is designed to facilitate minor customisation of the BCV. It consists of a 0.1" pitch array of plated-through holes located near the junction of digital and analogue planes. Key digital and analogue signals are brought to connectors that abut the area, allowing mixed-mode prototyping.

The P16 connector provides forty-two digital signals.With seventeen address lines, upper and lower data strobes, read/write control and a full 16-bit data bus, users may add extra or new types of memory, such as NVRAM. A chip select with programmable timing reduces the need for address decode and an interrupt line allows devices such as RTCs to be added. (The interrupt line connects to post which is located near the BCV's general interrupt jumpers). Two other signals, PROTO4 and PROTO5 could be used, via IC2, to control the selection of the VDSC clock source.

The P17 connector contains thirteen analogue signals and positive/negative 12 Volt supply rails. Pins 1 to 10 provide signals that may be used in the development of genlock circuitry. In order that support for the prototype area does not compromise the standard configuration video quality, Red, Green and Blue connectors are default to open-circuit. Jumpers 18-20 can be added to connect these to the RGB outputs of IC44.

TOP LEVEL BLOCK #2

The Motorola CD-i Development System (MCDS) P2 backplane is physically but not electrically compatible with the VME P2 backplane. This means it can use VME P2 hardware to provide connections to other CD-i Development System boards. However, under no circumstances should a VME card be connected to the development system's P2 backplane.

The MCDS system bus provides most of the signals necessary for the construction of a modular CD-i player. However, many inter-board signals that are implementation-specific, or are only used by two boards, are carried on separate connectors.

The BCV provides termination for all appropriate signals, independent of their use on the BCV. Two design features allow the termination to be easily optimised. Firstly, all termination resistors (R21-24, 28-37, 40-43) are socketed. Secondly, signals are assigned to specific resistor packs, based on their electrical characteristics. This means that active low control lines such as DS can have a different termination from clock signals such as SYS_CPUCLK.

Providing resistor sockets is also necessary, as the BCV is designed for use with the MPEG-V board. In this instance, the BCV sits between the MPEG-V and the CDIP boards. As only the furthest board from the CDIP should contain terminators, the BCV's termination resistors should be removed.

TOP LEVEL BLOCK #3

The BCV has fifteen off-board connectors, thirteen of which (P2-7, P9-P15), all shown on this sheet, are responsible for transmitting video information. As previously mentioned, the BCV is designed to support the MCDS MPEG-V board. The BCV provides 24 bits of RGB data plus control signals through P9, allowing the MPEG board to output a combined video signal. Via a 64-way IDC, P9 signals are alternated with ground lines to minimise cross-talk.

The other connectors are as follows: P5, 7, 11, 13-15 are 75 ohm, 90° BNCs; P3 is CENELEC (PERITEL); P4, 6, 10, 12 are 75 ohm,180° BNCs; P2 is S-VIDEO. Please note: P4, 6, 10, 12 are only provided on debug or test boards.

TOP LEVEL BLOCK # 4

Although the BCV has no VME interface, it can be considered VME P1 compatible, since the only VME signals it connects to are 5V and 12V supplies. This means that the BCV can connect to the same P1 backplane as the CDIP board.

TOP LEVEL BLOCK # 5

Apart from the specific decoupling used for components in the ANALOGUE VIDEO module, a variety of capacitors have been employed to reduce transients throughout the BCV. The TOP LEVEL BLOCK Sheet 5 shows these capacitors but does not indicate how they are distributed.

All FAST logic and PLA devices have 0.1 and 0.01µF decoupling capacitors across their supply pins. The VDSC uses 0.2µF caps across each supply pin, but DRAMs and termination resistor packs use only 0.1µF. Lastly, six 47µF "reservoir" capacitors are distributed across the BCV, alleviating the effects of large current surges.

MEMORY MAP OF BCV BOARD

Figure 2 shows the memory map of the BCV within the MCDS. Due to partial decoding, development boards and other system resources do not have unique locations in memory. It should also be noted that the Duplicate Status Register, Interrupt Enable line, Interrupt Priority Level Register and Interrupt Status Register are all specifically intended for use with IOP2-based systems and may not be implemented on all boards. In that case, accesses to the region 0xC80000 to 0xCFFE00 will complete without bus errors, but will be invalid.

The area assigned to Base Case Video and System DRAM reflects a 1Mx4 DRAM BCV implementation. For a different DRAM implementation, please consult the MCD210 Users' Manual to see how the memory map is affected.

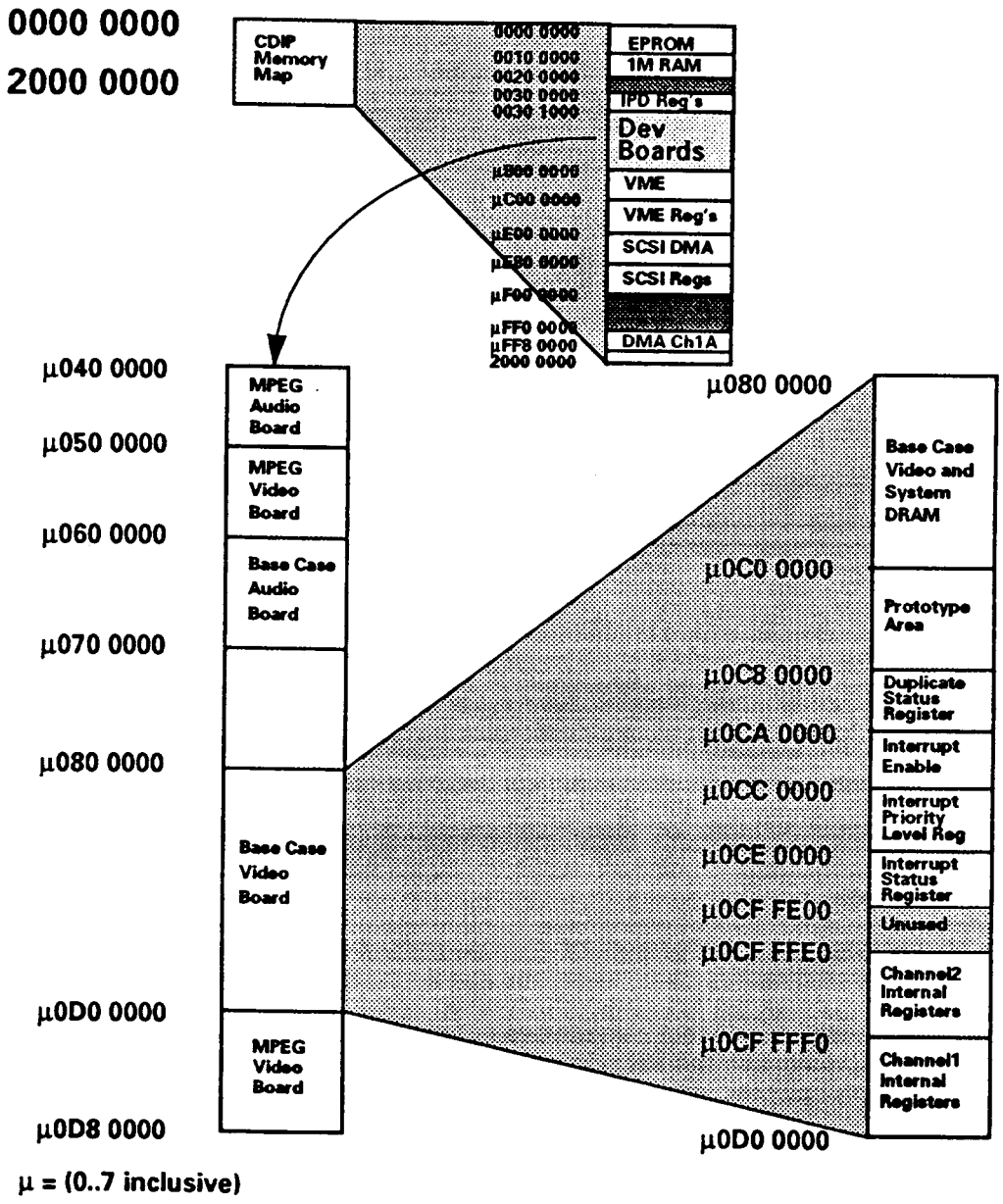


Figure 2. Memory map of the BCV within the MCDS

USAGE and ADDITIONAL DESIGN NOTES

VDSC OSCILLATOR

The frequency of the VDSC oscillator is determined by NTSC or PAL line frequency and associated VDSC timing. With an NTSC colour TV line period of 63.556 µS (CCIR 601) and 120x16 MCD210 clocks per line, the MCD210 requires a 30.2097 MHz oscillator module to produce correct timing. Similarly, for PAL operation, a 30.0 MHz crystal is required.

The oscillator circuitry allows for a number of different clock drive implementations. The VDSC's XTAL input can be driven directly from OSC1 through J6 and J7 using a CMOS or TTL module. If a TTL module is used, resistor R1 and R2 should be chosen so that CMOS input logic levels are achieved. If a CMOS module is chosen, R1 and R2 are not required, other than to shape the waveform. The VDSC's XTAL input may also be driven by IC28 pin 16 and although BUFCTL4 simply routes pin 7 through to pin 16, this PLA can be easily modified to use CLK_VCO from the prototype area.

GENLOCK

The VDSC can operate in MASTER or SLAVE TV mode. The BCV, as described, is configured for MASTER mode operation – the MCD210's MSn pin is pulled high by BUFCTL4 (IC28). In this mode, the VDSC drives HSYNC and VSYNC. However, by reprogramming IC28, the MSn line can be driven low and SLAVE TV mode entered. In this mode, the VSYNC becomes an input to the VDSC.This is useful if the BCV's output is to be synchronised to another video source, or "genlocked".

The BCV has been designed to facilitate the addition of genlock circuitry to the board, primarily through the routing of useful signals to the prototype area. Connector P7, for example, allows a composite video signal to be driven to prototype connector P17. By feeding this signal through a phase splitter – implemented on the prototype area – HSYNC and VSYNC signals can be generated. While VSYNC can be used to supply the VDSC, HSYNC, along with the VDSC's HSYNC output, can be used as the control inputs of a PLL, whose output is used as the VDSC clock, thus providing feedback. By using a PLA (IC28) to route or condition many of the control signals involved, additional flexibility is provided to the genlock circuit designer.

A fully specified CENELEC TV allows rapid switching of video source between its tuner output and SCART RGB[3] inputs. The BCV is designed to support this "mixing", thus allowing TV broadcasts to be overlaid with BCV video. This overlaying requires the BCV to be put into slave mode, as described above. Then, by using the CENELEC VIDEO INPUT line (connector P3 pin 20) to provide a composite video signal to the prototype area and by implementing genlock as outlined above, the BCV video can be synchronised to the TV tuner output. The selection of video source for display is controlled by the CENELEC "BLANKING" line, which can be driven by the MCD210s VSAn line [4]. In this case, TV tuner output will be visible whenever VSAn is low.

DUAL DRAM BUS and LOADING

The VDSC to DRAM interface is very much a "worst case" design. It uses additional circuitry and layout techniques that may not always be required (or desired), especially in the design of a production system. In particular, the BCV uses series termination resistors (RN 3-5, 8, 9) on memory address lines, and Thevenin termination (RN 1, 2) on DRAM control lines. Both of these are used to reduce ringing, something that should only occur if the distance between the VDSC and DRAM is large.

---

[3] See CENELEC PERITELEVISION section

[4] See CENELEC PERITELEVISION section

An 74F244 (IC22) provides buffering for all DRAM control signals and is used to doubly buffer the DRAM strobes. This buffering ensures that there is adequate capacitive drive capability, even with all DRAM sites socketed and all DRAM configurations supported. However, in socketless, single configuration designs this should not be necessary, as the MCD210 was designed to drive DRAM directly.
As mentioned above, layout techniques were also used to improve the DRAM signal integrity. As series termination works more effectively with lumped capacitive loads, each line on the VDSC memory address bus {MA(9:0)} connects to two series resistors. This effectively creates two duplicate address buses {MA1(9:0) and MA2(9:0)}, halving the number of DRAMs per address bus, thus allowing closer grouping of the loads. Again, by placing the VDSC and DRAM close together, this technique should not be necessary.

Finally, please note that the VDSC is designed to work with 512, not 1024, cycle refresh DRAMs such as the 514270.

RF-MODULATOR OUTPUT

The UM1233 RF video modulator (IC46), requires the input levels shown in the following table

Item	Description	Name	Min	Typ	Max	Unit
1	Video Input Level Sync Tip	Vs	2.54	2.6	2.66	V
2	Video Input Level Peak White	Vwh	3.18	3.24	3.3	V
3	Video Input Level Black	Vbk		2.83		V
4	Sync Level Ratio	(Vbk-Vs)/ (Vwh-Vs)	0.3	0.35	0.4	

Table 6. ASTEC UM1233 Specification for Composite Video Signal

Potentiometer R28 makes one half of a potential divider, and so controls the peak-to-peak voltage of the composite video waveform. R29 determines the output voltage of regulator circuit and so can be used to level shift the input. Experimentally, a good quality video output has been achieved by adjusting the R28/ R29 as follows:

i) Adjust R29 until the black level is 2.83V

ii) Adjust R28 until the SYNC tip to peak is 52mV

CENELEC PERITELEVISION

The CENELEC or European "SCART" interface can be used to deliver composite or RGB video to a full specification CENELEC TV. The interface uses two control lines to select the video source. Line 16, BLANKING, selects either the SCART's RGB lines or the TV's amplifier output for display. Line 8, FUNCTION SWITCHING, selects either the TV's antenna input or SCART composite video input to go to the TV's amplifier input. See Figure 3.

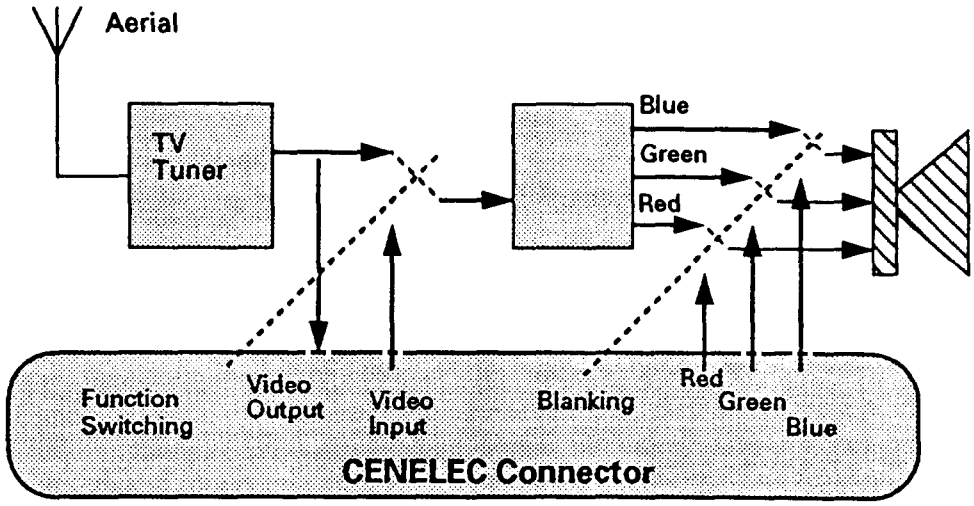


Figure 3. CENELEC TV Switching

The BCV supports three CENELEC configurations which are listed below in Table 7. Configuration C-1 produces a composite video output, useful for either general display or for monitoring the TV encoder output. Configuration C-2 gives a better quality output via the RGB lines. Lastly, C-3 can be used to overlay a BCV image onto a TV broadcast, as discussed earlier. The jumper settings needed for all these configurations are shown in Appendix C, Table 11.

Configuration	Description	SCART P19	Blanking	Function Switching
C-1	Comp Video	Comp Video	Asserted	Negated
C-2	RGB	Comp Sync	Negated	Asserted
C-3	RGB Overlay	Comp Sync	Negated	Negated

Table 7. CENELEC Configuration Options

JUMPERS

Appendix C describes how to configure jumpers for a particular operating mode. Note: pin one of three pin jumpers is denoted by a white dot on the silk screen, as shown in Figure 4.

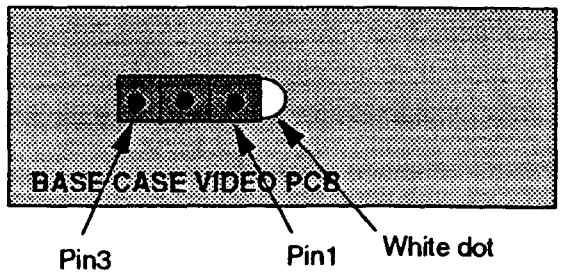


Figure 4. The position of pin one

Connectors

Below is a summary of connectors on the Base Case Video Board. Figure 5 shows the approximate position of off-board connectors.

Connector	Description	Function
P1	96-way Eurocard DIN41612	Connects to CD-i System Backplane Bus (VME P2)
P2	5-pin shielded S-VHS socket	S-VHS (Y/C) video output
P3	21-pin CENELEC video socket	RGB/composite video output
P4	180° 75 ohm bayonet BNC	Video sync input (for test)
P5	90° 75 ohm bayonet BNC	Composite sync output
P6	180° 75 ohm bayonet BNC	Blue video input (for prototyping/test)
P7	90° 75 ohm bayonet BNC	Composite video input (for prototyping)
P8	96-way Eurocard DIN41612	Connects to VME P1 backplane (power only)
P9	64-way IDC socket	BCV to MPEGV cable connector
P10	180° 75 ohm bayonet BNC	Green video input (for prototyping/test)
P11	90° 75 ohm bayonet BNC	Blue video output
P12	180° 75 ohm bayonet BNC	Red video input (for prototyping/test)
P13	90° 75 ohm bayonet BNC	Green video output
P14	90° 75 ohm bayonet BNC	Red video output
P15	90° 75 ohm bayonet BNC	Composite video output
P16	42x 0.1" plated through holes	Digital connector to prototype area
P17	15x 0.1" plated through holes	Analogue connector to prototype area.

Table 8. Connector Summary

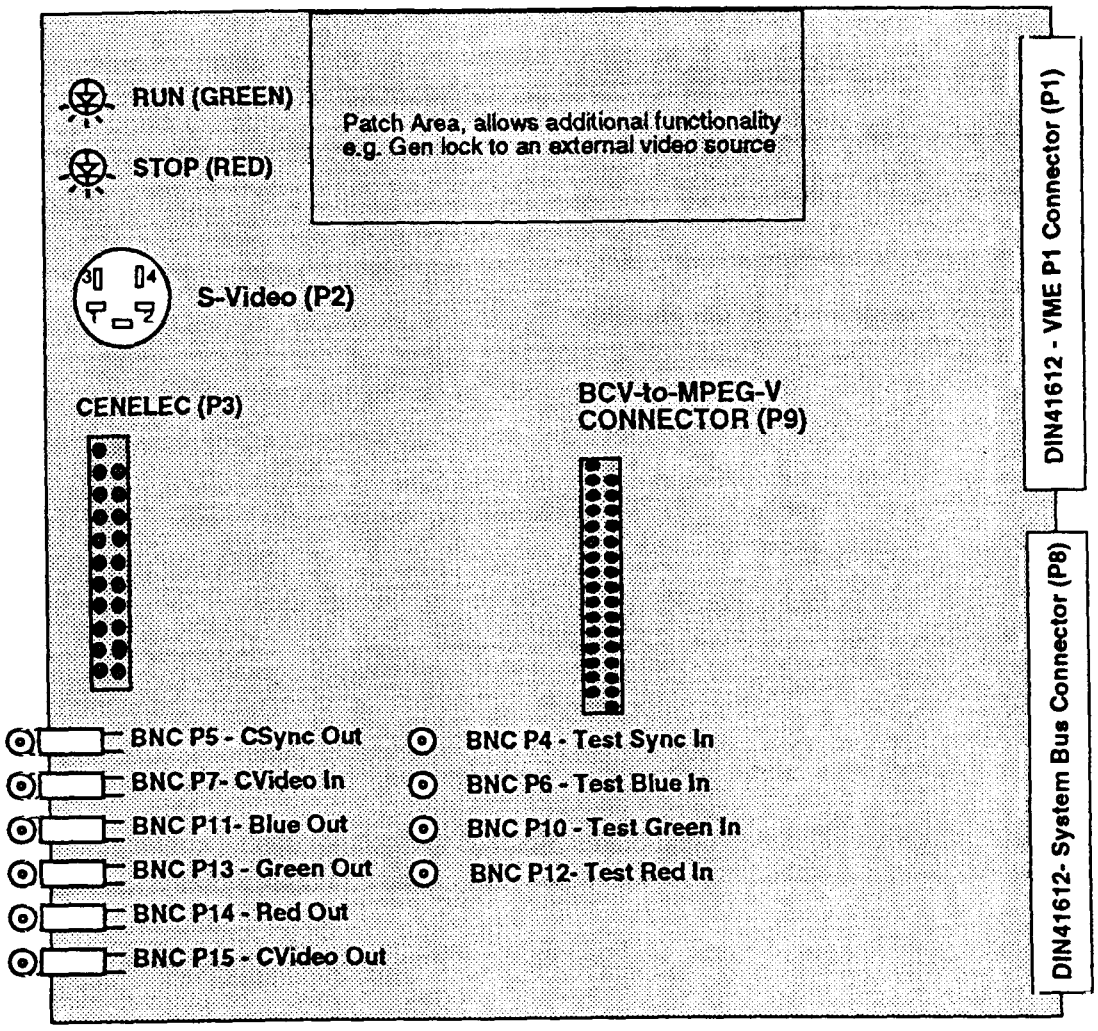


Figure 5. Layout of BCV connectors

APPENDIX A—PAL EQUATIONS FOR THE BCV BOARDS

Appendix A.1 – IC21, PAL16L8-7nS, BCVCTL2.PDS

Appendix A.2 – IC28, PAL16L8-7nS, BUFCTL4.PDS

Appendix A.3 – IC32, PAL22V10-10nS, CLKCTL9.PDS

Appendix A.4 – IC27,PAL20L8-7nS, DECODE7.PDS

Appendix A.5 – IC8, PAL22V10-10nS, IRQENC3.PDS

Appendix A.6 – IC1, PAL22V10-10nS, IRQPRI4.PDS

Appendix A.7 – IC38, PAL20L8-7nS, STBCNV8.PDS

Appendix A.1 – IC21, PAL16L8-7nS, BCVCTL2.PDS

```
TITLE     BCVCTL
PATTERN   BASE-CASE VIDEO OUTPUT CONTROL PAL
REVISION  0.2
AUTHOR    COLIN MAC DONALD
COMPANY   MOTOROLA, EAST KILBRIDE
DATE      25/05/92
CHIP      IC21       PAL16L8

;*********************************************************************
;*********************************************************************


; PINS    1          2          3          4          5
          XT2        /HSYNC_N   /VSYNC_N   /CSYNC_N   /VSD_N

; PINS    6          7          8          9          10
          /VSA_N     /BLANK_N   TTLH1      TTLH2      GND

; PINS    11         12         13         14         15
          TTLH3      /BBLANK_N  /BVSA_N    /BXT2_N    /BVSD_N


; PINS    16         17         18         19         20
          /BCSYNC_N  /BVSYNC_N  /BHSYNC_N  /BXT2      VCC

EQUATIONS

;*********************************************************************
; This PAL operates as a buffer but gives the flexibility to
; change control line signals if required.
;*********************************************************************

BXT2 = XT2

BXT2_N = /XT2

BHSYNC_N = HSYNC_N

BVSYNC_N = VSYNC_N

BCSYNC_N = CSYNC_N

BBLANK_N = BLANK_N

BVSD_N = VSD_N

BVSA_N = VSA_N
```

Appendix A.2 – IC28, PAL16L8-7nS, BUFCTL4.PDS

```
TITLE     BUFCTL4
PATTERN   BASE-CASE VIDEO CLOCK CONTROL PAL
REVISION  0.4
AUTHOR    COLIN MAC DONALD
COMPANY   MOTOROLA, EAST KILBRIDE
DATE      28/01/93
CHIP      IC28       PAL16L8

;*********************************************************************
;*********************************************************************


; PINS    1          2          3          4          5
          OSC_CK1    VC_XT2_N   PROTO4     VC_XT4     /VSA_N

; PINS    6          7          8          9          10
          /VSD_N     OSC_CK2    VCO_CLK    MASTER     GND

; PINS    11         12         13         14         15
          /OE_N      PROTO5     /OUTBUF_N  XT2_A      XT2_B

; PINS    16         17         18         19         20
          CLK_OUT    XT2_OUT    /VC_OE_N   VC_MS      VCC

EQUATIONS
```

```
;*********************************************************************
; HISTORY
; REV 0.1: Original

; REV 0.2: VC_XT2 changed to VC_XT2_N in schematics, changed
;        : PAL equations accordingly.
; REV 0.3: For enabled outputs, VC_OE_N should be high
; REV 0.4: Data buffers should be enabled on the high phase
;        : of XT2 not XT2_N. To meet worst case timing we need
;        : to delay assertion and negation of OUTBUF_N.
;*********************************************************************
; VDSC MASTER/SLAVE SELECT
; The VDSC is configured for MASTER operation if the JUMPER attached
; to P9 is pulled high
;*********************************************************************

VC_MS =  MASTER

;*********************************************************************
; VDSC OUTPUT ENABLE
; The VDSC OE pin is usually driven by VSD, however as the BCV board
; buffers the RGB outputs prior to multiplexing, the OE is permanently
; pulled low.
;*********************************************************************

VC_OE_N = GND

;*********************************************************************
; OUTPUT BUFFER ENABLE
; The RGB outputs are only driven during the high phase of XT2N to
; give sufficient tristate time. The outputs are not driven when VSD_N
; is low
;
; Clocks XT2_A and XT2_B are delayed versions of XT2_OUT used to
; generate correct timing. XT2_OUT is the buffered inverted VC_XT2_N.
;*********************************************************************

XT2_OUT = /VC_XT2_N

XT2_A = XT2_OUT

XT2_B = XT2_A

OUTBUF_N = XT2_A*/VSD_N
         + OUTBUF_N*XT2_B

;*********************************************************************
; CLK_OUT
; For double speed clock module, bufctl is 16r4 and clk_out is
; divide by two, using osc_clk1.
; For single speed bufctl.pds is 16L8
;*********************************************************************

CLK_OUT = /CLK_OUT
```

Appendix A.3 – IC32, PAL22V10-10nS, CLKCTL9.PDS

```
TITLE     CLKCTL10
PATTERN   BASE-CASE VIDEO CLOCK CONTROL PAL
REVISION  1.0
AUTHOR    COLIN MACDONALD / JOHN TRAILL
COMPANY   MOTOROLA, EAST KILBRIDE
DATE      13/01/93
CHIP      IC32       PAL22V10

;*********************************************************************
;*********************************************************************


; PINS    1          2          3          4          5
          SYS_CLK    XT2        SYS_RW     INV_AS     /CPU_RST_N

; PINS    6          7          8          9          10
          /DTACK_N   /BERR_N    PB_RST1    PB_RST2    /IVR_N
```

```
; PINS    11          12          13          14          15
         AS_DELAY    GND         JUMPER      RW          25MDTK

; PINS    16          17          18          19          20
         /SYS_DTK_N  LTCH_DTK    RST_TERM    /WR_TERM_N  /RESET_N

; PINS    21          22          23          24          25
         /SYS_BERR_N OC2         DELAY_CLK   VCC         V10_SET
EQUATIONS

; *********************************************************************
; HISTORY:
; REV 0.1: Original
; REV 0.2: ACS changed to AS
;        : Added 25MDTK term for 25MHz CPU operation
; REV 0.3: Changed PAL to 22V10 to allow more clocked and unclocked
;        : terms. Added DTACK delay for READ cycles
; REV 0.4: Added RESET term to DTACK control terms
; REV 0.5: Fixed DTK equations
; REV 0.6: Include push-button RESET, and control register termination
;        : for write cycles.
; REV 0.7: Assert dtack for IACK cycles. Change some pin positions.
; REV 0.8: Modify DTACK and BERR tristate control
; REV 0.9: PCB Mnfctr put Reset switch back-to-front -> swap PB_RST1
;        : and PB_RST2 to correct.
; REV 1.0: Modified DSACK generation for only 16MHz host processor
;        : operation.
; *********************************************************************


; *********************************************************************
; DELAY CLOCK
;
; The address/data strobe and chip select delaying clock can either
; be the system CPU clock (SYS_CLK) or the VDSC XT2 clock.
; *********************************************************************

DELAY_CLK = SYS_CLK
DELAY_CLK.TRST = VCC


; *********************************************************************
; READ/WRITE CONTROL
;
; Base Case Video RW is simply a buffered version of SYS_RW.
; *********************************************************************

RW = SYS_RW

RW.TRST = VCC


; *********************************************************************
; SYSTEM DTACK
;
; System DSACK1 is a buffered version of DTACK terminated on AS rise.
; Assertion is held until 25MDTK is asserted
; Assertion during READ cycles is delayed until LTCH_DTK is asserted.
; *********************************************************************

SYS_DTK_N = DTACK_N*/RESET_N*INV_AS

SYS_DTK_N.TRST = DTACK_N*/RESET_N


; *********************************************************************
; 25MHZ DTACK
;
; This term stops VDSC dtack terminating the next cycle.
; Asserts when DTACK is negated and AS is asserted. Since this
; term negates SYS_DTACK_N, and 1st line negates with DTACK_N
; need to sustain 'till end of cycle
; *********************************************************************

25MDTK = /DTACK_N*INV_AS*/RESET_N
       + 25MDTK*INV_AS*/RESET_N
25MDTK.TRST = VCC


; *********************************************************************
```

```
; LATCHED DTACK
;
; This term is used to delay the VDSC dtack on READ CYCLEs to ensure
; the microprocessor dtack asserted to data valid time is met. Used
; alone this term cannot guarantee adding a significant delay to an
; asynchronous input. It should therefore be used with sys_clk.
; ***************************************************************************

LTCH_DTK := DTACK_N*/RESET_N
          + IVR_N*/RESET_N
LTCH_DTK.TRST = VCC


; ***************************************************************************
; SYSTEM BUS ERROR
;
; System Bus Error not used in prototype - tri-state
; ***************************************************************************

SYS_BERR_N = BERR_N
SYS_BERR_N.TRST = GND


; ***************************************************************************
; CONTROL REGISTER WRITE TERMINATION
;
; On writes to DUPLSR_REG and IPL_REG, the strobe should negate
; with dtack to ensure valid data is latched into the registers.
; A raw DTACK should be sufficient to accomplish this, but using
; a discrete PAL output gives added flexibility.
; ***************************************************************************

WR_TERM_N = DTACK_N
WR_TERM_N.TRST = VCC


; ***************************************************************************
; BASE CASE VIDEO RESET
;
; RESET is driven either after assertion of SYS_RESET_N or after
; switch S1 is depressed.
; ***************************************************************************

RESET_N = PB_RST1*RST_TERM
        + CPU_RST_N
RESET_N.TRST = VCC


; ***************************************************************************
; RESET TERM
;
; RST_TERM is used to form a debounced, push-button reset
; ***************************************************************************

RST_TERM = /PB_RST2 + RESET_N
RST_TERM.TRST = VCC


; ***************************************************************************
; V10 MACRO CELL RESET
;
; System Bus Error is a buffered version of BERR_N
; ***************************************************************************

V10_SET.SETF =  GND
V10_SET.RSTF =  RESET_N
```

Appendix A.4 – IC27,PAL20L8-7nS, DECODE7.PDS

```
TITLE     DECODE6
PATTERN   BASE-CASE VIDEO DECODE PAL
REVISION  0.6
AUTHOR    COLIN MACDONALD / JOHN TRAILL
COMPANY   MOTOROLA, EAST KILBRIDE
DATE      22/08/92
CHIP      IC27       PAL20L8
```

```
;*****************************************************************
;*****************************************************************

; PINS   1           2           3           4
        /SYS_RST_N   /CSROM_N     RW          /SYS_AS_N
; PINS   5           6           7           8
        IOP_A28      IOP_A29     IOP_A30     A17

; PINS   9           10          11          12
        A18          A19         A20         GND

; PINS   13          14          15          16
        A21          A22         /IRQSTA_N   /WR_TERM_N

;PINS    17          18          19          20
        /IRQ_EN_N    /DUPLSR_N   /PRO_N       /RESET_N

;PINS    21          22          23          24
        IPLREG       VC_ADD      A23         VCC

EQUATIONS

; *******************************************************************
; HISTORY:
; REV 0.1: Original
; REV 0.2: A26-A30 denoted as IOP_A26-IOP_A30
;        : IRQMSK_SEL_N removed, RW added
;        : Moved RW and VC_ADD
; REV 0.3: Ensure output buffers are enabled
;        : Added RESET_N output equation
; REV 0.4: Terminate control register writes on WR_TERM_N
;        : Change polarity of control register strobes.
; REV 0.5: Changed sel address lines from low order
;        : to high order, added proto area chip select.
; REV 0.6: Condition chip selects with AS for 25M operation.
; *******************************************************************


; *******************************************************************
; VDSC ADDRESS DECODE
;
; If a system bus chip select is not used in STBCNV to generate the
; VDSC chip select, then this pal will provide a suitable address.
; Address lines A27-230 are only available from the IOP2 processor
; board, not the CDIP. Their use is therefore optional.
;
; For the CDIP based system the Base Case Video board is located
; at $0080 0000 - $00CF FFFF
; *******************************************************************

VC_ADD = A23*/A22*/A21*/A20*/SYS_RST_N
       + A23*/A22*/A21*A20*/SYS_RST_N
       + A23*/A22*A21*/A20*/SYS_RST_N
       + A23*/A22*A21*A20*/SYS_RST_N
       + A23*A22*/A21*/A20*/SYS_RST_N
VC_ADD.TRST = VCC

; *******************************************************************
; INTERRUPT PRIORITY LEVEL REGISTER      Address $CC 0000 - $CD FFFF
;
; The IPLREG is a 16-bit wide write-only register, which contains
; the IRQ priority level for 3 interrupt sources. This allows
; an IOP2 based system to use backplane counter control lines as
; interrupts. With a CDIP based system this register is not required.
; *******************************************************************

IPLREG = CSROM_N*A19*A18*/A17*/RW*/WR_TERM_N*SYS_AS_N
IPLREG.TRST = VCC


; *******************************************************************
; DUPLICATE STATUS REGISTER              Address $C8 0000 - $C9 FFFF
;
; The DUPLSR register is an 8-bit wide write-only register, which contains
; a copy of the processor status register. This allows
; an IOP2 based system to use backplane counter control lines as
; interrupts. With a CDIP based system this register is not required.
; *******************************************************************
```

```
DUPLSR_N = CSROM_N*A19*/A18*/A17*/RW*/WR_TERM_N*SYS_AS_N*/SYS_RST_N
DUPLSR_N.TRST = VCC

; ****************************************************************
; PROTOTYPE AREA CHIP SELECT              Address $C0 0000 - $C7 FFFF
;
; PRO_N allows selection of 0.5Mbyte (non-read protected) memory
; using VDSC CS_ROM_N.
; ****************************************************************

PRO_N = CSROM_N*/A19*SYS_AS_N*/SYS_RST_N
PRO_N.TRST = VCC


; ****************************************************************
; INTERRUPT STATUS REGISTER               Address $CE 0000 - $CF FDFF
;
; The IRQSTA register is an 8-bit wide read-only register, which
; contains status information on BCV board IRQs. This allows
; an IOP2 based system to use backplane counter control lines as
; interrupts. With a CDIP based system this register is not required.
; ****************************************************************

IRQSTA_N = CSROM_N*A19*A18*A17*/RW*SYS_AS_N*/SYS_RST_N
IRQSTA_N.TRST = VCC


; ****************************************************************
; IRQ ENABLE                              Address $CA 0000 - $CB FFFF
;
; Asserting the IRQ_EN_N line sets a term in IC28 which enables
; interrupts handled by the BCV to be passed through to the system
; backplane.
; ****************************************************************

IRQ_EN_N = CSROM_N*A19*/A18*A17*/RW*SYS_AS_N*/SYS_RST_N
IRQ_EN_N.TRST = VCC


; ****************************************************************
; BASE CASE RESET
;
; Address Buffers will normally be permanently enabled.
; ****************************************************************

RESET_N = SYS_RST_N
RESET_N.TRST = VCC
```

Appendix A.5 – IC8, PAL22V10-10nS, IRQENC3.PDS

```
TITLE     IRQENC3
PATTERN   BASE-CASE VIDEO INTERRUPT ENCODE PAL
REVISION  0.3
AUTHOR    COLIN MAC DONALD
COMPANY   MOTOROLA, EAST KILBRIDE
DATE      06/08/92
CHIP      IC8       PAL22V10

;*****************************************************************
;*****************************************************************


; PINS   1          2          3          4
         TTL_HI     /DUPLSR_N  /IRQVC_N   /IRQB_N

; PINS   5          6          7          8
         /IRQC_N    IPLVC0     IPLVC1     IPLVC2

; PINS   9          10         11         12
         IPLB0      IPLB1      IPLB2      GND

; PINS   13         14         15         16
         IPLC0      IPL2       IPL1       IPL0

;PINS    17         18         19         20
         AGTB       /RESET_N   /BCIRQ_N   BCIRQT_N

;PINS    21         22         23         24
```

```
        DUPL_WR        IPLC1        IPLC2        VCC

;PINS    25
        V10_SET


; ********************************************************************
; HISTORY:
; REV 0.1: Original
; REV 0.2: Added assertion qualifying for DUPLSR updates
;        : Changed to V10 for positive polarity output terms
;        : Moved pins to simplify schematics for 22V10
; REV 0.3: Modified sustain terms of IPL(2:0)
;        : Fixed RESET polarity in pin list
; ********************************************************************
EQUATIONS

; ********************************************************************
; INTERRUPT PRIORITY LINE ZERO
; IPL0 is generated if one of the IRQs with level corresponding to
; IPL0 set is active and DUPL_WR is negated. IPL0 is sustained if one
; of the IRQs with level corresponding to IPL0 is active.
; ********************************************************************

IPL0 =  IRQVC_N*IPLVC0*/IRQB_N*/IRQC_N*/DUPLSR_N*/RESET_N
     +  IRQB_N*IPLB0*/IRQVC_N*/IRQC_N*/DUPLSR_N*/RESET_N
     +  IRQC_N*IPLC0*/IRQVC_N*/IRQB_N*/DUPLSR_N*/RESET_N
     +  IPL0*IRQVC_N*IPLVC0*/IRQB_N*/IRQC_N*/RESET_N
     +  IPL0*IRQB_N*IPLB0*/IRQVC_N*/IRQC_N*/RESET_N
     +  IPL0*IRQC_N*IPLC0*/IRQVC_N*/IRQB_N*/RESET_N


; ********************************************************************
; INTERRUPT PRIORITY LINE ONE
; IPL1 is generated if one of the IRQs with level corresponding to
; IPL1 set is active and DUPL_WR is negated. IPL1 is sustained if one
; of the IRQs with level corresponding to IPL1 is active.
; ********************************************************************

IPL1 =  IRQVC_N*IPLVC1*/IRQB_N*/IRQC_N*/DUPLSR_N*/RESET_N
     +  IRQB_N*IPLB1*/IRQVC_N*/IRQC_N*/DUPLSR_N*/RESET_N
     +  IRQC_N*IPLC1*/IRQVC_N*/IRQB_N*/DUPLSR_N*/RESET_N
     +  IPL1*IRQVC_N*IPLVC1*/IRQB_N*/IRQC_N*/RESET_N
     +  IPL1*IRQB_N*IPLB1*/IRQVC_N*/IRQC_N*/RESET_N
     +  IPL1*IRQC_N*IPLC1*/IRQVC_N*/IRQB_N*/RESET_N

; ********************************************************************
; INTERRUPT PRIORITY LINE 2
; IPL2 is generated if one of the IRQs with level corresponding to
; IPL2 set is active and DUPL_WR is negated. IPL2 is sustained if one
; of the IRQs with level corresponding to IPL2 is active.
; ********************************************************************

IPL2 =  IRQVC_N*IPLVC2*/IRQB_N*/IRQC_N*/DUPLSR_N*/RESET_N
     +  IRQB_N*IPLB2*/IRQVC_N*/IRQC_N*/DUPLSR_N*/RESET_N
     +  IRQC_N*IPLC2*/IRQVC_N*/IRQB_N*/DUPLSR_N*/RESET_N
     +  IPL2*IRQVC_N*IPLVC2*/IRQB_N*/IRQC_N*/RESET_N
     +  IPL2*IRQB_N*IPLB2*/IRQVC_N*/IRQC_N*/RESET_N
     +  IPL2*IRQC_N*IPLC2*/IRQVC_N*/IRQB_N*/RESET_N

; ********************************************************************
; TIMER GATE INTERRUPT
; This output generates a rising edge to form a processor interrupt
; through the Timer TGATE pin. As the 7485 will output an active high
; signal, this output is simply a buffered version of that.
; ********************************************************************

BCIRQT_N = AGTB

; ********************************************************************
; SYSTEM INTERRUPT
; This output generates a falling edge to form a processor interrupt
; through (normally) SYS_IRQA_N.
; ********************************************************************

BCIRQ_N = AGTB

; ********************************************************************
; DUPLICATE STATUS REGISTER WRITE CONTROL
; ********************************************************************
```

```
DUPL_WR = DUPLSR_N

; ********************************************************************
; V10 MACRO CELL RESET
; Enables outputs
; ********************************************************************

V10_SET.SETF =  GND
V10_SET.RSTF =  RESET_N
```

Appendix A.6 – IC1, PAL22V10-10nS, IRQPRI4.PDS

```
TITLE     IRQPRI4
PATTERN   PRIORITISE INTERRUPTS
REVISION  0.4
AUTHOR    COLIN MAC DONALD
COMPANY   MOTOROLA, EAST KILBRIDE
DATE      10/07/92
CHIP      IC1       PAL22V10

;********************************************************************
;********************************************************************


; PINS    1         2          3          4           5
          XT2       /IRQB_N    /IRQC_N    /IRQVC_N    VCGTB

; PINS    6         7          8          9           10
          VCGTC     BGTC       /RESET_N   /IRQ_ENBL_N /IVR_N

; PINS    11        12         13         14          15
          PU1       GND        PU2        IRQC_TERM   IRQB_TERM

; PINS    16        17         18         19          20
          ENBL_TERM /IRQC_O_N  /IRQB_O_N  /IRQVC_O_N  NC1

;PINS     21        22         23         24          25
          NC2       NC3        NC4        VCC         V10_SET

EQUATIONS

; ********************************************************************
; History:
; REV 0.1: Original
; REV 0.2: XT2_N -> XT2 reducing loading on XT2_N
;          PALCOUT and CLRIRQ added.
; REV 0.3: IRQ outputs are not enabled until ENBL_TERM is asserted.
; REV 0.4: Change to 22V10 for flexibility. Add IVR_N to block
;        : changes during IACK cycle.
; ********************************************************************

; ********************************************************************
; IRQB VALIDATION TERM
; IRQB is latched by XT2 clock to ensure interrupt is not caused
; by backplane noise. The validation term is then used to produce
; the IRQB output.
; ********************************************************************

IRQB_TERM := IRQB_N

; ********************************************************************
; IRQC VALIDATION TERM
; IRQC is latched by XT2 clock to ensure interrupt is not caused
; by backplane noise. The validation term is then used to produce
; the IRQC output.
; ********************************************************************

IRQC_TERM := IRQC_N

; ********************************************************************
; IRQB OUTPUT
; An interrupt from SYS_IRQB is propagated if it has a greater priority
; than any active IRQ.
; ********************************************************************

IRQB_O_N = IRQB_N*IRQB_TERM*/VCGTB*BGTC*ENBL_TERM*/IVR_N*/RESET_N
         + IRQB_N*IRQB_TERM*/IRQVC_O_N*BGTC*ENBL_TERM*/IVR_N*/RESET_N
         + IRQB_N*IRQB_TERM*/IRQC_O_N*/VCGTB*ENBL_TERM*/IVR_N*/RESET_N
         + IRQB_O_N*IRQB_N*IRQB_TERM*/VCGTB*BGTC*ENBL_TERM*/RESET_N
```

```
        + IRQB_O_N*IRQB_N*IRQB_TERM*/IRQVC_O_N*BGTC*ENBL_TERM*/RESET_N
        + IRQB_O_N*IRQB_N*IRQB_TERM*/IRQC_O_N*/VCGTB*ENBL_TERM*/RESET_N

; ********************************************************************
; IRQC OUTPUT
; An interrupt from SYS_IRQC is propagated if it has a greater priority
; than any active IRQ.
; ********************************************************************

IRQC_O_N = IRQC_N*IRQC_TERM*/VCGTC*/BGTC*ENBL_TERM*/IVR_N*/RESET_N
         + IRQC_N*IRQC_TERM*/IRQVC_O_N*/BGTC*ENBL_TERM*/IVR_N*/RESET_N
         + IRQC_N*IRQC_TERM*/IRQB_O_N*/VCGTC*ENBL_TERM*/IVR_N*/RESET_N
         + IRQC_O_N*IRQC_N*IRQC_TERM*/VCGTC*/BGTC*ENBL_TERM*/RESET_N
         + IRQC_O_N*IRQC_N*IRQC_TERM*/IRQVC_O_N*/BGTC*ENBL_TERM*/RESET_N
         + IRQC_O_N*IRQC_N*IRQC_TERM*/IRQB_O_N*/VCGTC*ENBL_TERM*/RESET_N

; ********************************************************************
; IRQVC OUTPUT
; An interrupt from the  VDSC  is propagated if it has a greater priority
; than any active IRQ.
; ********************************************************************

IRQVC_O_N = IRQVC_N*VCGTB*VCGTC*ENBL_TERM*/IVR_N*/RESET_N
          + IRQVC_N*/IRQC_O_N*VCGTB*ENBL_TERM*/IVR_N*/RESET_N
          + IRQVC_N*/IRQB_O_N*VCGTC*ENBL_TERM*/IVR_N*/RESET_N
          + IRQVC_O_N*IRQVC_N*VCGTB*VCGTC*ENBL_TERM*/RESET_N
          + IRQVC_O_N*IRQVC_N*/IRQC_O_N*VCGTB*ENBL_TERM*/RESET_N
          + IRQVC_O_N*IRQVC_N*/IRQB_O_N*VCGTC*ENBL_TERM*/RESET_N

; ********************************************************************
; ENBL_TERM
; The ENBL_TERM is set, by writing to the IRQEN_REG address in VDSC CSIO
; address space (thus asserting IRQ_ENBL_N). This is written once after
; the IPL_REG and DUPLSR_REG have been initialised, shortly after RESET.
; Once set, this term is asserted until RESET is driven low. This
; function is required to prevent interrupts being generated before the
; BCV interrupt vector register has been initialised giving non-valid
; interrupt vectors.
; ********************************************************************

ENBL_TERM := IRQ_ENBL_N*/RESET_N
           + ENBL_TERM*/RESET_N


; ********************************************************************
; V10 MACRO CELL RESET
; System Bus Error is a buffered version of BERR_N
; ********************************************************************

V10_SET.SETF =  GND
V10_SET.RSTF =  RESET_N
```

Appendix A.7 – IC38, PAL20L8-7nS, STBCNV8.PDS

```
TITLE     STBCNV9
PATTERN   BASE-CASE VIDEO STROBE CONVERSION PAL
REVISION  0.9
AUTHOR    COLIN MACDONALD / JOHN TRAILL
COMPANY   MOTOROLA, EAST KILBRIDE
DATE      16/10/92
CHIP      IC38       PAL20L8

;*****************************************************************


; PINS   1          2           3           4
        /SYS_AS_N   /SYS_AS68_N  /SYS_DS_N   /SYS_UDS_N

; PINS   5          6           7           8
        /SYS_LDS_N  SYS_SIZ0     SYS_SIZ1    SYS_A0

; PINS   9          10          11          12
        RW          DS_DELAY     AS_DELAY    GND

; PINS   13         14          15          16
        VC_ADD     /RESET_N     DBUFF_DIR   /VC_CS_N

;PINS   17         18          19          20
        /LDS_N      /UDS_N      /DBUFF_EN_N  INV_DS
```

```
;PINS    21          22          23          24
        INV_AS      ADD0        NODELAY     VCC

; ********************************************************************
; HISTORY:
; REV 0.1:  Original. P2 Backplane with chip selects
;       :   A0 input from f244 buffer.
; REV 0.2:  Backplane with additional IRQ's no chip selects
;       :   A0 input from SYS.
; REV 0.3:  Simulation problems - ensure outputs are enabled.
; REV 0.4:  Fix INV_DS_N comment error
; REV 0.5:  Fix polarity of VDSC CS
;       :   Fix BUFFEN_N vs BUFDIR
; REV 0.6:  No UDS,LDS delay, selected by NODELAY input from jumper
;       :   connected to PIN23.
; REV 0.7:  IOP2 version - SYS_UDS_N = IACK, include in DBUFF equations.
; REV 0.8:  CDIP version - SYS_UDS_N = SYS_UDS_N!
; ********************************************************************

EQUATIONS

; ********************************************************************
; Data Buffers Enable - IOP2 version (now saved as version 0.7)
; ********************************************************************
;
;DBUFF_EN_N    = VC_ADD*SYS_AS_N*/NODELAY*AS_DELAY*/RESET_N
;              + VC_ADD*SYS_AS_N*NODELAY*/RESET_N
;              + SYS_UDS_N*SYS_AS_N*/NODELAY*AS_DELAY*/RESET_N
;              + SYS_UDS_N*SYS_AS_N*NODELAY*/RESET_N
;              + DBUFF_EN_N*SYS_DS_N*/RESET_N        ; Sustain with DS

; ********************************************************************
; Data Buffers Enable
; Databus buffers are enabled after SYS_AS_N and sustained
; with SYS_DS_N. If NODELAY is negated then output is dependent on
; AS_DELAY.
; ********************************************************************
;
DBUFF_EN_N    = VC_ADD*SYS_AS_N*/NODELAY*AS_DELAY*/RESET_N
              + VC_ADD*SYS_AS_N*NODELAY*/RESET_N
              + DBUFF_EN_N*SYS_DS_N*/RESET_N        ; Sustain with DS

; ********************************************************************
; Inverse Address Strobe
; If a backplane chip select IS NOT used for decode of this board, then
; INV_AS_N is EITHER the inverse of SYS_AS_N or of SYS_AS68_N.
; INV_AS_N is used to combat backplane noise. It is used to delay
; the address validation signal until signals are stable.
; ********************************************************************

INV_AS = SYS_AS_N*/RESET_N          ; Inverse of System AS_N


; ********************************************************************
; Inverse Data Strobe
; INV_DS_N is used to combat backplane noise. It is used to delay
; the data validation signal until signals are stable.
; If 68000 processor signals are to be used to access this board
; INV_DS_N is programmed to be the inverse or SYS_UDS_N and SYS_LDS_N.
; If 68340 processor signals are to be used to access this board
; INV_DS_N is programmed to be the inverse or SYS_DS_N.
; Delayed INV_DS until CS is valid to prevent potential timing
; violation.
; ********************************************************************

INV_DS  = SYS_DS_N*VC_CS_N*/RESET_N          ; 68340 access

;INV_DS_N = SYS_LDS_N*/RESET_N
;         + SYS_UDS_N*/RESET_N         ; 68000 access


; ********************************************************************
; Upper Data Strobe
; If 68000 processor signals are to be used to access this board
; UDS_N is created from SYS_UDS_N
; If 68340 processor signals are to be used to access this board
; UDS_N is created from SYS_DS, SYS_A0, SYS_SIZ0 and SYS_SIZ1.
;
; UDS_N = SIZ1*DS*/RESET      (word aligned)
;       + /SIZ1*/A0*DS*/RESET ( even byte & long word aligned)
```

```
;
; If NODELAY is negated, UDS assertion is delayed by DS_DELAY
; whose timing is selectable via jumpers. If NODELAY is asserted,
; UDS is independent of DS_DELAY.
; ********************************************************************

UDS_N =  SYS_SIZ1*SYS_DS_N*/NODELAY*DS_DELAY*/RESET_N      ; 68340 signal access
      + /SYS_SIZ1*SYS_DS_N*/NODELAY*DS_DELAY*/SYS_A0*/RESET_N
      +  SYS_SIZ1*SYS_DS_N*NODELAY*/RESET_N               ; 68340 signal access
      + /SYS_SIZ1*SYS_DS_N*NODELAY*/SYS_A0*/RESET_N

;UDS_N = SYS_UDS_N*/NODELAY*DS_DELAY*/RESET_N             ; 68000 signal access
;      + SYS_UDS_N*NODELAY*/RESET_N                       ; 68000 signal access


; ********************************************************************
; Lower Data Strobe
; If 68000 processor signals are to be used to access this board
; LDS_N is created from SYS_LDS_N
; If 68340 processor signals are to be used to access this board
; LDS_N is created from SYS_DS, SYS_A0, SYS_SIZ0 and SYS_SIZ1.
;
; LDS_N = SIZ0*A0*DS*/RESET     (odd byte)
;       + /SIZ0*DS*/RESET       (word & long word aligned)
;
; If NODELAY is negated, UDS assertion is delayed by DS_DELAY
; whose timing is selectable via jumpers. If NODELAY is asserted,
; UDS is independent of DS_DELAY.
; ********************************************************************

LDS_N =   SYS_SIZ0*SYS_DS_N*/NODELAY*DS_DELAY*SYS_A0*/RESET_N  ; 68340 signal access
      +  /SYS_SIZ0*SYS_DS_N*/NODELAY*DS_DELAY*/RESET_N         ;
      +   SYS_SIZ0*SYS_DS_N*NODELAY*SYS_A0*/RESET_N           ; 68340 signal access
      +  /SYS_SIZ0*SYS_DS_N*NODELAY*/RESET_N                  ;

;LDS_N = SYS_LDS_N*/NODELAY*DS_DELAY*/RESET_N               ; 68000 signal access
;LDS_N = SYS_LDS_N*NODELAY*/RESET_N                         ; 68000 signal access

; ********************************************************************
; VDSC Chip Select
; The VDSC CS not only allows access to the video and system
; DRAM, it also allows selection of control registers through
; the VDSC's CSIOn. The VDSC CS is derived from a decoded address
; signal from IC30. The VDSC CS can use either a 68000 AS or a 68340 AS.
;
; If NODELAY is negated, VDSC CS assertion is delayed by AS_DELAY
; whose timing is selectable via jumpers. If NODELAY is asserted,
; UDS is independent of AS_DELAY.
; ********************************************************************

VC_CS_N = VC_ADD*SYS_AS_N*/NODELAY*AS_DELAY*/RESET_N       ; 68340 AS
        + VC_ADD*SYS_AS_N*NODELAY*/RESET_N                 ; 68340 AS

;VC_CS_N = VC_ADD*SYS_AS68_N*/NODELAY*AS_DELAY*/RESET_N    ; 68000 AS
;        + VC_ADD*SYS_AS68_N*/NODELAY*/RESET_N             ; 68000 AS

; ********************************************************************
; Data Buffer Direction Control
; Data bus dir asserts (active high) during write cycles. DBUFF_EN_N
; must therefore be disabled except during
; ********************************************************************

DBUFF_DIR = /RW*/RESET_N

; ********************************************************************
; ADDRESS LINE ZERO
;; Data bus buffers can be enabled for all writes but only on board
; decode for reads.
; ********************************************************************

ADD0 = SYS_A0
```

Appendix B—Schematic Diagrams of the BCV Board

Appendix B.1 TOP LEVEL BLOCK-1, Top level block diagram

Appendix B.2 DIGITAL VIDEO, Digital video module including MCD210

Appendix B.3 DRAM, DRAM module

Appendix B.4 INTERFACE, System interface

Appendix B.5 INTERRUPT, Interrupt control circuitry

Appendix B.6 ANALOGUE VIDEO, Analogue circuitry for video input/output

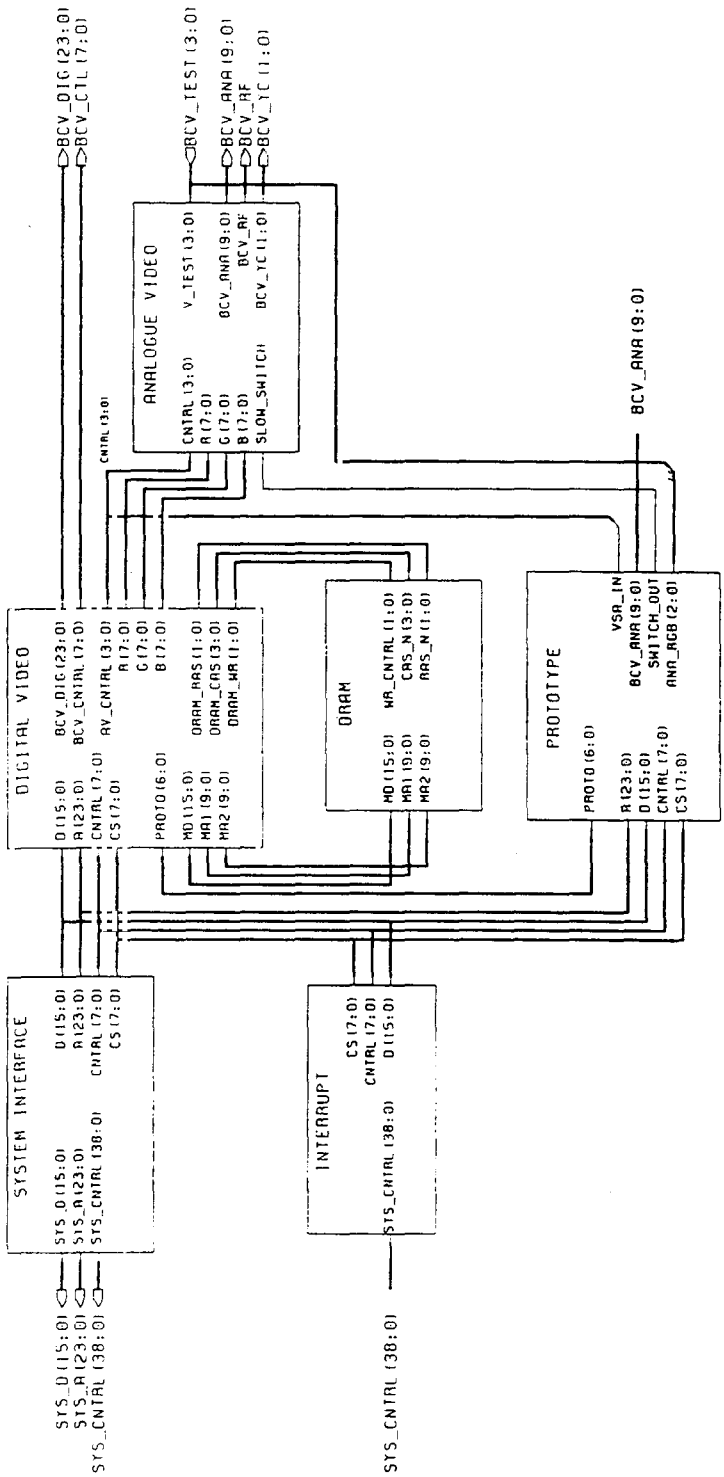
Appendix B.7 PROTOTYPE, Mixed signal prototype

Appendix B.8 TOP LEVEL BLOCK-2, System bus connector

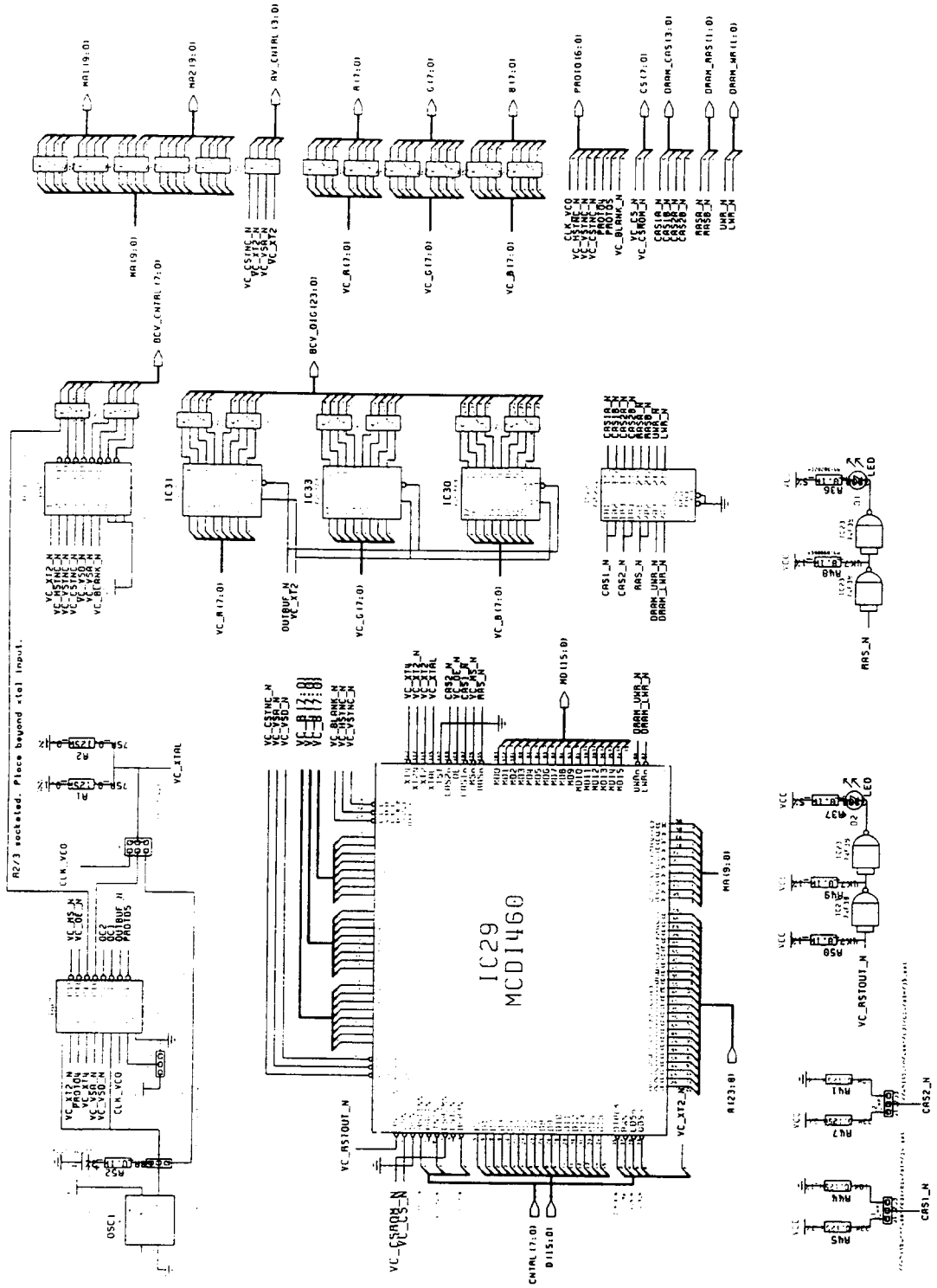
Appendix B.9 TOP LEVEL BLOCK-3, Video connectors including MPEG

Appendix B.10 TOP LEVEL BLOCK- 4, VME P1 connector

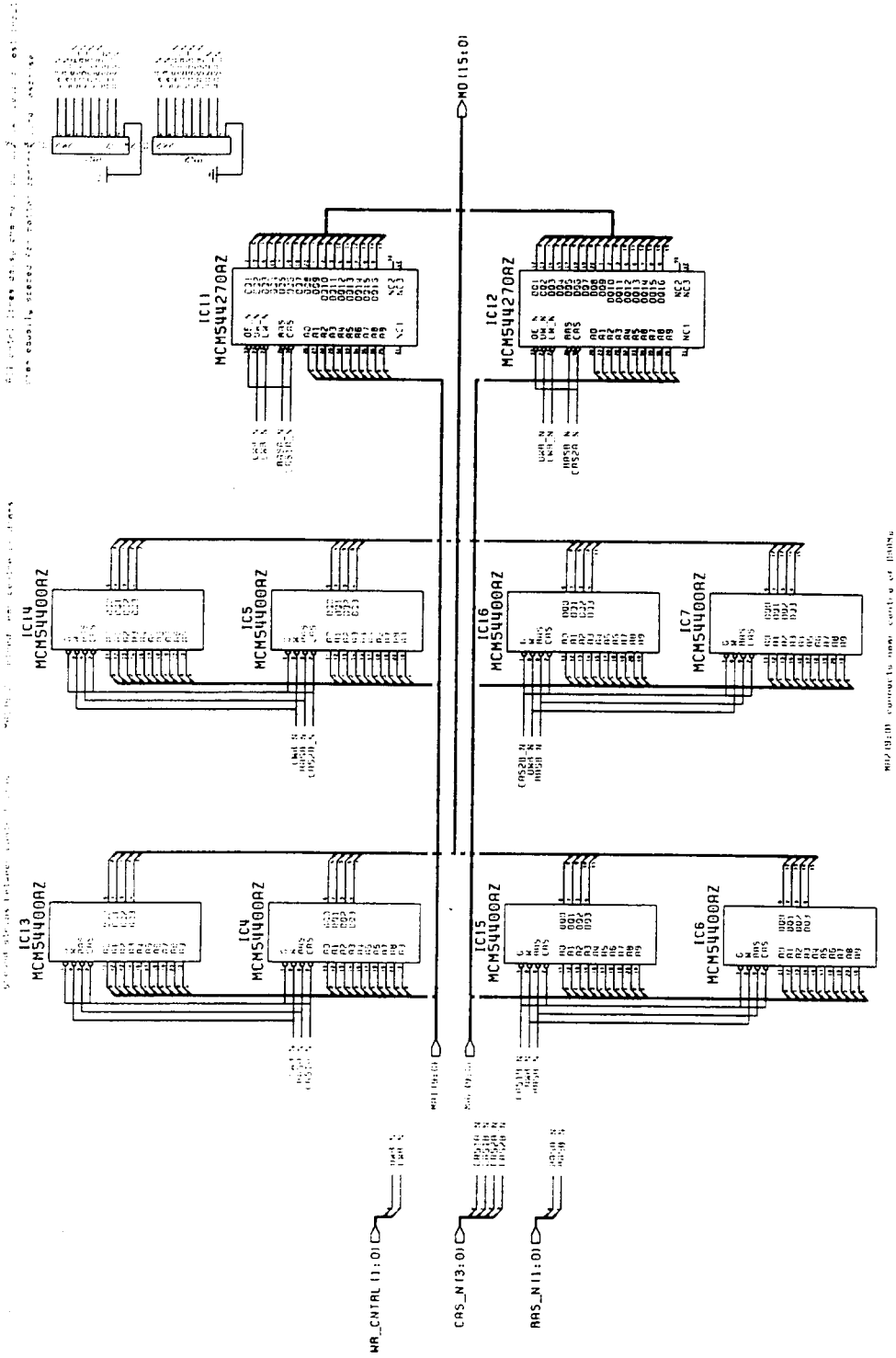
Appendix B.11 TOP LEVEL BLOCK- 5, Decoupling capacitors



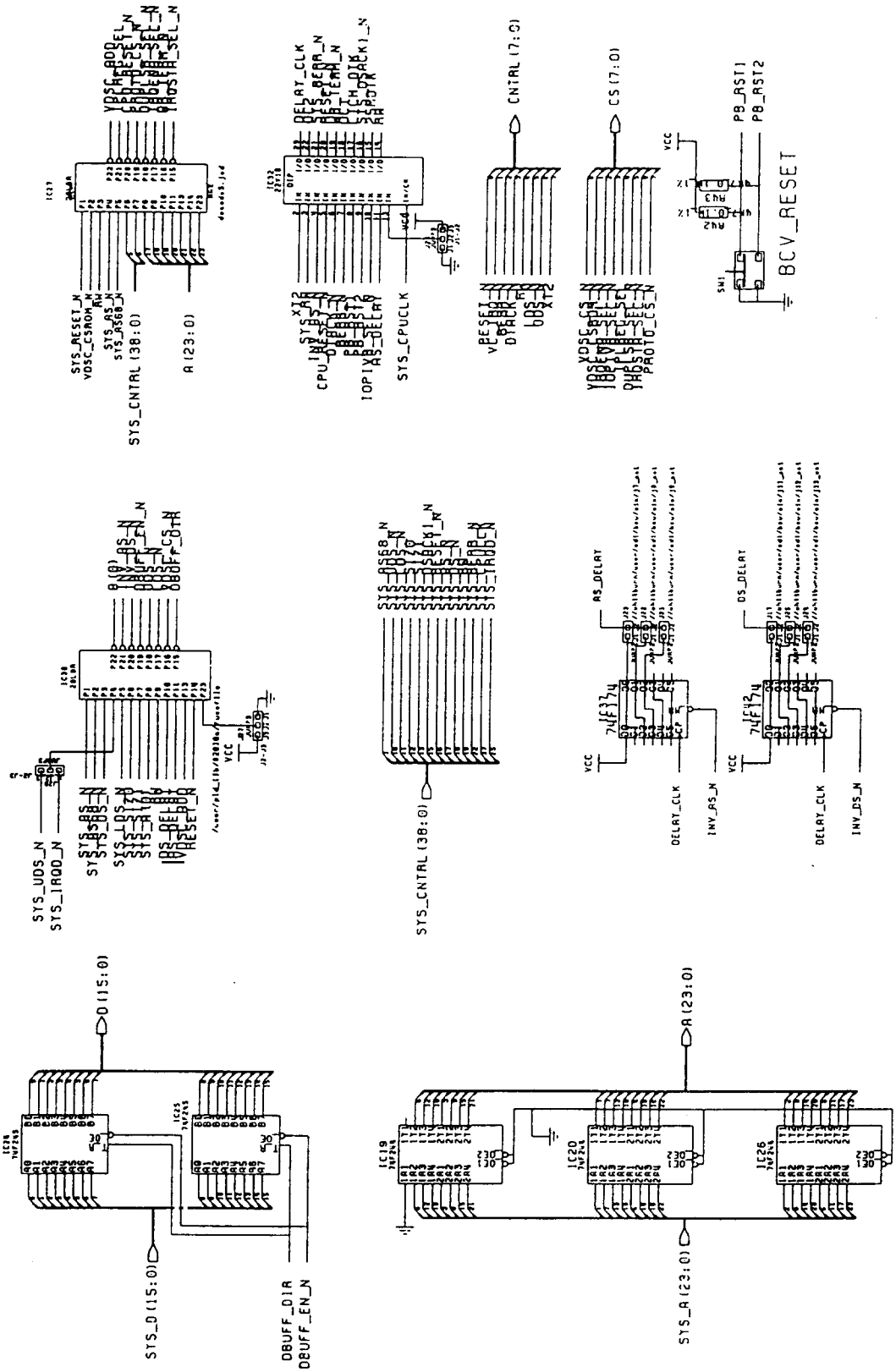
Appendix B.1 TOP LEVEL BLOCK-1, Top level block diagram



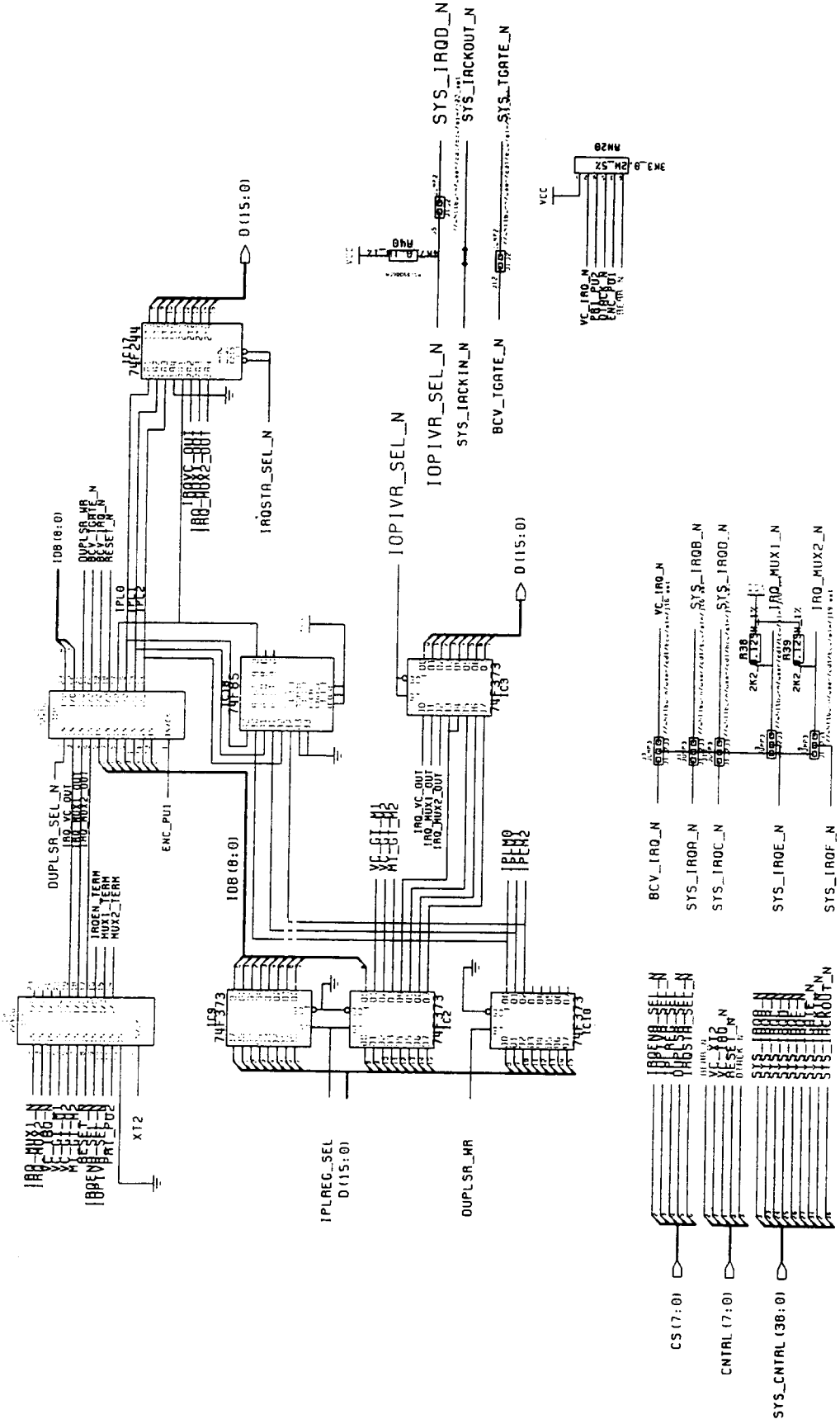
Appendix B.2 DIGITAL VIDEO, Digital video module including MCD210



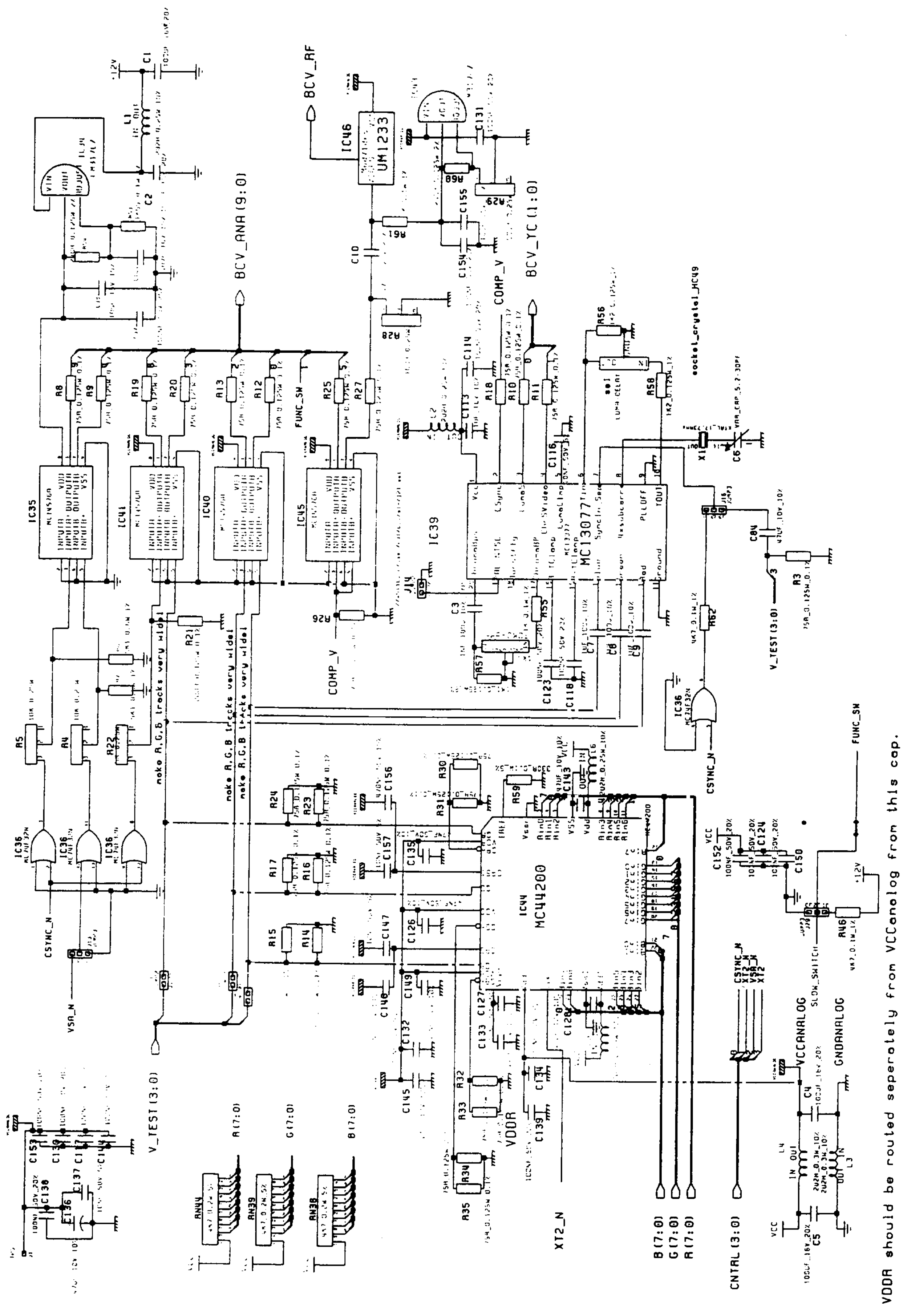
Appendix B.3 DRAM, DRAM module



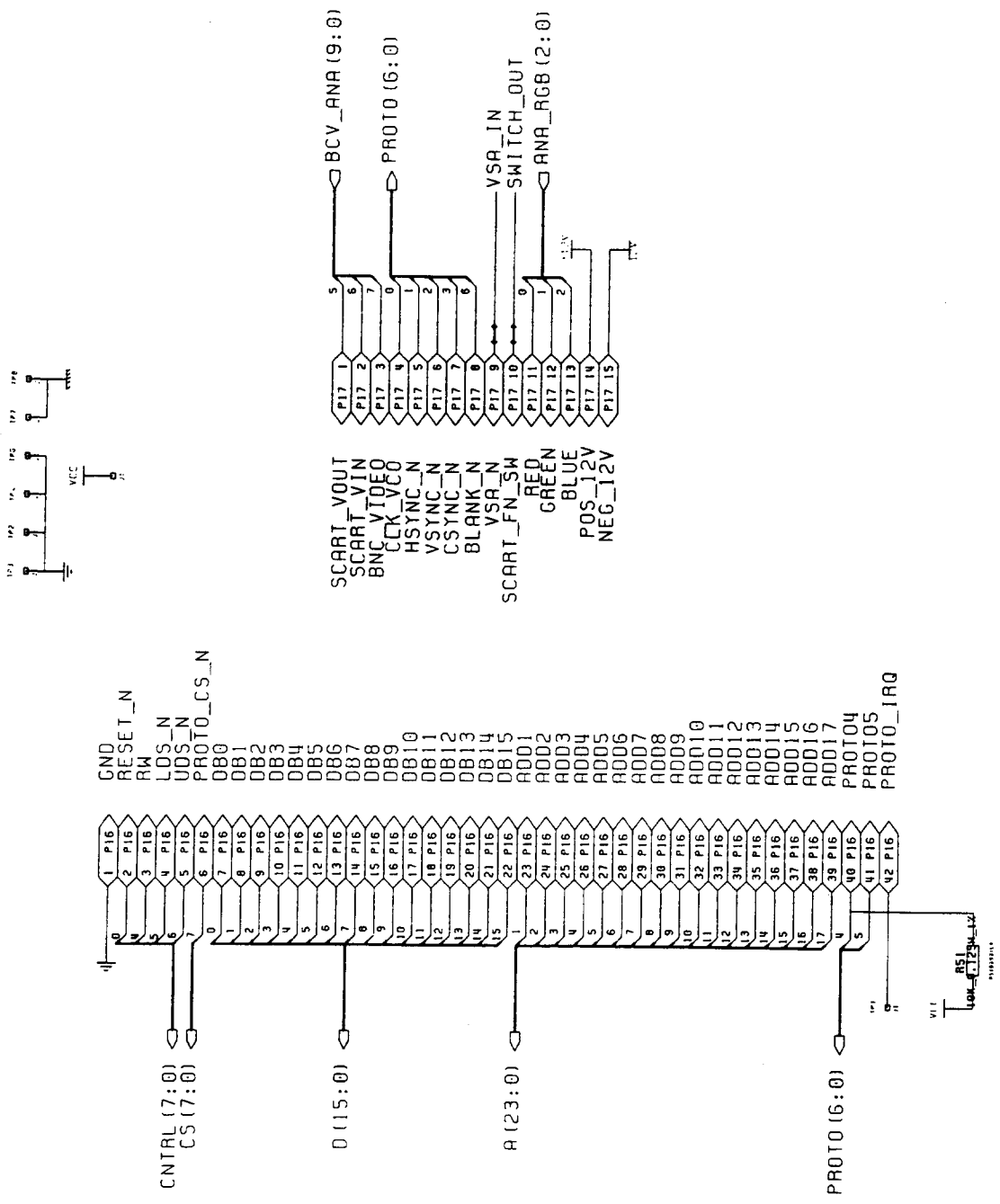
Appendix B.4 INTERFACE, System interface



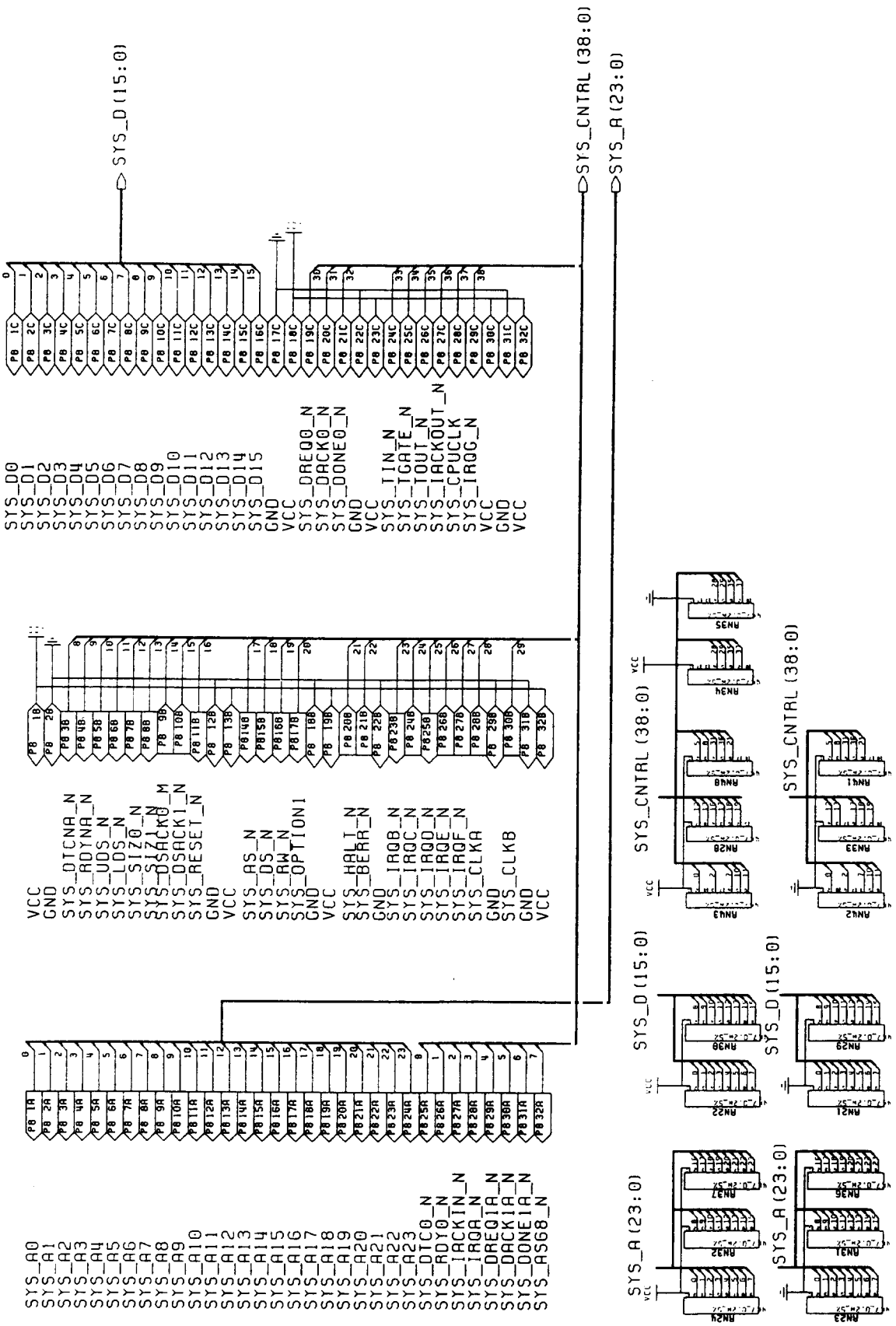
Appendix B.5 INTERRUPT, Interrupt control circuitry



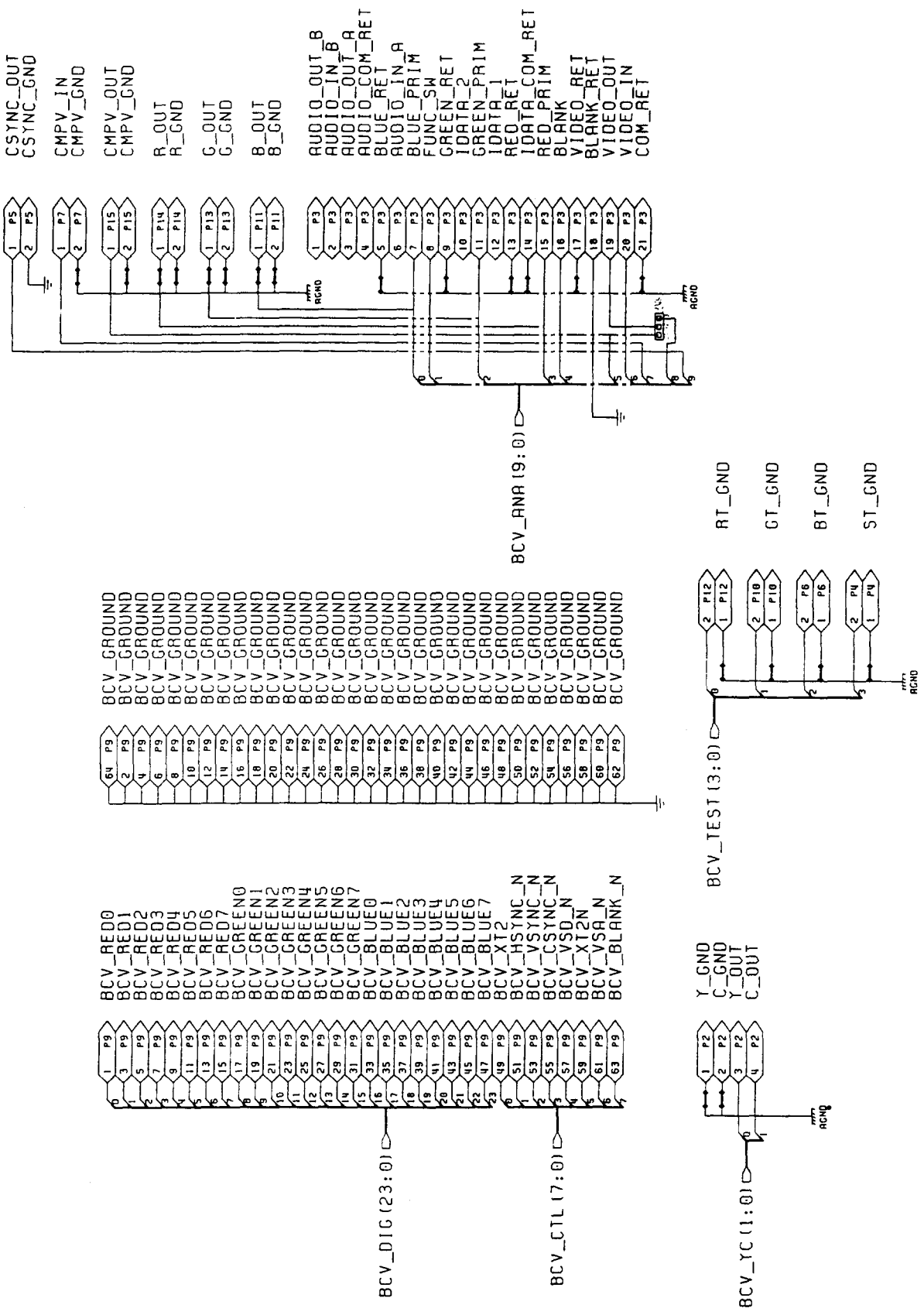
Appendix B.6 ANALOGUE VIDEO, Analogue circuitry for video input/output



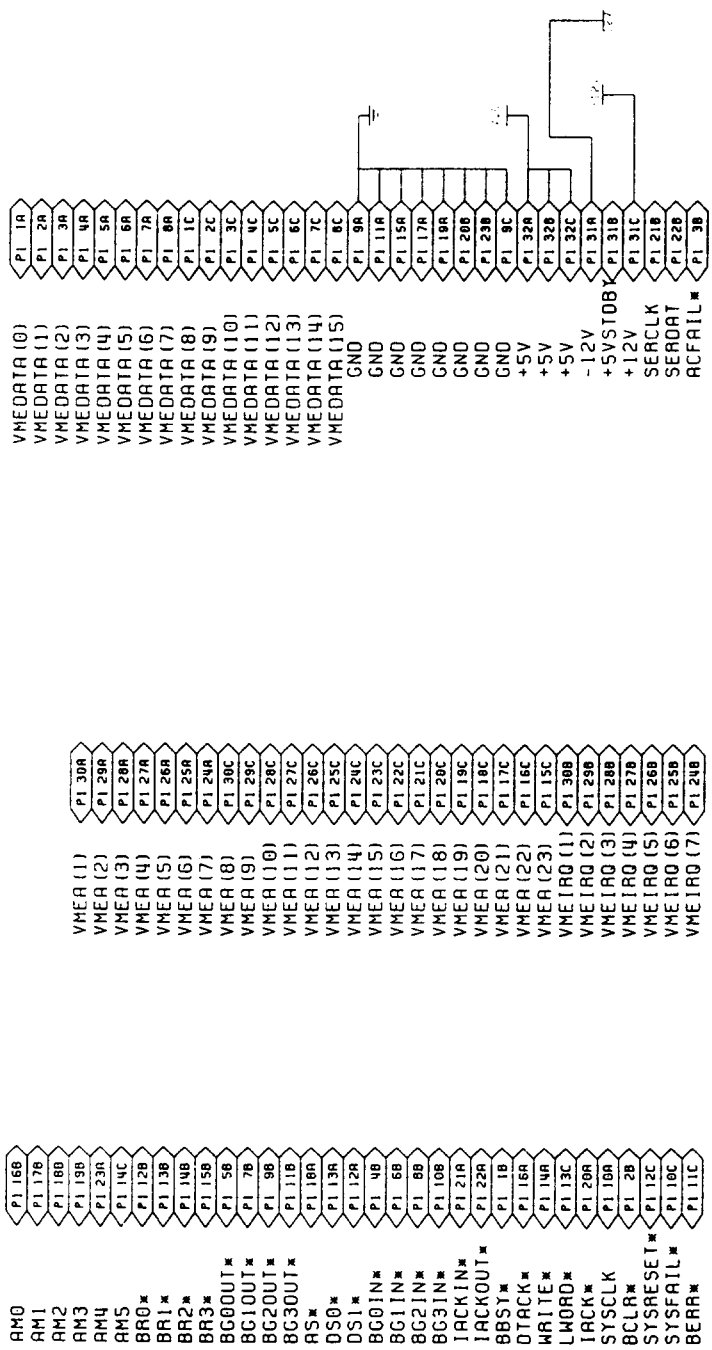
Appendix B.7 PROTOTYPE, Mixed signal prototype



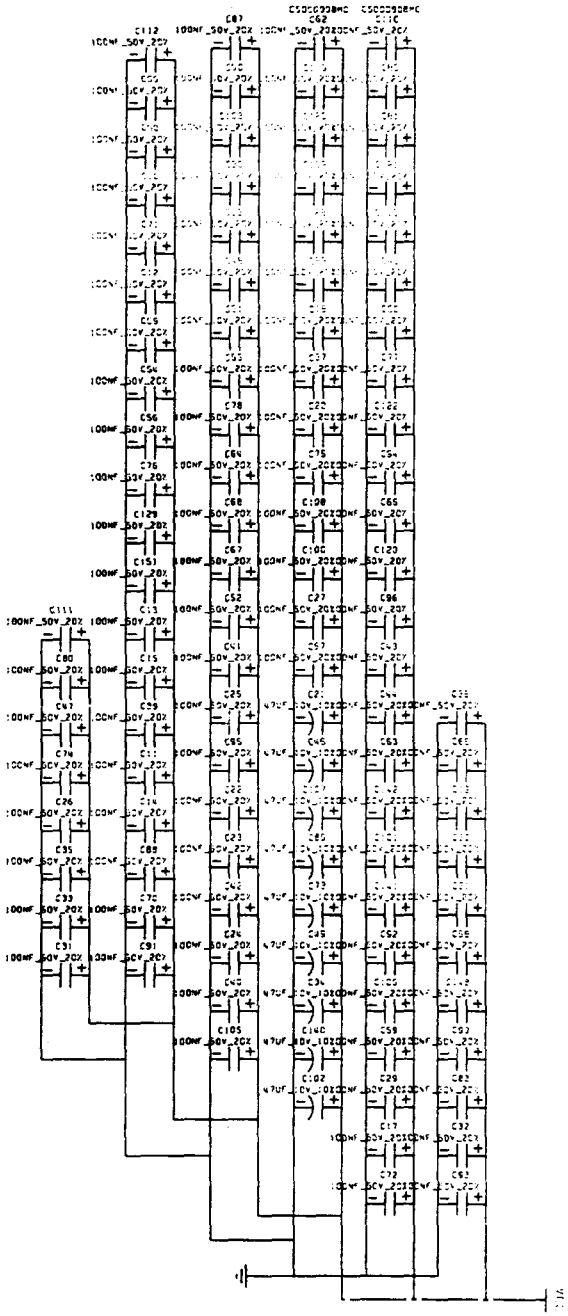
Appendix B.8 TOP LEVEL BLOCK-2, System bus connector



Appendix B.9 TOP LEVEL BLOCK-3, Video connectors including MPEG



Appendix B.10 TOP LEVEL BLOCK- 4, VME P1 connector



Appendix B.11 TOP LEVEL BLOCK- 5, Decoupling capacitors

APPENDIX C—Tables of jumper settings for the BCV board

Function	J6	J4	J7	J1	J2	J23	J22	J21	J27	J17	J25	J24	J28
Direct osc. drive	2-3	-	5-6	-	-	-	-	-	-	-	-	-	-
Devide/2 osc drive	1-2	-	3-4	-	-	-	-	-	-	-	-	-	-
Proto osc drive	-	-	1-2	-	-	-	-	-	-	-	-	-	-
Validate bank1 DRAM	-	-	-	1-2	-	-	-	-	-	-	-	-	-
In-validate bank1 DRAM	-	-	-	2-3	-	-	-	-	-	-	-	-	-
Validate bank2 DRAM.	-	-	-	-	1-2	-	-	-	-	-	-	-	-
In-validate bank2 DRAM	-	-	-	-	2-3	-	-	-	-	-	-	-	-
No AS/DS delay	-	-	-	-	-	√	x	x	1-2	√	x	x	-
Delay AS by 1 clock (max)	-	-	-	-	-	√	x	x	2-3	-	-	-	-
Delay AS by 1-2 clocks	-	-	-	-	-	x	√	x	2-3	-	-	-	-
Delay AS by 2-3 clocks	-	-	-	-	-	x	x	√	2-3	-	-	-	-
Delay DS by 1 clock (max)	-	-	-	-	-	-	-	-	2-3	√	x	x	-
Delay DS by 1-2 clocks	-	-	-	-	-	-	-	-	2-3	x	√	x	-
Delay DS by 2-3 clocks	-	-	-	-	-	-	-	-	2-3	x	x	√	-
CD-i System Bus based	-	-	-	-	-	-	-	-	-	-	-	-	2-3

NB: "x-y" means connect pins x & y of jumper. "!x-y" means don't connect pins x & y of jumper ."√" means connect. "x" means do not connect. "-" means don't care. "!y" means don't connect pin y of jumper.

Table 9. General Configuration

Function	J3	J12	J11	J10	J9	J8	J5	J28
IRQ output on IRQA	2-3	x	1-2	x	x	x	x	2-3
IRQ output on IRQB	2-3	x	2-3	x	x	x	x	2-3
IRQ output on IRQC	2-3	x	x	1-2	x	x	x	2-3
IRQ output on IRQD	2-3	x	x	2-3	x	x	x	2-3
IRQ output on IRQE	2-3	x	x	x	1-2	x	x	2-3
IRQ output on IRQF	2-3	x	x	x	x	1-2	x	2-3

Table 10. Interrupt Configuration

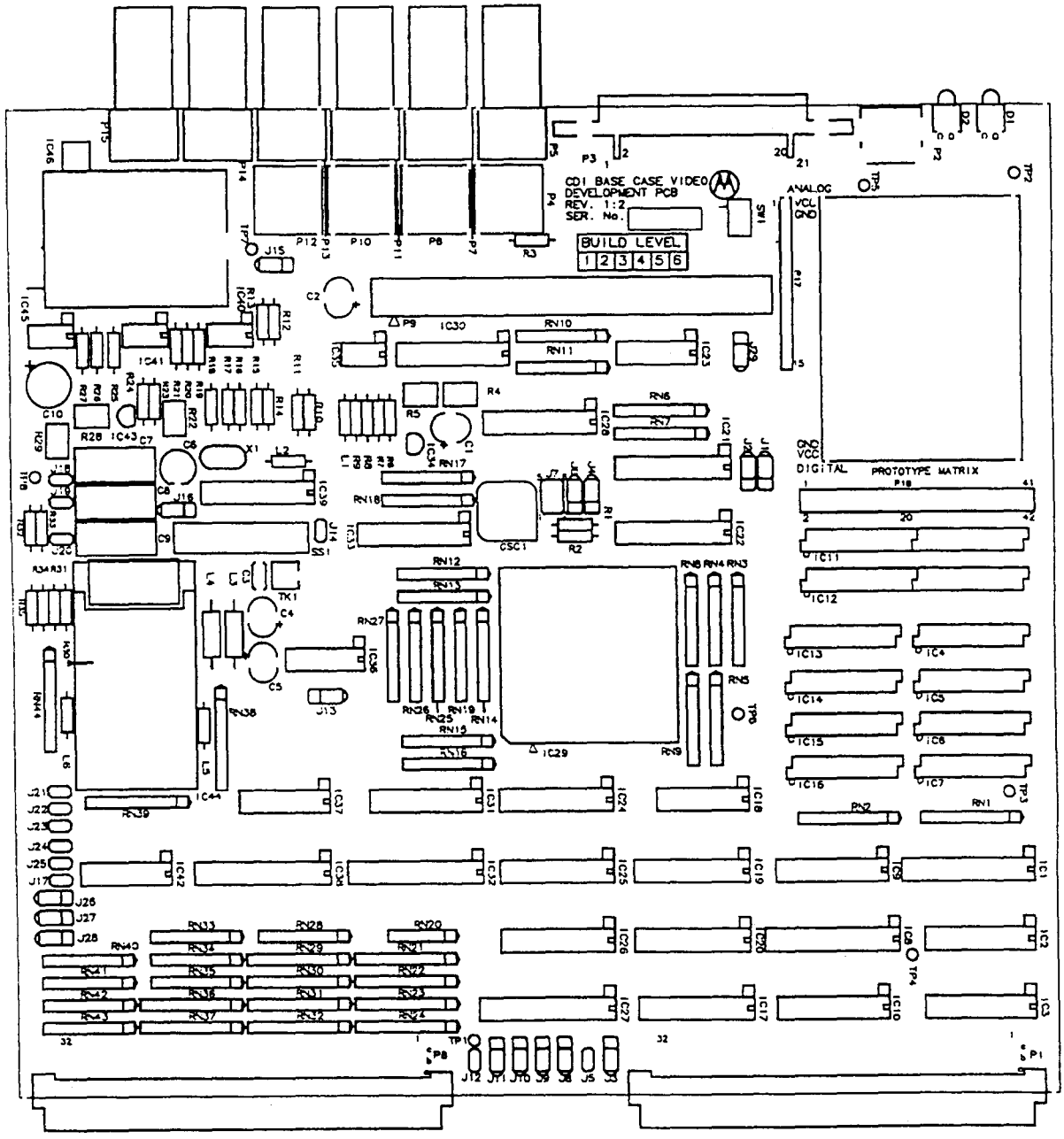
Function	J3	J12	J11	J10	J9	J8	J5	J28
IRQ input on IRQE (IOP only)	1-2	-	x	!3	2-3	!1	-	!3
IRQ input on IRQF (IOP only)	1-2	-	x	!3	!1	2-3	-	!3
IRQ output on IRQC (IOP only)	1-2	x	x	1-2	!1	!1	-	1-2
IRQ output on TGATE (IOP only)	1-2	√	x	x	!1	!1	x	x
IACK support (IOP only)	1-2	x	-	1-2	!1	!1	√	1-2
NB: "x-y" means connect pins x & y of jumper. "!x-y" means don't connect pins x & y of jumper ."√" means connect. "x" means do not connect. "-" means don't care. "!y" means don't connect pin y of jumper.								

Table 10. Interrupt Configuration

FUNCTION	J13	J14	J15	J16	J18	J19	J20	J29
BCV PAL Video	-	x	-	-	x	x	x	-
BCV NTSC Video	-	√	-	x	x	x	x	-
Test Video	2-3	-	-	√	√	√	√	-
SCART P19 = Composite video (C-2)	2-3	-	1-2	-	-	-	-	1-2
SCART P19 = Sync, RGB video (C-1)	1-2	-	2-3	-	-	-	-	1-2
SCART P19 =Sync, RGB overlay (C-3)	1-2	-	2-3	-	-	-	-	2-3
NB: "x-y" means connect pins x & y of jumper. "!x-y" means don't connect pins x & y of jumper ."√" means connect. "x" means do not connect. "-" means don't care. "!y" means don't connect pin y of jumper.								

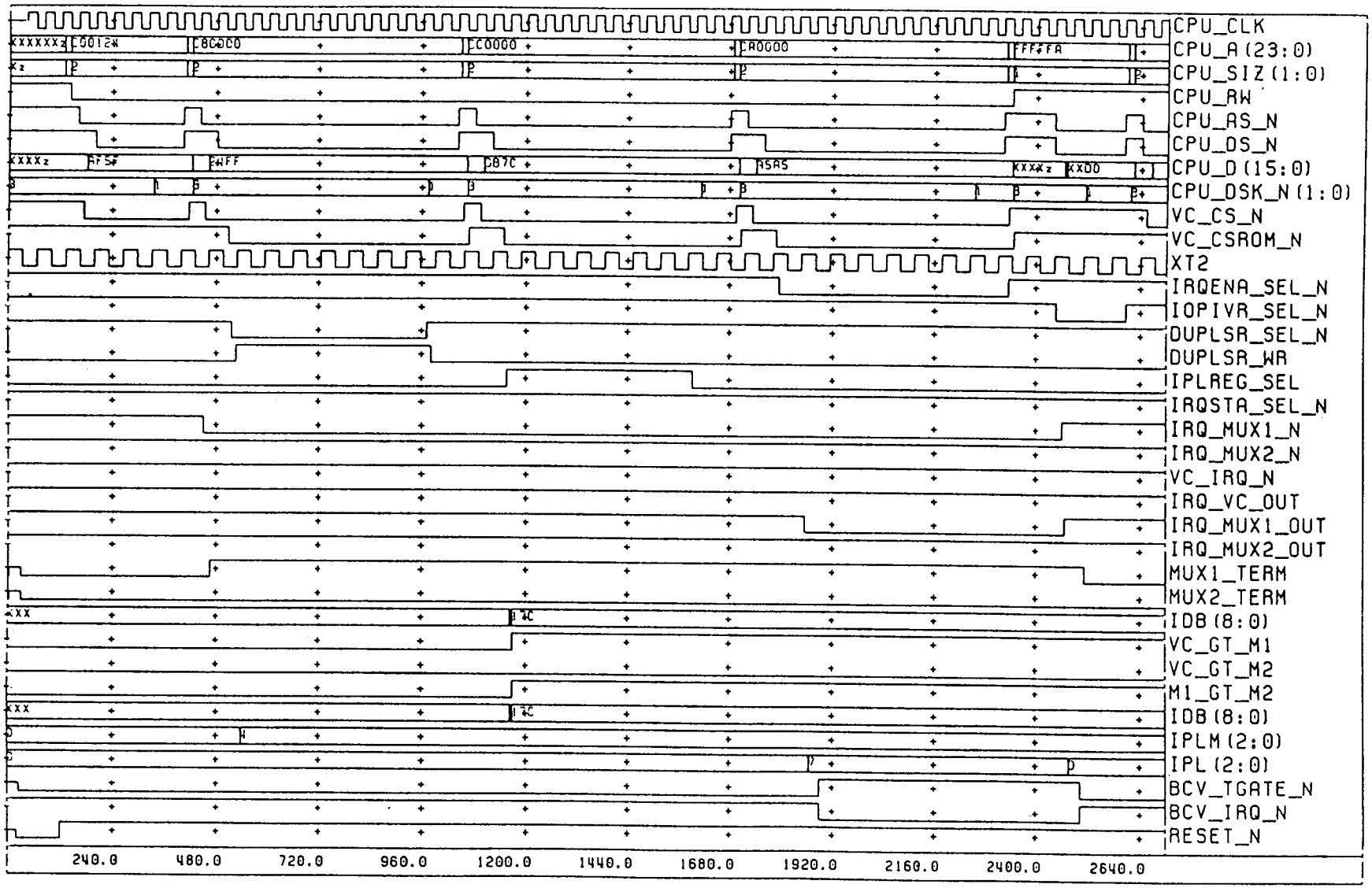
Table 11. Video Configuration

APPENDIX D – Layout of Base Case Video Board



APPENDIX E – Simulation of IRQ Circuitry

The following simulation shows the operation of the BCV's IRQ circuitry working in an IOP2 environment. The simulation conditions are as follows: 25MHz IOP2 operation, zero AS/DS delay on BCV board and worst case PAL and VDSC timing. The simulation shows an interrupt on SYS_IRQF_N occurring during an access to DRAM. The next cycle is a write of $24FF to the duplicate status register at address $C80000, setting the interrupt mask level to 4. Cycle three is a write of $DB7C to the interrupt priority level register. BCV interrupts are enabled in the next cycle, by accessing location $CA0000, which causes IRQENA_SEL_N to be asserted. During this forth bus cycle BCV_IRQ_N is asserted, causing the processor to run an interrupt acknowledge (IACK) cycle in the next bus slot. This IACK cycle causes IOPIVR_N to be asserted, which negated the BCV interrupt output and uses the IOPIVR to provide an interrupt vector ($DD). The final slot is another access to DRAM.


CPU_CLK
CPU_A(23:0)
CPU_SIZ(1:0)
CPU_RW
CPU_AS_N
CPU_DS_N
CPU_D(15:0)
CPU_DSK_N(1:0)
VC_CS_N
VC_CSROM_N
XT2
IRQENA_SEL_N
IOPIVR_SEL_N
DUPLSR_SEL_N
DUPLSR_WR
IPLREG_SEL
IRQSTA_SEL_N
IRQ_MUX1_N
IRQ_MUX2_N
VC_IRQ_N
IRQ_VC_OUT
IRQ_MUX1_OUT
IRQ_MUX2_OUT
MUX1_TERM
MUX2_TERM
IDB(8:0)
VC_GT_M1
VC_GT_M2
M1_GT_M2
IDB(8:0)
IPLM(2:0)
IPL(2:0)
BCV_TGATE_N
BCV_IRQ_N
RESET_N
240.0
480.0
720.0
960.0
1200.0
1440.0
1680.0
1920.0
2160.0
2400.0
2640.0

APPENDIX F – Base Case DRAM Test Program

The following assembly language program (RT1.A) can be used to test BCV DRAM in an MCDS running OS-9. As written, it tests a 0.5 MByte area of memory between locations $A00000-A7FFFF, but by editing header definitions, any range can be chosen.The program can also be easily modified to perform more tests using different data test values. The program, which is fairly simple and has many comments, can also be used as a learning vehicle for someone wishing to start programming in OS-9 assembly language.

RT1MAK is a simple example command file that can be used to assemble the program, by issuing the following command at the OS-9 prompt: make -uf=rt1mak <return>

```
* ------------------------------------------------------------------------
* RT1MAK (RAMTEST1 Makefile)
* OS-9 Makefile for assembly language programs
* Author     :   Colin Mac Donald
* Date       :   9th-Nov-1992
* Revision   :   0.1
* Outline    :   Takes source files (rt1.s) from /dd/USR/CM
*            :   Places relocatables (rt1.r) in  /dd/CMDS/CM
*            :   Generates and places stbs in /dd/CMDS/CM/STB
*            :   User needs to create dummy file v4.lst and place
*            :   in source directory
* ------------------------------------------------------------------------

RC      = r68
LC      = l68
CC      = cc

SDIR    = .
ODIR    = /dd/cmds/cm
RDIR    = /dd/cmds/cm
SLIB    = /dd/lib
DDIR    = /dd/defs

RFLAGS = -q -s -m=0
LIBS = -l=$(SLIB)/usr.l
LFLAGS = -g -s
RFILES = $(RDIR)/rt1.r
AFILES = $(SDIR)/rt1.a
DFILES = $(DDIR)/oskdefs.d
MAKER  = asl

rt1 : $(RDIR)/rt1.r $(SDIR)/$(MAKER)
    @echo "Found linker dependencies ..................."
    @echo "Processing explicit commands ................"
    $(LC) -g -s $(LIBS) $(RFILES) -o=$(ODIR)/$* -m >+$(SDIR)/$*.lst
    @echo "Linking complete, augmented listing ........"
    @echo ""

$(RDIR)/rt1.r : $(AFILES) $(DFILES) $(SDIR)/$(MAKER)
    @echo  "Found assembly dependencies................."
    @echo  "Deleting old assembling listing ............"
    copy   v4.lst $*.lst -r
    del    $(SDIR)/$*.lst
    @echo  "Processing explicit commands ................"
    $(RC)  $(RFLAGS) rt1.a -o=$(RDIR)/$*.r -l >$(SDIR)/$*.lst
    @echo  "Assembly complete, new listing generated ..."
    @echo  ""
* ------------------------------------------------------------------------
* R A M T E S T 1
* ------------------------------------------------------------------------
*
* File Name          :   RT1.A (RamTest1.A)
* Author             :   Colin Mac Donald
* Description        :   Memory Test program for the Motorola
*                    :   CD-i Development System (CDS) Base Case
*                    :   Video Board
* Revision           :   0.1
* Date               :   92/11/10
* History            :   0.1 Prerelease
*                    :   RT1.A copied from V6.A, will include write addres
* Description        :   This program is written to throughly test a
*                    :   memory area defined in its header. The program
*                    :   demonstrates how os9 alarms and variable sections
*                    :   may be used. Add more tests by editting MAIN1 area of
```

```
*                   :   Program CMcD 93/01/10
* ------------------------------------------------------------------------

        nam rt1.a
        ttl CDS BCV RAM test program
        use /dd/defs/oskdefs.d

Edition     equ 1
TypLng      equ (Prgrm<<8)+Objct
AttRev      equ (ReEnt<<8)+4


        psect ramtest,TypLng,AttRev,Edition,1024,main1
StdIn       equ 0
StdOut      equ 1
StdErr      equ 2
ramsta      equ $A00000
ramstp      equ $A7FFFE
alrmsig     equ 260
interval    equ 4
pass        equ 0

* ---------------------- OS-9 Variable Section ---------------------------
* The following section is used for storage of variables, that can both
* be defined at assembly time via dc.x commands and can accessed for
* reading/writing during program execution via register A6
* ------------------------------------------------------------------------

        vsect
alarm_flag: dc.w 0
fail_c:     dc.w 0
* The following '8's represent backspace chars for VT100
gstrt:      dc.b 8,8,8,8,8,8,8,8
            dc.b "INITADDR"
gend:
gsiz        equ *-gstrt
        ends

* ------------------------------------------------------------------------
* The following data is only ever read and so can be kept in the psect
* ------------------------------------------------------------------------

mess0:  dc.b "Starting RAM test ..................",C$CR,C$LF
msiz0   equ *-mess0
mess1:  dc.b "Writing 0000 to address ....00000000"
msiz1   equ *-mess1
mess2:  dc.b "Writing FFFF to address ....00000000"
msiz2   equ *-mess2
mess3:  dc.b "Writing A5A5 to address ....00000000"
msiz3   equ *-mess3
mess4:  dc.b "Writing 5A5A to address ....00000000"
msiz4   equ *-mess4
mess5:  dc.b "Writing 1234 to address ....00000000"
msiz5   equ *-mess5
mess6:  dc.b "Writing addr to address ....00000000"
msiz6   equ *-mess6
mess7:  dc.b "Fail occured at address ....FAILADDR"
msiz7   equ *-mess7

p_mess: dc.b 8,8,8,8,8,8,8,8
        dc.b "..PASSED"
p_size  equ *-p_mess

f_mess: dc.b 8,8,8,8,8,8,8,8
        dc.b "..FAILED"
f_size  equ *-f_mess

cr_mess:    dc.b C$CR,C$LF
cr_size     equ *-cr_mess

read_mess:  dc.b $1B,$5B,"23D","fm"
            dc.b $1B,$5B,"15D","Reading"
            dc.b $1B,$5B,"29C"
read_size   equ *-read_mess


* ---------------- START_ICPT - Start Intercept Sub Routine ----------------
* Subroutine Name   :   START_ICPT
* Purpose           :   Install the intercept routine that will
*                   :   handle the cyclic alarm generated signals
* ------------------------------------------------------------------------
```

```
* Inputs            :   iroot = label of intercept routine
* ------------------------------------------------------------------------
* Ouputs            :   Signals sent to this process will cause iroot to
*                   :   be called
* ------------------------------------------------------------------------
* Register Usage    :   A0      =   Address of intercept routine
*                   :   A6      =   Address of RT1's data area
* ------------------------------------------------------------------------
start_icpt:
        move.l  a0,-(sp)            stack a0
        lea     iroot(pc),a0        place routine add in a0
        os9     F$Icpt              call intercept init
        move.l  (sp)+,a0            unstack a0
        rts                         and return
* ------------------------------------------------------------------------



* ---------------- START_ALRM - Start Alarm Sub Routine --------------------
* Subroutine Name   :   START_ALRM
* Purpose           :   Starts a cyclic alarm via an os-9 service request
*                   :   NB OS-9 timekeeping functions need to be running
* ------------------------------------------------------------------------
* Inputs            :   alrmsig =   signal no identifying this alarm
*                   :   interval=   time between signals (ticks/256secs)
* ------------------------------------------------------------------------
* Ouputs            :   Recurring periodic signal sent every INTERVAL
*                   :   with signal number ALRMSIG
* ------------------------------------------------------------------------
* Register Usage    :   D0      =   Reserved param for OS9 call
*                   :   D1      =   A$Cycle  param for OS9 call
*                   :   D2      =   signal code for OS9 call
*                   :   D3      =   time interval for OS9 call
* ------------------------------------------------------------------------

start_alrm:
        movem.l d0-d3,-(sp)         stack used registers
        moveq   #0,d0               cyclic alarm must be 0
        move.w  #A$Cycle,d1         standard mnemonic
        move.l  #alrmsig,d2         defined in header to be 260
        move.l  #interval,d3        fast update
        os9     F$Alarm             os9 call
        bcs     error_exit          exit on error, check clock
        movem.l (sp)+,d0-d3         unstack used registers
        rts
* ------------------------------------------------------------------------


* ---------------- MESSAGE - Message Sub Routine --------------------
* Subroutine Name   :   MESSAGE
* Purpose           :   Prints a message to standard ouput (VT100)

* ------------------------------------------------------------------------
* Inputs            :   d0      =   path number
*                   :   d1      =   number of bytes to write
*                   :   a0      =   address of message area
* ------------------------------------------------------------------------
* Ouputs            :   Message sent to standard output
* ------------------------------------------------------------------------

message:
        movem.l d0-d1/a0,-(sp)      stack used registers
        moveq   #StdOut,d0          load output path into d0
        os9     I$Write             call I$Write no EOL
        bcs     error_exit          carry == error
        movem.l (sp)+,d0-d1/a0      unstack used registers
        rts                         return
* ------------------------------------------------------------------------


* ---------------- OUTPUT_VAL - Output Value Sub Routine ------------------
*
* Subroutine Name   :   OUTPUT_VAL
* Purpose           :   Converts a address supplied in address register
*                   :   A4 to eight ACSII characters. These characters
*                   :   are stored at a label offset (gstrt) from A6
*                   :   in a VSECT.
* ------------------------------------------------------------------------
* Inputs            :   A4      =   Address to be converted
* ------------------------------------------------------------------------
```

```
* Outputs          :   ASCII representation of A4 -> standard out (VT100)
* ------------------------------------------------------------------------
* Register Usage   :   D0      =   not used
*                  :   D1      =   message size for I$Write
*                  :   D2      =   address store for masking
*                  :   D3      =   mask for address nibbles
*                  :   D4      =   counter for nibble loop
*                  :   D5      =   ASCII output location offset
*                  :   D6      =   address store
*                  :   A0      =   offset to converted address
* ------------------------------------------------------------------------

output_val:
        movem.l a0/d1-d6,-(sp)      save registers
        move.l  a4,d2               address for conv -> d2
        move.l  a4,d6               save address in d6
        moveq.l #$0F,d3             setup mask ls 4 bits
        moveq.l #$7,d4              setup loop count - 8 times
        move.l  #gend,d5            offset (from A6 vsect) -> d5
out_loop:
        and.l   d3,d2               D4 (3:0) -> ascii
        cmpi.b  #$9,d2              check d2 gt 9?
        ble     add_$30             if not, skip adding extra7
        addi.l  #$7,d2              numbers>9 add $37

add_$30:
        addi.l  #$30,d2             number <9 add $30
        subq    #$1,d5              working backwards
        move.b  d2,(a6,d5)          move to out area
        lsr.l   #$4,d6              next ls nibble
        move.l  d6,d2               next nibble in d2
        dbf     d4,out_loop         process next nibble

        lea     gstrt(a6),a0        load mess address in a0
        move.l  #gsiz,d1            mess siz -> d1
        jsr     message(pc)         output message
        movem.l (sp)+,a0/d1-d6      restore registers
        rts                         return
* ------------------------------------------------------------------------



* ---------------- FLT_SR - Alarm Flag Sub Routine ----------------------
*
* Subroutine Name   :   FLT_SR
* Purpose           :   Checks to see if the cyclic alarm has set a flag
*                   :   indicating time to output address value. If set
*                   :   a routine to output address to VT100 is called.
* ------------------------------------------------------------------------
* Inputs            :   alrmsig     =   signal value of cyclic alarm
*                   :   alarm_flag  =   address (offset) of cyclic flag
* ------------------------------------------------------------------------
* Outputs           :   current address held in A4 if flag set
* ------------------------------------------------------------------------
* Register Usage    :   D6      =   flag signal value holding reg
*                   :   D7      =   alarm signal value comp reg
*                   :   A6      =   offset to converted address
* ------------------------------------------------------------------------

flt_sr:
        movem.l d6-d7/a6,-(sp)      stack used registers
        move.l  #alrmsig,d7         ramtest signal value -> d7
        move.w  alarm_flag(a6),d6   flg locn signal value -> d6
        cmp.w   d6,d7               compare alrmsig with received
        bne.s   noalrm              if != alrmsig, then ignore
        clr.w   alarm_flag(a6)      clear flag
        jsr     output_val(pc)      if = alrmsig, then output address
noalrm:
        movem.l (sp)+,d6-d7/a6      unstack registers
        rts
* ------------------------------------------------------------------------



* ---------------- RAM_TEST Ram Test Sub Routine ------------------------
*
* Subroutine Name   :   RAM_TEST
* Purpose/Function  :   Tests a block of RAM up to 16MBytes in size
*                   :   with zeros,ones,a's and 5's.
* ------------------------------------------------------------------------
```

```
* Inputs            :   RAM start address declared as ramsta
*                   :   RAM stop address declared as ramstp
*                   :   D1      = 1st message size
*                   :   A0      = 1st message location
* ------------------------------------------------------------------------
* Outputs           :
* ------------------------------------------------------------------------
* Register Usage    :   D0      = Write value
*                   :   D1      = low-order count safe
*                   :   D2      = lo-order counter
*                   :   D3      = Read value
*                   :   D4      = hi-order count safe
*                   :   D5      = hi-order counter
*                   :   D6      = test type identifier
*
*                   :   A0      = message location
*                   :   A1      = pass value (0)
*                   :   A4      = memory pointer
*                   :   A5      = end location
*                   :   A6      = os9 data pointer
* ------------------------------------------------------------------------

ram_test:
        movem.l d1-d7/a0-a6,-(sp)
        move.l  #ramsta,a4          start test address
        move.l  #ramstp,a5          end test address
        move.w  #pass,a1            hold pass val in a1 for compare
        move.l  a5,d1               load end address into d1
        sub.l   a4,d1               subtract start address
        lsr.l   #$1,d1              divide by two -> words

        move.l  d1,d4               d1 -> hi cnt safe
        lsr.l   #$8,d4              shift hi-order to 0-15
        lsr.l   #$8,d4              shift hi-order to 0-15
        move.l  d4,d5               initialise hi counter

        andi.l  #$0000FFFF,d1       ensure 31-16 of d2 -> 0
        move.l  d1,d2               initialise lo counter
        cmpi.w  #$0,d1              is lo cnt zero
        beq     zentry              zero entry pos
        sub.l   #$1,d1              take account of -1 dbcc
        move.l  d1,d2               mod lo cntr

write_loop:
        jsr     write_mem(pc)       jump to mem write sr
        jsr     flt_sr(pc)          check the alarm flag
        dbf     d2,write_loop       loop back
zentry:
        move.w  #$FFFF,d2           after initial value
        dbf     d5,write_loop       loop back
        jsr     output_val(pc)      complete write output
        move.l  d1,d2               reset lo counter
        move.l  d4,d5               reset hi counter

read_init:
        move.l  d1,-(sp)            save d1
        lea     read_mess(pc),a0    xng mess to "Reading"
        move.l  #read_size,d1       message length
        jsr     message(pc)         output message
        move.l  (sp)+,d1            restore d1

read_loop:
        jsr     read_comp(pc)       jump to read/compare routine
        cmpa.w  fail_c(a6),a1       check for any fails
        bne     skip_update         if any, no update (overwrite)
        jsr     flt_sr(pc)          check the alarm flag
skip_update:
        dbf     d2,read_loop        loop if lo cnt > -1
        move.l  #$FFFF,d2           fill in lo cnt
        dbf     d5,read_loop        loop if hi cnt > -1
        cmpa.w  fail_c(a6),a1       check for any fails
        bne     no_pass_mess        if any, no pass message
        lea     p_mess(pc),a0
        move.l  #p_size,d1
        jsr     message(pc)
no_pass_mess:
        movem.l (sp)+,d1-d7/a0-a6   read fm stack,cc not affect
        rts
* ------------------------------------------------------------------------
```

```
* ------------ WRITE_MEM - Memory Write Sub Routine ----------------------
*
* Subroutine Name   :   WRITE_MEM
* Purpose/Function  :   Writes a word value to a memory location
*                   :   Only A4 is changed, as wished -> no stacking
* ---------------------------------------------------------------------
* Inputs            :   D0      = write data for non-incremental test
*                   :   D6      = test type (incr/non-incr)
*                   :   A4      = pointer to current mem locn
* ---------------------------------------------------------------------
* Outputs           :   D0 -> (a4) or a4->(a4)
* ---------------------------------------------------------------------

write_mem:
        cmpi.w  #$0,d6              if d6=1 then incremental write
        beq     non_incr_write      if not goto standard routine
        move.w  a4,(a4)+            write address to address
        rts                         return - no stacking required
non_incr_write:
        move.w  d0,(a4)+            write d0 to pointer address
        rts                         return - no stacking required
* ---------------------------------------------------------------------


* ------------ READ_COMP - Memory Read/Compare Sub Routine --------------
*
* Subroutine Name   :   WRITE_MEM
* Purpose/Function  :   Writes a word value to a memory location
*                   :   Only A4 is changed, as wished -> no stacking
* ---------------------------------------------------------------------
* Inputs            :   D0      = write data for non-incremental test
*                   :   D6      = test type (incr/non-incr)
*                   :   A4      = pointer to current mem locn
*                   :   A6      = os9 data pointer
* ---------------------------------------------------------------------
* Outputs           :   fail_c(a6) -> fail_c(a6)+1
*                   :   error message -> vt100
* ---------------------------------------------------------------------
* Register Usage    :   D3      = read data
* ---------------------------------------------------------------------

read_comp:
        move.w  -(a4),d3            read from pointer location
        cmpi.w  #$0,d6              if d6=1 then incremental read
        beq     non_incr_read       if not goto standard routine
        cmp.w   a4,d3               compare data with locn address
        bne     error_jump          if not equal - do not return yet
        rts                         return - no stacking required
non_incr_read:
        cmp.w   d3,d0               compare with written value
        bne     error_jump          if not equal - do not return yet
        rts                         return - no stacking required
error_jump:
        addq.w  #$1,fail_c(a6)      increment fail count
        jsr     error_mess(pc)      print out fail locn
        rts                         return - no stacking required
* ---------------------------------------------------------------------


* ------------ ERROR_MESS - Error Message Sub Routine ------------
*
* Subroutine Name   :   ERROR_MESS
* Purpose/Function  :   Outputs "FAILED" message to standard output
*                   :   if the first fail in group. For first and subs
*                   :   -sequent failures, a fail address is output
* ---------------------------------------------------------------------
* Inputs            :   fail_c  = locn (offset) fail count flag
*                   :   f_mess  = locn of "FAILED" message
*                   :   f_size  = size in bytes of "FAILED" message
*                   :   cr_mess = locn of CR,LF message
*                   :   cr_size = size in bytes of CR,LF message
*                   :   mess7   = locn of fail address message
*                   :   msiz7   = size of fail address message
* ---------------------------------------------------------------------
* Outputs           :   "FAILED" message to standard out on 1st fail
*                   :   "Fail occured at address....(A4)" message
* ---------------------------------------------------------------------
* Register Usage    :   A0      =   location of message data
*                   :   D1      =   size of message
```

```
* ------------------------------------------------------------------------

error_mess:
        movem.l a0/d1,-(sp)
        cmpi.w  #$1,fail_c(a6)
        bgt     skip_fmess
        lea     f_mess(pc),a0
        move.l  #f_size,d1
        jsr     message(pc)
skip_fmess:
        lea     cr_mess(pc),a0
        move.l  #cr_size,d1
        jsr     message(pc)
        lea     mess7(pc),a0
        move.l  #msiz7,d1
        jsr     message(pc)
        jsr     output_val(pc)
        movem.l (sp)+,a0/d1
        rts
* ------------------------------------------------------------------------


* ------------ CRLF_SR - Carridge Ret, Line Feed Sub Routine ------------
*
* Subroutine Name   :   CRLF_SR
* Purpose/Function  :   Prints out ASCII carridge return/ linefeed chars
*                   :   which for some reason, have to be sent seperately
*                   :   from related message.
* ------------------------------------------------------------------------
* Inputs            :   cr_mess =   locn of CR,LF chars
*                   :   cr_size =   cr,lf message size
* ------------------------------------------------------------------------
* Outputs           :   CR,LF to standard output
* ------------------------------------------------------------------------
* Register Usage    :   A0      =   location of message data
*                   :   D1      =   size of message
* ------------------------------------------------------------------------
crlf_sr:
        movem.l d1/a0,-(sp)     save d1 and a0
        lea     cr_mess(pc),a0  CR and LF
        move.l  #cr_size,d1     CR and LF size
        jsr     message(pc)     exec CR,LF
        movem.l (sp)+,d1/a0     recall d1 and a0
        rts
* ------------------------------------------------------------------------


* ---------------- MAIN1 - Mainline Program ----------------------------
*
* Subroutine Name   :   MAIN1
* Purpose/Function  :   Mainline of RAMTEST1 program
*                   :   Mainline is written for clarity - not for
*                   :   compact code. Tests can be easily added or
*                   :   subtracted as there are not loop counts to
*                   :   be affected. There are two possible endings
*                   :   continous loop or single-shot. The ending is
*                   :   choosen at assembly time
* ------------------------------------------------------------------------

main1:
        move.l  #alrmsig,d3     put alarm signal no into D3
        jsr     start_icpt(pc)  set up intercept routine
        jsr     start_alrm(pc)  set up alarms
        lea     mess0(pc),a0    Initial message location
        move.l  #msiz0,d1       Initial message size
        jsr     message(pc)     print initial message
        moveq   #$0,d6          setup for non-incremental testing

        clr.w   fail_c(a6)      clear fail count
        move.w  $#0000,d0       test data = 0000
        lea     mess1(pc),a0    address of 0000 mess
        move.l  #msiz1,d1       size of 0000 message
        jsr     message(pc)     print test message
        jsr     ram_test(pc)    run ram test for 0000
        jsr     crlf_sr(pc)     print CR,LF

        clr.w   fail_c(a6)      clear fail flag
        move.w  #$FFFF,d0       test data = FFFF
        lea     mess2(pc),a0    address of FFFF mess
        move.l  #msiz2,d1       size of FFFF message
```

```
        jsr     message(pc)     print test message
        jsr     ram_test(pc)    run ram test for FFFF
        jsr     crlf_sr(pc)     print CR,LF

        clr.w   fail_c(a6)      clear fail flag
        move.w  #$A5A5,d0       test data = A5A5
        lea     mess3(pc),a0    address of A5A5 mess
        move.l  #msiz3,d1       size of A5A5 message
        jsr     message(pc)     print test message
        jsr     ram_test(pc)    run ram test for A5A5
        jsr     crlf_sr(pc)     print CR,LF

        clr.w   fail_c(a6)      clear fail flag
        move.w  #$5A5A,d0       test data = 5A5A
        lea     mess4(pc),a0    address of 5A5A mess
        move.l  #msiz4,d1       size of 5A5A message
        jsr     message(pc)     print test message
        jsr     ram_test(pc)    run ram test for 5A5A
        jsr     crlf_sr(pc)     print CR,LF

        clr.w   fail_c(a6)      clear fail flag
        move.w  #$1234,d0       test data = 1234
        lea     mess5(pc),a0    address of 1234 mess
        move.l  #msiz5,d1       size of 1234 message
        jsr     message(pc)     print test message
        jsr     ram_test(pc)    run ram test for 5A5A
        jsr     crlf_sr(pc)     print CR,LF

        clr.w   fail_c(a6)      clear fail flag
        moveq   #$01,d6         incremental test
        lea     mess6(pc),a0    address of incr mess
        move.l  #msiz6,d1       size of incr message
        jsr     message(pc)     print test message
        jsr     ram_test(pc)    run ram test for incr
        jsr     crlf_sr(pc)     print CR,LF

* ---------------------- Continuous Loop Ending -----------------------
end:    bra.w   main1
* ---------------------------------------------------------------------

* ------------------------ Single Shot Ending -------------------------
* end:      moveq   #0,d1
*           os9     F$Exit
* ---------------------------------------------------------------------

error_exit:
        os9     F$Exit


* ---------------------------------------------------------------------
*
* Subroutine Name   :   IROOT
* Purpose/Function  :   Sets a flag in the RAMTEST1 data area if a
*                   :   signal with signal no ALRMSIG is received
*                   :   If a signal != ALRMSIG is received the program
*                   :   is exited
* ---------------------------------------------------------------------
* Inputs            :   alrmsig = desired signal number for flag set
*                   :   alarm_flag = locn (offset) of flag
* ---------------------------------------------------------------------
* Outputs           :   alarm_flag in RAMTEST vsect is set
* ---------------------------------------------------------------------
* Register Usage    :   A6      =   os9 data pointer
*                   :   D1      =   received signal number
* ---------------------------------------------------------------------

iroot:  cmpi.l  #alrmsig,d1
        bne     error_exit
        move.w  d1,alarm_flag(a6)
        os9     F$RTE

        ends
```

APPENDIX G—VDSC Display Program

The following OS-9 'C' program, "FADE.C", is a simple example of how to initialise and control the VDSC and ICA/DCA areas in order to generate and manipulate a video image. This can be accomplished by compiling FADE.C using the Microware 'C' compiler. However, as this program sets up the VDSC for PAL mode, the BCV must also be configured for PAL mode to obtain the correct video output.

In this program, both VDSC planes A and B are configured for 7-bit CLUT mode. The plane images consist of four identical horizontal bars of 100% saturation: red, green, blue and white. The only difference between the plane A and B images is the order of the colours. The program starts with one plane at full weight and the other at zero weight. The planes constantly fade from one to another in a see-saw fashion, creating a range of colours in the bars. As the weights of each plane are changed, their value is output to the standard output device.

```
/*****************************************************************/
/*  "FADE.C"                                                     */
/*  MCDS Base Case Video Test Program                            */
/*****************************************************************/
/*  Program Name     : FADE.C                                    */
/*  Author           : Colin Mac Donald                          */
/*  Date             : 3rd March 1993                            */
/*  Purpose          : Test Demo of VDSC Board                   */
/*  Description      : This program sets up the MCD210 for PAL   */
/*                   : timing. It then displays 4 parallel coloured */
/*                   : bars on-screen (red,green,blue,white) it then */
/*                   : uses the dual VDSC planes and mixing control */
/*                   : to fade to these bars in a different position */
/*****************************************************************/

#include    <stdio.h>

/*****************************************************************/
/*  General Definitions                                          */
/*****************************************************************/

#define     VDSC    0x00800000
#define     P1_ODD  0x00040000      /* 840000 - 870000          */
#define     P2_ODD  0x00240000      /* A40000 - A70000          */

#define     P1_DT0  0x0000          /* 1st quarter screen       */
#define     P1_DT1  0x0101          /* 2nd quarter screen       */
#define     P1_DT2  0x0202          /* 3rd quarter screen       */
#define     P1_DT3  0x0303          /* 4th quarter screen       */

#define     P2_DT0  0x0101          /* 1st quarter screen       */
#define     P2_DT1  0x0202          /* 2nd quarter screen       */
#define     P2_DT2  0x0303          /* 3rd quarter screen       */
#define     P2_DT3  0x0000          /* 4th quarter screen       */

#define     P1_ICA  0x00010000      /* 810000 - 810xxx          */
#define     P2_ICA  0x00210000      /* A10000 - A10xxx          */
#define     P1_DCA  0x00020000      /* 820000 - 830000          */
#define     P2_DCA  0x00220000      /* A20000 - A30000          */

#define     DCDATA  0x10000000      /* NOP                      */
#define     DCSIZ   280             /* 240 = NTSC / 280 = PAL   */
#define     SCSIZE  53760           /* (384 x 280)/2 = 53760    */

/*****************************************************************/
/* ICA/DCA Instruction Register Stems                            */
/*****************************************************************/

#define     CL0 0x80000000          /* CLUT Address 0           */
#define     CL1 0x81000000          /* CLUT Address 1           */
#define     CL2 0x82000000          /* CLUT Address 2           */
#define     CL3 0x83000000          /* CLUT Address 3           */
#define     ICR 0xC0000000          /* Image Control Register   */
#define     TCR 0xC1000000          /* Transparency Control     */
#define     POR 0xC2000000          /* Plane Order Register     */
#define     CBR 0xC3000000          /* CLUT Bank Register       */
#define     TCA 0xC4000000          /* Transparent Color A      */
#define     TCB 0xC6000000          /* Transparent Color B      */
#define     MCA 0xC7000000          /* Mask Color A             */
#define     MCB 0xC9000000          /* Mask Color B             */
#define     DSA 0xCA000000          /* DYUV Start A             */
#define     DSB 0xCB000000          /* DYUV Start B             */
#define     CPR 0xCD000000          /* Cursor Position Register */
#define     CCR 0xCE000000          /* Cursor Control Register  */
#define     CPT 0xCF000000          /* Cursor Pattern Register  */
#define     BCR 0xD8000000          /* Backdrop Color Register  */
#define     MHA 0xD9000000          /* Mosaic Hold A            */
#define     MHB 0xDA000000          /* mosaic Hold B            */
#define     WFA 0xDB000000          /* Weight Factor Plane A    */
#define     WFB 0xDC000000          /* Weight Factor Plabe B    */

/*****************************************************************/
/* VDSC Memory Mapped Register Values                            */
/*****************************************************************/

#define     CSR1R   0x004FFFF1      /* Status Reg 1 (read)      */
#define     CSR1W   0x004FFFF0      /* Control Reg 1 (write)    */
#define     DCR1    0x004FFFF2      /* Display Command Reg 1    */
#define     VSR1    0x004FFFF4      /* Video Start Reg1         */
#define     DDR1    0x004FFFF8      /* Display Decode Reg1      */
```

```
#define     DCP1    0x004FFFFA      /* DCA Pointer 1                */

#define     CSR2R   0x004FFFE1      /* Status Reg2 (read)           */
#define     CSR2W   0x004FFFE0      /* Control Reg2 (write)         */
#define     DCR2    0x004FFFE2      /* Display Command Reg2         */
#define     VSR2    0x004FFFE4      /* Video Start Reg2             */
#define     DDR2    0x004FFFE8      /* Display Decode Reg2          */
#define     DCP2    0x004FFFEA      /* DCA Pointer 2                */

/*********************************************************************/
/* ICA/DCA Instructions                                              */
/*********************************************************************/

#define     ICR_V   0x00000303      /* -ext vid, +ch1 +ch2 CLUT7    */
#define     TCR_V   0x00000808      /* +mix, +no transp             */
#define     POR_V   0x00000000      /* +plane A 1st                 */
#define     CB0_V   0x00000000      /* +CLUT bank0                  */
#define     CB2_V   0x00000002      /* +CLUT bank2                  */
#define     TCX_V   0x00000000      /* Transparent Col A/B          */
#define     MCX_V   0x00000000      /* +mask color zs               */
#define     CPR_V   0x00200200      /* +no corner posn              */
#define     CCR_V   0x00DB000C      /* +c,+on/cmp,+red              */

#define     CL0_V   0x00FCFCFC      /* CLUT 0 is White              */
#define     CL1_V   0x00FC0000      /* CLUT 1 is Red 100%           */
#define     CL2_V   0x0000FC00      /* CLUT 2 is Green 100%         */
#define     CL3_V   0x000000FC      /* CLUT 3 is Blue 100%          */
#define     BCR_V   0x00000000      /* black bacdrop                */
#define     MHX_V   0x00000001      /* no mosaic hold               */
#define     WFA_V   0x003F          /* Weight Factor Pl A = 3F      */
#define     WFB_V   0x0000          /* Weight Factor Pl B = 0       */

/*********************************************************************/
/* ICA/DCA Instruction Register Values                               */
/*********************************************************************/

#define     CSR1W_V 0x8000          /* -IRQ,Dtk12,256k* DRAM        */
#define     CSR2W_V 0x8000          /* & -ST, -BE                   */
#define     DCR1_V1 0x4000          /* -DE,+30M,-il,VSA=0,-CM       */
#define     DCR1_V2 0xC000          /* +DE,+30M,-il,VSA=0,-CM       */
#define     DCR1_V3 0xC200          /*  DCR1_V2,+IC1                */
#define     DCR2_V3 0xC200          /*  DCR1_V2,+IC2                */
#define     DCR1_V4 0xC300          /*  DCR1_V3,+DC1                */
#define     DCR2_V4 0xC300          /*  DCR2_V3,+DC1                */
#define     DDR1_V  0x0000          /*  +MF=2,FT=BtM,DCA1_hi=0      */
#define     DDR2_V  0x0000          /*  +MF=2,FT=BtM,DCA2_hi=0      */

#define     ICA1_PTR    0x00000400  /* ICA Plane1(A) Even Field Ptr */
#define     ICA2_PTR    0x00200400  /* ICA Plane2(B) Even Field Ptr */

/*********************************************************************/
/* ICA/DCA Instruction Register Stems                                */
/*********************************************************************/

#define     ICA_STP     0x00000000      /* ICA Stop Instruction     */
#define     ICA_NOP     0x10000000      /* ICA NOP Instruction      */
#define     ICA_RLV     0x40000000      /* ICA Reload VSR Inst      */
#define     ICA_RVS     0x50000000      /* ICA Reload VSR & Stop    */
#define     ICA_RDC     0x20000000      /* Reload the DCP reg       */
#define     ICA_IRQ     0x60000000      /* Generate Interrupt       */

#define     DCA_STP     0x00000000      /* DCA Stop Instruction     */
#define     DCA_NOP     0x10000000      /* DCA NOP Instruction      */
#define     DCA_RDS     0x30000000      /* DCA Reload DCP & STOP    */
#define     DCA_IRQ     0x60000000      /* Generate Interrupt       */

#define     DA_MSK      0x80
#define     IRQ_MSK     0x06


/*********************************************************************/
/* Program Start and Initialise Values                               */
/*********************************************************************/

int main()
{
    char CSR1R_V=0x00;
    short DA_SET, IRQ_SET, arise =0;
    int i, ICA, DCA;
    int plaw = WFA_V;
    int plbw = WFB_V;
```

```
    char    *rdchar, *csr1reg, *csr2reg;
    unsigned short *wrreg, *rdreg, *dcr1reg, *dcr2reg, *image;
    unsigned long *(icaptr++), *(dcaptr++);


/*********************************************************************/
/* Initialise Pointers to Frequently Accessed Variables              */
/*********************************************************************/

    dcr1reg = (unsigned short*)(VDSC + DCR1);
    dcr2reg = (unsigned short*)(VDSC + DCR2);
    csr1reg = (char*)(VDSC + CSR1R);
    csr2reg = (char*)(VDSC + CSR2R);


/*********************************************************************/
/* Create Plane1 Screen Data                                         */
/*********************************************************************/

    image   = (unsigned short*)(VDSC + P1_ODD);
    puts(" Filling in Plane1 data ");
    for ( i=0; i < (SCSIZE/4); i++)
        *(image++) = P1_DT0;
    for ( i=0; i < (SCSIZE/4); i++)
        *(image++) = P1_DT1;
    for ( i=0; i < (SCSIZE/4); i++)
        *(image++) = P1_DT2;
    for ( i=0; i < (SCSIZE/4); i++)
        *(image++) = P1_DT3;


/*********************************************************************/
/* Create Plane2 Screen Data                                         */
/*********************************************************************/

    puts(" Filling in Plane2 data ");
    image   = (unsigned short*)(VDSC + P2_ODD);
    for ( i=0; i < (SCSIZE/4); i++)
        *(image++) = P2_DT0;
    for ( i=0; i < (SCSIZE/4); i++)
        *(image++) = P2_DT1;
    for ( i=0; i < (SCSIZE/4); i++)
        *(image++) = P2_DT2;
    for ( i=0; i < (SCSIZE/4); i++) {
        *(image++) = P2_DT3;


/*********************************************************************/
/* Create Plane1 DCA Area                                            */
/*********************************************************************/

    puts(" Filling in Plane1 DCA ");
    dcaptr  = (unsigned*)(VDSC + P1_DCA);
    *(dcaptr++) = DCA_NOP;
    *(dcaptr++) = DCA_NOP;
    *(dcaptr++) = DCA_NOP;
    *(dcaptr++) = (DCA_RDS + P1_DCA);
    for ( i=0; i < (DCSIZ-5); i++)
        *(dcaptr++) = DCA_NOP;
    *(dcaptr++) = (DCA_RDS + P1_DCA);

/*********************************************************************/
/* Create Plane2 DCA Area                                            */
/*********************************************************************/

    dcaptr  = (unsigned*)(VDSC + P2_DCA);
    puts(" Initialising Plane2 DCA ");
    *(dcaptr++) = DCA_NOP;
    *(dcaptr++) = DCA_NOP;
    *(dcaptr++) = DCA_NOP;
    *(dcaptr++) = (DCA_RDS + P2_DCA);
    for ( i=0; i < (DCSIZ-5); i++)
        *(dcaptr++) = DCDATA;
    *(dcaptr++) = (DCA_RDS + P2_DCA);

/*********************************************************************/
/* Initialise Plane1 Odd/Even Pointers to ICA Area                   */
/*********************************************************************/

    puts(" Setting Plane1 ICA ptrs for odd/even fields ");
```

```
    icaptr = (unsigned long*)(VDSC +  ICA1_PTR);
    *(icaptr++) = (ICA_RLV + P1_ICA);
    *(icaptr++) = (ICA_RLV + P1_ICA);


/*********************************************************************/
/* Create Plane1 ICA Area                                            */
/*********************************************************************/

    puts(" Filling in Plane 1 ICA data ");
    icaptr = (unsigned long*)(VDSC +  P1_ICA);
    *(icaptr++) = (WFA + WFA_V);
    *(icaptr++) = (ICR + ICR_V);
    *(icaptr++) = (TCR + TCR_V);
    *(icaptr++) = (POR + POR_V);
    *(icaptr++) = (MCA + MCX_V);
    *(icaptr++) = (CPR + CPR_V);
    *(icaptr++) = (CCR + CCR_V);
    *(icaptr++) = (MHA + MHX_V);
    *(icaptr++) = (BCR + BCR_V);
    *(icaptr++) = (CBR + CB0_V);
    *(icaptr++) = (CL0 + CL0_V);
    *(icaptr++) = (CL1 + CL1_V);
    *(icaptr++) = (CL2 + CL2_V);
    *(icaptr++) = (CL3 + CL3_V);
    *(icaptr++) = (ICA_RDC + P1_DCA);
    *(icaptr++) = (ICA_RVS + P1_ODD);

/*********************************************************************/
/* Create Plane2 ICA Area                                            */
/*********************************************************************/

    puts(" Setting Plane2 ICA ptrs for odd/even fields ");
    icaptr = (unsigned long*)(VDSC +  ICA2_PTR);
    *(icaptr++) = (ICA_RLV + P2_ICA);
    *(icaptr++) = (ICA_RLV + P2_ICA);

/*********************************************************************/
/* Create Plane2 Screen Data                                         */
/*********************************************************************/

    icaptr  = (unsigned long*)(VDSC +  P2_ICA);
    *(icaptr++) = (WFB + WFB_V);
    *(icaptr++) = (MCB + MCX_V);
    *(icaptr++) = (MHB + MHX_V);
    *(icaptr++) = (ICA_NOP);
    *(icaptr++) = (CBR + CB2_V);
    *(icaptr++) = (CL0 + CL0_V);
    *(icaptr++) = (CL1 + CL1_V);
    *(icaptr++) = (CL2 + CL2_V);
    *(icaptr++) = (CL3 + CL3_V);
    *(icaptr++) = (ICA_RDC + P2_DCA);
    *(icaptr++) = (ICA_NOP);
    *(icaptr++) = (ICA_RVS + P2_ODD);

/*********************************************************************/
/* Initialise the VDSC                                               */
/*********************************************************************/

    puts(" Configuring VDSC registers");
    puts(" Configuring CSR1W");
    wrreg = (unsigned short*)(VDSC + CSR1W);
    *wrreg = CSR1W_V;

    while (CSR1R_V) {
        CSR1R_V = (short)*csr1reg;
        CSR1R_V &= DA_MSK;
        }

/*  Wait DA-rise */
    while (!CSR1R_V) {
        CSR1R_V = (short)*csr1reg;
        CSR1R_V &= DA_MSK;
        }
    *dcr1reg = DCR1_V1;
    puts(" Initialised  DCR1");

    while (CSR1R_V) {
/*  puts(" Wait DA-fall"); */
        CSR1R_V = (short)*csr1reg;
        CSR1R_V &= DA_MSK;
```

```
        }

    *dcr1reg = DCR1_V2;
    puts(" Set the DE bit in DCR1");

    while (!CSR1R_V) {
/*  puts(" Wait DA-rise");  */
        CSR1R_V = (short)*csr1reg;
        CSR1R_V &= DA_MSK;
        }

    *dcr1reg = DCR1_V3;
    *dcr2reg = DCR2_V3;
    puts(" DCR1(IC1)->1, DCR2(IC2)->1");

    while (CSR1R_V) {
/*  puts(" Wait DA-fall"); */
    CSR1R_V = (short)*csr1reg;
    CSR1R_V &= DA_MSK;
    }

    *dcr1reg = DCR1_V4;
    *dcr2reg = DCR2_V4;
    puts(" DCR1(DC1)->1, DCR2(DC2)->1");

/*********************************************************************/
/* Plane weight timing and control                                   */
/*********************************************************************/

    while (1) {
            for ( i=0; i < 5; i++) {
                CSR1R_V = 0;
                while (!CSR1R_V) {
                    CSR1R_V = (short)*csr1reg;
                    CSR1R_V &= DA_MSK;
                    }
                while (CSR1R_V) {
                    CSR1R_V = (short)*csr1reg;
                    CSR1R_V &= DA_MSK;
                    }
            }
        while (!CSR1R_V) {
            CSR1R_V = (short)*csr1reg;
            CSR1R_V &= DA_MSK;
        }

        icaptr = (unsigned long*)(VDSC +  P1_ICA);
        *icaptr = (WFA + plaw);
        icaptr = (unsigned long*)(VDSC +  P2_ICA);
        *icaptr = (WFB + plbw);
        printf( "Plane A weight %X  Plane B weight %x.\n", plaw, plbw);

        if (arise) {
            plaw = plaw+1;
        plbw = plbw-1;
        }
        else {
            plaw = plaw-1;
            plbw = plbw+1;
        }
        if (plaw > 0x3E)
        arise = 0;
        if (plaw < 0x01)
        arise = 1;
    }
}
```

References

CENELEC EN50 049, November 1983 – Domestic or Similar Electronic Equipment Interconnection Requirements: Peritelevision Connector.

CCIR Report 624-3, 1986.

Motorola Application Note AN451: An MC68340-based Input/Output Processor

Motorola Semiconductor Technical Data, Dynamic RAMs and Memory Modules, DL155/D

Motorola Semiconductor Technical Data, Product Preview – MC13077 Advanced PAL/NTSC Encoder

Motorola Semiconductor Technical Data MCD210/D – MCD210 Video Decoder and System Controller

Motorola Semiconductor Technical Data, Product Preview – MC44200 Triple 8-bit Video DAC

Literature Distribution Centres:
EUROPE:Motorola Ltd., European Literature Centre, 88 Tanners Drive, Blakelands, Milton Keynes, MK14 5BP, England.
ASIA PACIFIC:Motorola Semiconductors (H.K.) Ltd., Silicon Harbour Center, No. 2, Dai King Street, Tai Po Industrial Estate, Tai Po, N.T., Hong Kong.
JAPAN:Nippon Motorola Ltd., 4-32-1, Nishi-Gotanda, Shinagawa-ku, Tokyo 141, Japan.
USA:Motorola Literature Distribution, P.O. Box 20912, Phoenix, Arizona 85036.

MOTOROLA

AN492/D