D32Kitty

Digital Signal Processor (DSP32c) based Data-acquisition board

ProScope GmbH
Arnold-Sommerfeld-Ring 2
D-52499 Baesweiler
GERMANY
www.proscope.de

1. Introduction

The *D32Kitty* Digital Signal processor data acquisition board enables fast data acquisition and control task. The core of this ISA and PCI compatible plug-in board is a digital floating point signal processor (DSP32C) with a computing power of 20 MIPS and 40 MFLOPS respectively. The digital signal processor is loaded with programs and controlled by the personal computer. The DSP has a fast, local 128 Kbytes memory for program code and data. The DSP host port interface with DMA capabilities enables a direct writing and reading of the complete DSP memory.

Furthermore the *D32Kitty* board supports 4 analog Input and Output channels as well a 8 bit digital Input and Output.

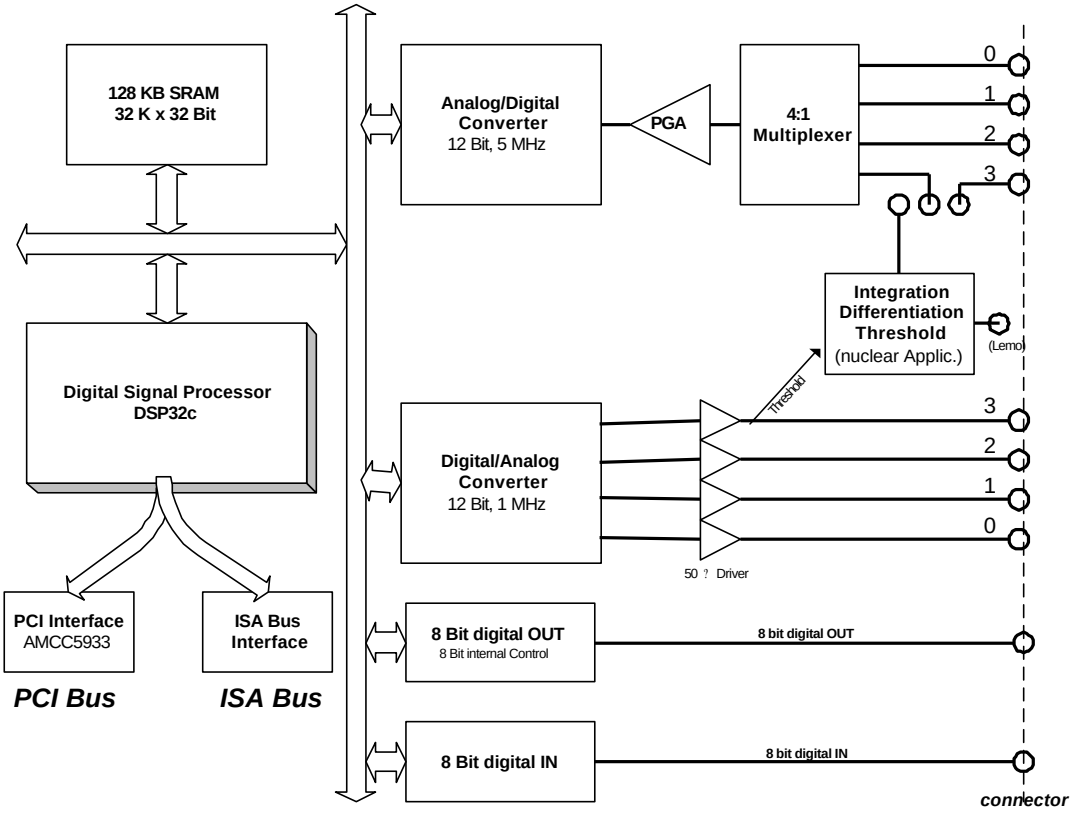
D32Kitty Components

- ? ? ISA Bus Interface (16 Bit, PC-ISA Bus)
- ? ? PCI Bus Interface (Plug and Play)
- ? ? Lucent DSP-32C (floating point DSP, 80 MHz, 20 MIPS, 40 MFLOPS)
- ? ? 128 Kbytes SRAM (15ns, organized as 32 K * 32 Bit)
- ? ? 4 analogue Inputs (12 Bit, 5 MHz, multiplexed)
- ? ? Programmable Gain Amplifier
  Analog input range: Gain =1: +/- 10.00 Volt
  2: +/- 5.00 Volt
  4: +/- 2.50 Volt
  8: +/- 1.25 Volt
- ? ? 4 analogue Outputs (12 Bit, +/- 10 Volt)
- ? ? 8 Bit Digital Output (TTL level)
- ? ? 8 Bit Digital Input (TTL level)

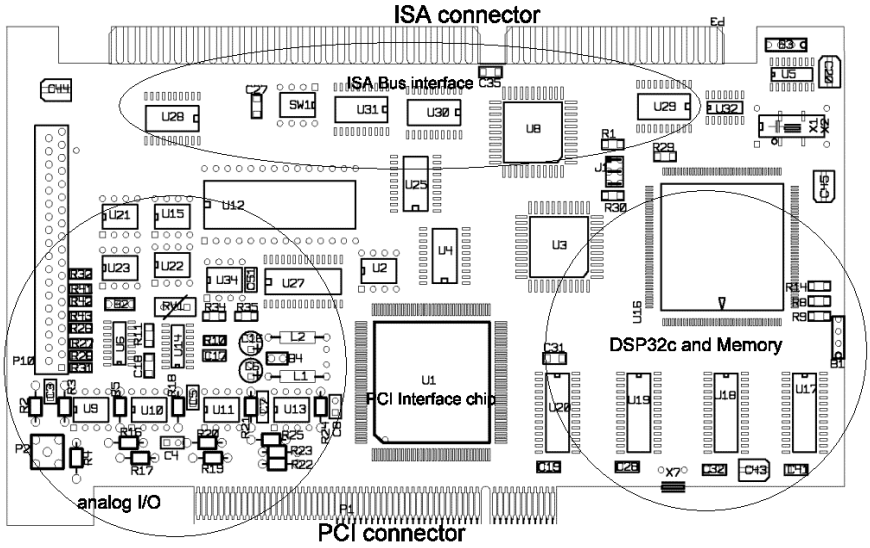
Programming the *D32Kitty* board :

Windows 3.1 / 95 / 98 / NT Drivers and Interface DLL are available.

1.1. Block diagram of D32Kitty

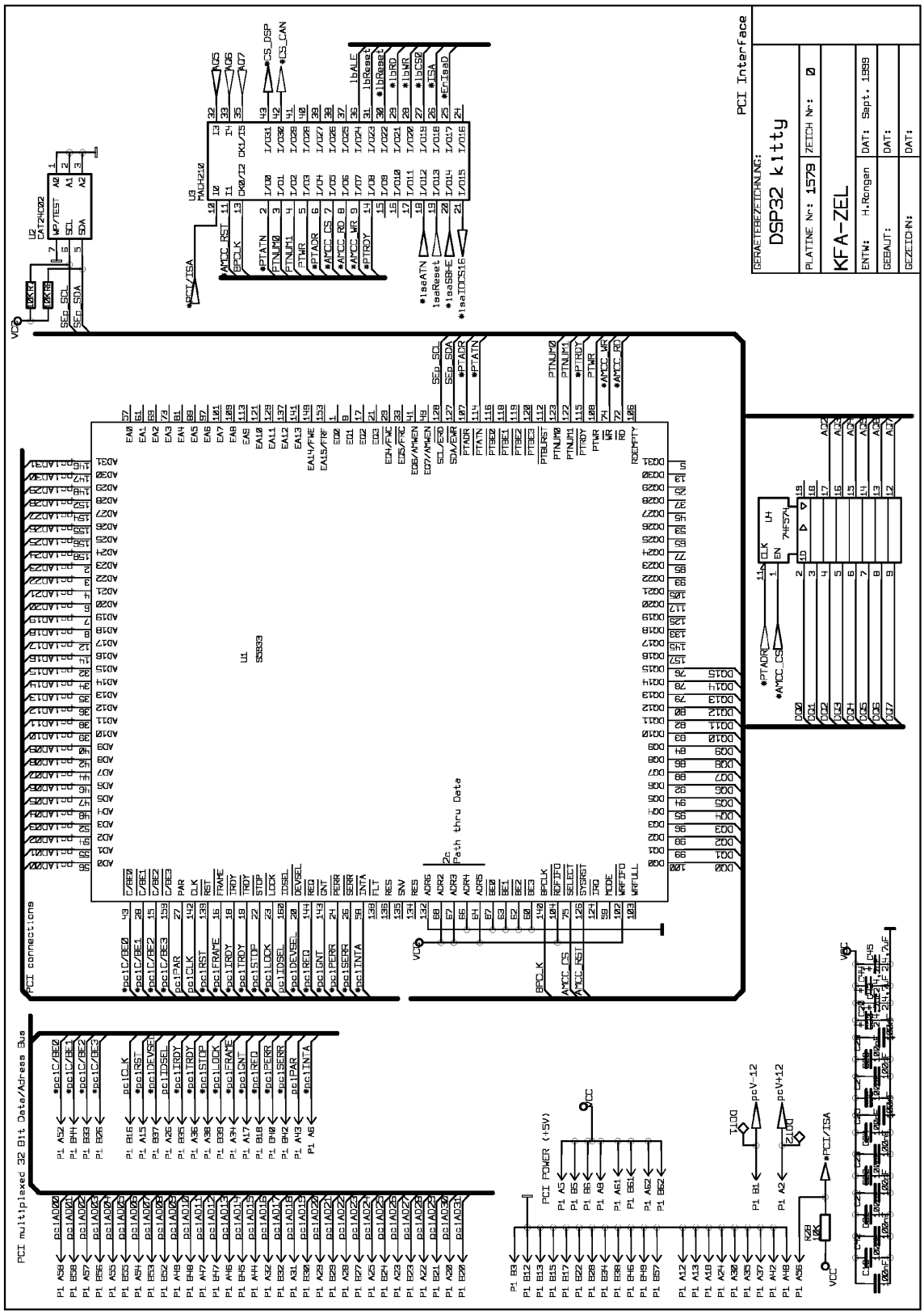


1.2. Board layout of D32Kitty

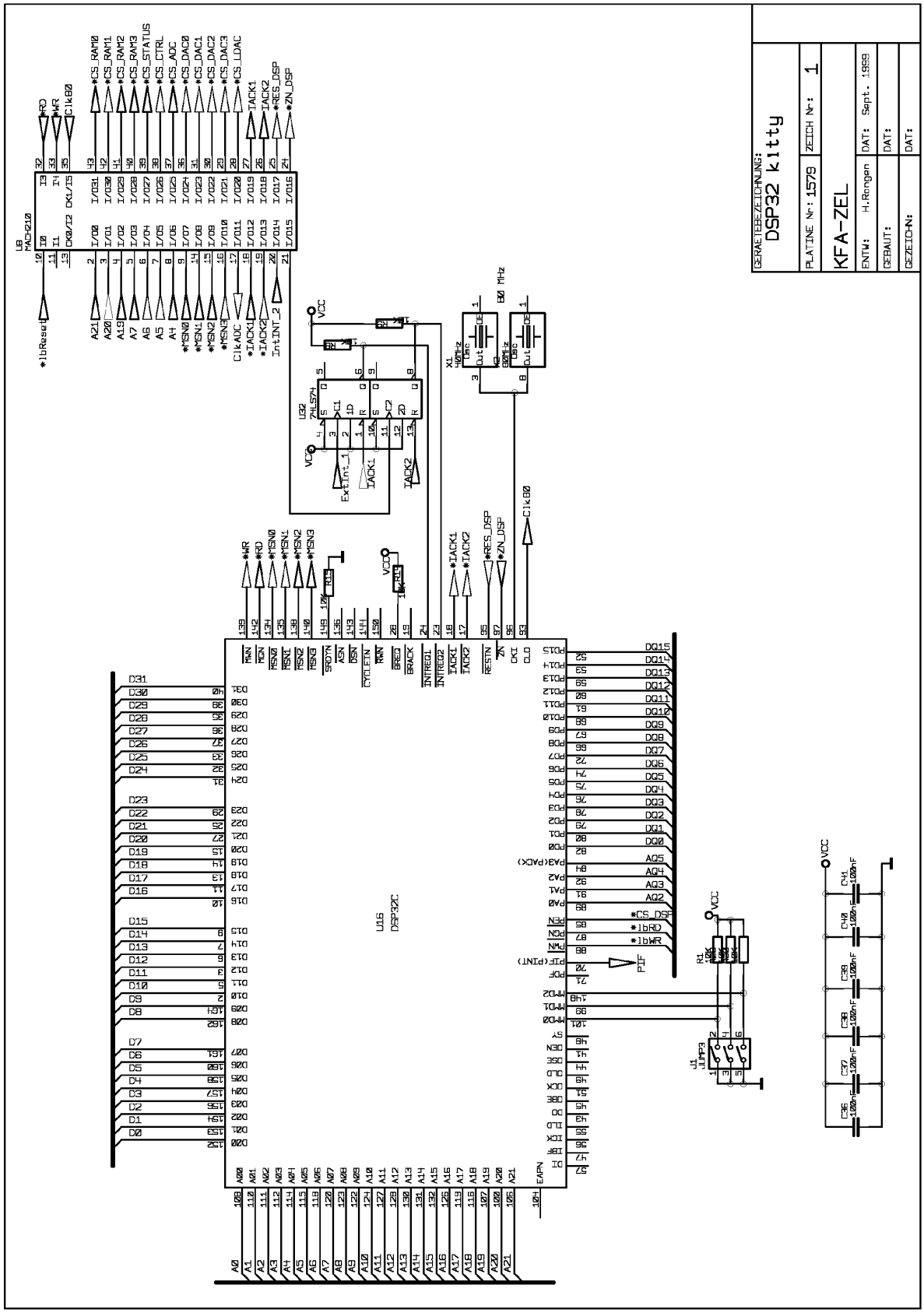


2. Circuit Diagram

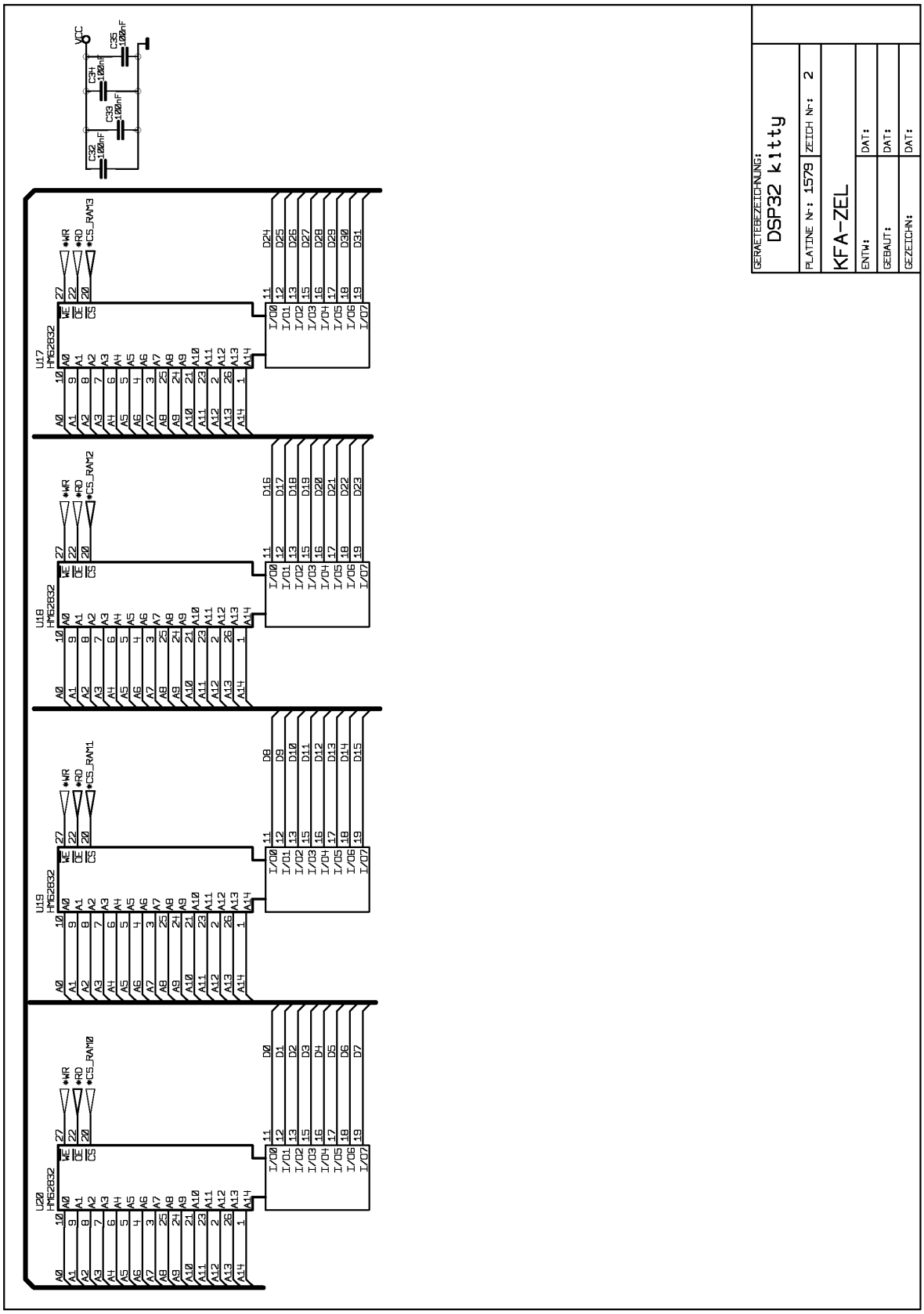
(PCI Interface)



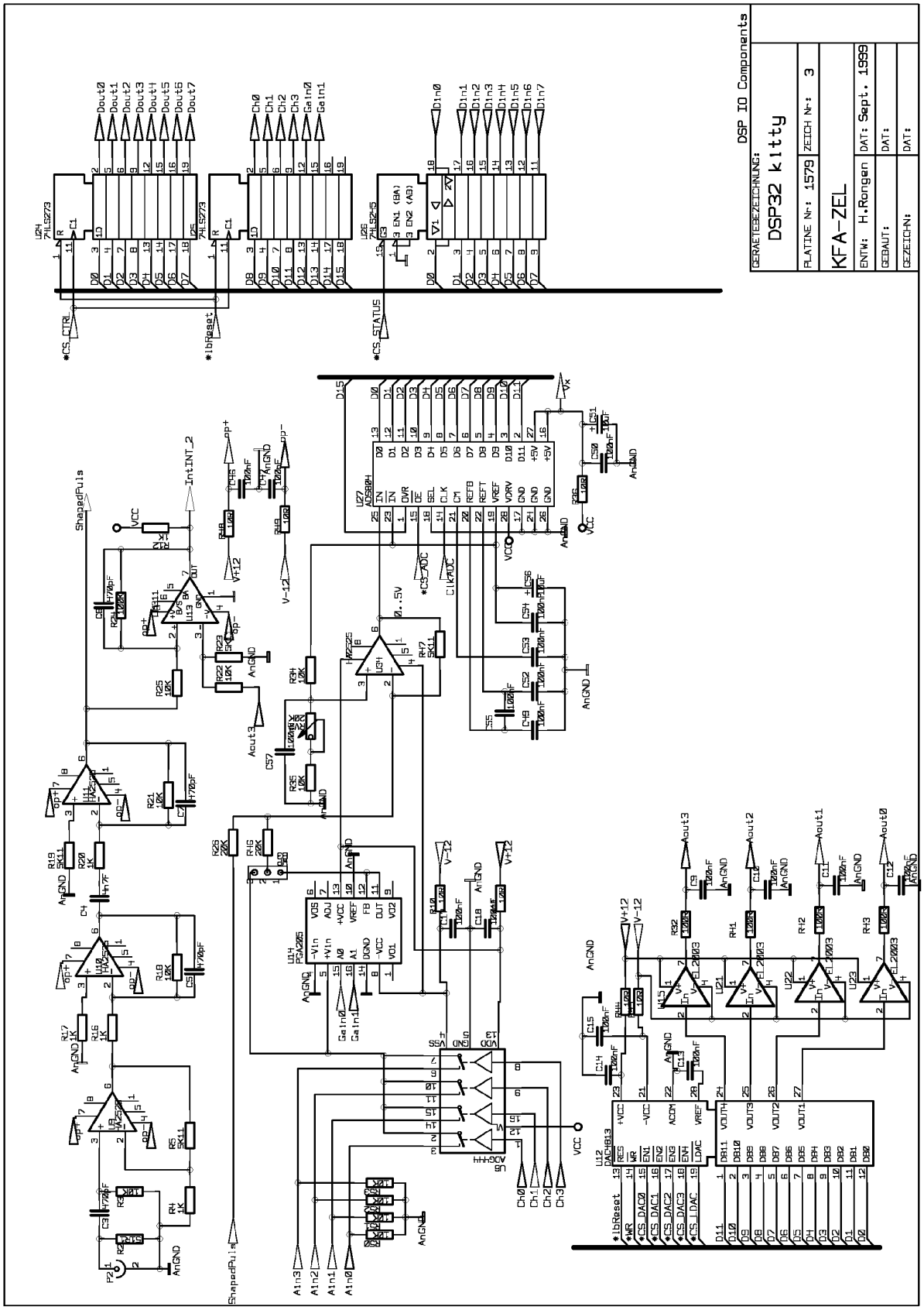
Circuit Diagram (DSP32c Digital Signal Processor)



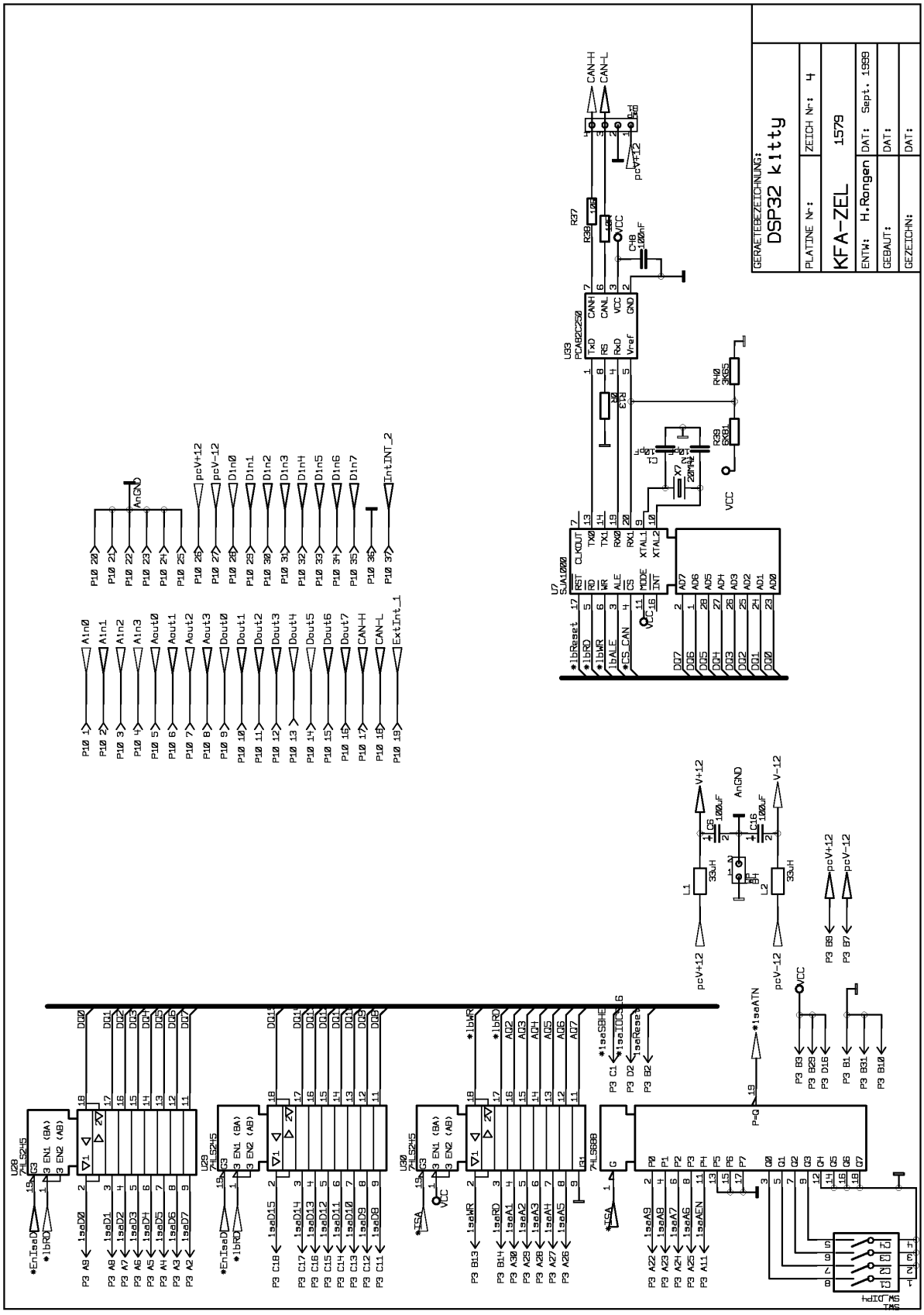
Circuit Diagram (Memory)



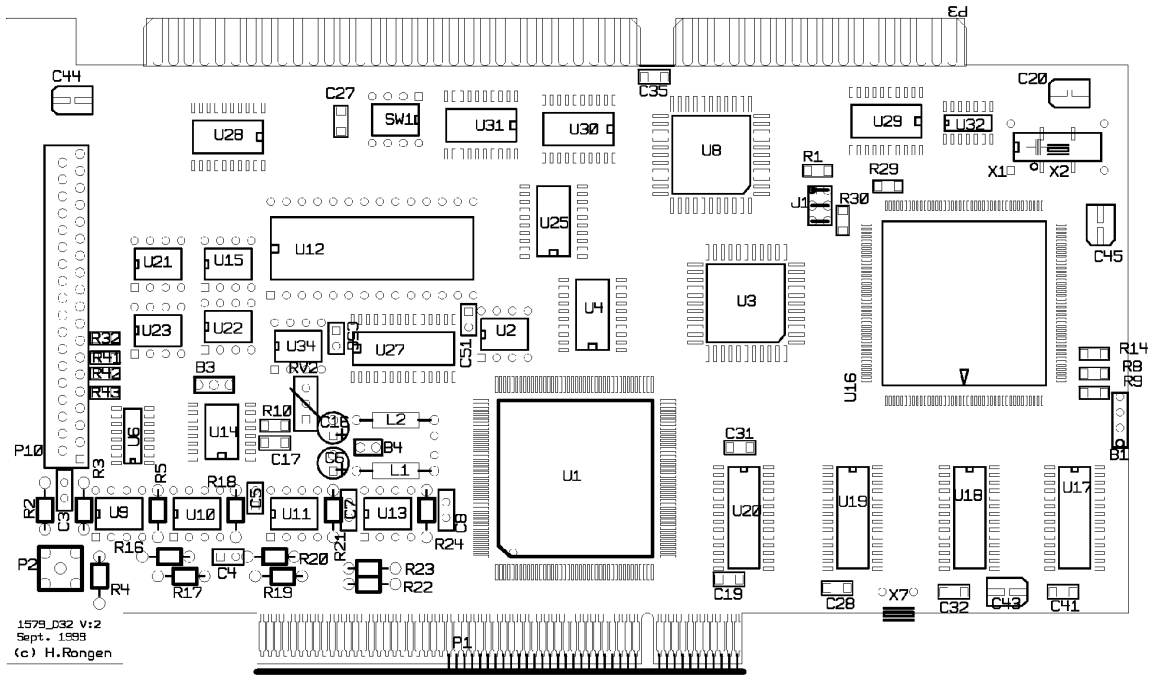
Circuit Diagram (Analog and digital I/O)



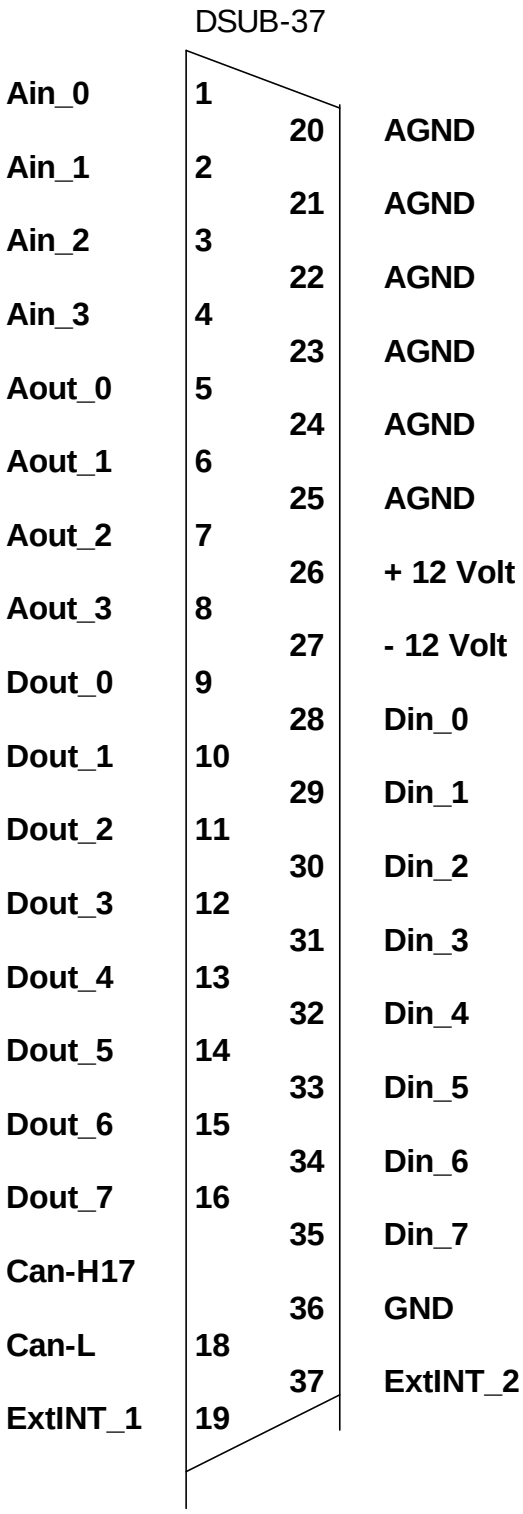
Circuit Diagram (ISA Bus-Interface, connector, CAN-Interface)



2.1. Component layer



3. Connector (P10)



4. Installation

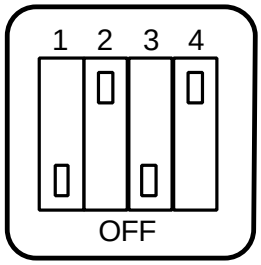
The D32Kitty board is a PCI and ISA compatible PC-slot board. The board automatic detects the used bus. The driver software is fully transparent for both interfaces.

Installation:

? ?open the PC
? ?check for a free PCI or ISA 16 bit slot
? ?change the D32Kitty slotpanel for the selected bus
? ?if ISA, select a proper I/O Base address
? ?insert D32Kitty in the PC bus slot
? ?fasten the slotpanel screw
? ?close the PC

For ISA Bus: select a proper I/O Base address:

The D32Kitty board is mapped in the I/O address area of the host CPU and uses 64 bytes of the I/O area starting a base address. This base address is selectable with the onboard DIP switches and must be unique (not used by another hardware) in the whole system.



SW 1

S4 : Adressline A6
S3 : Adressline A7
S2 : Adressline A8
S1 : Adressline A9

The default base address is 280 hex.

Examples:

Baseaddress	200h : off, on, on, on
	240h : off, on, on, off
	280h : off, on, off, on
	2C0h : off, on, off, off
	300h : off, off, on, on
	340h : off, off, on, off

4.1. Spezifikationen:

? ?ISA bus Interface
- PC-ISA Bus
- 16 Bit Databus
- IO mapped, 64 bytes addressarea

? ?PCI bus Interface
- IO mapped, 256 bytes addressarea
- 16 bit databus
- AMCC5944 PCI controller („Path Thru Slave Mode“)
- VendorID: 2810 hex
- ProductID: 8011 hex
- BaseClass: 0B hex Processor)
- SubClass: 80 hex

? ?Digital Signalprocessor
- Lucent „DSP32c“
- 32 bit Floatingpoint processor
- 20 MIPS (16 bit Integer arithmetic)
- 40 MFLOPS (32 bit Floating point arithmetic)

? ?Memory
- 128 Kbytes Static RAM
- 12 ns access time
- Dual ported for DSP and PC

? ?I/O Components
- 4 analog Outputs
  - one four channel DAC (DAC-4813, BurrBrown)
  - 1 Mhz (1 µs)
  - +/- 10 Volt, 50 Ohm Outputs
- 4 analog Inputs
  - one 12 bit ADC (ADS804, BurrBrown)
  - 5 Mhz (200 ns)
  - Programmable Gain Amplifier (Gain=1, 2, 4, 8)
  - analog input multiplexer
  - four Inputs (+/- 10, 5, 2.5 or 1.25 Volt)
- 8 Bit digital OUT (TTL Level)
- 8 Bit digital IN (TTL Level)

? ?Main amplifier and Puls shaper
- One input for nuclear pulses
- Positive polarity
- 5 µs shaping time
- Low Level Discriminator for Interrupt generation

5. Programming the D32Kitty

```
int     WINAPI D32Kitty (int function, int value);

void    WINAPI D32k_StopDSP          (int D32board);
void    WINAPI D32k_StartDSP         (int D32board);
BOOL    WINAPI D32k_InitDSP          (int D32board);

BOOL    WINAPI D32k_CoffLoad         (int D32board, LPSTR fn, BOOL start);
long    WINAPI D32k_GetAdr           (int D32board, LPSTR LabelName, LPSTR DspProgName);
void    WINAPI D32k_SetMemPointer    (int D32board, int dspMemAdr);
BOOL    WINAPI D32k_SetMemPointerLabel (int D32board, LPSTR LabelName, LPSTR DspProgName);
int     WINAPI D32k_GetMemPointer    (int D32board);

void    WINAPI D32k_WriteInt         (int D32board, int dspMemAdr, int value);
int     WINAPI D32k_ReadInt          (int D32board, int dspMemAdr );
float   WINAPI D32k_ReadFloat        (int D32board, int dspMemAdr );
void    WINAPI D32k_WriteFloat       (int D32board, int dspMemAdr, float value );

void    WINAPI D32k_RdDSPblkInt16    (int D32board, short *daten, int anzahl);
void    WINAPI D32k_WrDSPblkInt16    (int D32board, short *daten, int anzahl);
void    WINAPI D32k_RdDSPblkInt32    (int D32board, int   *daten, int anzahl);
void    WINAPI D32k_WrDSPblkInt32    (int D32board, int   *daten, int anzahl);
void    WINAPI D32k_RdDSPblkFloat    (int D32board, float *daten, int anzahl);
void    WINAPI D32k_WrDSPblkFloat    (int D32board, float *daten, int anzahl);

WORD    WINAPI D32k_GetPCR           (int D32board);
int     WINAPI D32k_ReadPIR          (int D32board);
BOOL    WINAPI D32k_IsPIR            (int D32board);
int     WINAPI D32k_WaitPIR          (int D32board, DWORD timeoutMS);


//- D32Kitty IO components are mapped into DSP32c memory at Offset 0x60.xxxx -
WORD    WINAPI D32k_ReadIOPage       (int D32board, int Reg );
void    WINAPI D32k_WriteIOPage      (int D32board, int Reg, WORD Val );
#define  RDADC                       0x600000
#define  WRDAC0                      0x600000
#define  WRDAC1                      0x600040
#define  WRDAC2                      0x600080
#define  WRDAC3                      0x6000c0
#define  WRLDAC                      0x600100
#define  WRPARIO                     0x600140
#define  RDPARIO                     0x600140

void    WINAPI D32k_SetPio           (int D32board, int d);
void    WINAPI D32k_SetPioBit        (int D32board, int bit);
void    WINAPI D32k_ClrPioBit        (int D32board, int bit);

int     WINAPI D32k_GetPio           (int D32board);
BOOL    WINAPI D32k_IsPioBitSet      (int D32board, int bit);

void    WINAPI D32k_SetDAC           (int D32board, int channel,  int value);

void    WINAPI D32k_SetChannelGain   (int D32board, int channel, int Gain);
#define AinChannel0          0x0E00
#define AinChannel1          0x0D00
#define AinChannel2          0x0B00
#define AinChannel3          0x0700
#define AinChannelOff        0x0f00
#define Gain1                0x0000
#define Gain2                0x1000
#define Gain4                0x2000
#define Gain8                0x3000

int     WINAPI D32k_GetADC           (int D32board);
```

The Parameter D32board describes the board to be used, if more then one board is installed in the computer. Normaly this parameter is always 0 (zero).

void WINAPI D32k_StopDSP (int D32board);
the DSP will be stopped.

void WINAPI D32k_StartDSP (int D32board);
the DSP starts execution of the programm at memory address 0000000.

BOOL WINAPI D32k_InitDSP (int D32board);
the Dsp will be initalized.

BOOL WINAPI D32k_CoffLoad (int D32board, LPSTR fn, BOOL start);
a DSP programm (generated with the Dsp32 Support Software) will be downloaded to the specified DSP. The parameter **fn** gives the path and name of the Dsp programm on the disk. If **start** is true, the DSP will be started after download.

long WINAPI D32k_GetAdr (int D32board, LPSTR LabelName, LPSTR DspProgName);
this calls returns the address of a Label (**Labelname**) in a Dsp programm.This address can be used to access the data of the variable with the name **labelname**.

void WINAPI D32k_SetMemPointer (int D32board, int dspMemAdr);
the memorypointer will be set to address **dspMemAdr**. This call is used to initialize the Dsp memory access from the host side.

BOOL WINAPI D32k_SetMemPointerLabel (int D32board, LPSTR LabelName, LPSTR DspProgName);
this is a combination of *"D32k_GetAdr"* and *"D32k_SetMemPointer"*. The memorypointer will be set to the address given by **Labelname** of the Dsp program **DspProgName.**

int WINAPI D32k_GetMemPointer (int D32board);
this call returns the actual address of the memorypointer.

void WINAPI D32k_WriteInt (int D32board, int dspMemAdr, int value);
writes one Integervalue (**value**) to the Dsp memory at address **DspMemAdr**. If DspMemAdr is –1, then the current address of the memorypointer will be used. (Set by a former call to D32k_SetMemPointer.)

int WINAPI D32k_ReadInt (int D32board, int dspMemAdr);
reads one Integervalue from the Dsp memory at address **DspMemAdr**. If DspMemAdr is –1, then the current address of the memorypointer will be used. (Set by a former call to D32k_SetMemPointer.)

void WINAPI D32k_WriteFloat (int D32board, int dspMemAdr, float value);
writes one Floating point value (**value**) to the Dsp memory at address **DspMemAdr**. If DspMemAdr is –1, then the current address of the memorypointer will be used. (Set by a former call to D32k_SetMemPointer.)

float WINAPI D32k_ReadFloat (int D32board, int dspMemAdr);
reads one Floating point value from the Dsp memory at address **DspMemAdr**. If DspMemAdr is –1, then the current address of the memorypointer will be used. (Set by a former call to D32k_SetMemPointer.)

void WINAPI D32k_RdDSPblkInt16 (int D32board, short *data, int count);
reads a block of Integer values from the Dsp memory. The address must be set by a call to D32k_SetMemPointer. **Count** values will be read into a local buffer of short integers (16 bit), pointed to by **data**.

void WINAPI D32k_WrDSPblkInt16 (int D32board, short *data, int count);
writes a block of Integer values to the Dsp memory. The address must be set by a call to D32k_SetMemPointer. **Count** values will be written to the Dsp, from a local buffer of short integers (16 bit), pointed to by **data**.

void WINAPI D32k_RdDSPblkInt32 (int D32board, int *data, int count);
reads a block of Integer values from the Dsp memory. The address must be set by a call to D32k_SetMemPointer. **Count** values will be read into a local buffer of long integers (32 bit), pointed to by **data**.

void WINAPI D32k_WrDSPblkInt32 (int D32board, int *data, int count);
writes a block of Integer values to the Dsp memory. The address must be set by a call to D32k_SetMemPointer. **Count** values will be written to the Dsp, from a local buffer of long integers (32 bit), pointed to by **data**.

void WINAPI D32k_RdDSPblkFloat (int D32board, float *data, int count);
reads a block of floating point values from the Dsp memory. The address must be set by a call to D32k_SetMemPointer. **Count** values will be read into a local buffer of float variables (32 bit), pointed to by **data**.

void WINAPI D32k_WrDSPblkFloat (int D32board, float *data, int count);
writes a block of floating point values to the Dsp memory. The address must be set by a call to D32k_SetMemPointer. **Count** values will be written to the Dsp, from a local buffer float variables (32 bit), pointed to by **data**.

WORD WINAPI D32k_GetPCR (int D32board);
this call returns the current state of the Dsp Processor Controll Register (PCR). This Register is bit encoded. See the DSP32c Manual.

BOOL WINAPI D32k_IsPIR (int D32board);
this call checks if the Dsp Processor Interrupt Register (PIR) has been written by the DSP. (Synchronisation of DSP and PC)

int WINAPI D32k_ReadPIR (int D32board);
this call returns the current value of the Dsp Processor Interrupt Register (PIR). The Dsp can write to this register. Writing to this register signals the PC that a action must be performed. (Synchronisation of DSP and PC)

int WINAPI D32k_WaitPIR (int D32board, DWORD timeoutMS);
this call waits for a maximal time (timeoutMS) for the PIR event. (Synchronisation of DSP and PC)

D32Kitty IO components are mapped into DSP32c memory at Offset 0x60.xxxx

WORD WINAPI D32k_ReadIOPage (int D32board, int Reg);
read a memory mapped IO address.

void WINAPI D32k_WriteIOPage (int D32board, int Reg, WORD Val);
write to a memory mapped IO address.

```
#define RDADC       0x600000
#define WRDAC0      0x600000
#define WRDAC1      0x600040
#define WRDAC2      0x600080
#define WRDAC3      0x6000c0
#define WRLDAC      0x600100
#define WRPARIO     0x600140
#define RDPARIO     0x600140
```

void WINAPI D32k_SetPio (int D32board, int d);
write **d** to the digital outputs.

void WINAPI D32k_SetPioBit (int D32board, int bit);
set the **bit** (0..7) of the digital output.

void WINAPI D32k_ClrPioBit (int D32board, int bit);
clear the **bit** (0..7) of the digital output.

int WINAPI D32k_GetPio (int D32board);
read the digital inputs.

BOOL WINAPI D32k_IsPioBitSet (int D32board, int bit);
Check if **bit** of the digital input is set.

void WINAPI D32k_SetDAC (int D32board, int channel, int value);
write a **value** (-2048 .. +2047) to the DAC **channel** (0..3). This results in a analog voltage of –10 .. + 10 Volt at the output of the DAC channel.

void WINAPI D32k_SetChannelGain (int D32board, int channel, int Gain);
set the **channel** (0..3) and the **Gain** for next ADC conversion.

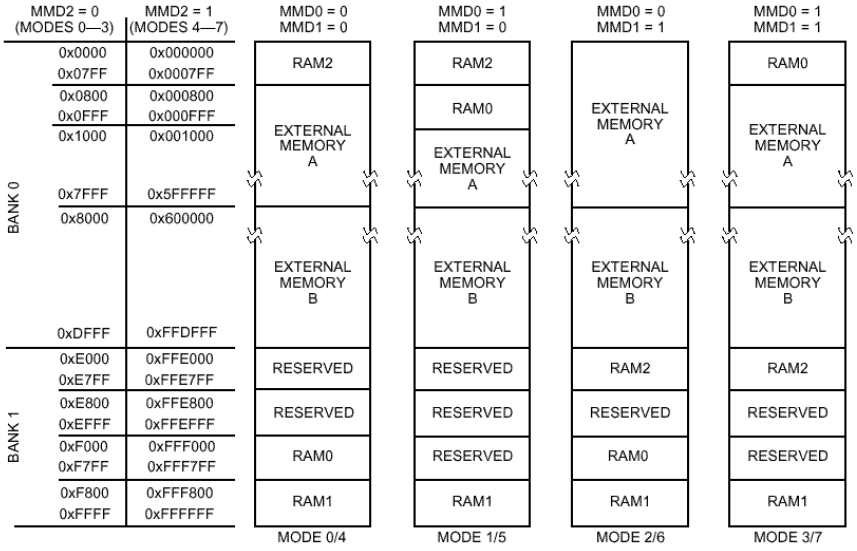
```
#define AinChannel0     0x0E00
#define AinChannel1     0x0D00
#define AinChannel2     0x0B00
#define AinChannel3     0x0700
#define AinChannelOff   0x0f00
#define Gain1           0x0000
#define Gain2           0x1000
#define Gain4           0x2000
#define Gain8           0x3000
```

int WINAPI D32k_GetADC (int D32board);
reads the current ADC value. (-2048 .. +2047)

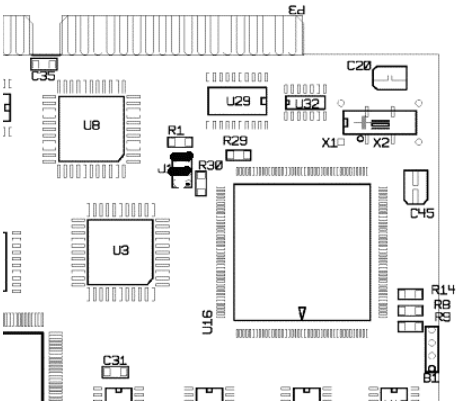
6. D32Kitty DSP Resourcen

6.1. DSP32c Memory

DSP32c has internal 6 Kbyte RAM. Onboard is additional 128 Kbyte, fast static Ram.
The DSP32c couldbe configured to different Memory modes. For the D32Kitty we choose memory mode 4.



For this the Jumper J1 should be set to:



In this mode we have:

```
2 Kbyte, internal fast Ram    at address 0x000000..0x000800
126 Kbyte, external SRAM      at address 0x000800..0x020000
4 Kbyte, internal fast Ram    at address 0xFFF000..0xFFFFFF
```

We use the lower Ram for the .text (program code) and .data section. The upper memory is used for the .bbs (Stack) section.
This result in the following memory map file for the DSP32c Support Software:

```
MEMORY {
     .mtext:        o=0x000000, l=0x00800
     .mdata:        o=0x000800, l=0x02000
     .mbss:         o=0xfff000, l=0x01000
}

SECTIONS {
     .text: {} > .mtext
     .data: {} > .mdata
     .bss:  {} > .mbss
}
```

The wait States for the memory banks are define in the Processor Control Word (PCW)

6.1.1. Processor Control Word

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Field	INTER						PEND	MGN	PIOPH	PIOPL	MEMA		MEMB		WA	WB

Bit(s)	Field	Description
0	WB	If 0, disable wait-state generator for external memory partition B. If 1, enable wait-state generator for external memory partition B.
1	WA	If 0, disable wait-state generator for external memory partition A. If 1, enable wait-state generator for external memory partition A.
3, 2	MEMB	These bits select the number of wait-states that will be generated for the external memory in partition B. 00 — 1 wait-state* 01 — 2 wait-states 10 — 3 wait-states 11 — 2 or more wait-states (controlled by SRDYN signal)
5, 4	MEMA	These bits select the number of wait-states that will be generated for the external memory in partition A. 00 — 1 wait-state* 01 — 2 wait-states 10 — 3 wait-states 11 — 2 or more wait-states (controlled by SRDYN signal)
6	PIOPL	If 0, PIOP3—PIOP0 are inputs. If 1, PIOP3—PIOP0 are outputs when **pcr[9]**(PIO16) = 0.
7	PIOPH	If 0, PIOP7—PIOP4 are inputs. If 1, PIOP7—PIOP4 are outputs when **pcr[9]**(PIO16) = 0.
8	MGN	If 0, the MGN pin acts as a memory output enable signal. If 1, the MGN pin is used by the bus arbitration protocol to indicate that an external memory access is pending.
9	PEND	If **pcw[8]**(MGN) = 0, this bit is not used. If **pcw[8]** = 1, the logical value of this bit is ORed with an internal signal that indicates an external access is pending. It is then inverted to produce the signal at the MGN pin (i.e., if **pcw[8]**(MGN) = 1 and **pcw[9]**(PEND) = 1, then the pin MGN = 0).
15—10	INTER	Each bit in this field corresponds to one of the six sources for an interrupt. A value of 1 in a position enables the corresponding interrupt source; a value of 0 disables the source. Bit — Interrupt — Source 10 — INTREQ2 — Interrupt 2 pin 11 — OBE — Serial output buffer empty 12 — IBF — Serial input buffer full 13 — PDE — Parallel data empty (output) 14 — PDF — Parallel data full (input) 15 — INTREQ1 — Interrupt 1 pin

6.1.2. The memory mapped IO Components

The D32Kitty IO components (ADC, DAC, digital IO) are memory mapped in the DSP32c address space. The Baseaddress of the IO components is 0x600000.

The file `<d32kitty\d32kitty.h>` describes these IO components addresses:

```
#define RDADC  0x600000      // Read the ADC

#define WRDAC0 0x600000      // write to DAC 0
#define WRDAC1 0x600040      // write to DAC 1
#define WRDAC2 0x600080      // write to DAC 2
#define WRDAC3 0x6000C0      // write to DAC 3
#define WRLDAC 0x600100      // DAC load impuls

#define WRPIO  0x600140      // write digital output
#define RDPIO  0x600140      // read digital input
```

IO components are accessed by setting up a pointer to the resource:

```
r1e = WRPIO         /* register 1 points to the digital output
register*/
r2  = 0xf0ff   /* register 2 is loaded with the data */
*r1 = r2       /* write the value of r2 to the mem location
               /* pointed to by r1 */
```

In „C“ this is done by accessing the memory location with a pointer:

```
#include <d32kitty\d32kitty.h>

     *(short*)WRPIO = 0xf0ff;
```

6.2. DSP32c Interrupts

Beside the DSP32c internal Interrupts (Interrupts on parallel and/or serial port register) the D32Kitty board has two external Interrupts, usable by the user.
Both Interrupt lines are available on the connector P10. (pin 19 and 37)

- INTREQ1: external interrupt from connector P10, pin 19
- INTREQ2: external interrupt from pulse Discriminator

The external interrupts are sensitiv on the rising edge!

Interrupts are programmed this way:

```
#include <d32kitty\d32kitty.h>

#define IVTP   r22e                /* Interrupt Vector Table */

/*-------------------------------------------------------------
--*/
.rsect          ".text"

Init:           r1   = 0x0013      /* memory wait states */
          pcw  = r1

          IVTP = IntTable
          r1   = pcw
          nop
          r1   = r1 | 0x8000       /* enable INT1 */
          pcw  = r1

          goto Start
          nop

/******* the Interrupt Vector Table
*****************************/
IntTable: goto ExInt1                   /* Ext. Interrupt 1 */
          nop
          ireturn                  /* PIO Buffer full  */
          nop
          ireturn                  /* PIO Buffer empty */
          nop
          ireturn                  /* SIO Input Buffer full*/
          nop
          ireturn                  /* SIO Output Buffer empty*/
          nop
          ireturn                  /* Ext. Interrupt 2 */
          nop

/*----Interrupt Service Routine for External INT1: -----------
--*/
ExInt1:   r1e = WRPIO
          r2  = 0xf0ff
          *r1 = r2
```

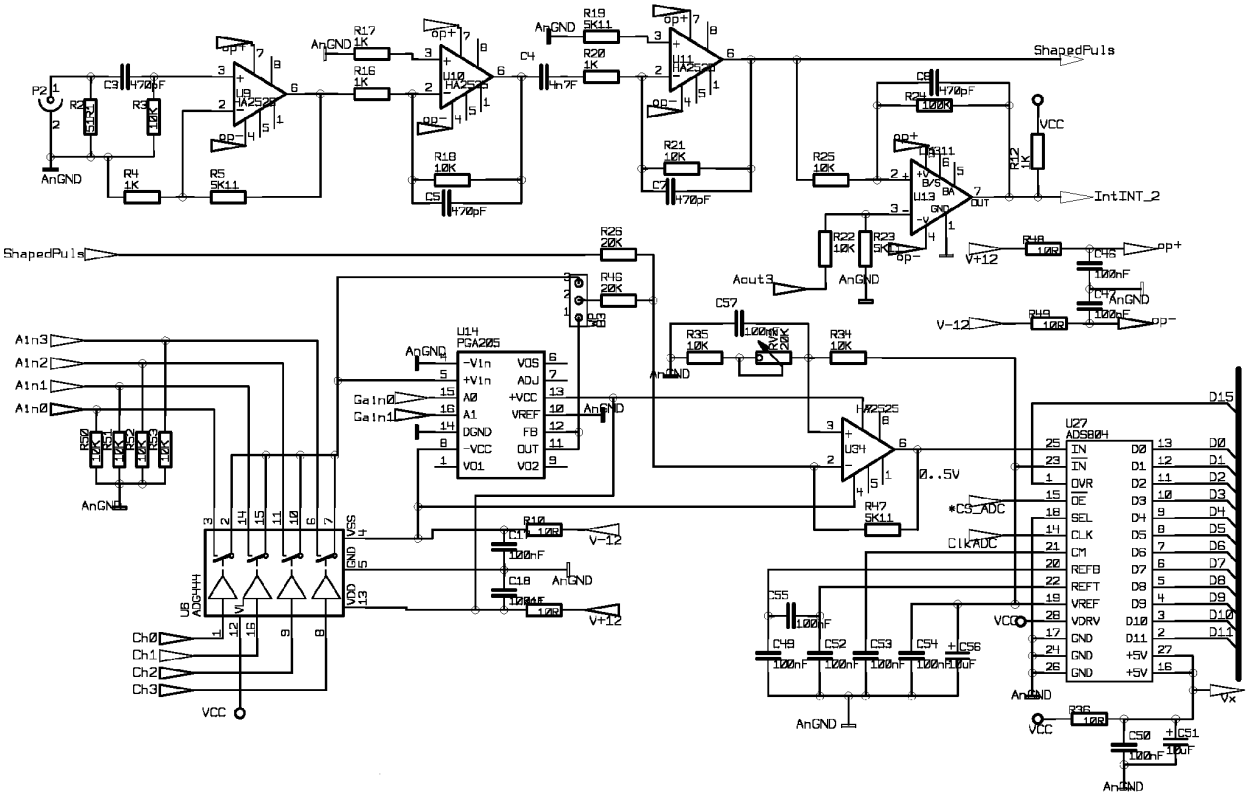
```
            r2  = 0xf000
            *r1 = r2
            ireturn
            nop

    /*-------- Mainprogram is only a infinite Loop ---------------
    -----*/
    Start:
    LOOP:          goto LOOP;          nop
```

6.3. D32Kitty Analog Inputs

D32Kitty has 4 general purpose analog inputs (+/- 10 Volt) and one input for nuclear pulses.

The four analog inputs are multiplexed (U6) and amplified with a PGA (Programmable Gain Amplifier, U14). Behind the PGA the puls signal is added and the signal is offset-shifted to fit the ADC (U27) input specification of 0..5 Volt.

The shaper for the nuclear pulses consist of a 2 level Differentiator and a 2 level Integrator (U9, U10, U11) with a shaping time about 5 µs. The shaped puls is feed into a Discriminator (Comparator, U13). The discriminator level is set by the DAC channel 3. The discriminator output feeds the DSP External Interrupt 2.



The ADC is running with a constant clock of 10 Mhz. So the DSP could read a new digitized value every time, without polling any status bit.

The current analog Input channel and the gain for the next conversion is set by the parallel output register.

The ADC is memory mapped at the address 60_0000 hex.

```
short *RDADC = 0x600000;
value = *RDADC;
```

Any read at this address returns a new digitized value of the current selected analog input channel. Because of the linear bit coding of the ADC, the value read must be converted into a signed integer number by „2047 – value“.

a typical Assembler routine for reading the ADC:

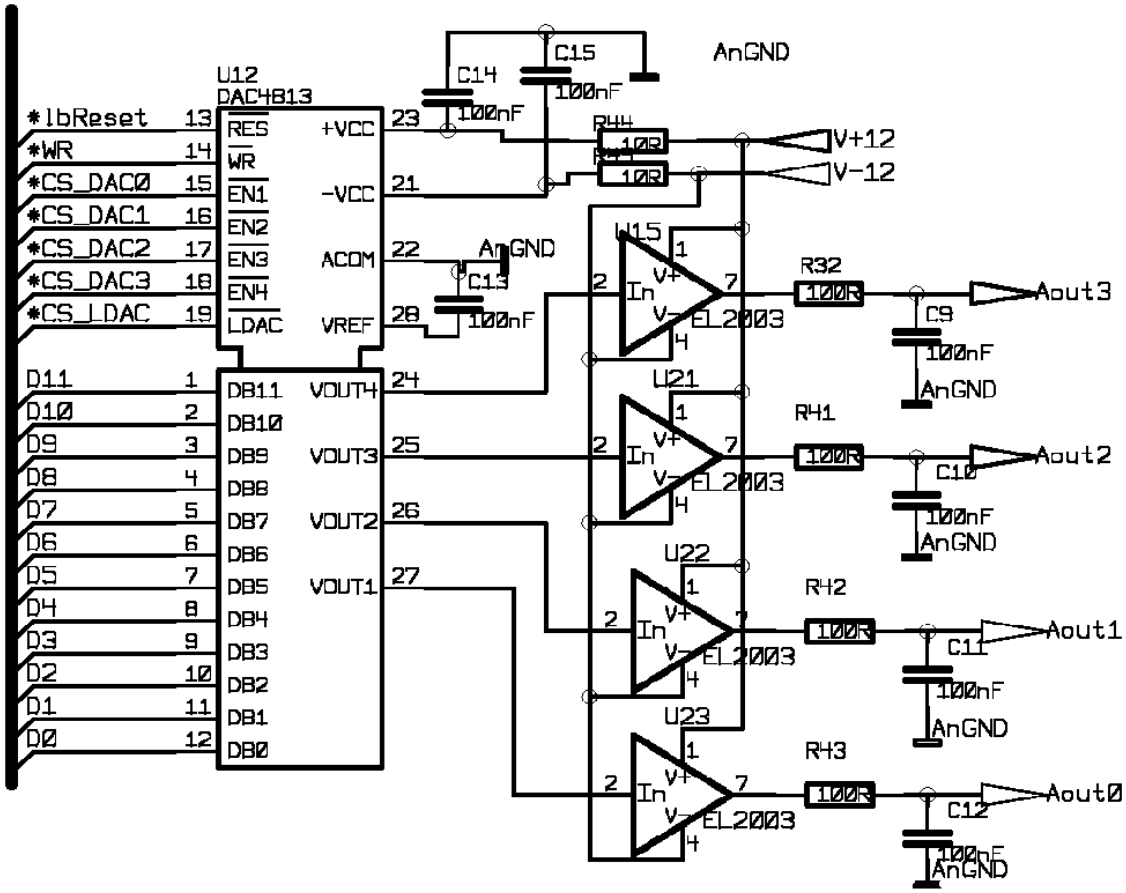
```
r1e  = RDADC
r2 = *r1
nop
r2 = r2 & 0x0fff    /* 0..4096 */
r2 = 2047 – r2      /* -2048 .. 2047 */
```

a typical C routine for reading the ADC:

```
#define RDADC  0x600000

short GetADC ()
{
short d;
   d = *(short*)RDADC & 0xfff;
   return (2047 - d );
}
```

6.3. D32Kitty Analog Outputs

D32Kitty has four general purpose analog outputs, +/- 10 Volt. The DAC (U12) four outputs are feed via a 50 Ohm driver to the output pins of P10.



Channel Aout_3 is also used as the Discriminator level.

The DAC is memory mapped into the DSP at address:

```
#define WRDAC0 0x600000
#define WRDAC1 0x600040
#define WRDAC2 0x600080
#define WRDAC3 0x6000C0
#define WRLDAC 0x600100
```

every DAC channel has his own Dataregister:

```
WRDAC0, WRDAC1, WRDAC2, WRDAC3
```

after loading the data, the DAC need a strobe puls for transfering the new DAC data to the output. At this time the analog output is updated:

```
WRLDAC
```

a typical Assember program:

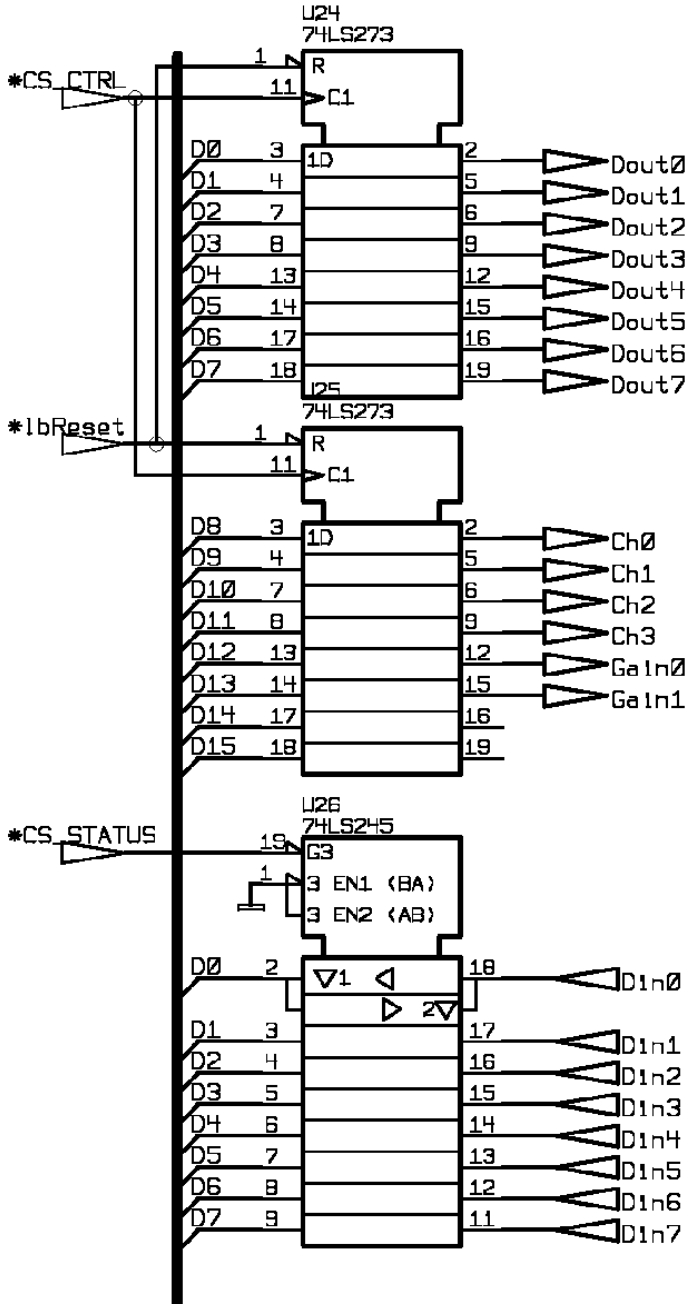
```
          r1   = 0
          r2e  = WRDAC0
          r3e  = WRLDAC

LOOP:     r4   = r1 + 2047
          r1   = r1 + 1
          *r2  = r4
          *r3  = r4

          goto LOOP
          nop
```

a typical C program:

```
#include <d32kitty\d32kitty.h>

void main ()
{
register short d;
  d = 0;
  while(1)
  {
    d = d + 1;
     *(short*)WRDAC0 = d+2047;
     *(short*)WRLDAC = 0;
  }
}
```

6.3. D32Kitty Digital IO

D32Kitty has a 16 bit digital output register and a 8 bit digital input register.
The lower 8 bits of the output register and the input register is feed to the connector P10 as general purpose digital (TTL level) IO pins.



The upper 8 bits of the output register controls the analog input multiplexer and the programmable Gain Amplifier:

Bit	Operation	by logic LOW = 0	by logic HIGH = 1
bit 8	select Channel 0	0 = Channel ON	1 = Channel OFF
bit 9	select Channel 1	0 = Channel ON	1 = Channel OFF
bit 10	select Channel 2	0 = Channel ON	1 = Channel OFF
bit 11	select Channel 3	0 = Channel ON	1 = Channel OFF
bit 12	Gain bit 0	| bit coded	
bit 13	Gain bit 1	| bit coded	
bit 14	free		
bit 15	free		

The Gain is binary bit coded:

binary code	Gain	analog input range
0	Gain = 1	+/- 10 Volt
1	Gain = 2	+/- 5 Volt
2	Gain = 4	+/- 2.5 Volt
3	Gain = 8	+/- 1.25 Volt

The digital IO register are memory mapped into the DSP at address:

```
#define WRPIO  0x600140
#define RDPIO  0x600140
```

a typical Assember program:

```
r1e  = WRPIO
r1   = r2
```

a typical C program:

```
#include <d32kitty\d32kitty.h>

void main ()
{
     * (short*)WRPIO = 0x01ff;
}
```