IrDA connection between an AVR and a Palm Pilot

Imagine the following practical example. It is necessary to measure some type of data somewhere outside and process them from time to time. A typical example is the building trade, where it is necessary to capture certain kinds of power at building parts, or capture the temperature curve to determine the maturity of concrete.
For this kind of applications it is necessary to create a device, which is very easy to handle with a minimum number of control buttons, and needs also to be strong. Therefore a display device is not recommendable due to the proneness to mechanical damages. The question is how to collected the data and process them? A wired connection is also excluded on a building stage. A wireless connection looks like the best solution, but the problem is that existing applications which guarantee a save data transfer are still in development, expensive or not useable world wide (e.g. DECT).
The most convenient seems to be infrared data transfer via the broadly used IrDA standard. For instants, all Palm Pilot systems and its clones have an infrared interface. In addition, they got a big display, which can be programmed with data. Therefore the ideal control-unit, and also protected in the pocket of the user.
He comes to the logger and with only a few touches on the touch screen of the Palm he transfers the collected data to its memory or the other way around, from the Palm e.g. configuration data are set and transferred to the sensing device.
This describes the idea behind our data logger based on the Atmel AVR and equipped with an infrared interface.

Today mobility is one of the key features. Available RF techniques like Bluetooth, DECT are complicated and require a lot of design experience in this area. In addition, development and devices are still expensive. Therefore we were looking for an easy way for design and implementation of data transmission. Let's say from a sensing system to a Palm Pilot, because of the fact that Palm Pilots are used broadly.

Our design shows the possibility of communication between an Atmel AVR and a PC or Palm Pilot via IrDA. This kind of embedded solution of infrared communication with a Palm is ideal for applications where for instance parameters of a system need to be set or for data acquisition.
The first example, setting system parameters, is also interesting under the aspect of using the AVR processor as the configuration memory.
Handling can be done with small and light equipment based on PalmOS. In addition, typical issues of cable connections like different connectors, reduction, etc. doesn't exist.

The described solution works up to a speed of 115 kbps (for Palm hand-held only up to 56.2 kbps given by the Palm) and uses in addition to normal IrDA transceiver also a 3/16 Encoder-/Decoder-IC HSDL7001.
This IC could be replaced by a software solution, but this would require an interrupt input and reduces the speed to max. 9.6 kbps.

The IrDA AVR Stack is build up by low-level layers like framing layer, IrLAP, IrLMP, and also the high-level IrCOMM protocol for the secondary only device.

IrLAP reacts on the Discovery paket (XID) and takes care about activating the connection. After connecting it operates the Supervisory packets and controls the Information packets.
IrLMP has minimum implementation. It does the multiplex between two services, which are IAS and IrCOMM.

Code of this solution requires ca. 3KB, therefore it is possible to store it at the following AVR types: 4414, 4433,4434, 8515, 8534, 8535, and all Atmega types.

In practical, strain gauge data are collected in pre-set intervals (defined by the user) and written into the dataflash.
This is possible by using in addition to the AVR itself and the circuits for the Ir-physical-layer also a DATAFLASH AT45D041, RTC DS1302 and a precision analog part made by 3 x a differential amplifier, the 24-bit analog-to-digital converter LTC2400.

By using an easy Palm demo-program it is possible to set the real time of the system in the RTC, the interval of data collection and also write the measured data stored in the Dataflash to the Palm memory.

The whole system is design as a low-power solution and therefore only to by powered by a battery. The analog part is connected directly, all other circuits by a low-dropout stabilizator of 3.3 V.
The stage of the battery is controlled in set time intervals. The RTC is backed-up by a Supercap and used mainly to bridge the time while changing the battery.
The analog part is connected to the supply voltage only at the measurement times. A bistable relay is used as the switch.
Together with other loads like the LED, buzzer, etc. it is handled via a serial shift register TPIC6595, which has a 250 mA, sink capability.

Picture of the whole system

Circuit description

The core is an Atmega 103L with a 3,6864-MHz tact, a 3/16 encoder/decoder HSDL7001 with its own generator of clock impulses. The HSDL-7001 modulates and demodulates electrical pulses from HSDL-1001 Infrared transceiver module. The speed of the transfer is switched by the three inputs A0, A1 and A2. The IR transceiver circuit is based on the HSDL1001 (both Agilent). During sleep mode both circuits are consuming very low power (shut down). An external IR diode can be added to extend the range.
The HW SPI of the ATmega connects the microcontroller and the Dataflash AT45D041 memory. The AT45D041 is the first device in the Serial DataFlash family comprising 2048 equal length pages. The AT45D041's page size is 264 bytes, rather than 256 bytes. Therefore, the total density (4,325,376 bits) of the device is 128K bits larger than 4M bits. The AT45D041 incorporates two on-chip, bi-directional buffers to expedite the flow of data to and from the device. These buffers provide a built-in "pseudo" cache memory and allow the AT45D041 device to receive data during erase/program operations. Each buffer is 264 bytes long, the same size as a Flash page, and function independently from each other.

Using a software based SPI, to the ATmega is connected a RTC DS1302 which is backed-up with a Supercap. It would be possible to use instead of the RTC also the Timer2, but this was kept as a resource for future extensions.
Another circuit, which is SW SPI connected/handled, is the shift register TPIC6595, eight DMOS switches with the possibility of a continuous sink of 250 mA per channel. This circuit mainly controls a bi-stable relay for the power supply of the analog part and the sensors. The relay is used for the reason of higher loads for different types of sensors.
The TPIC also controls two LED diodes. LED1 flashes shortly in a 2-second tact during active mode. LED2 indicates the IrDA transfer. Another signaling device, which is controlled via this logic shift register, is a buzzer for confirmation of the function when using the control buttons.
The buttons to start and activate the measurement initiate external interrupts.
The analog part is a single board to increase the flexibility for further extensions, e.g. other types of sensors, which require different conversion rate, resolution, etc. The analog part (see schematics Precision Analog Part) consists of a 3 times differential to single ended bridge amplifier done by three INA122s. The precision virtual ground TLE2426 supplies bias voltage for level shifting. Each signal is further filtered through a low pass filter and converted via a 24-bit sigma-delta analog-to-digital converter LTC2400. The software created SPI controls these converters. The serial resistors in the SPI

control line are integrated to avoid interferences and for digital level shifting. The INA and LTC devices are powered directly from a battery voltage. All other devices of the whole system are using 3.3-V voltage.
In addition, on this analog part is integrated a voltage reference LT1019-2.5 V for reference of the battery voltage measurement. The measurement of this constant voltage (1.00-V divider) is done on the input of ADC0 of the integrated 10-bit converter, and the reference voltage of this converter is obtained from the non-regulated battery voltage.
For lower requirements it is possible to use instead the 10-bit, 8-channel built-in AD converter.

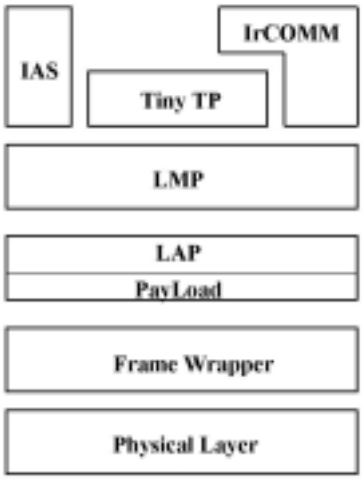
The system is powered from 4 cells (not over-passing 5.5 V). The core is supplied from a LM2931 low-dropout regulator, which is set for 3.3 V. The voltage supervisor MAX809 secures the reliable start of the microprocessor.

Main board with IrDA adapter

For detailed schematics of the whole system see schematics at the end of this document.

Implementation of the IrDA lite stack on the AVR

The major issue of the system design was to implement the IrDA Lite stack for AVR processors. The reason behind programming in Assembler was to save resources that the use of 4-kB program memory devices would be possible. The implementation is based on the IrDA Lite specifications for a secondary only device with support for connection speeds up to 115200 bps and uses the AVR HW UART.



IrDA protocol stack

Physical Layer

IR communication is inherently half-duplex.

The physical layer consists of an IrDA UART to convert octets of data to an IR pulse stream and to process the received pulse stream into octets of data. The bit format is 1start bit, 8 data bits and 1 stop bit (no parity).

On the hardware side it consists of the IR transceiver HSDL 1001, 3/16 encoder/decoder IC HSDL7001 and the HW UART of the Atmega. The Interrupt UART Receive Complete, UART Transmit Complete of the AVR initiates the processing of the IrDA stack into the higher layers.

Frame Wrapper

Frame Wrapper detects the beginning of the frame, deletes the symbols BOF, EOF, CE from the frame and calculates the 16-bit CRC. Into the PayLoad layer transfers only the information data from the frame. When finishing the frame informs the PayLoad about the validity, or about error within the information data.
In the transfer stage it is creating the initial beginning-of-frame (BOF) bytes, calculating CRC, securing data transparency, and finishing the frame with sending the CRC and EOF.

State diagram Frame Wrapper Receive

Current State	Event	Action	Next State
IDLE	fwData receive	IF fwData==BOF	START
		IF fwData==other	IDLE
START	fwData receive	IF fwData==BOF	START
		IF fwData==EOF	IDLE
		Init CRC Init FIFO	
		IF fwData==CE	CE
		Calc new CRC put fwData to PayLoad Layer throw 2byte FIFO	DATA
DATA	fwData receive	IF fwData==BOF	START
		IF fwData==CE	CE

		IF fwData==EOF THEN{
		IF CRC==0xF0B8 THEN send fw_plRxCorrect
		ELSE send fw_plRxError
		}
			IDLE

		Calc new CRC put fwData to PayLoad Layer throw 2byte FIFO	DATA
CE	fwData recieve	fwData=fwData XOR 0x20 Calc new CRC put fwData to PayLoad Layer throw 2byte FIFO	DATA
Any	pl_fwRxAbort		IDLE

PayLoad Layer

The PayLoad layer is part of the LAP Layer, controls the LAP address of the frame and analyses the individual LAP instructions for the slave. It reports the beginning and the end of receiving the frame to the LAP layer. If tak set, it transfers information data to the LMP layer.
In the send stage it generates the LAP frame (ADDRESS, COMMAND, DATA).

State Diagram PayLoad Layer Receive

Current State	Event	Action	Next State
ADDRESS	plData receive	IF plData==plConnAddr	COMMAND
		IF fwData!=plConnAddr THEN pl_fwRxAbort	ADDRESS
COMMAND	plData receive	IF P==0 THEN pl_fwRxAbort	ADDRESS

		IF Information frame THEN pl_lapRxFrame	IFRAME
		IF Supervisory frame THEN pl_lapRxFrame	WAIT
		IF XID frame THEN pl_lapRxFrame	XID
		IF SRNM frame THEN pl_lapRxFrame	SRNM
		IF DISC frame THEN pl_lapRxFrame	WAIT
		IF other frame THEN pl_lapRxFrame	WAIT
IFRAME	plData recieve	IF lapSendToLMP==True THEN send plData to LMP Layer (pl_lmpRxData)	IFRAME
XID	plData receive	IF plData==1 (format=1)	XIDSRC
		IF plData!=1 pl_fwRxAbort	ADDRESS
XIDSRC	plData receive	Store master address	XIDDST
XIDDST	plData receive	IF broadcast address (0xFFFFFFFF) or our address	XIDSLOT
		ELSE pl_fwRxAbort	ADDRESS
XIDSLOT	plData receive	slot:=plData	XIDVER
XIDVER	plData receive	IF plData==0	WAIT
		ELSE pl_fwRxAbort	ADDRESS
SNRM	plData receive	Connect Address:=plData	SNRMPAR
SNRMPAR	plData receive	IF plData==0x01 THEN set BaudRate	SNRMLEN
SNRMLEN	plData receive	plCounter:=plData	SNRMDAT

SNRMDAT	plData receive	IF BaudRate==True THEN {
		BaudRate:=plData
		BaudRate&=0x1E (115200 not supported)
		}
		Decrement plCounter
		IF plCounter==0
			SRNMPAR

		ELSE	SNRMDAT
WAIT	plData receive		WAIT
Any	fw_plRxCorrect	send pl_lapRxCorrect	ADDRESS
Any	fw_plRxError	send pl_lapRxError	ADDRESS
Any	lap_plRxAbort	pl_fwRxAbort	ADDRESS

LAP Layer

This layer decides which answer is sent to the master. LAP is moving in three stages:

NDM – Normal Disconnect Mode
The AVR is not connected to the master. It answers to the first discovery frame and can generate a new identification address if requested by the master. It also accepts the request for connection if realized by the SNRAM from the master.

NRM(S) - Normal Response Mode (Slave)
The AVR is connected. The connection is done in conjunction with the new communication parameters as agreed at both sides. In this stage the frames are transferred into the LMP layer and the numbering of the frames is controlled. In case of an error the LMP is requesting a repeated transfer of the last frame. Or LMP informs about an error within the current frame.

SCLOSE
The AVR is requesting Disconnect using the RD frame and now it is waiting for the end of the link DISC frame from the master.

Depending from the stage and from the received frame it is decided which answer is send. In the case of information frame the data are send into LMP or if sending, the data are requested from the LMP. In the case of TimeOut it controls the end of the link and the correct return to NDM.

Current State	Event	Action	Next State

NDM	pl_lapRxFrame	IF not (XID or SRNM frame) THEN lap_plRxAbort	NDM
	pl_lapRxCorrect	IF SRNM Frame THEN { set lapGoToNRM NsWaited:=0 NrWaited:=0 NrLast:=0 Set Timer0 1sec lap_plTxUARsp}	NDM

		IF XID Frame THEN{
		Set Timer0 500ms
		IF AddressConflictFlag==True THEN Generate New Device Address
		IF slot==0 THEN lap_plTxXIDRsp
		}
			NDM

	pl_lapTxComplete	IF lapGoToNRM==True THEN {clear lapGoToNRM Apply New Baud Rate }	NRM
NRM	pl_lapRxFrame	IF Iframe THEN set lapSendToLMP	NRM
	pl_lapRxCorrect	Set Timer0 3sec IF RR frame THEN clear lapMasterBusy send NRM Response	NRM
		Set Timer0 3sec IF RNR frame THEN set lapMasterBusy send NRM Response	NRM
		IF DISC Frame THEN set lapGoToNDM	NRM
		IF I Frame THEN send NRM Response	NRM
	pl_lapTxComplete	IF lapGoToNDM==True THEN {clear lapGoToNDM Apply default baud rate }	

pl_lapRxFrame – The PayLoad layer sends identification of the received COMMAND within (I, XID, SNRM, RR, RNR, DISC, RD)
pl_lapRxCorrect – The PayLoad layer reports the errorless end of receiving the frame.
send NRM Response – controls if the last numbered frame send correspondents with the frame received by the master. If yes, than it informs the LMP that the last frame was accepted by the master. LMP is waiting for this confirmation. If not, than it controls, if at least the numbered frame before the last was confirmed. In the case it wasn't than it requests the end of the link.
In the case of an Information frame it controls if the order number, it is waiting for, is in line with the number of the received frame. If yes, than it informs the LMP about the correct received Information package.
If the master is not busy than the LMP is asked to transfer the Information frame. If the LMP does not have data to send or the master is busy a Supervisory RR (Receive Ready) frame is sent, otherwise the LMP is sending an Information frame.
pl_lapTxComplete – The PayLoad informs that the transfer of the frame was finished.

LMP Layer, IAS, IrCOMM

The LMP directs the Information frame to the application or reacts on send instruction Connect. The LMP knows only two services IAS (Information Access Server) and IrCOMM interface. After the start of the link (confirmation of the Connect instruction) follows the identification via the IAS service. If the master is requesting IrCOMM link or TinyTP (Palm), IAS answers with a defined identification number of the service (LSAP). This way the IrCOMM link is created and LMP is sending frames with the belonging LSAP into the application layer. The TinyTP is realized as most as possible simple. The TinyTP ignores control frames, and only information frames are transferred. This is possible because of the size of the receiving buffer within the Palm, and also because the application is using it's own protocol working as COMMAND-RESPOND.

Application layer

The application layer is the final point of the IrDA. With the LMP it is communicating via different calls. The application has to wait for the different events and process them correctly.
SendDataToCOMM – sends to the application one information byte (in register A). Within one frame can be max. 62 byte.
DataToCOMMCorrect – informs the application that the received data are valid (after the end of the frame it controls the CRC and order number of the information frame)
DataToCOMMError – informs the application that the received data are invalid

lmp_startCommTx – LMP detects if application wants to start transmitting data. If application wants to send data, it sets the ZERO flag.
lmp_GetTxData – the application is requesting one data byte. This calling will be repeated until the max number of data (62 byte) is reached, or the application is showing with Z=1 that the last data byte is transferred.
lmp_AVRTxCorrect – is showing that the last send frame was accepted by the master
lmp_AVRTxError – reports that the master was not accepting the last send frame.

Within the received data the start of the instruction (character STARTCHAR='c') is looked for by **SendDataToCOMM.** All characters following after character STARTCHAR are put into the receiving buffer ir_in_buffer, which has a 32-byte length. After **DataToCOMMCorrect** the AVRRxState=2 is informing the main program that the receiving buffer is containing an instruction. The main program analyses the instruction and performs it. Usual the instruction requests to send back some data. The main program puts the sent data into the output buffer ir_out_buffer and changes AVRTxState into a non-zero value.
Via the following **lmp_startCommTx** the LMP is notified that the application request to send data and begins with the transmission.
With the following **lmp_GetTxData** all data are transferred step by step (finish character 0x00).
The following frame from the master is reporting to the LMP if all data are received and accepted without any error. The application is informed via **lmp_AVRTxCorrect** or **lmp_AVRTxError.** In the case of
lmp_AVRTxError the transfer is repeated. In the case of correct transfer the SENDNEXT is tested. It is set if the application needs to send further data. In this case SENDAGAIN is set, which is the signal for the main program that more data can be send.

Function description

The main function of the demo program to demonstrate the IrDA functionality is the measurement of three strain gauges, and the collection of the data into the DataFlash in a pre-set interval.
The signal from the strain gauges is amplified by a precision instrumentation amplifier and converted by a 24-bit sigma-delta ADC. The result of the conversion is only using the 16 most significant bits.

The AVR DemoDatalogger integrates the following functions

Handling through buttons:
1. START Button – data collection start in adjustable intervals. In the set moment is the supply for the analog part switched on, the converter activated and the measurement result saved to the DataFlash. Supply of the analog part is switched off.
2. STOP Button – Data collection End

Handling via the Palm:
1. Time setting. The system time of the PalmOS device is transferred into the RTC on the board.
2. Measurement interval setting (1-255 sec).
3. Reading the time out of the RTC and the measurement interval
4. Start of 8 measurement series, arithmetical average and visualization on the Palm display
5. Transfer and visualization of the measured data saved in the DataFlash on the Palm display.
6. Dataflash erasing

The board is continuously powered from 4xNiCd cells. The datalogger is waked up every second through an interrupt from sleep mode to test the handlers and after their service put again into sleep mode.

To demonstrate the IR connection between the AVR and the Palm an easy program is added (AVR.prc). After its installation and start on the Palm display is shown

The different buttons mean the following:

1. Set time, the system time of the Palm is transferred into the RTC
2. Set repeat time, setting the interval of the measurement (for simplicity 1-255 sec.). On the display is appears

The value can be entered with the touch pen or over the keyboard.

3. Get settings, shows the time read out of the RTC to compare with the system time of the Palm and to set the measurement interval
4. Make conversion, starts the set of 8 measurements. The arithmetical average is saved back into the memory of the Palm, and on the display is shown

Afterwards the arithmetical average of the last 8 measurements is displayed.

5. View saved data, allows to read simultaneously the saved data out of the Dataflash into the Palm memory and their numerical visualization.
6. Erase all data, erases all data saved in the Dataflash

Note: Under the name Palm is understood a handheld device based on PalmOS. For lower versions PalmOS under 3.3 it is necessary to use enhanced ir-software, e.g. IrLink of iscomplette.com
For PalmOS3.3 and higher versions no additional software is necessary.
The software was tested on Palm3 with PalmOS3.0 and Palm3x with PalmOS3.11

Flow Chart of the application program

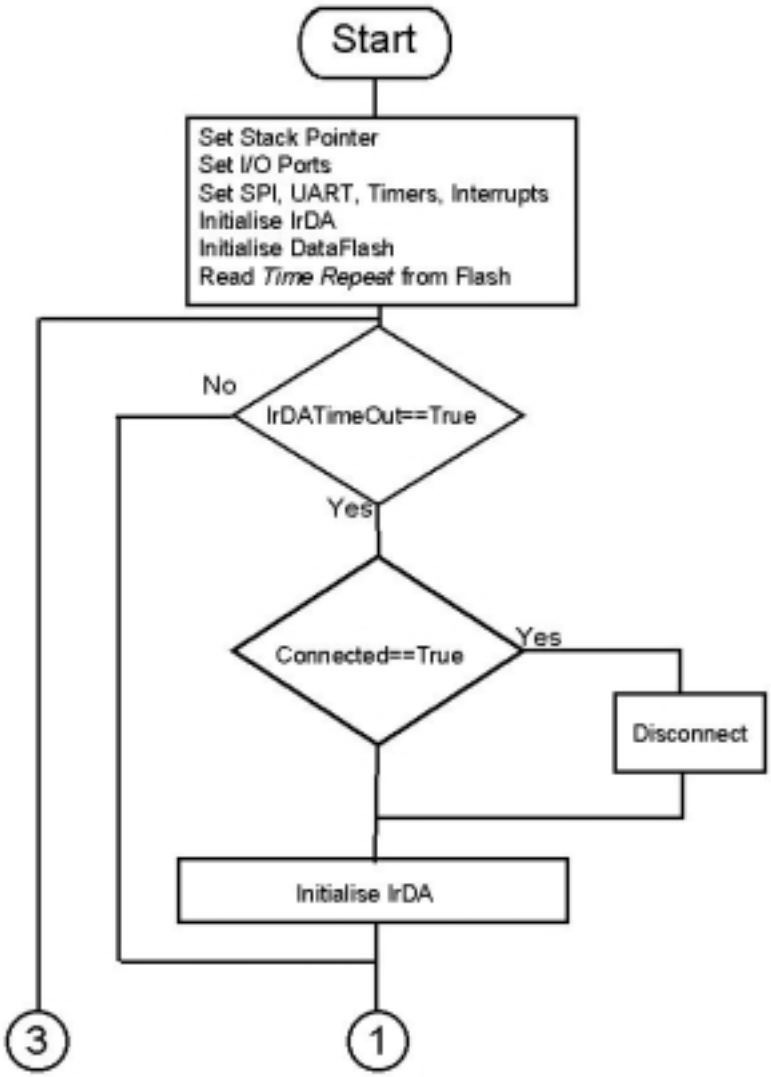
Asynchronous events serviced by interrupts

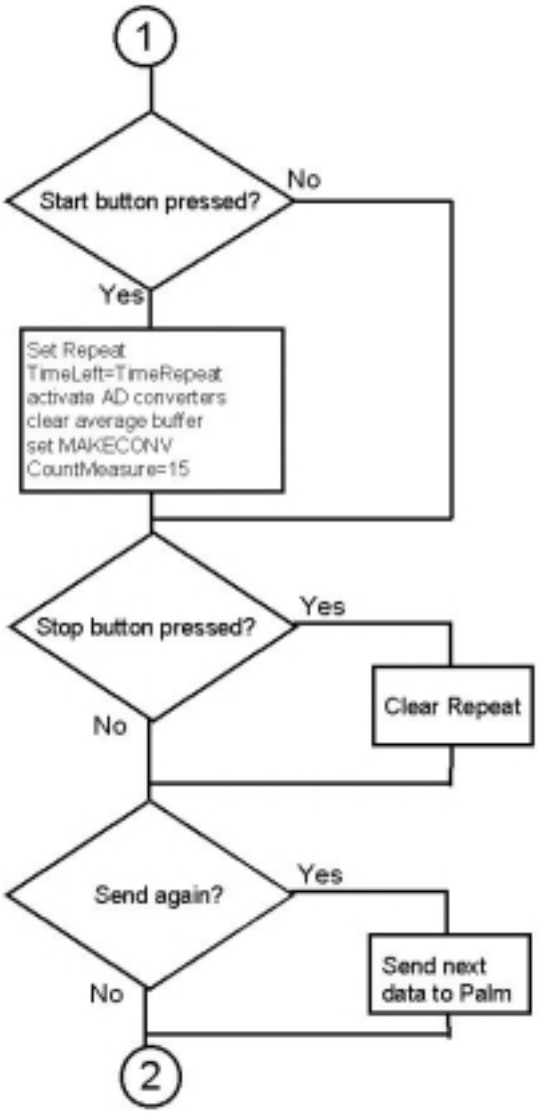
Timer 0 overflow
1. decrement TimeOut
2. IF TimeOUT==0 THEN set IrDATimeOut

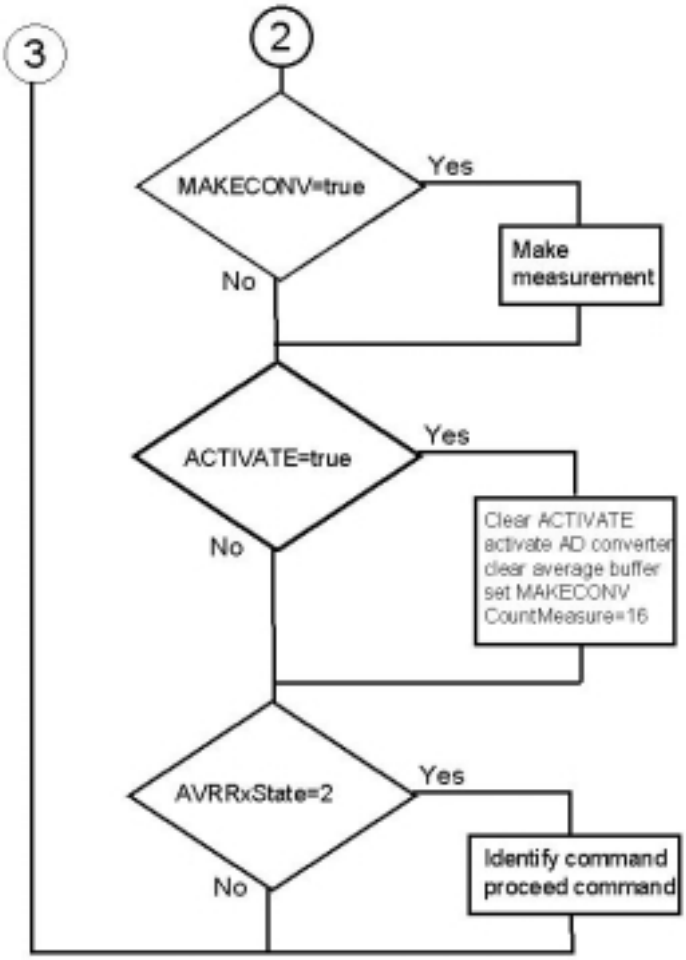
Timer 1 overflow
1. Increment blik_counter
2. IF REPEAT==True THEN
   decrement TimeLeft
   IF TimeLeft==0 THEN set ACTIVATE; TimeLeft:=TimeRepeat

UART Receive Complete, UART Transmit Complete - pass control to IrDA stack

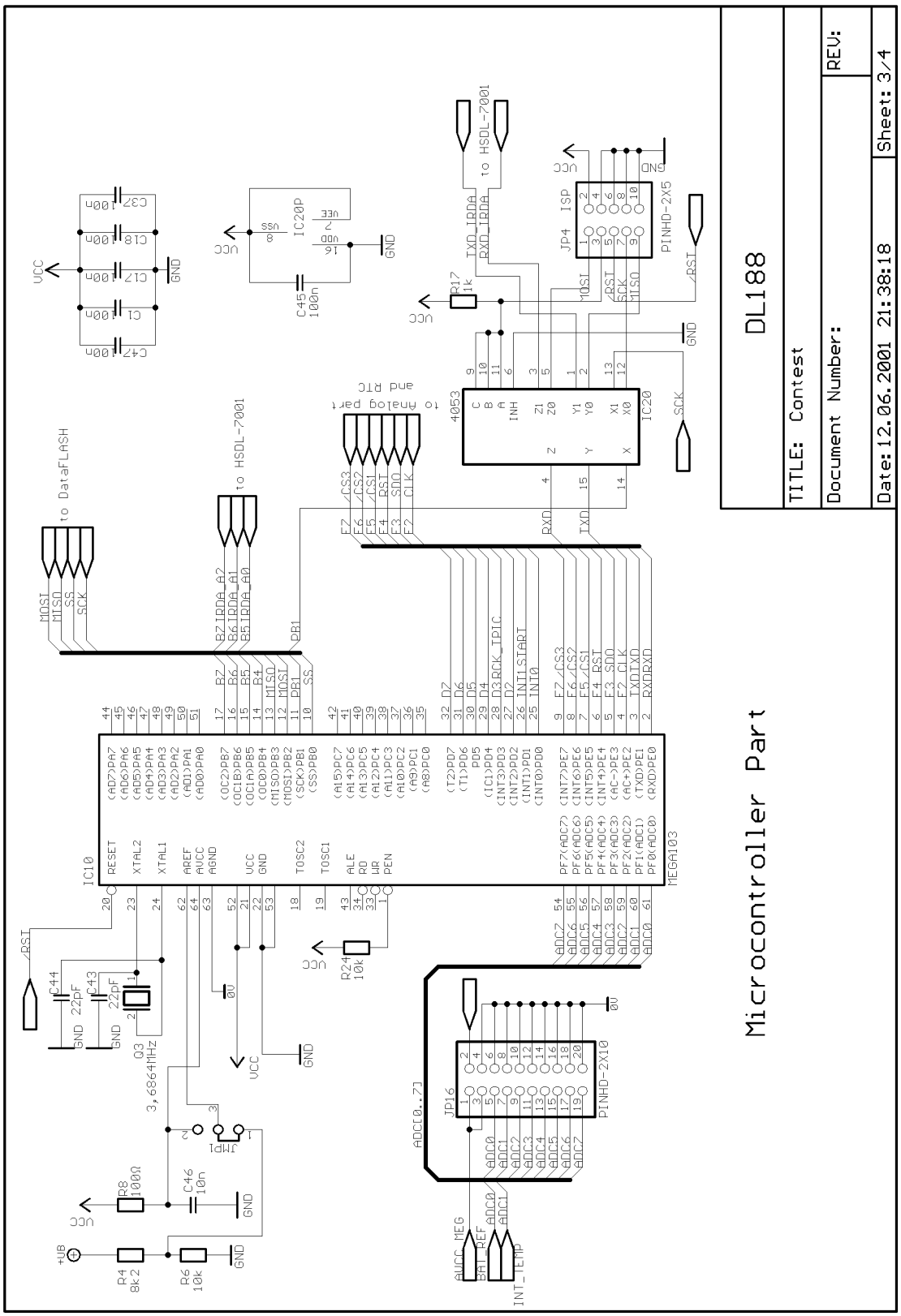
See the following pages for the flow chart of the main program.


Start
Set Stack Pointer
Set I/O Ports
Set SPI, UART, Timers, Interrupts
Initialise IrDA
Initialise DataFlash
Read *Time Repeat* from Flash
No
IrDATimeOut==True
Yes
Connected==True
Yes
Disconnect
Initialise IrDA
3
1

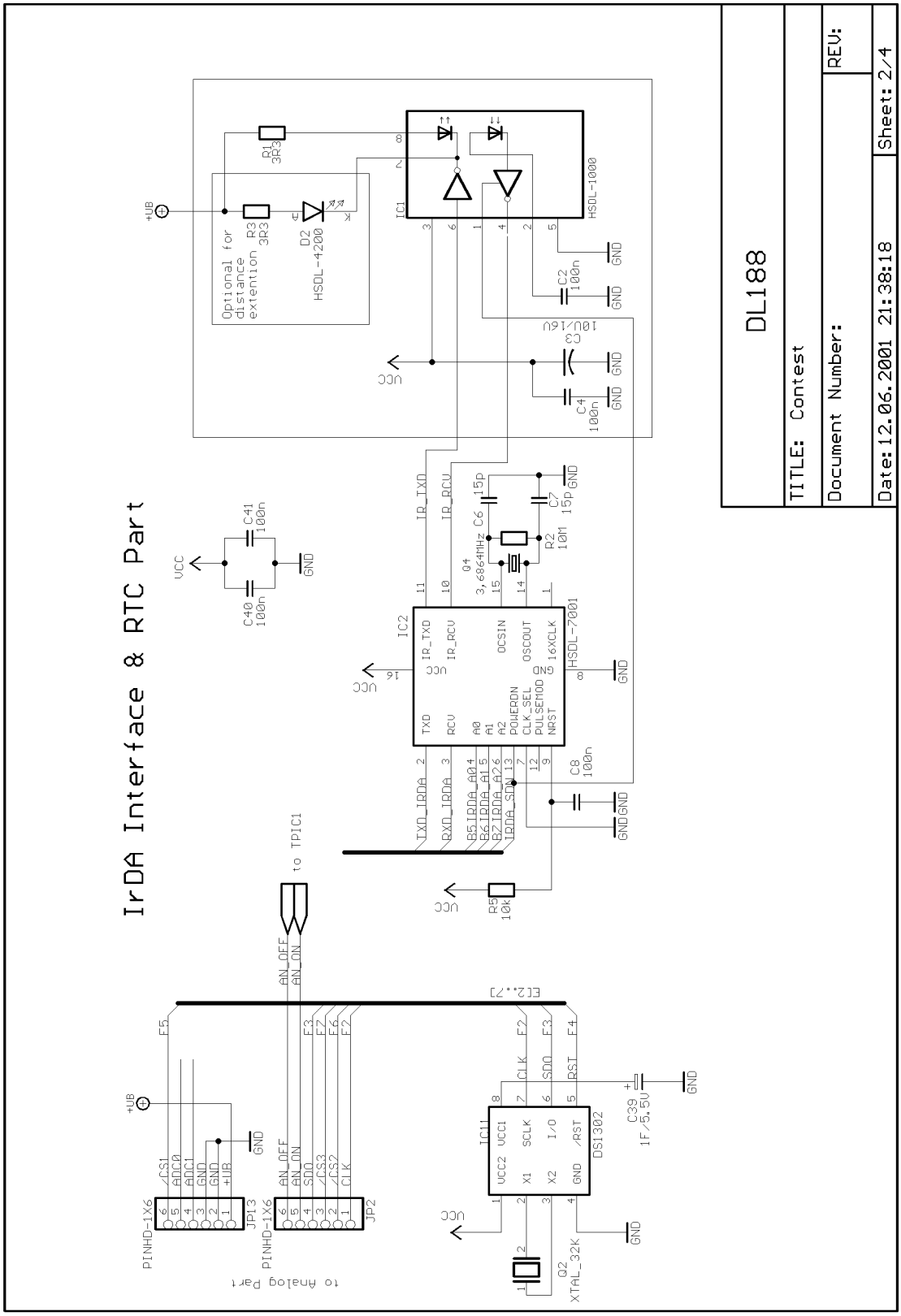

1
Start button pressed?
No
Yes
Set Repeat
TimeLeft=TimeRepeat
activate AD converters
clear average buffer
set MAKECONV
CountMeasure=15
Stop button pressed?
Yes
Clear Repeat
No
Send again?
Yes
Send next
data to Palm
No
2


3
2
MAKECONV=true
Yes
No
Make measurement
ACTIVATE=true
Yes
No
Clear ACTIVATE
activate AD converter
clear average buffer
set MAKECONV
CountMeasure=16
AVRRxState=2
Yes
No
Identify command
proceed command

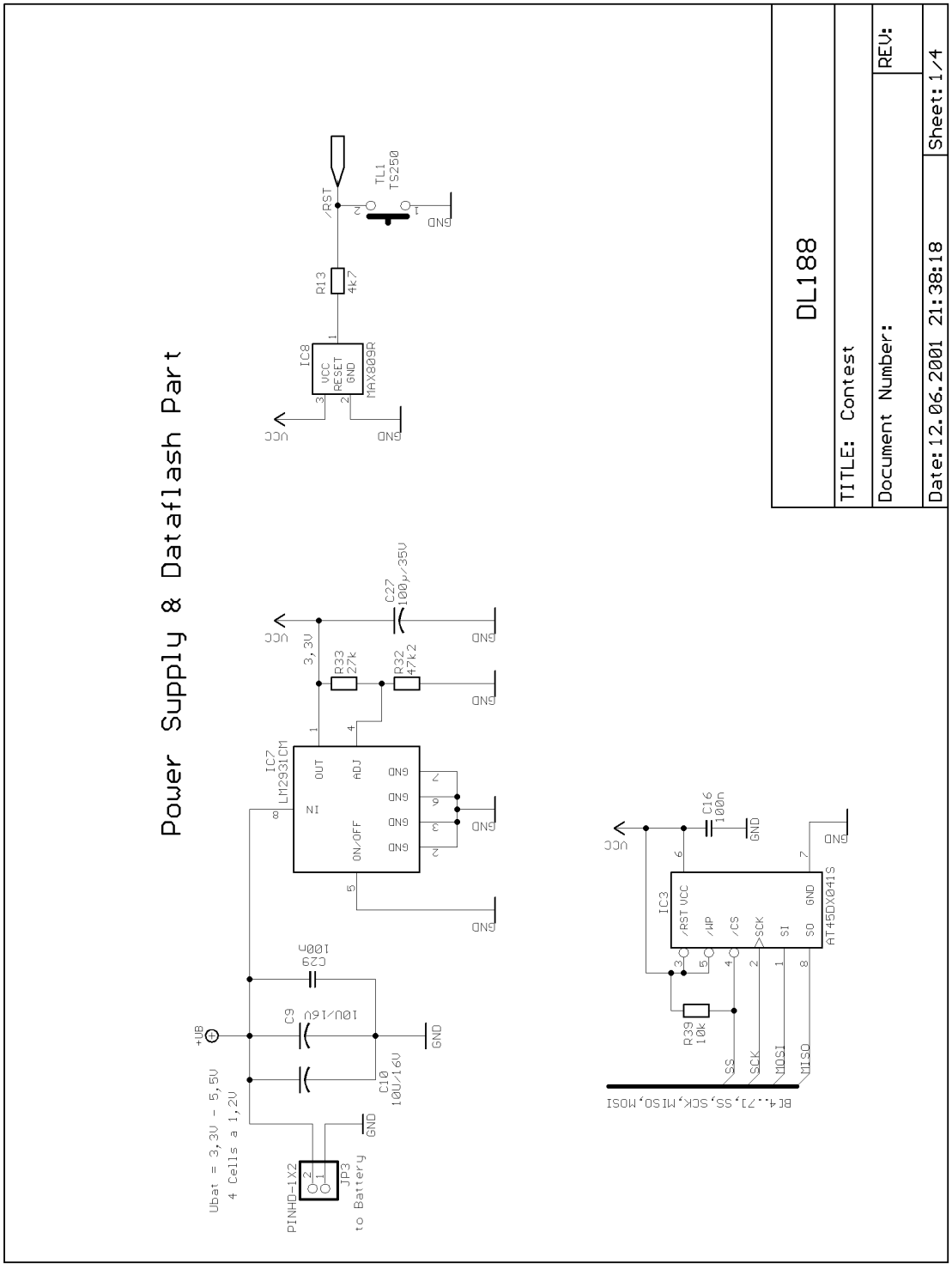
Microcontroller Part



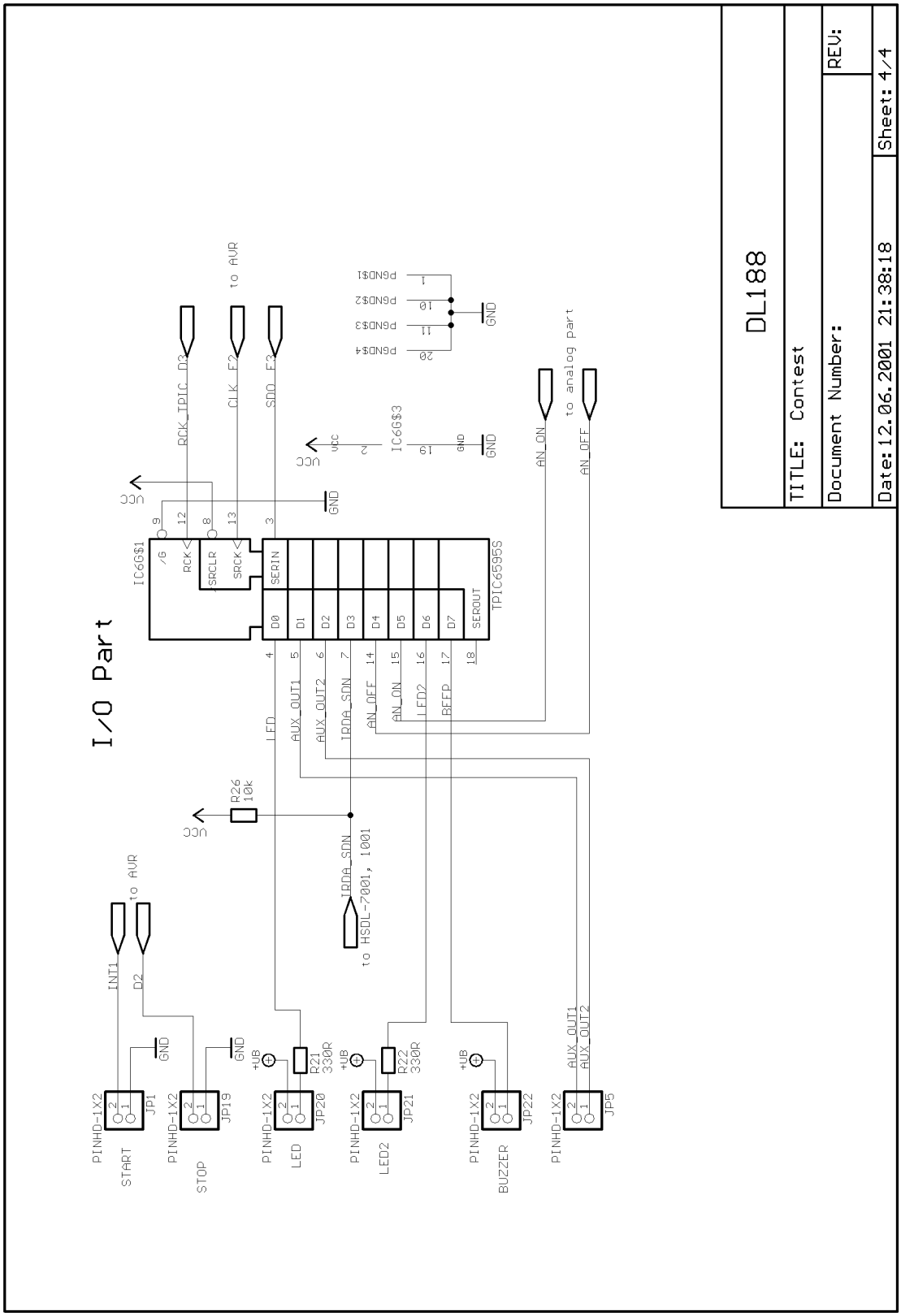
IrDA Interface and RTC Part



Power Supply and Dataflash Part



Digital Input/Output Part



Precision Analog Part