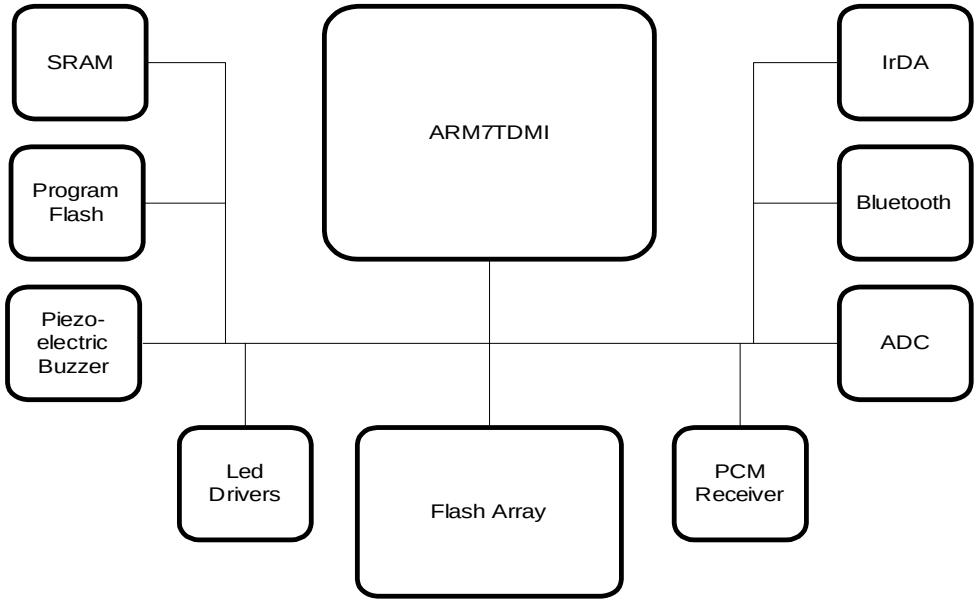
Department of Information Technology
Lund University

Design of an ASIC with ARM7TDMI

NetDrive - A Wireless Pocket-sized Harddisk

Tord Jakobsson



Lund, December 1998

1.0 NetDrive Overview

1.1 Introduction

Information between electronic appliances is usually transmitted by wires or via infrared light. Recently, a new standard for radio communications called *Bluetooth* has been developed by Ericsson and supported by many leading companies. When using Bluetooth technology all electronic equipment may connect spontaneously and exchange information without any free air or angles restrictions.

FIGURE 1. NetDrive from the outside.

NetDrive is one step towards achieving the goal of making your personal computer-files available wherever you go. It’s a pocket-size, wireless hard drive capable of storing 200 megabytes of data. It consists of no moving parts and is very energy efficient. It communicates with both Bluetooth and IR.

1.2 NetDrive Overview

The major advantage of the NetDrive is its lack of features. It’s a simple little box with nothing but a battery-charging socket, IR window and blinking LED. Using it is just as easy; bring it close to a terminal with Bluetooth or IR and use your favorite web browser to transfer your files from and to the NetDrive.

1.3 The Project

The project was carried out within a very short time by using *The Wavefront Method*, developed by Professor Lars Philipson at Lund University. The main goal of the method is to achieve maximum efficiency by splitting the work into several parallel *tracks*. One person is responsible for each track and for bringing necessary resources into the project. These resources include both material and specialists in that particular field.

TABLE 1. The test benches used in the NetDrive project.

Test bench		Description
1	PC - Flash board 1	The Flash board 1 is connected directly to a PC by the parallel port. Used for testing the Flash board and original FlashFx software.
2	PC - Wire - ARM board - Flash board 1	The PC is connected with a wire to the ARM board to which Flash board 1 is connected. Used for testing PPP connection between a PC and the ARM development board running a OSE example.
3	PC - Bluetooth 1 - Bluetooth 1 - ARM - Flash board 1	Similar with Test bench 2 but the cable is replaced by two Bluetooth development boards. Used to test the Bluetooth connection.
4	PC - Wire - Wire-wrap board - Flash board 1	Similar with Test bench 2 but the ARM development board is replaced by the first version of our Wire-wrap board. Used to test the Wire-wrap board.
5	PC - Bluetooth 1 - Bluetooth 1 - Wire-wrap board - Flash board 1	Similar with Test bench 4 but the cable is replaced with two Bluetooth development boards. Used to test the Bluetooth connection with the Wire-wrap board.
7	PC - IR - Lab board	A PC is connected via IR media to the Lab board. Used to test the Lab board hardware and IR software.
8	PC - IR - Board 1	Similar with Test bench 7 but the Lab board is replaced with Board 1. Used to test hardware on Board 1.

TABLE 2. Explanation of the test bench components.

PC	Standard PC running Windows 95.
Flash board 1	Laboratory board with flash devices. The board is connected to the PC (or other board) with a standard parallel port.
Wire	Standard cable connecting two serial ports.
ARM board	One-board computer development board with build-in RAM from ARM.
Bluetooth 1	Development board with bluetooth technology from Ericsson.
Wire-wrap board	Wire-wrapped version of the hardware. Contains everything except power supply and flash devices.
Lab board	The mother board in a big development stage.
Board 1	Complete mother board in final version.

The purpose of the pre-project is to plan and gather as much information as possible about the upcoming project. Typical issues the pre-project deals with are making a detailed *project plan* (both from technical and management point of view), investigation of what type of complications the participants might come across and building up a network of contacts that can solve these problems.

The project plan consists of a number of *checkpoints*. Every checkpoint is associated with a deadline and description. Some of the checkpoints serve as synchronization points between different tracks. To evaluate if a checkpoint is reached, the responsible person proves it on special *test benches*. The test benches are carefully chosen to be useful even during the different development phases, see Table 1.

During the project, detailed plans are made for the next upcoming checkpoint. *Risk analysis* and alternative technical solutions are always included in the detailed plans. This short-range planning is an essential part of the Wavefront Method. By using continuous risk analysis, very few unpleasant surprises can affect the project plan. The NetDrive project plan contains 31 checkpoints spread over a period of 7 months, it is presented in Table 3.

The project was initiated and funded by Ericsson Mobile Communications AB. Four students at the Department of Information Technology at Lund University, were the only permanent staff. Several specialists in a broad range of areas were engaged. A list of all participants is presented in Appendix B.

Half way through the project it became evident that Bluetooth would not be ready in time for this project's deadline. This changed the conditions of the project plan and a decision was made to replace Bluetooth link by an IrDA connection.

TABLE 3. The project plan used in the NetDrive Project.

Track	Jun 8	Jun 19	Jun 26	Jul 10	Jul 16	Jul 22	Jul 24	Jul 31	Oct 16	Oct 27	Nov 3	Nov 20	Nov 28	Dec 4	Dec 16
Mechanics	1 2			10	13						25				30
File access	1	3 4 5		10	14	17	20	21 22					28		31
Electronic Hardware	1	3	7	9 10 12	13 14			21	23	24	26	27	28	29	30
Communi-cation	1		8	10	16	18	19						28		31
Radio and Antenna	1		8	11	16	18	19								
Battery and Energy	1								23	24					31
Audio and sensors	1				15			22			25				31
Documen-tation	1		6	10											

TABLE 4. Explanation of the project's checkpoints.

1	Kick-off.	17	OSE with RAM-disk and www server on wire-wrap board (TB4).
2	Mechanical design v.0	18	Radio test on wire-wrap board.
3	FlashFx on DOS (TB1).	19	WWW connection from PC to wire-wrap board via Bluetooth (TB5).
4	Hello world on ARM board (TB2).	20	FlasFx under OSE on ARM board (TB2).
5	OSE with RAM-disk and www server on ARM board (TB2).	21	FlashFx under OSE on wire-wrap board (TB4).
6	WWW structure.	22	First application for sensors on wire-wrap board (TB4).
7	Component placement for wire-wrap board.	23	Test-board for power supply.
8	WWW connection from PC to ARM board via Blue-tooth media (TB3).	24	Component placement on Lab board.
9	Hello world on wire-wrap board (TB4).	25	Mechanical design v.1
10	First release of web-site.	26	Component placement on board 1.
11	First test of radio antenna.	27	Hello world on Lab board.
12	Test of Lauterbach emulator.	28	FlashFx under OSE with IR on Lab board (TB7).
13	First mechanical assembly v.0	29	Hello world on board 1 (TB8).
14	OSE with RAM-disk on wire-wrap board.	30	Prototype 1.
15	Receiving and playback of PCM sound from Blue-tooth (TB3).	31	Complete software for board 1 (TB8).
16	TCP/IP connection through Bluetooth.		

1.4 User Interface

The NetDrive is very easy to use, just place it close enough to an appliance with a bluetooth installed. Use a standard browser and enter the IP address of the NetDrive in the location field and the web page in Figure 2 will appear.

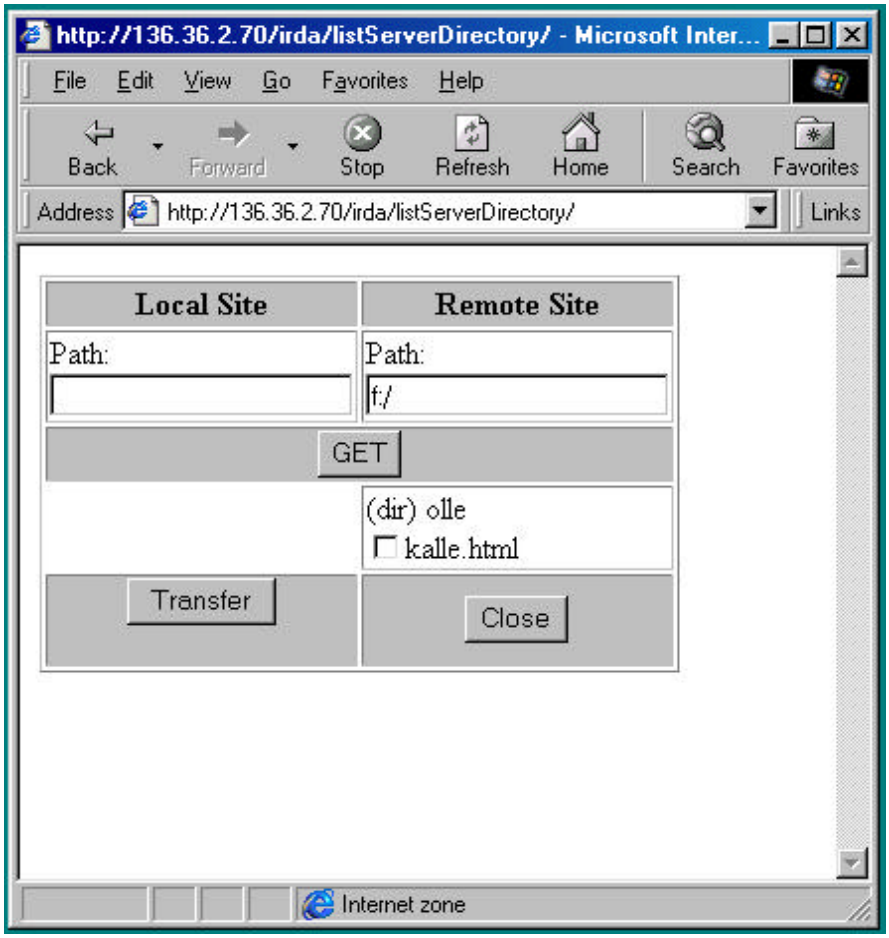


FIGURE 2. The NetDrive user interface.

The user interface consists of two frames; *Local site* and *Remote site*. The local site represents the host system and the remote site represents the NetDrive. Each of these frames also has a *Path* field indicating the path of the files on either frame.

In order to transfer files to and from the NetDrive you first enter the paths of the Local and Remote site. Then you select which files you want to transfer in the Local site / Remote site frames and press ‘TRANSFER’. The files selected in the Local site frame will be transferred to the NetDrive and the files selected in the Remote site frame will be transferred to the Local site.

NetDrive also has a build-in support for various sensors. The prototype comes with a temperature senor and software for logging values of the senor.

1.5 Electronic Hardware

The electronic hardware consists of components from various manufacturers. Figure 3 shows a general block diagram. The processing element of NetDrive is an ASIC developed by Ericsson containing an ARM7TDMI processor core.

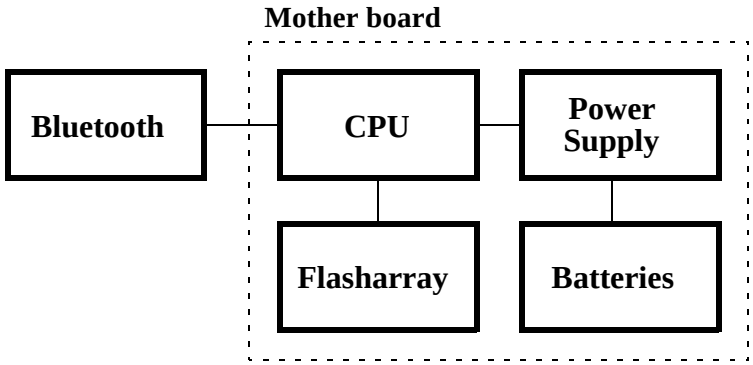


FIGURE 3. Overall block diagram.

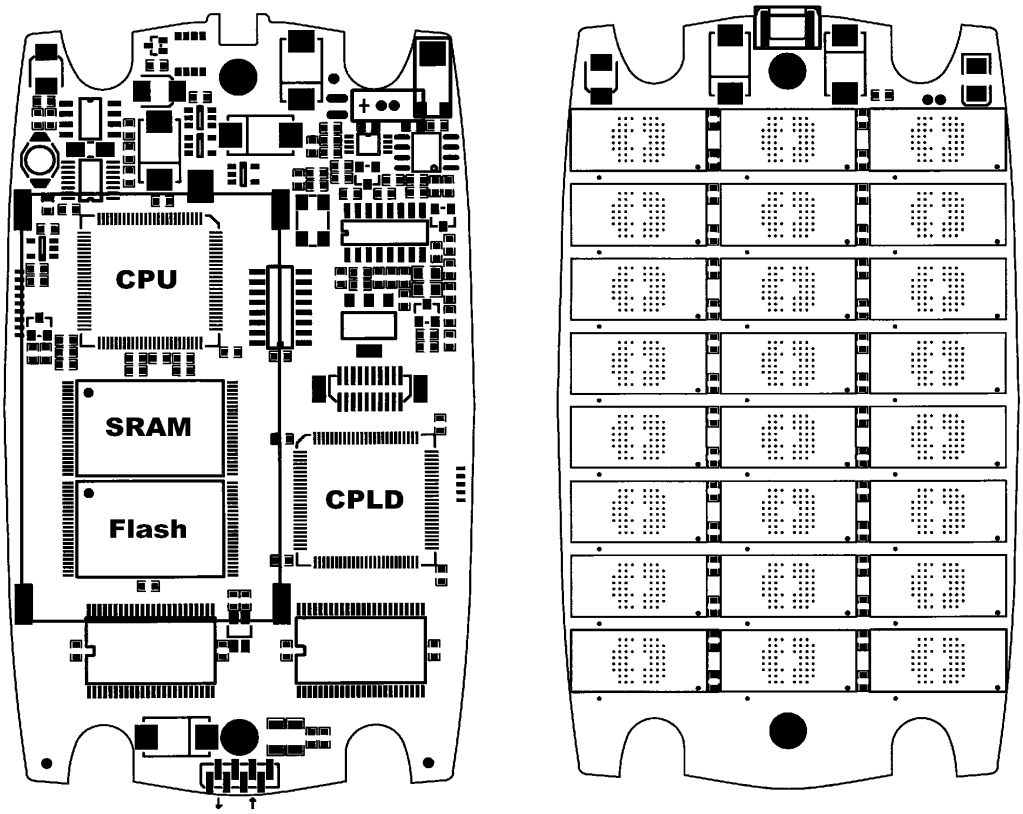


FIGURE 4. Drawing of NetDrive motherboard. Top view (right) and bottom view (left).

1.5.1 Flash Disk Hardware

The flash disk hardware consists of 64Mbit flash devices from Intel, StrataFlash in BGA, Ball Grid Array, packages. On the board 24 of these are mounted which gives a total storage capacity of 192Mbyte to the flash disk.

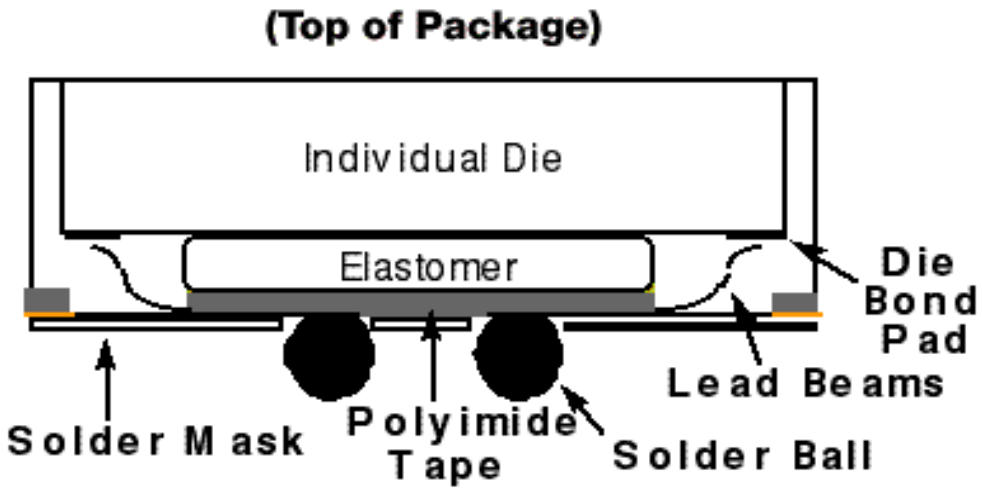


FIGURE 5. Cross-section of BGA package. [1]

The BGA is a packages with the pins under the package and not on the sides as other surface mounted packages, FIGURE 5. This allows BGA package to be smaller. With no pins that points out on the sides devices in BGA packages can be mounted much closer to each other. This makes BGA package much more space efficient.

1.5.2 Flash Memory

NetDrive stores its files on an array of flash memory devices. A flash memory differs somewhat from a conventional storage device such as RAM or hard disk. The individual bits of the flash device can only be programmed in one direction, from a one to a zero, and cannot be programmed from a zero to a one without an erase operation.

An erase operation for a flash device requires that a large section of bits, an *erase zone*, be programmed at the same time. This erase zone, see Figure 6, sometimes referred to as an erase unit, is typically 64Kbytes in size, but can range from 512 bytes to 128Kbytes, depending on the type of flash device used.

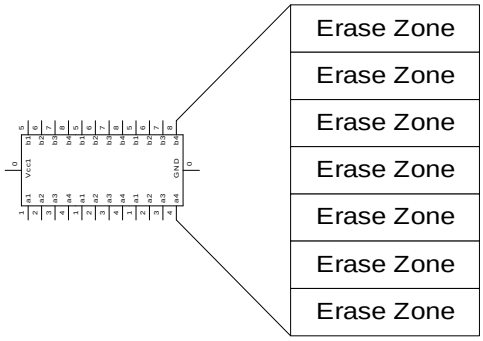


FIGURE 6. Flash Memory Device Overview.

When using RAM or conventional disk for storage, data of any size can be written and re-written into the same location without incident. Since flash is not capable of re-writing individual bits of data, all data must be written, or re-written, into an unused area of the flash device.

Another aspect of flash devices that must be considered is its limited life. For any given flash device, there is a limit to the total number of erase operations that may be performed on a particular erase zone before it becomes unreliable or damaged. Flash device lifetimes range from 10,000 cycles to 1,000,000 cycles, with most rated around 100,000.

1.6 Bluetooth

Bluetooth technology [2] allows for the replacement of the many proprietary cables that connect one device to another with one universal short-range radio link. It is a result of a cooperation between leaders in the telecommunications and computer industries, including Ericsson, IBM, Intel, Nokia, and Toshiba. They are determined to take the necessary action to make the technology a global standard for wireless connectivity.

Bluetooth is targeted at mobile and business users who need to establish a link, or a small network, between their computer, cellular phone and other peripherials. It operates in the international 2.45 GHz ISM band, at a gross data rate of 1 Mbit/s, and features low energy consumption for use in battery operated devices. The air interface is based on a nominal antenna power of 1 mW resulting in a range of about 10 meters.

A connection can support one asynchronous data channel, up to three simultaneous synchronous voice channels, or simultaneously data and voice. Each voice channel supports 64 kbit/s data rate. The data channel can support an asymmetric link of maximally 721 kbit/s in either direction while permitting 57.6 kbit/s in the return direction, or a 432.6 kbit/s symmetric link. To sustain this transfer rate in busy radio environments a packet switching protocol with frequence hopping and advanced coding techniques are emplyed. A connection features graceful degradation of both voice and data transfer rates in busy environments.

A Bluetooth network supports both point-to-point and multi-point connections. Several ad-hoc subnets can be established and linked together. All nodes are peer units with identical hardware and software interfaces except i unique 48 bit address. At the start of a connection, the initialization unit is temporarily assigned as a master and controls the traffic of up to a maximum of seven slave units in each subnet.

Bluetooth provides user protection and information privace mechanisms at the physical layer. Authentication and encryption is implemented in the same way in each Bluetooth device, appropriate for the ad hoc nature of the network. Connections may require a one-way, two-way, or no authentication.

To achieve highest possible robustness for noisy radio environments, Bluetooth uses a packet switching protocol based on a frequence hop scheme with 1600 hops/s. The entire frequence spectrum is used with 79 hops of 1 MHz bandwidth, defined analogous to the IEEE 802.11 standard. Frequency hopping (direct sequence spread spectrum) gives a reasonable bandwidth and the best interference immunity by utilizing the entire available spectrum of the ISM band. Virtual channels are defined using pseudo-random hop sequences.

Units can be dynamically connected and disconnected from the subnet at any time with a connection time of typically 0.64 s. A unit does not need to be connected at all times since only a typical delay of under one second is required to start a transaction. When not in use the unit can be in a standby mode where only a low power oscillator is running.

1.7 Software

1.7.1 Operating System

NetDrive uses a real-time operating system for embedded systems called OSE. It is developed by Enea OSE Systems and comes with a wide variety of additions, such as TCP/IP stack, www- and ftp-servers and the Embedded File System (EFS). All these additions are used in the NetDrive. An IrDA stack was added to OSE by Enea during the project.

1.7.2 Flash Translation Layer Software

In order to successfully maintain a file system on an array of flash memory devices, a special layer of software is needed. This software handles all the flash memory-specific properties and makes the flash array appear as a conventional storage device. The main differences between flash memory and RAM memory or other storage devices are;

- The inability to re-write on the same location in flash devices.
- The finite life time of the erase zones.
- The need for interrupt protection. Embedded hardware can be shut off at any time, this shouldn’t destroy the file system.

The software taking care of the flash memory-specific properties is often referred to as Flash Translation Layer, FTL. The NetDrive uses an FTL called FlashFx developed by Datalight.

1.8 Mechanics

Figure 7 shows an exploded view of NetDrive.

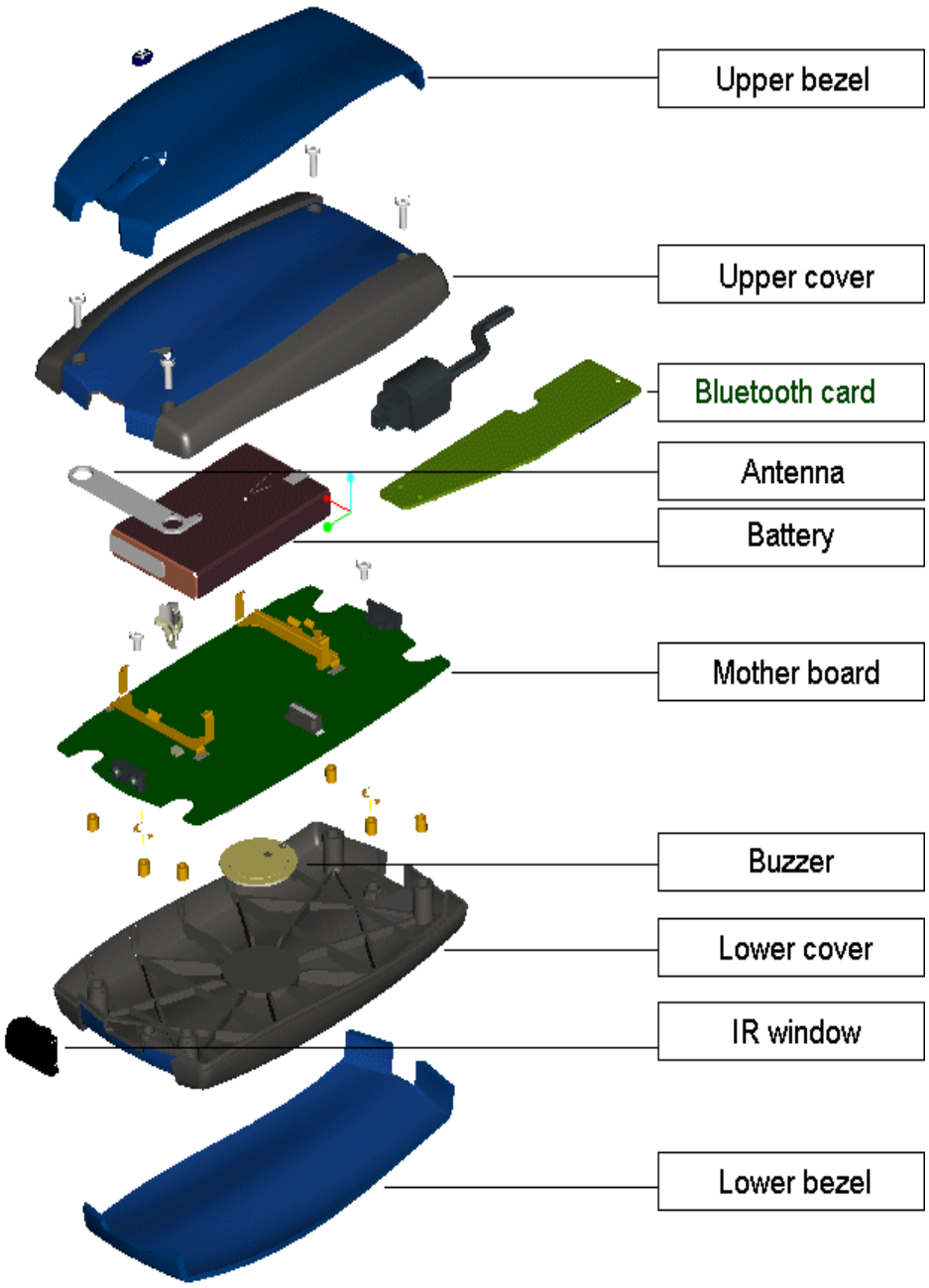


FIGURE 7. Exploded view of the NetDrive mechanics.

1.9 Technical Specifications

TABLE 5. Function Specifications.

Function	Specification
Storage Capacity	192Mbyte
Maximum Data Transfer Rate	115 Kbit/s
Sensors	Temperature Sensor

TABLE 6. Hardware Specifications.

Hardware	Specification
Processor	Ericsson with ARM7TDMI
Internal RAM	SRAM 512 Kbyte
Program Memory	Flash 1 Mbyte
Sound Element	Piezoelectric buzzer
Supply Voltage	3.3 V
Battery Type	Sanyo Lithium ion, Model 1UF653048
Non-volatile data storage	Flash 192 Mbyte (24 x 8 Mbyte)
LED Type	Citizen CL-155
Radio	Bluetooth technology
IrDA	115 Kbit/s
A/D converter	10bit

TABLE 7. Software Specifications.

Software	Specification
Operating System	OSE Delta version 3.1
FTL Software	Datalight FlashFx version 4.0
WWW server	OSE Delta webs WWW server.
FTP server	OSE Delta inet FTP server.

TABLE 8. Mechanics Specifications.

Mechanics	Specification
Size	105 x 60 x 20 mm
Weight	80g
Battery pack	Chargeable

2.0 Hardware Overview

2.1 Introduction

The hardware is developed in small stages. This makes debugging of the new hardware easier and allows software to be developed at the same time. When new hardware is added software can be tested immediately. The first step was the development of a Wire Wrap Board where it was easy to add components with time. When the Wire Wrap Board was completely developed the Lab Board was designed. Figure 8 shows the different blocks in the hardware.

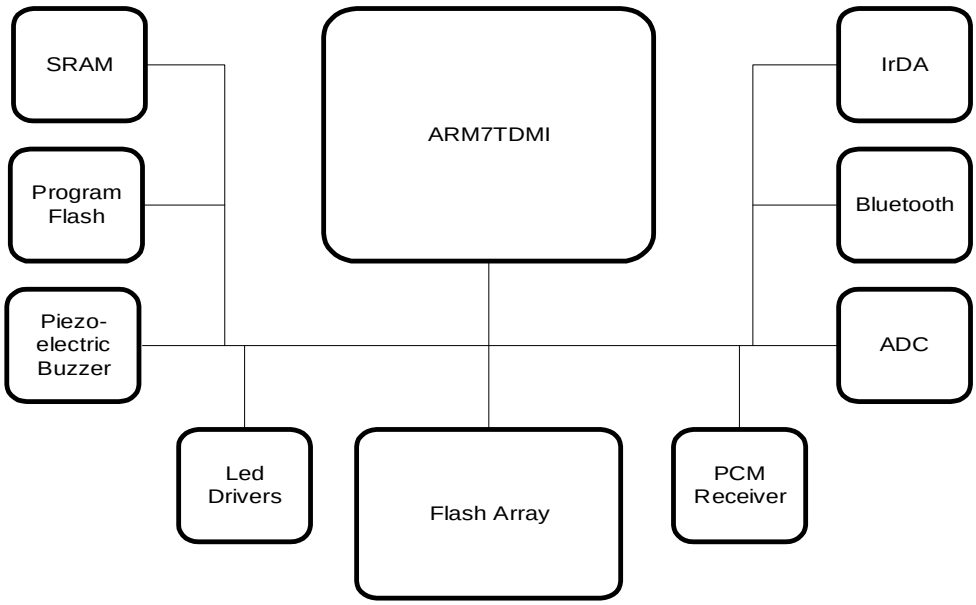


FIGURE 8. Block diagram of Netdrive

In the development of the Wire Wrap Board a big problem was to find suitable circuits for our purpose. To be able to mount the circuits they should be in PLCC or DIP packages to which it's easy to find sockets for a wire wrap board. The circuits should also be suited for 3.3V. This combination was not always the best and for some circuits adapters had to be used. The Lab Board was designed on a PCB, Printed Circuit Board, which eliminated these problems. The Lab Board has the same functions as the wire wrap board but smaller and more suitable circuits could be used. The final stage was the NetDrive Motherboard which is identical, except from some debug options, to the Lab Board.

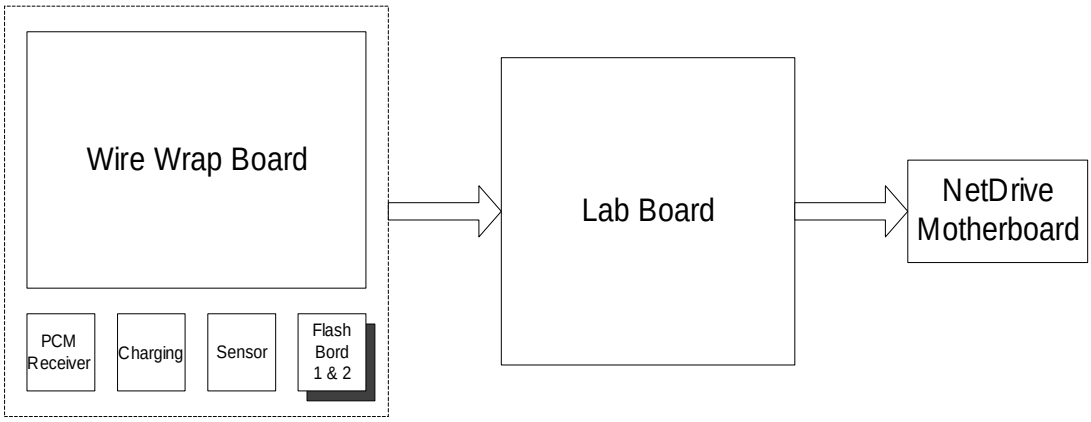


FIGURE 9. Development stages

The CPU in NetDrive is an ASIC developed by Ericsson that it includes an ARM7TDMI cpu core. In the ARM7TDMI is a debug block called JTAG. With JTAG programs can be downloaded to target from a PC. Programs can be loaded both to RAM and flash. When running programs on target it's possible to single step and put break points in the program. When the processor is halted the internal registers and the memory contents can be edited. To use the JTAG block the Embedded ICE, Figure 10, must be connected between the PC and target.

FIGURE 10. Embedded ICE, connected between PC and target when using JTAG debugging.

2.2 Wire Wrap Board

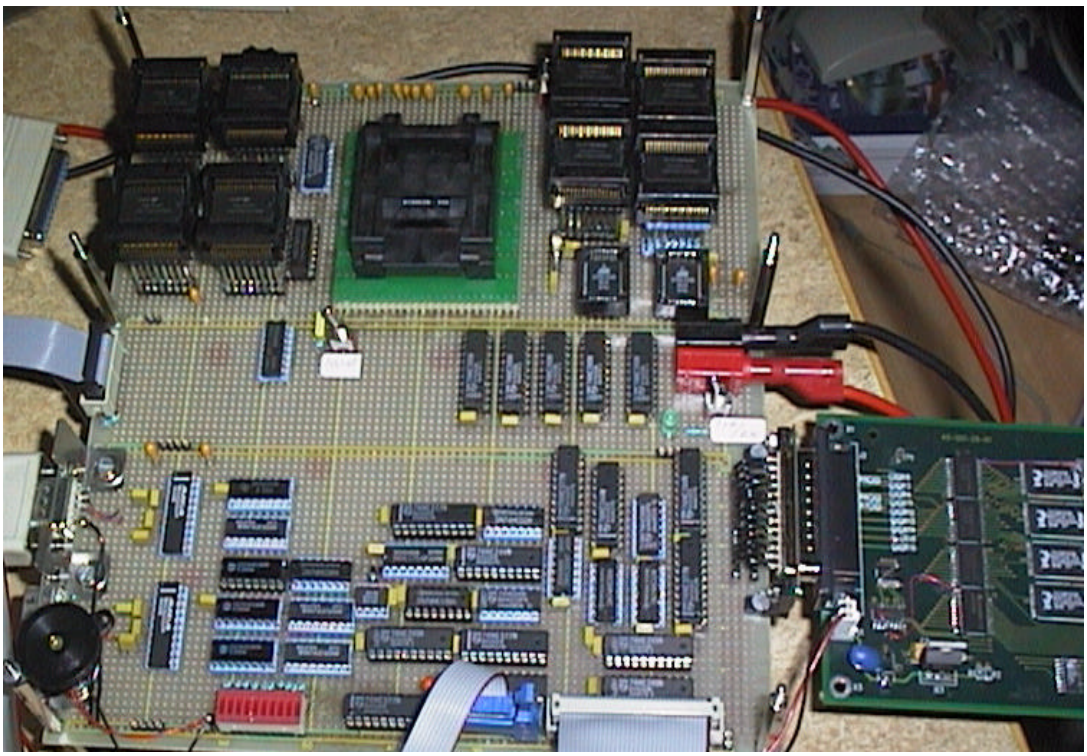


FIGURE 11. Wire Wrap Board

2.2.1 General

The Wire Wrap Board, Figure 11, was developed in small stages with test benches for each stage. This made debugging of the new hardware easier. Software development can also benefit from this. By adding hardware at the same time as the software was developed the software can be directly tested on hardware.

The first step in the development chain was a program called Hello World, Figure 12, on the CPU, i.e. to have the ARM7TDMI [3] running with a minimum of external hardware. The Hello World test was to send a char on one of the UART's to the COM port on a computer. To do this the only external components needed was a clock, JTAG and a RS232 interface. On the ASIC some pins had to be strapped to V_{cc} or GND to set the right boot mode.

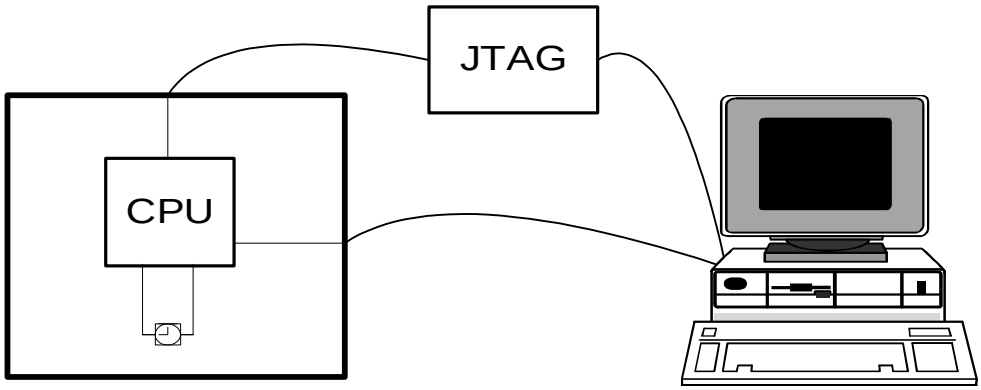


FIGURE 12. Hello World setup

When Hello World was completed SRAM memory was added so the software developers could start configuring OSE for our hardware. After that came program flash, parallellport interface to the flash disk, PCM receiver, A/D converter and temperature sensor, real-time clock, led and piezoelectric buzzer and last COM2 was changed to an IrDA module. The IrDA module is used to send data via IR when no Bluetooth is available.

2.2.2 I/O

The wire wrap board has three different I/O ports. There are JTAG for debugging and loading the program to the flash. Two Serial ports, COM 1 and COM 2, for connection to a PC [4]. COM 2 was after a while replaced with a IrDA module. The serial ports and IrDA was used to connect to the terminal or to the web browser on a PC.

Bluetooth was meant to be connected to COM 1 and COM 2 was for connection to a PC. COM 1 & 2 was connected to the Rx, receive, and Tx, transmit, signals on a UART via a RS232 interface. The purpose of the RS232 interface is to adjust the signal levels between the UART and the COM port.

With the delay of Bluetooth a decision to use a IR was taken. The IrDA module replaced the RS232 interface on COM 2. The IrDA module is connected to the Rx and Tx signals from the UART. When using IrDA a special block in the UART is used to code/decode the signals on the Rx and Tx lines. IrDA has a data transfer rate of 115 Kbit/s.

2.2.3 Memory

On the Wire Wrap Board there is 1Mbyte SRAM and 1Mbyte flash. Both SRAM and flash was hard to find in suitable packages and for 3.3V. With SRAM surface mounted packages had to be used together with an adapter socket.

The flash is used as program memory and consists of two 512k x 8bit flash. The program flash is organized to allow 16 bit access in one bus cycle, Figure 13. The SRAM consists of eight 128k x 8 bit devices. The SRAM is organized to allow 16 bit access but have also possibility to make 8bit access.

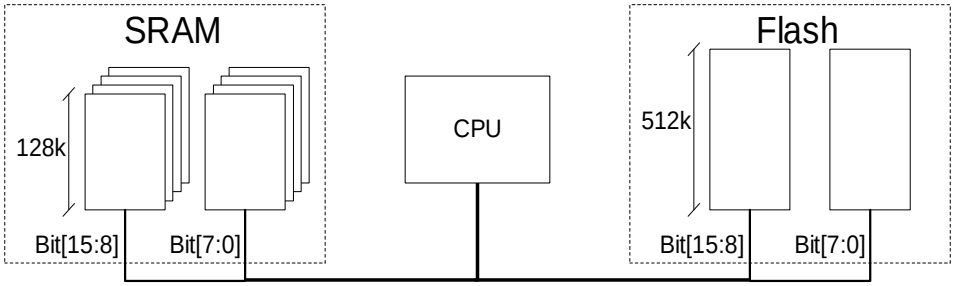


FIGURE 13. Memory organization

2.2.4 Flash Disk Interface

The flash board for the wire wrap board was developed by SVEP. It was designed with a parallel port in order to be tested with software on a PC. The flash disk has four 1Mbyte flash devices, total 4Mbyte. Later a second version of the flash disk was designed. This device was in function identical to the first one but the flash memory was in BGA packages and only developed to test mounting of the BGA packages,Figure 14.

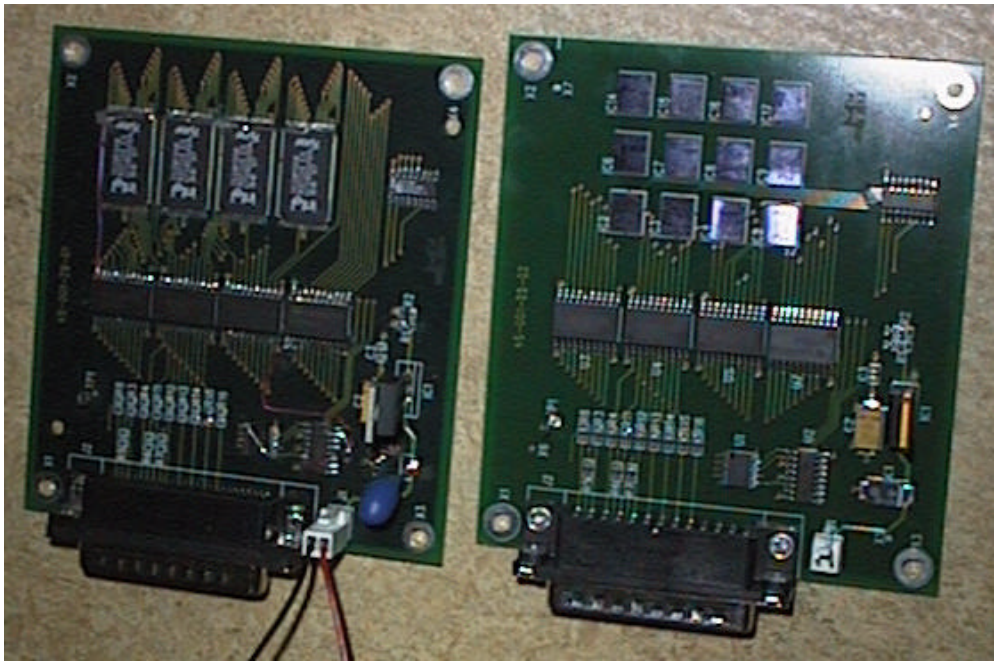


FIGURE 14. Flash disk 1 (right) & 2 (left).

Read and write operations to the flash disk has, due to the limits in the parallel port on the PC, to be done in stages. On the flash disk there are three registers to which the address and read/write signals to the flash devices first must be written before the flash can be accessed. To control if data is written to a register, written to the flash or read from flash there are also four control signals needed. Two of the signals, X and Y, enables write to registers or sets the buffer to the flash in read mode. The third signal, CSEN, is a chip enable signal to the flash. The fourth signal, STROBE, works as a clock that clocks the X and Y signals in to the flash card.

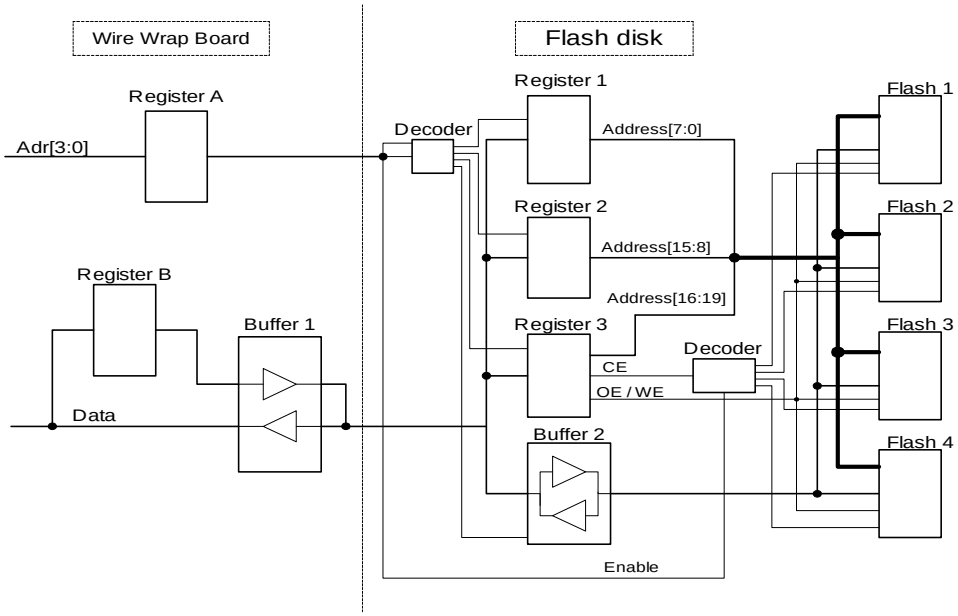


FIGURE 15. Flash interface. Register A is for the control signals and Register B is for data and addressbits to be written to the flash disk.

While the flash disk was originally designed for the parallel port on a PC a similar interface had to be built on the wire wrap board, Figure 15. This was built with register and tristate buffers. If data is read from the flash disk Buffer 1 opens for data to go from the disk to the CPU and tristates the data path from Register B. When data is not read from the flash disk Buffer 1 opens the data path from Register B to the flash disk and the path from the disk to the CPU is tristated.

2.2.5 AD Converter

The A/D converter, MAX 113, has four analog inputs and a 8bit parallel bus. To the A/D converter four control signals, Enable, Read and Adr[1:0], are connected. The control signals needs to be valid longer time than the memory bus allows therefor the signals are written to a register. To control if data is read from the A/D converter or written to the register a buffer is used in a similar way as in the flash interface, Figure 16.

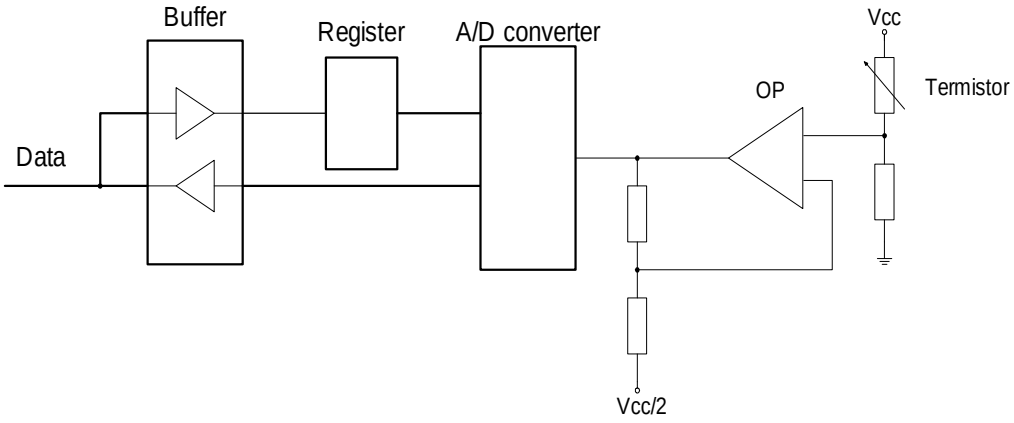


FIGURE 16. Block diagram for A/D converter and temperature sensor.

To the A/D converter different types of sensors can be connected. On the wire wrap board a temperature sensor was chosen. The temperature sensor changes its resistance with the temperature. The signal from the temperature sensor is amplified, x50, with an OP amplifier before the A/D converter.

2.2.6 PCM Interface

To sample sound via the PCM channel on Bluetooth a PCM receiver is needed. The PCM channel is a serial channel with four wires, clock, sync and one data line in each direction. In NetDrive data is only received from Bluetooth. Therefor only tree lines are needed. The pcmClk, pcmSync and pcmData signals are generated by Bluetooth. A data transfer is 8 or 16 bits long,. The PCM interface consists of a eight bit shiftregister, two eight bit registers and a counter, Figure 17. When the pcmSync comes the counter starts counting and pcmData is shifted in to the shift register. When the counter reaches 8 the data in the shift register is written to Register 1 and when it reaches 16 the data is written to Register 2 and a interrupt is generated. When 8bit PCM is used the extra 8 bits has to be removed by software.

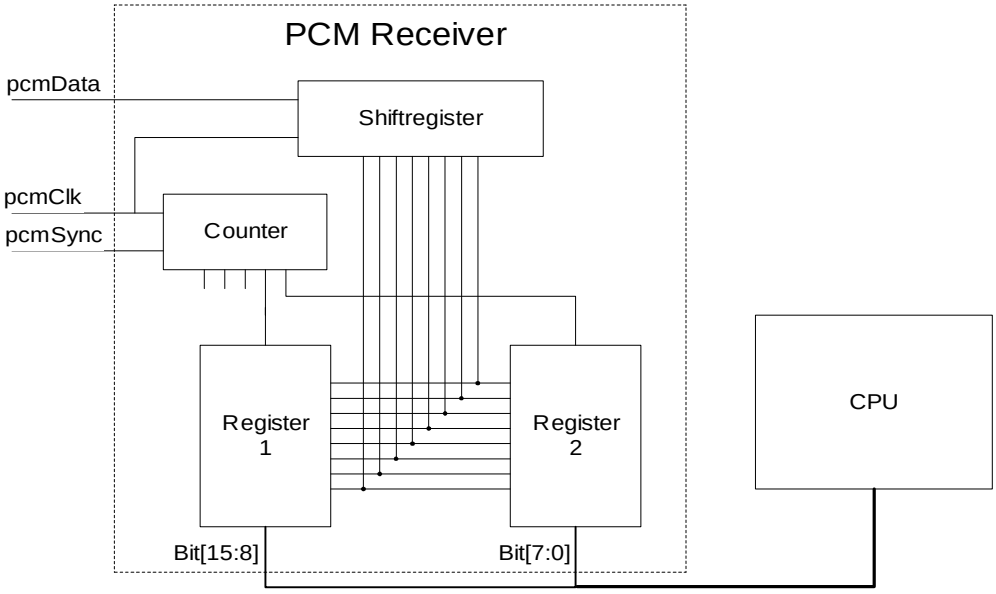


FIGURE 17. PCM receiver.

2.2.7 Real Time Clock

To keep track of time even when the battery has run out a real time clock, Dallas DS1302, with a small battery backup is used. To set the time in the real time clock or read time from the real time clock a serial communication protocol is used, Figure 18. Data becomes valid when the clock, SCLK, is low and data is read on the rising edge of SCLK. During a data transfer the first byte sent is a command byte that specifies if its a read or write and the internal address of the real time clock. Then data is sent in the second byte.

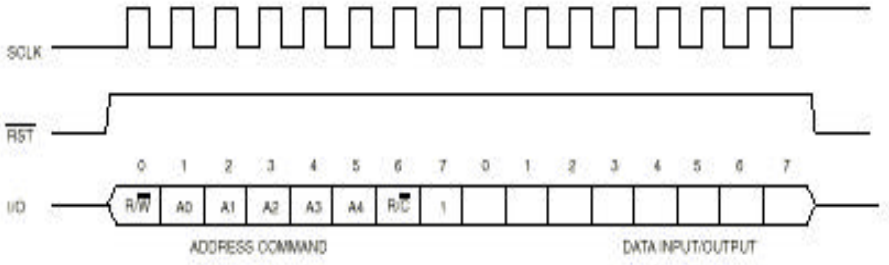


FIGURE 18. Serial protocol for the Real time clock. [5]

From the CPU there is no clock signal available so the whole protocol has to be generated by software. The interface to the real time clock has register for RST, SCLK and DataIO. The DataIO from the register to the real time clock is connected via a buffer to control input and output to the real time clock, Figure 19.

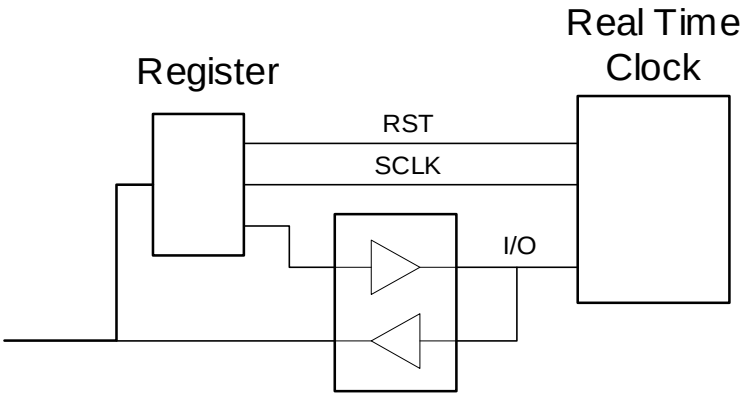


FIGURE 19. Block diagram for timekeeping chip

2.3 Lab Board

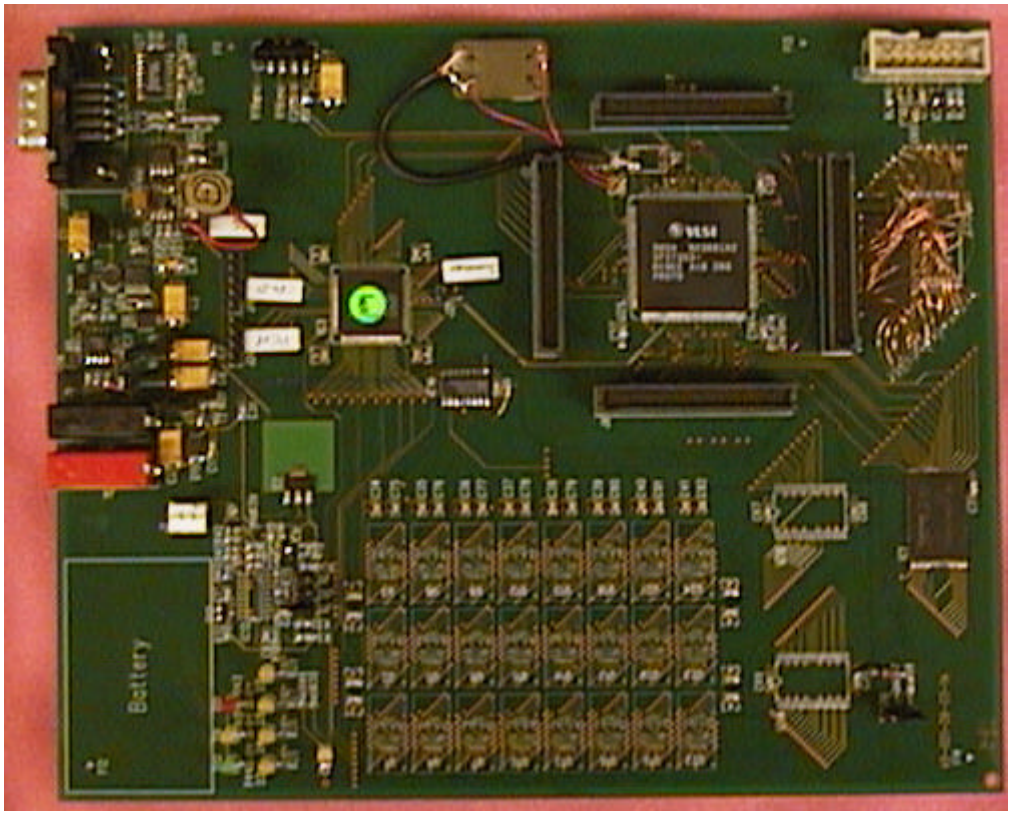


FIGURE 20. Lab Board

2.3.1 General

The LabBoard, Figure 20, is a large scale variant of the final product. The LabBoard is used to test all the components on the final card. Here all the functionality of the wire wrap board, flash card, PCM receiver, sensor, and charging electronics is built on the same board. The flash disk on the Lab board is full size, 192Mbyte, compared to 4Mbyte on the flash cards. On the LabBoard another debug feature is added with the possibility to connect a emulator from Lauterbach.

2.3.2 Changes From Wire Wrap Board

The most important changes compared to the wire wrap board is the CPLD that includes all off the address decoding to external units on the card and control of the Led's and the buzzer. The PCM receiver is also included in the CPLD. The SRAM is reduced to 512kbyte

2.3.3 Flash Disk

The flash disk consists of 64Mbit flash devices from Intel, StrataFlash [ref] in BGA, Ball Grid Array, packages. On the LabBoard 24 of these are mounted which gives a total storage capacity of 192Mbyte. The flash disk is designed identical to the final version to test mounting the BGA packages with only 1.5mm spacing and also to test routing on the card.

To access the new flash disk a new interface is designed, Figure 21. In the new interface two bus operations is needed. The first bus cycle specifies which flash device that will be accessed and the most significant bits of the internal address in the flash device. The second access specifies the lower 16 bits of the flash address and if data is read from or written to the flash memory. When the following flash access is near the last one, i.e. flash device and address bits specified in the first stage is the same, only stage two needs to be done.

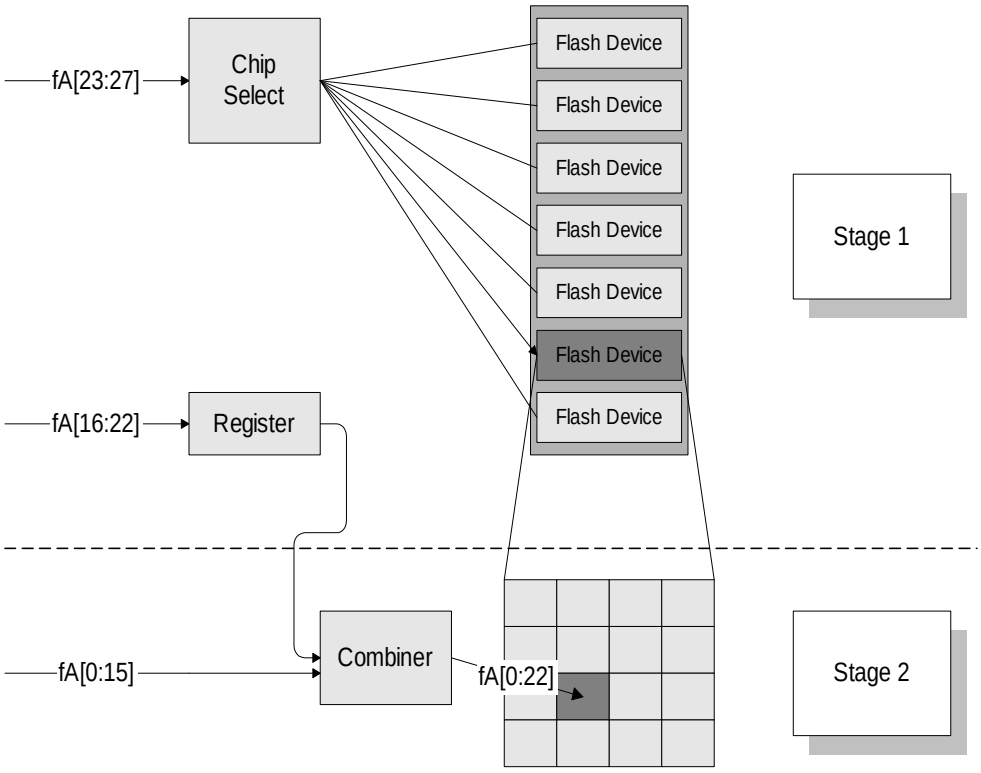


FIGURE 21. Flash interface.

2.3.4 Power Supply

The LabBoard has both 3.3V and 5V power supply. 5V is only used by the flash disk as a programming voltage. Its possible to switch of the power to all external units, both 5V and 3.3V, from the processor. The power to the processor is always on.

2.3.5 Lauterbach

With the connectors for the Lauterbach emulator, Figure 22, more options for software debugging was added compared to the Embedded ICE that were used in the wire wrap board. With the new emulator its possible to trace and record what happens when a program runs at full speed. With the ICE a breakpoint had to be set and then the memory contents could be checked.

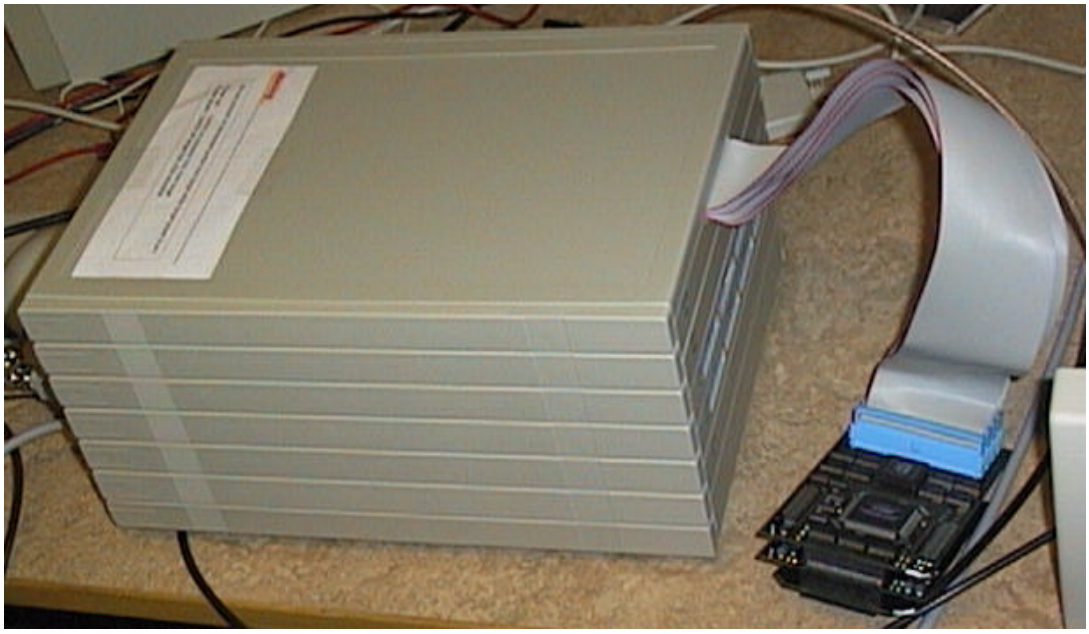


FIGURE 22. Lauterbach emulator.

2.4 NetDrive Motherboard

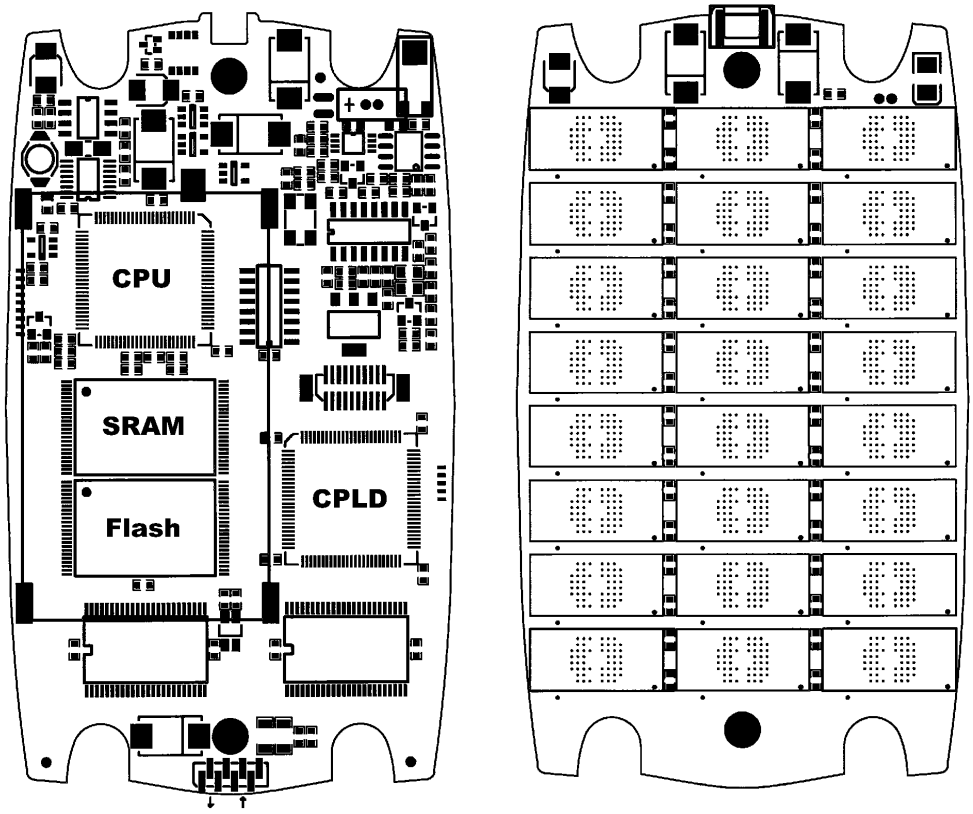


FIGURE 23. NetDrive motherboard. Top view (right) and bottom view (left).

When the functionality of the LabBoard was verified the design was transferred to the final NetDrive Motherboard. NetDrive Motherboard has all the functionality of the Lab-Board except for the debug options. NetDrive Motherboard is prepared for Bluetooth with a connector to a Bluetooth daughter card that Ericsson are developing.

3.0 ASIC Implementation

3.1 Presentation

The never ending struggle to build smaller and smaller devices often leads to development of an ASIC with most of the needed functionality included. The size of an ASIC is in most cases not decided by the amount of logic inside but of the number of pins. So when adding more functions to an ASIC care must be taken so that not more pins than necessary is added. In some cases the ASIC can be designed to have more than one function on some pins.

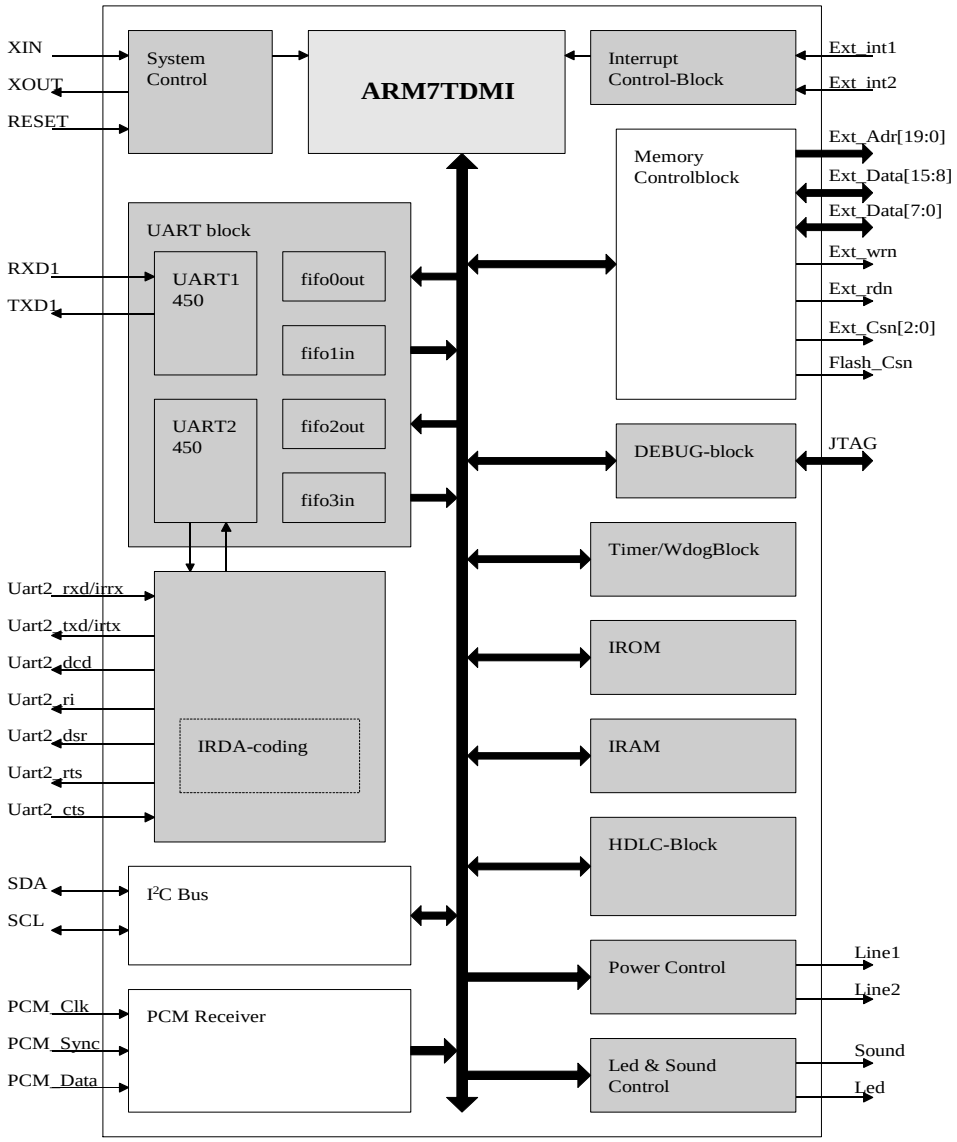


FIGURE 24. Block diagram of an ASIC for NetDrive, white blocks were implemented.

Another advantage with ASIC implementations is that power consumption can be reduced. In battery powered applications this is just as important as the size. Low power consumption increases the time until the batteries have to be recharged. Power consumption also affects the size, low power allows smaller batteries to be used.

In NetDrive most of the functions can be included in an ASIC. For NetDrive the ASIC should include system control, two UART block, internal RAM & ROM, debug block (JTAG), Timer/Watchdog block, HDLC-block, Power Control, Led & Sound Control, I^2C bus [6], PCM Receiver and a Memory Control block with a flash controller included, Figure 24. To implement all these blocks would take much more time than was available in this project. Therefor only the I^2C, PCM and Flash/Memory Control blocks are implemented here. The blocks are implemented in VHDL [7,8] code.

The Flash Controller is included in the Memory Control block. The Flash Controller when in 8bit mode uses the 8 unused data bits as high address bits. This allows the Flash Controller to access large memory areas in one bus cycle.

The I^2C bus is a general two-line serial data bus. The I^2C standard was developed by Philips in the 70's. The I^2C bus can have multiple slaves and also multiple masters connected. This allows external AD converter and timekeeping chip to be connected without any extra logic for address decoding. In the future more units can be added to the bus without unnecessary external logic.

The PCM receiver is used for the PCM channel from Bluetooth to store sound on the disk.

The UART block is thought to handle the serial communication with Bluetooth, UART 1, and IrDA, UART 2. To be able to reduce power consumption of external logic there are a Power control block. This block has two on/off signals to control the power supply. The Led & Sound control block is thought to handle light and sound signals to the user without interference from the processor. The processor is only supposed to give start and stop commands to this block. The JTAG block is included for debug purpose in software development.

3.2 Flashdisk Interface

3.2.1 General

The flashdisk interface is intended to be included in the memory controller. Here a memory controller is implemented with 8 and 16 bit data bus and wait states, 0-7. Wait states is used when the memory access needs more than one clockcycle. The number of waitstates indicates how many extra cycles that are added to the memory access. The address bus is 20 bit wide. There are also four Chip Select so that four different memory banks, or other external units, can be accessed.

To this is then the flash interface added. The flash interface has the same features as the memory controller. What the difference between the two is when a 8 bit bus is used. The

flash interface uses the remaining 8 data bits as high address bits for easier access to large memory areas.

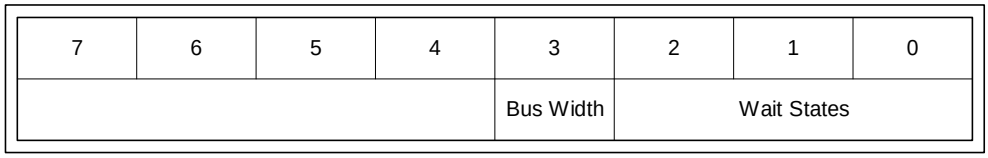
There are three enable signals, Enable_Mem, Enable_FlashD and Enable_Internal. To enable the different operation modes of the memory controller. Enable_Internal is used when data is read or written to the control registers. Enable_Mem and Enable_FlashD are used for memory respectively flash access.

3.2.2 Control Registers

The memory control block has five registers, Figure 25. One for each of the four Chip Selects and the Flash_Csn. The registers are 8 bits but only four are used. Three bits are used to set number of wait states and one for 8 or 16 bit bus, Figure 26.

Address[31:0]	Rgister
00800300	Ext_Csn[0]
00800304	Ext_Csn[1]
00800308	Ext_Csn[2]
0080030C	Ext_Csn[3]
00801000	Flash_Csn

FIGURE 25. Control registers and their addresses on the internal bus.



Bit	Name	Function
7:4		Spare for future use
3	Bus Width	0 = 8 bit, 1 = 16 bit
2:0	Wait States	000 = 7 100 = 3 001 = 6 101 = 2 010 = 5 110 = 1 011 = 4 111 = 0

FIGURE 26. Control Register

3.2.3 Wait States

Wait states are used to halt the processor when a slow memory is accessed. The processor is halted by pulling nWAIT low which stops the internal twophase clock in the processor, Figure 27.

```
IF ECLK = '1' AND ph2 = '1' THEN
   TmpWait:= 7;
ELSE
   IF ph1 = '1' THEN
```

```
      WaitState:= TmpWait;
   END IF
END IF;

IF WaitState > TO_INTEGER('0' & IOspeed(2 downto 0)) THEN
   nWAIT <= '0';
   IF ph2 = '1' THEN
      TmpWait:= WaitState - 1;
   END IF;
ELSE
   nWAIT <= '1';
END IF;
```

FIGURE 27. VHDL code to set number of wait states. IOspeed = bit[2:0] in the control register for the accessed memory or flash area.

The number of wait states is set in the control register. A counter, WaitState, that counts down from 7 holds nWAIT low while WaitState are greater than the value in the control register. During phase two of the cycle before the memory access the address is decoded and the control register of the accessed memory area is read, the number of wait states is put in the variable IOspeed. At that point the counter is also reset. During phase 1 Wait-State is updated and compared with IOspeed. If wait state is greater than IOspeed nWAIT is held low and during phase 2 the counter is decremented. Otherwise nWAIT is pulled high.

3.2.4 Read/Write

Read and write operations can be done in five different modes. The access can use a 8 or 16 bit bus. When a 8 bit bus is used the processor can read or write data in three different sizes, byte, half word (16 bit) and word (32 bit). With a 16 bit bus it can read or write data in two sizes, half word and word.

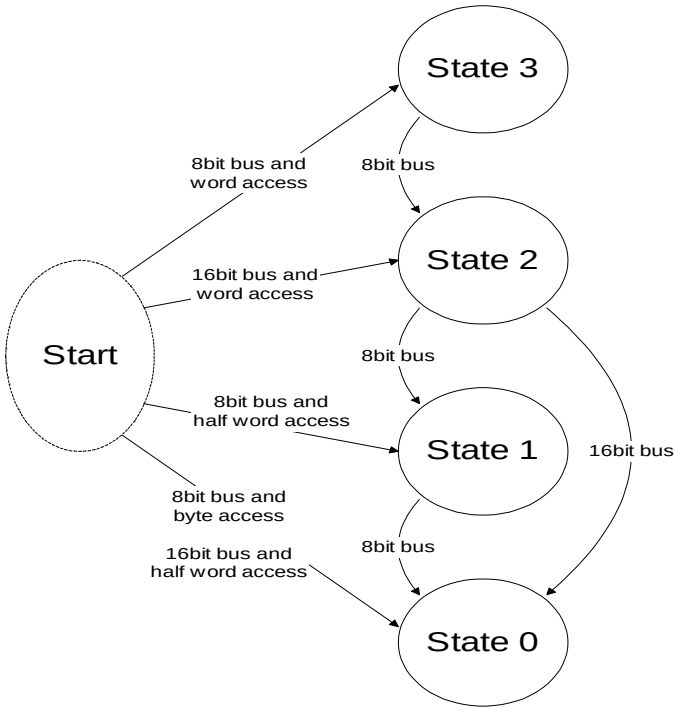


FIGURE 28. Different states during Read/Write operations.

The different states indicate offset in the memory access. When a word size read to a memory with 8bit bus occurs it starts in state 3, bit[31:24] is read, Figure 28. Next bus cycle the state machine is in state 2 and bit[23:16] is read. Third bus cycle bit [15:8] is read and the fourth bus cycle bit[7:0] is read, Figure 29. A word access with 16 bit bus starts in state 1 and reads bit[31:16].

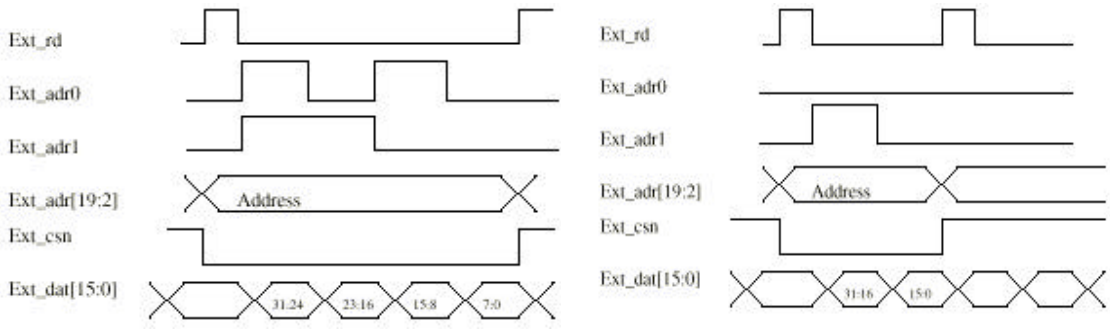


FIGURE 29. Word size read with 8 and 16 bit bus. [2]

During state 3, 2 and 1 nWAIT is constantly held low. In those cases nWait is not pulled high when WaitState equals IOspeed, only the state is changed, Figure 30.

During write operations the address and data signals must have time to stabilize before the nWRITE signal goes low.

```
IF nRW = '1' THEN
   IF WaitState = TO_INTEGER("01" & IOspeed(2 downto 1)) THEN
      IF IOspeed(0) = '1' AND TmpWait = 7 THEN
         nWRITE <= NOT ph2;
      ELSIF IOspeed(0) = '1' AND ph2 = '1' THEN
         nWRITE <= '0';
      ELSIF IOspeed(0) = '0' THEN
         nWRITE <= '1';
      END IF;
   ELSIF WaitState > TO_INTEGER("01" & IOspeed(2 downto 1)) THEN
      nWRITE <= '1';
   ELSIF TmpWait = 7 AND ph2 = '0' THEN
      nWRITE <= '1';
   ELSE
      nWRITE <= '0';
   END IF;
END IF;
```

FIGURE 30. VHDL code to set nWAIT.

3.2.5 Flash Access

A flash access can be with either a 8 or 16 bits data bus. When 16bit bus is used there are 20 addressbits available. When 16bit bus is used address bit[0] is not used. This allows access to 1Mbyte flash in one bus cycle. Larger areas needs external logic for address decoding and more than one bus cycle is needed for the access. When using 8bit bus the 8 unused data bits is used as high addressbits, total 28 addressbits. This allows access to 256Mbyte in only one bus cycle.

3.3 I^2C Block

3.3.1 General

The I^2C block is built as a Master with a two byte read and two byte write buffers. In the I^2C standard there is also support for multiple masters and 7 or 10 bit addressing, this is not implemented in this design.

The I^2C bus communication is bytewise; i.e. a byte is shifted out with the MSB first and finishes with the LSB. Each sent byte must then be acknowledge by the receiver otherwise the transmission will be aborted. When waiting for acknowledgment the sender leaves the dataline high and the receiver pulls the line down. When a master wants to communicate with a slave the first byte include 7 addressbits and a read/write bit. If a slave's address matches the sent one it sends an acknowledge bit. Depending on if it was a read or a write command, the master or the addressed slave then starts sending data to the other. When the slave is the sender the master terminates the data transfer by not sending an acknowledge after the last byte, Figure 31.

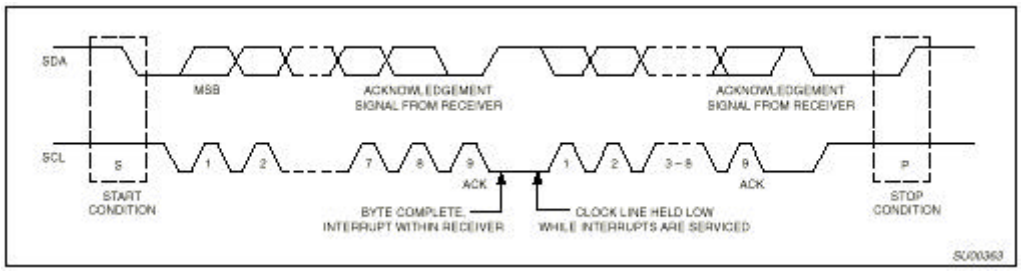


FIGURE 31. Bus protocol. [6]

The I^2C bus works as a wired AND, Figure 32. It means that the data and clock lines are default high and goes low as soon as a unit pulls the signal low. To hold the lines high all units connected to the bus must hold the signal high. as soon as one unit pulls the signal down all other units are affected.

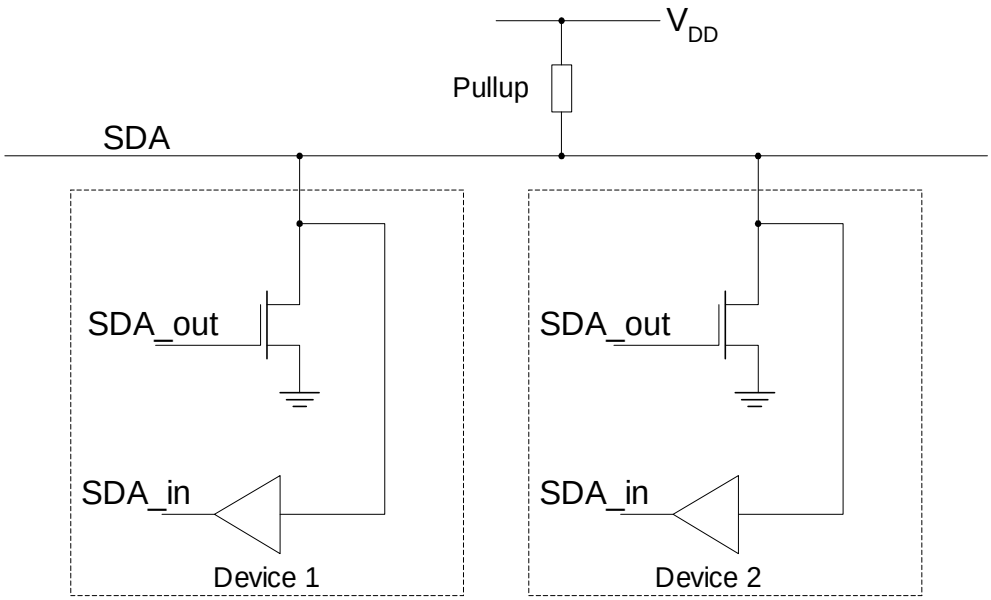


FIGURE 32. Wired AND for the serial data line. When the output transistor is closed the data line holds a high level. When a transistor opens the data line is pulled low.

To solve this in the ASIC design both a tristate output buffer and an input buffer are used for each of the two lines, Figure 33. The input always reads from the data or clock line. The output buffer can either be in tristate or open. When in tristate the data or clock line is left high, unless some other unit on the bus pulls the line low. When the buffer is open the line is pulled low.

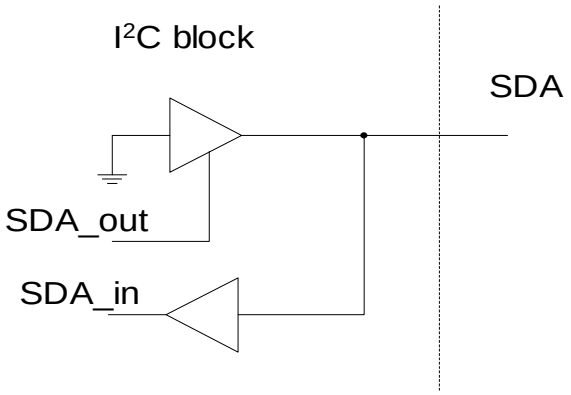


FIGURE 33. In/Out buffer on I²C block in ASIC. When the output buffer is in tristate the data line is unaffected. When the buffer is open the data line is pulled low.

3.3.2 Control Block

The Control block in the I²C interface has a 9bit register to control clock division and interrupt. Bit[6:0] is the clock divider. Bit[8:7] is mask bits for the interrupts, the interrupt is enabled if the corresponding bit is 0 and disabled if it's 1.

3.3.3 Clock Generator

All units connected to the I²C bus has an internal clock that affects the clock line, SCL, on the bus. To synchronize the different units the wired AND behavior of the SCL is used. When the clock on one of the connected units goes low SCL goes low and it affects all the other units. When SCL goes low all units starts counting off its low period and then goes high again. The units doesn't start counting off its high period until all other also has reached a high state. The time SCL is high is therefor decided by the unit with the shortest high period and the time SCL is low is decided by the unit with longest low period.

The I2C bus uses a 100kHz clock therefor the internal clock has to be divided down to the right frequency. How much the internal clock shall be divided is specified in the control register. A counter increments on every clock cycle and when it reaches the value specified in the register the counter resets and the SCL_out signal is inverted.

The wired AND behavior of the SCL line helps the different units on the bus to synchronize. If SCL goes low before the high period is counted the counter is reset and SCL_out goes low. If one or more of the other units is still in it's low state, SCL is low, when SCL_out goes high the counter is halted until SCL goes high, Figure 34.

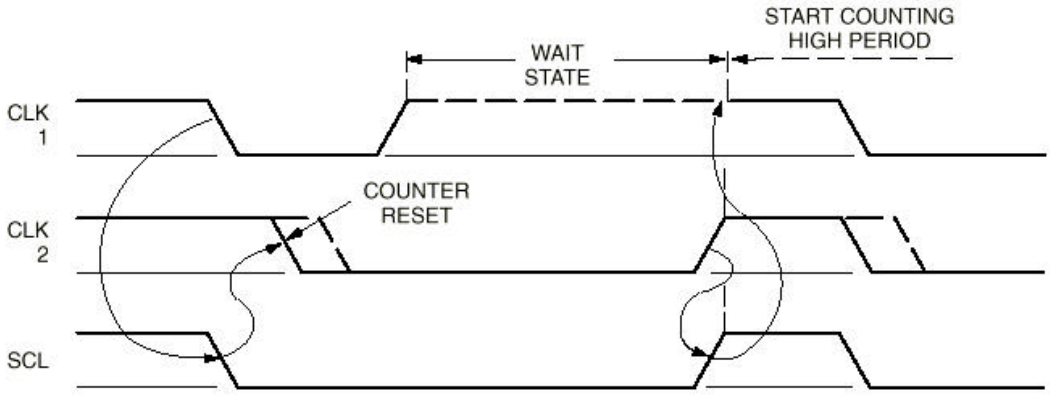


FIGURE 34. Clock synchronization. [6]

The clock generator also sends a signal to the data controller to indicate the middle of the high and low periods.

3.3.4 Data Control

For each read or write operation on the bus the data control reads 16bit data from the output fifo. The most significant byte is the address and the read/write bit. If it's a read operation the other byte is ignored otherwise it's sent after the address.

The data line, SDA, changes value during the low period of SCL and keeps that value throughout the high period of SCL. The only times SDA changes value during high period of SCL is when a start or stop sign is sent, Figure 35. A high to low transition on SDA while SCL is high indicates a start sign and a low to high transition is a stop sign. A signal from the clock generator indicates when the value on SDA shall change value for start and stop signs.

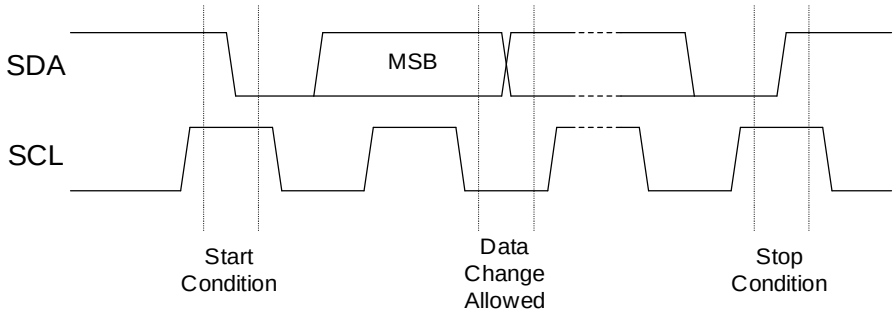


FIGURE 35. Start, stop conditions and change of data on the bus.

The Data Control has six states which represents the actions Hold, Send Data, Receive Data, Wait, Acknowledge and Stop, Figure 36. The states is changed during the high period of SCL.

Hold waits for a bus transfer to be initiated. When a transfer is initiated the state changes to Send Data.

Send Data shifts out address or data with the most significant bit first. As the least significant bit is sent the state changes to Wait.

Wait stops the data sending to allow the slave to send the acknowledge bit. This state is only held for one bus cycle and is then changed to Stop or Acknowledge depending on if transmission is completed or not.

Acknowledge state checks if the acknowledge bit is sent from the slave. If no acknowledge is sent a stop sign will be sent and the state changes to Hold. If a acknowledge signal is detected and it's a read operation the state is changed to Receive Data. Otherwise the sate changes to Send Data

Receive Data shifts in data from the slave with the most significant bit first. When all eight bits is received the state changes to Stop.

Stop sends a stop sign and changes the state to Hold.

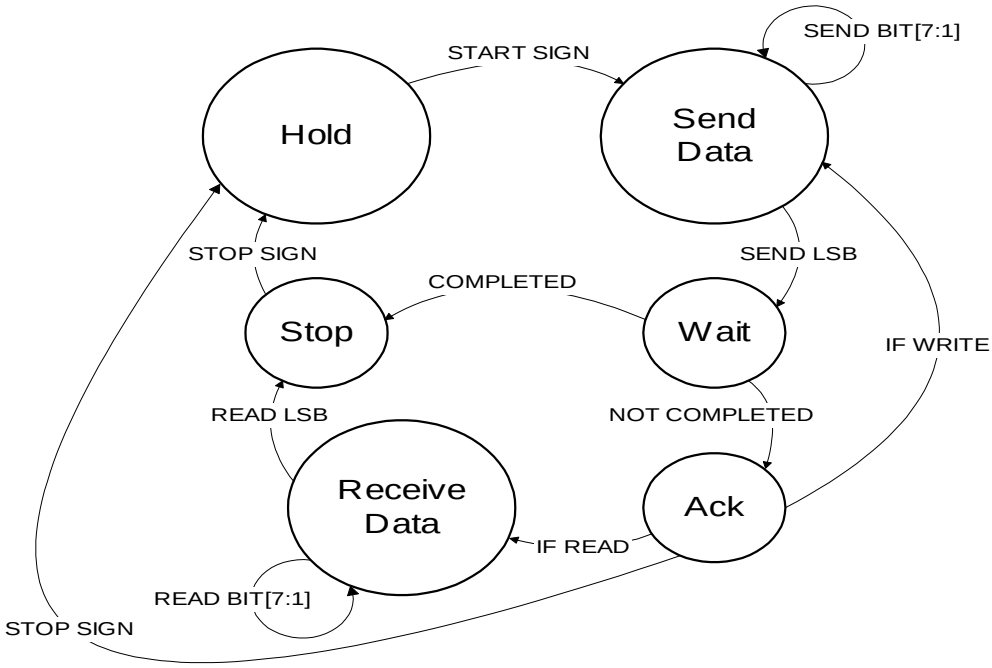


FIGURE 36. Function of the Data Control.

3.3.5 Fifo

The output fifo is four byte. The Data Controller reads 16 bit data from the fifo, bit[15:8] is address and read/write bits and if write operation bit[7:0] is data to send otherwise these bits are ignored. Data can be written to the fifo as byte, half word (16 bit) or word (32 bit). When byte format is used data to the fifo must be written with bit[15:8] first and then bit[7:0]. When word format is used bit[31:16] are the first bus access and bit[15:0] are the second. When the fifo is empty an Interrupt is sent.

The input fifo is only one byte. When the Data Controller reads data from the I^2C bus it's written to the fifo. When the fifo has new data an interrupt is sent.

In the future the sizes of the fifo's can easily be expanded.

3.3.6 Test Block

The I^2C interface is designed as a master and to test it, Figure 37, a slave was built. After a start sign is sent the slave receives one byte and compares it to the internal address register. If the address received is the same as in the register the slave sends or receives one byte depending on the read/write bit. The procedure for send and receive data is identical to the master.

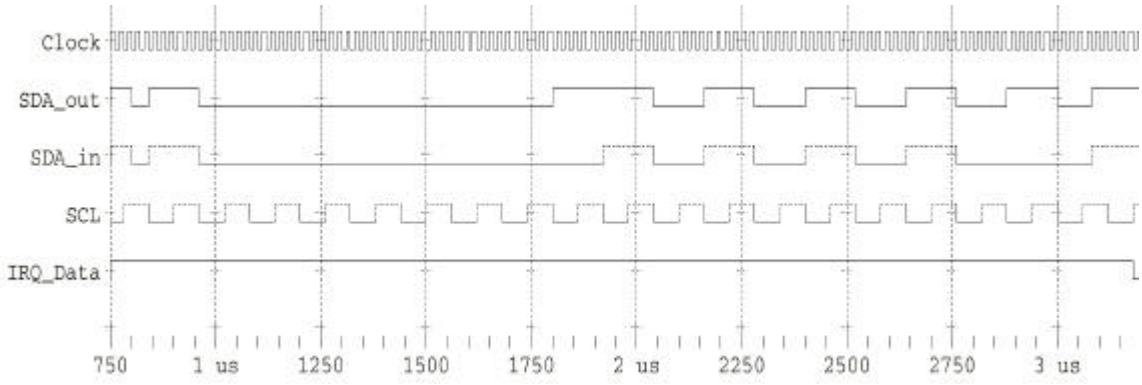


FIGURE 37. Test of I^2C block. The Master sends a Write to a Slave. The Slave acknowledge the write command and the Master starts sending Data (0xAAAA). When the transmission is completed the Slave acknowledge again.

3.4 PCM Receiver

3.4.1 General

To sample sound via the PCM channel on Bluetooth a PCM receiver is needed. The PCM channel is a serial channel with four wires, clock, sync and one data line in each direction. In NetDrive data is only received from Bluetooth. Therefor only tree lines are needed. The pcmClk, pcmSync and pcmData signals are generated by Bluetooth. A data transfer is 8 or

16 bits long. The interface implemented here has only a 16bit mode. When 8bit PCM is transferred the extra eight bits can be removed by software. The transfer is initialized by pcmSync that becomes valid the cycle before data is sent. The pcmSync and pcmData becomes valid during the high phase of pcmClk, Figure 38 and Figure 41.

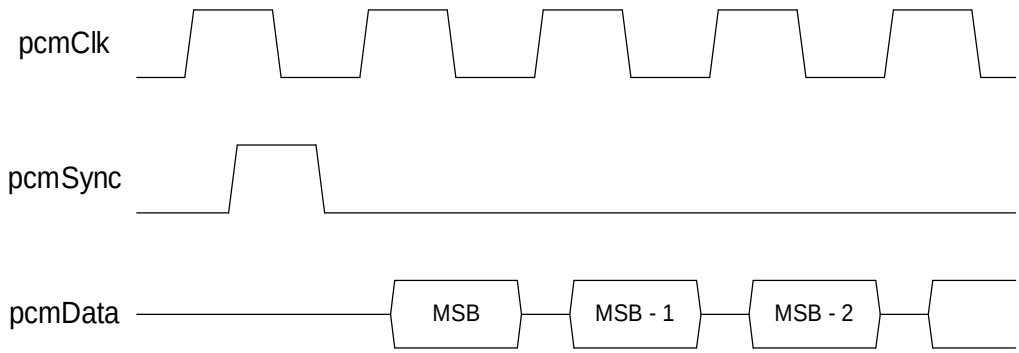


FIGURE 38. Timing of PCM

The PCM receiver has three blocks, control, fifo and receiver. The control block controls interrupt and data output from the fifo. The Receiver shifts in 16bit data from the PCM bus and writes it to the fifo. The fifo has one position. If interrupts are enabled an interrupt is generated when new data is written to the fifo. In the future the fifo can easily be expanded.

3.4.2 Control Register

The control register is a eight bit register located at address 0x00801008. Of the eight bits in the register only one is used, bit[0] = En_IRQ. Bit[0] = 1 disables and 0 enables interrupt.

When a read from the PCM receiver occurs the control sends a write signal to the fifo to send out data.

3.4.3 Receiver

The receiver is a small statemachine with two states, Hold and Read, Figure 39. When the pcmSync comes the state changes from Hold to Read and 16 data bits are shifted in on the falling edge of pcmCLK. When all 16 bits are shifted in the data is written to the fifo and the state changes back to Hold.

```
IF PCM_CLK = '0' AND PCM_CLK'EVENT THEN
  CASE STATE IS
    WHEN 0 =>                                   --   HOLD
      IF PCM_SYNC = '1' THEN
        NextState := 1;
```

```
        ELSE
          NextState := State;
        END IF;
        NextBitPos := 15;
        Data := NextData;
      WHEN 1 =>                                    --  READ
        Data(0) := PCM_DATA;
        Data(15 downto 1) := NextData (14 downto 0);
        NextBitPos := BitPos - 1;
        IF BitPos = 0 THEN
          NextState := 0;
        ELSE
          NextState := STATE;
        END IF;
      WHEN OTHERS =>
        null;
  END CASE;
ELSIF PCM_CLK = '1' AND PCM_CLK'EVENT THEN
  State := NextState;
  BitPos := NextBitPos;
  NextData := Data;
END IF;
```

FIGURE 39. VHDL code for shifting data in to the PCM receiver

3.4.4 Test Block

The test block designed to test the PCM receiver sends data from a small internal memory, 256 x 17 bit. In the memory bit[16] are pcmSync and bit[15:0] are the pcmData. The pcm-Clk is driven with half the frequency of the system clock, i.e. pcmClk is inverted on the rising edge of MCLK, Figure 40.

```
IF MCLK = '1' AND MCLK'EVENT THEN
  Clk := NOT Clk;
  State := NextState;
  Adr := NextAdr;
  Count := NextCount;
ELSIF MCLK = '0' AND Clk = '1' THEN
  CASE STATE IS
    WHEN 0 =>
      PCM_SYNC <= mem(Adr)(16);
      NextState := STATE + 1;
      NextCount := 15;
    WHEN 1 =>
      PCM_SYNC <= '0';
      PCM_DATA <= mem(Adr)(Count);
      NextCount := Count - 1;
      IF COUNT = 0 THEN
        NextAdr := Adr + 1;
        NextState := 0;
      END IF;

    WHEN OTHERS =>
      null;
  END CASE;
ELSIF MCLK = '0' AND Clk = '0' THEN
```

```
   PCM_SYNC <= '0';
END IF;
PCM_CLK <= Clk;
```

FIGURE 40. VHDL code for the PCM test block.

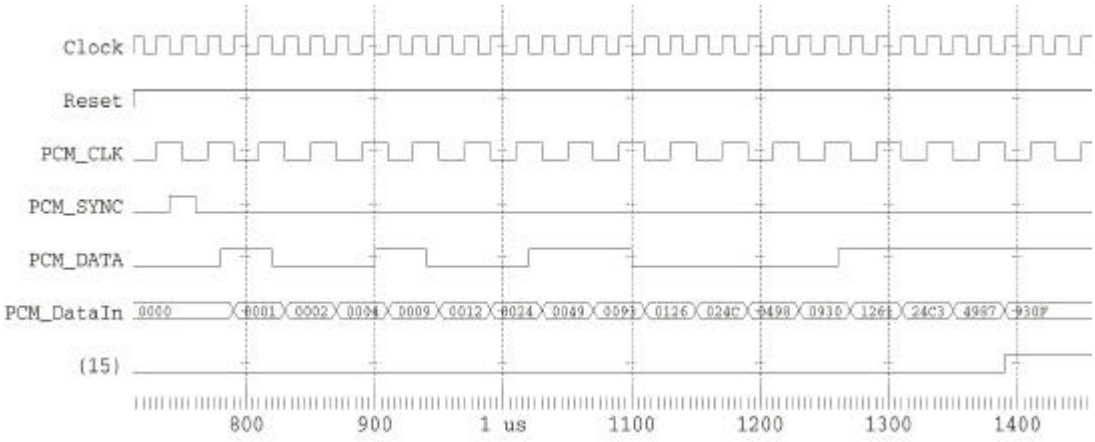


FIGURE 41. Test of PCM. 0x930f sent from test bock to PCM receiver.

4.0 RisARM

4.1 Introduction

The central part of the ASIC design is the ARM7TDMI core. In order to communicate with peripheral units, all units must use the same bus protocol as the ARM core. Therefor all peripheral units had to be designed carefully and then meticulously tested. To test the VHDL designs the VHDL code for the ARM7TDMI or something that simulated the behavior of the ARM7TDMI bus was needed. Because the VHDL code for the ARM7TDMI was not available the RisARM, Reduced Instruction Set ARM, was created. The RisARM is built to have the same bus protocol and timing as the ARM processor. Since the bus protocol of the ARM7TDMI is very complex and can work in different modes, special care has to be taken in that part of the design. The RisARM consists of an instruction interpreter with a shell for the bus timing, Figure 42. The RisARM is designed to execute 16 bit THUMB code [9]. THUMB code is one of the two instructionset the ARM7TDMI can execute, the other one is 32 bit ARM code. THUMB code is specially designed for applications where there are restrictions in code density. In the RisARM THUMB code is implemented because NetDrive is supposed to execute THUMB code and that gives the potential to execute real code designed for NetDrive. Today that is not possible because the RisARM has only 14 instructions implemented. More instructions can easily be added.

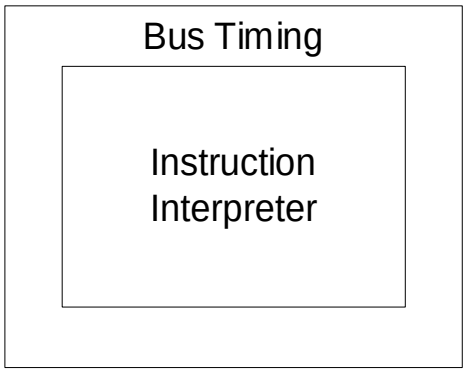


FIGURE 42. Structure of RisARM. Instruction interpreter with a bus timing interface outside.

4.2 ARM7TDMI

4.2.1 General

The ARM7TDMI architecture, Figure 44, is a 32bit RISC processor with a three-stage pipeline. The processor has 37 registers, 31 general-purpose 32bit registers and six status-registers. All of these registers are not visible at the same time, which registers that are visible are dictated by the processor state. In ARM mode 16 registers and two status registers are visible and in THUMB mode register[7:0], Program Counter (PC), Stack Pointer

(SP), Link Register (LR) and two status registers are visible, Figure 43. In both ARM and THUMB mode Stack Pointer, Link Register and PC are mapped to register 13-15. The two statusregisters are CPSR, Current Processor Status Register, and SPSR, Saved Processor Status Register.

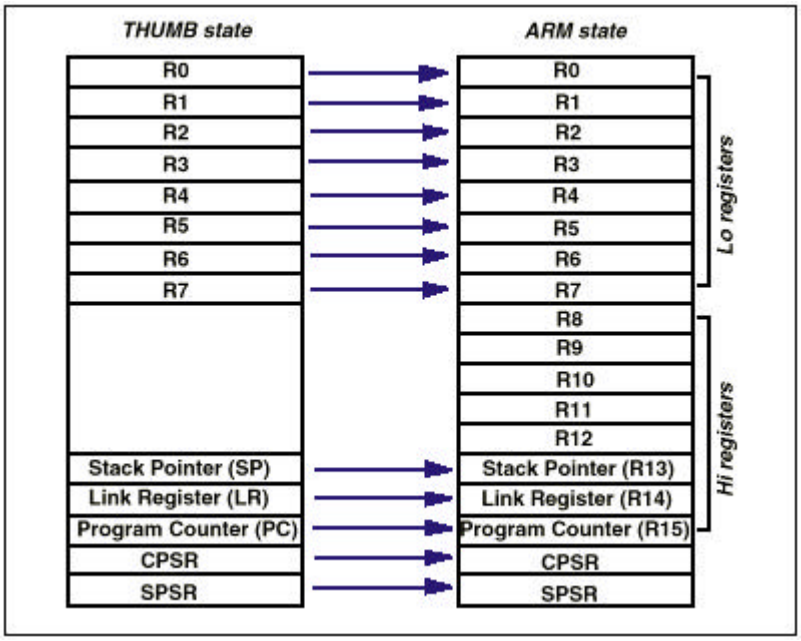


FIGURE 43. Mapping of registers in THUMB and ARM modes. [3]

The ARM7TDMI can execute both 32bit ARM code and 16bit THUMB code. THUMB code is well suited for applications where there are restrictions with memory access, 8 or 16 bit memory busses, and applications with restrictions in code density.

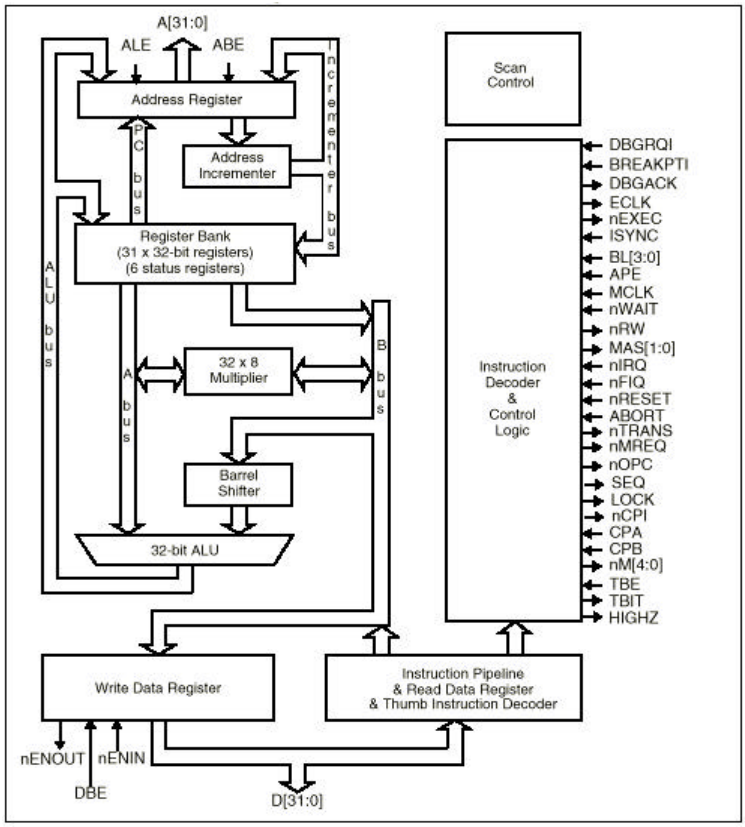


FIGURE 44. ARM7TDMI core. [3]

The processor is clocked with a single clock, MCLK. Internally a non-overlapping twophase clock is used. In the twophase clock phase one goes high when MCLK is low. When MCLK goes high phase 1 goes low and then phase 2 goes high, Figure 45. Non-overlapping means that one phase can’t go high until the other phase is low. This is to avoid race through in the pipeline. The internal twophase clock can be slowed down by pulling nWAIT low. With nWAIT low phase 1 will be stretched until nWAIT is released again. The internal clock is reflected in ECLK.

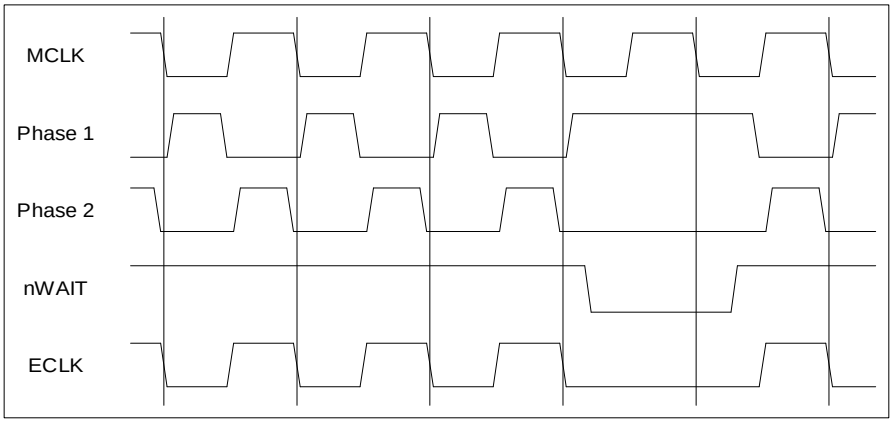


FIGURE 45. Clock signals. [3]

The processor has both a bidirectional data bus and unidirectional Detain and DataOut busses. The unidirectional data busses are provided because bidirectional busses are prohibited in some ASIC design methodologies. When using a bidirectional data bus special care must be taken to avoid clashes on the bus. Clashes are avoided by adding some logic outside the ARM core controlled by nENOUT, Figure 46. Most of the time the processor reads instructions from the memory, therefore with nENIN default high allows data to be read from memory. When a write operation occurs nENIN is driven low which pulls nEN1 high and disables input. When nEN1 goes high nEN2 will go low and enable the processor to drive the bus.

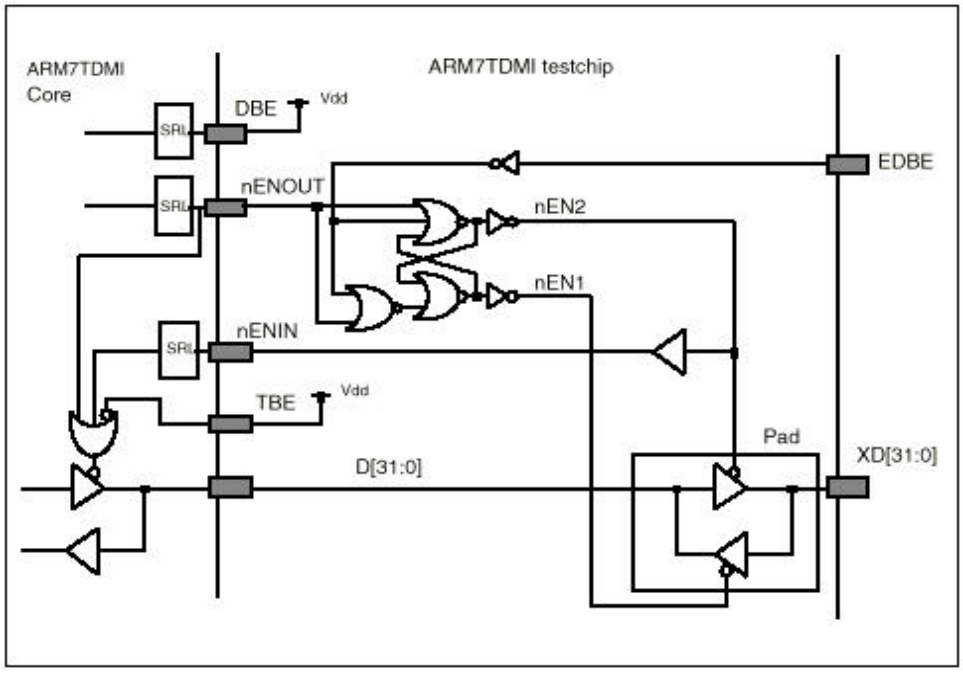


FIGURE 46. Bus control logic. [3]

4.2.2 Bus Timing

The ARM7TDMI memory bus has two modes, pipelined and depipelined. This is controlled by APE, address pipeline enable. To put it in depipelined mode APE is pulled low. APE controls the ADDRESS, MAS, nRW, nTRANS, LOCK and nOPC signals.

When in depipelined mode the signals become valid during phase 1 of the actual cycle and remains valid until phase 1 the following cycle, Figure 47.

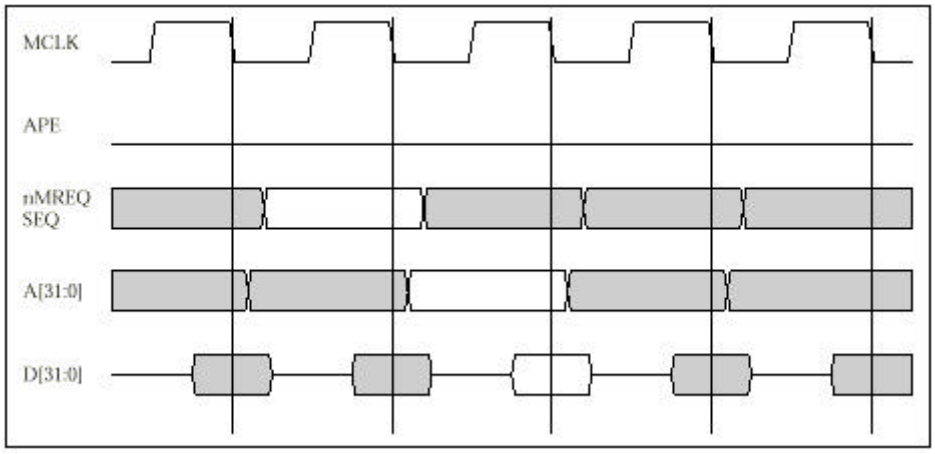


FIGURE 47. De-pipelined addressing. [3]

When in pipelined mode the signals become valid during phase 2 of the cycle before the memory cycle it refers to. The signals remain valid until phase 2 of the referring cycle, Figure 48.

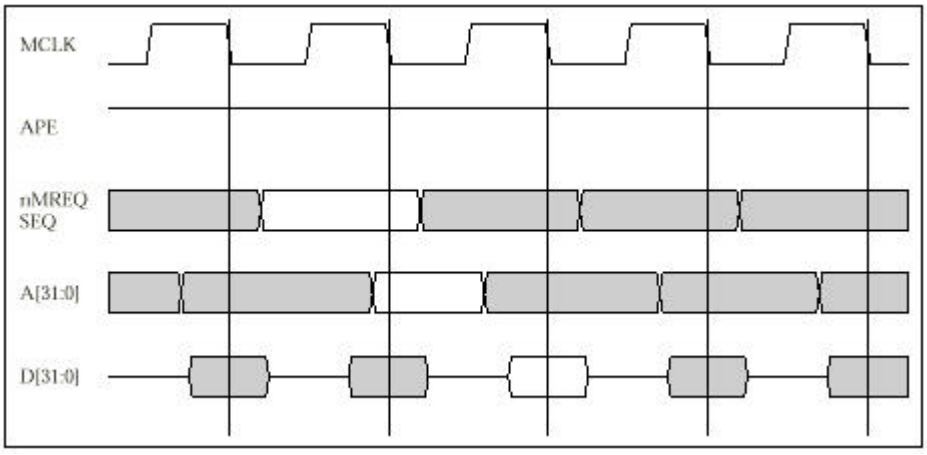


FIGURE 48. Pipelined addressing. [3]

In pipelined mode it’s also possible to stretch the time the signals are valid. This is done by pulling ALE low which freezes ADDRESS, MAS, nRW, nTRANS, LOCK and nOPC until ALE is released, Figure 49. This has to be used with care and is only kept for backward compatibility.

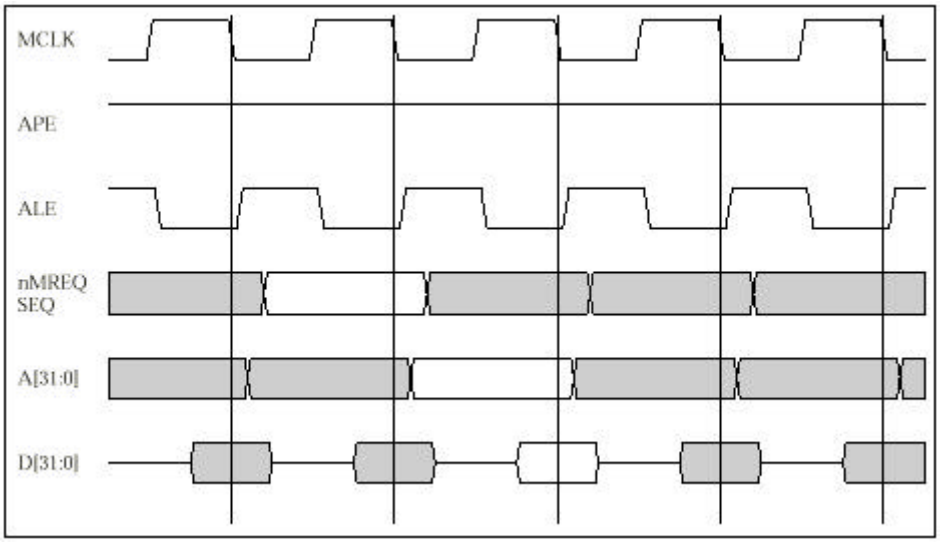


FIGURE 49. Latched addressing. [3]

4.2.3 Write Operation

The 32bit ARM architecture allows word, 32bit, half-word, 16bit and byte size write operations. The access size is defined in MAS, 00 = byte, 01 = half-word, 10 = word. 11 is saved for future use. The data lines changes value during phase 1 and remains valid through phase 2.

If the accessed unit, memory etc., needs more time than one clock cycle the signals can be held valid for a longer time by pulling nWAIT low.

4.2.4 Read Operation

Read operations can in the same way as write operations be word, half word or byte size. During read Data must be valid during the falling edge of phase 2. When the external memory bus is smaller than 32 bits, the memory controller can use BL, Byte Latch, to latch in data, Figure 50. By asserting appropriate bits in BL only valid data will be latched into the processor. If for example a word size read is issued and the memory has a 16bit bus, the memory controller can halt the processor with nWAIT and perform two 16bit reads. First set BL=0011and read bit 15-0, then set BL=1100 and read bit 31-16. Default BL=1111.

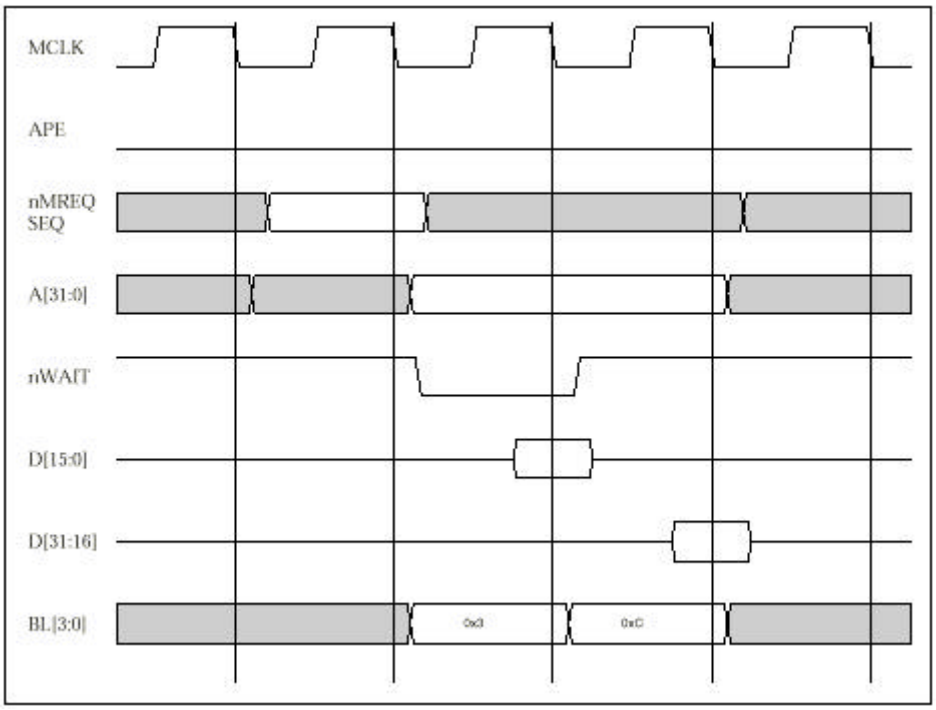


FIGURE 50. Data read where BL is used. [3]

4.3 RisARM Core

4.3.1 Structure/Dataflow

The RisARM is built to execute a small amount of instructions for the ARM processor. Therefor the internal structure of the RisARM is different than in the ARM processor. The RisARM uses a multicycle implementation and is clocked on the rising edge of phase1. During the first and second cycle, instructions are fetched. The second cycle is needed for the bus interface to simulate the timing of the ARM processor. The third cycle, instructions are decoded and executed. If a load or store operation is detected a fourth and fifth cycle is added for the memory access. In case of a load instruction data will be written to register, Figure 51.

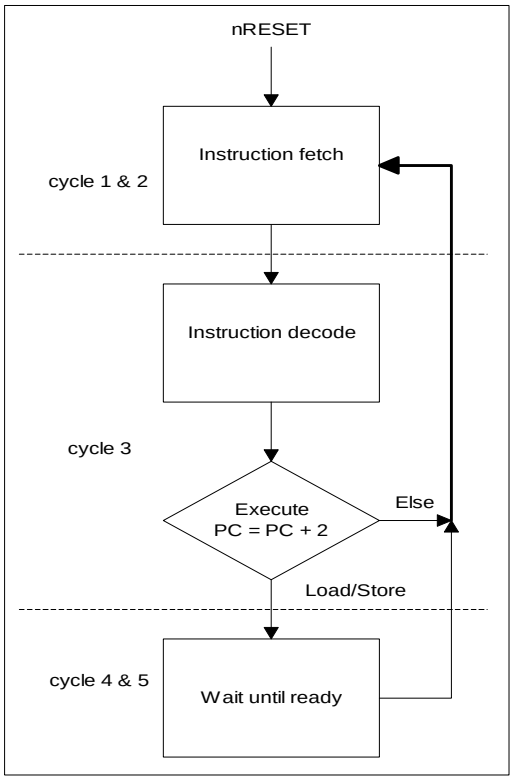


FIGURE 51. RisARM structure

The RisARM has 16 registers with PC in register 15. Stack Pointer and Link Registers are not used and status registers are not implemented.

To control the databus CoreControl is set to either ’1’ (read) or ’0’ (write). CoreControl controls nENOUT and the output buffers on the datalines, Figure 52.

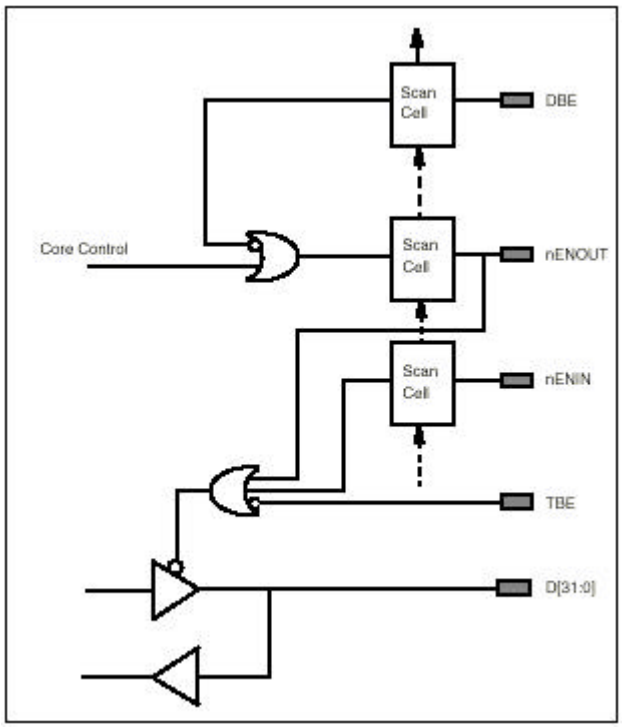


FIGURE 52. Control of the data output buffer. [3]

4.3.2 Instruction Set

The RisARM has 14 instructions implemented, Table 9. The instructions implemented is what was required to test the different designed blocks. The instruction set can easily be expanded with more instructions when needed.

TABLE 9. RisARM instruction set

Instruction	Operation	Note
LDR Rd, [Rn, #5bit offset]	Rd <= Rn + (5bit offset * 4)	Load Word
LDRH Rd, [Rn, #5bit offset]	Rd <= Rn + (5bit offset * 4)	Load Half Word
LDRB Rd, [Rn, #5bit offset]	Rd <= Rn + (5bit offset * 4)	Load Byte
STR Rd, [Rn, #5bit offset]	Rn + (5bit offset * 4) <= Rd	Store Word
STRH Rd, [Rn, #5bit offset]	Rn + (5bit offset * 4) <= Rd	Store Half Word
STRB Rd, [Rn, #5bit offset]	Rn + (5bit offset * 4) <= Rd	Store Byte
MOV Rd, #8bit imm.	Rd <= 8bit imm.	Move 8bit immediate to Rd
LSL Rd, Rm, #5bit shift imm.	Rd <= RM << 5bit shift imm.	Rm shifted left
ADD Rd, Rm, #3bit imm.	Rd <= Rn + 3bit imm	Add immediate
ADD Rd, Rn, Rm	Rd <= Rn + Rm	Add registers
ADD Rd, #8bit imm.	Rd <= Rd + 8bit imm.	Add immediate

TABLE 9. RisARM instruction set

Instruction	Operation	Note
ADD Rd, Rm	h1 & Rd <= h1 & Rd + h2 & Rm	Add high registers
ADD Rd, PC, #8bit imm.	Rd <= 0xfffffffc & PC + 8bit imm.	Add PC and Immediate
NOP		

4.4 RisARM, Bus Interface

The bus interface simulates the timing of the ARM7TDMI bus. All signals from the core are latched in during phase1. nRW are latched in as the inverse of nWRITE. When the memory access is completed nREADY will go low. This signal is set low if nWRITE or nREAD are low, “nREADY <= nWRITE AND nREAD;”.

4.4.1 Write Operation

Depending on ALE and APE ADRESS, MAS, nRW, LOCK, nOPC and nTRANS is then latched out on phase 2 or the following phase 1. SEQ, nMREQ, MODE and BUSDIS are latched in during phase 1 and direct latched out. The rest of the signals are latched out during phase 2 the following cycle, Figure 53.

```
PROCESS(D_in3, A_in2, A_in3, APL, ALE, ABE) IS
  IF ABE = ‘0’ THEN
    A_OUT <= “ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ”;
  ELSE
    IF APE = ‘1’ AND ALE = ‘1’ THEN
      A_OUT <= A_in2;
    ELSIF APE = ‘0’ AND ALE = ‘1’ THEN
      A_OUT <= A_in3;
    ELSE
      null;
    END IF;
  END IF;
  D_OUT <= D_in3;
END PROCESS;

PROCESS(ph1, D_in, D_in2, A_IN, A_in2) IS
  IF ph1 = ‘1’ THEN
    A_in1 <= A_IN
    A_in3 <= A_in2
  END IF;
END PROCESS;

PROCESS(ph2, D_in1, A_in1) IS
  IF ph2 = ‘1’ THEN
    A_in2 <= A_in1
    D_in2 <= D_in1;
  END IF;
END PROCESS;
```

FIGURE 53. VHDL code for address and data timing. MAS, nRW, LOCK, nOPC and nTRANS has similar timing as the address. CoreControl has the same timing as data.

4.4.2 Read Operation

During read operations all signals from the core are latched in the same way as during write operations. Data from external units is latched in asynchronously with BL, byte latch. BL is a 4-bit signal where the different bits latches in one byte each from the data-lines. Data latched in with BL is then forwarded to the core on phase 2, Figure 54.

```
PROCESS(BL, DataIn from external units) IS
  IF BL(0) = '1' THEN
    Data(7 downto 0) <= DataIn(7 downto 0);
  END IF;

  IF BL(1) = '1' THEN
    Data(15 downto 8) <= DataIn(15 downto 8);
  END IF;

  IF BL(2) = '1' THEN
    Data(23 downto 16) <= DataIn(23 downto 16);
  END IF;

  IF BL(3) = '1' THEN
    Data(31 downto 24) <= DataIn(31 downto 24);
  END IF;
END PROCESS;

PROCESS(ph2, Data) IS
  IF ph2 = '0' THEN
    DataOut <= Data;
  END IF;
END PROCESS;
```

FIGURE 54. VHDL code for data read.

The asynchronous read of data with BL allows data to be latched in byte for byte when the processor is halted with nWAIT. When nWAIT is released, data will be latched in to the core. This it useful when word size data is read from a memory with 8 or 16 bit access or when half word data is read from memory with 8 bit access.

5.0 Conclusions

The hardware in NetDrive consists of an ARM7TDMI processor, SRAM, program flash, flash array for data storage, A/D converter, Real Time Clock, PCM receiver and some I/O. Of all the units in the design very few uses the same bus protocol. In the hardware design much of the work was to design the interface between these peripheral units and the ARM processor.

In the suggested ASIC design these problems is solved by including most of the peripheral units and by inserting a I^2C bus and a special interface for the flash disk. The I^2C bus also makes it easy to add more external units in the future without adding any unnecessary logic. The Flash interface allows access to a large flash area in one bus cycle.

The implementation of some blocks in the ASIC made it necessary to implement the RisARM. The RisARM was designed to test the implemented blocks and their communication with the ARM core. The most important part of the RisARM implementation was the design of the bus interface. The ARM/TDMI can use different bus protocols which mad the design of the RisARm more complex. Except for simulating the bus protocol of the ARM7TDMI core the RisARM also executes a part of the instruction set in the ARM7TDMI. The most important instructions to implement was LOAD and STORE instructions to communicate with the blocks that were being tested. To make the testing easier some extra instructions as Add, MOVE and SHIFT were also implemented.

Appendix A Acknowledgments

We would like to thank the following people:

Our mentor Professor Lars Philipson for being and big source of knowledge and inspiration. Without his guidance, this project would not have been accomplished.

Per Svensson and Mats Lindoff at Ericsson Mobile Communications for their moral support and for initiating the project.

The staff at the Department of Information Technology, especially Bertil Lindvall, Lennart Magnusson, Josef Wajnblom and Magnus Jönsson.

Appendix B Project Participants

Project Masterminds		
Lars Philipson	Pre-project manager, mentor	IT Department, Lund University
Per Svensson	Customer and cheer leader	Ericsson Mobile, Lund
Project Core Team		
Martin Trojer	Project Manager, coordinator of File Access.	IT Department, Lund University
Carl-Johan Hansson	Coordinator of Radio & Antenna and Audio & Sensors plus Documentation	IT Department, Lund University
Mikael Caleres	Coordinator of Mechanics & Design and Data communication	IT Department, Lund University
Tord Jakobsson	Coordinator of Electronic Hardware and Battery & Energy.	IT Department, Lund University
Mechanics		
Tom Waldner	Design & styling	Lindahl Design, Malmö
Magnus Örnhem	Master mechanic	Sigma Teknik, Växjö
File Access		
Ola Liljedahl	OSE guru and moral support	Enea OSE Systems, Täby
Fredrik Markström	OSE FTL support	Enea OSE Systems, Täby
Søren Lanng	FlashFx support	Lanng & Co, Denmark
Keith Garvin	FlashFx support	Datalight, Arlington, USA
Tony Questad	FlashFx support	Datalight, Arlington, USA
Dan Mattson	First FlashFx FIM-software	Svep, Lund
Chaowana Kusonkhum	CGI programming	Enea Data, Täby
Electronic Hardware		
Roland Larsson	Circuit design	Svep, Lund
Johannes Pihlström	CAD	Svep, Lund
Nils Arne Nilsson	Battery charging	Svep, Lund
Fredrik Karlberg	Solder guru	Svep, Lund
Mikael Schiller	BGA mounting	Vellinge Electronics, Vellinge
ARM and ARM-ASIC		
Helen Collins	ARM Development Board supply	ARM Europe, Cambridge, England
Jean De Oliveira	ARM Development Board support	ARM Europe, Cambridge, England
Ronan Synnott	ARM Development Board support	ARM Europe, Cambridge, England
Tom Cronk	ARM Development Board support	ARM Europe, Cambridge, England
Rune Knutsen	Ericsson ASIC expert	Ericsson, Grimstad, Norway
Torbjörn Grahm	Ericsson ASIC support	Ericsson Mobile, Lund
ARM Emulator		
Magnus Nemell	Lauterbach supply	Nohau Elektronik, Malmö

Magnus Åman	Lauterbach installation	Nohau Elektronik, Malmö
Karl Kolbinger	Lauterbach support	Lauterbach, Hofolding, Germany
Ralf Berberich	Lauterbach support	Lauterbach, Hofolding, Germany
Data Communication		
Jonas Harris	IrDA stack provider	Ericsson Mobile, Lund
Tobias Lindkvist	IrDA stack	Ericsson Mobile, Lund
Peter Marshall	IrDA stack	Enea Data, Täby
Ulf Höglund	IrDA stack	Enea Data, Täby
Bluetooth		
Bengt Falkenberg	Antenna magician	Telecom, Lund
Örjan Johansson	Bluetooth manager	Ericsson Mobile, Lund
Lars Nord	Bluetooth hardware support	Ericsson Mobile, Lund
Magnus Sandgren	Bluetooth installation	Ericsson Mobile, Lund
Hans Lindkvist	PCM hints	Ericsson Mobile, Lund
Components		
Mats Selin	Flash memory supply	Intel, Stockholm
Johan Uggmark	Flash memory middleman	Ericsson Mobile, Lund
Göran Schack	Charger contact supply	Ericsson Mobile, Lund
Plastic housing		
Kaj Larsen	Finish	Larsens modellverkstad
?	Rapid prototyping	GT Prototyper, Ystad
?	Plastic moulding	Plastic Design & Service
Local Support		
Bertil Lindvall	PC support	IT Department, Lund University
Björn Breidegard	Moral support	IT Department, Lund University
Josef Wajnblom	PADS support	IT Department, Lund University
Magnus Jönsson	PC support	IT Department, Lund University
Mats Brorsson	FrameMaker support	IT Department, Lund University
Others		
Karl Elmer	Moral support	
Anna Piper	Salary issues	Ericsson Mobile, Lund
Patrik Olsson	Moral support	Ericsson Mobile, Lund
Robert Hed	Moral support	Ericsson Mobile, Lund
Jerry Forsius	OSE teaching	Enea Data, Täby
Thomas Barnå´	OSE teaching	Enea Data, Täby

Appendix C References

1 www.intel.com

2 www,bluetooth.com

3 ARM7TDMI data Sheet, Advanced RISC Machines LTD, 1995

4 PC Intern, Michael Tischer, Abacus, 1994

5 DS1302 Data Sheet, Dallas Semiconductor, 1998

6 I2C Bus Specification, Philips, 1995

7 Designer’s Guide to VHDL, Peter J Ashendens, Morgan Kaufman Publishers, 1996

8 A VHDL Primer, J Bhasker, Prentice Hall, 1995

9 ARM Architecture Reference Manual, Advanced RISC Machines LTD, 1996