Development of an Advanced 8051 based CPU Sub-System for an LF Transmitter IC

MA669

Master's Thesis

at

Graz University of Technology

submitted by

René Allmeier

Institute for Electronics
Graz University of Technology
A-8010 Graz, Austria

December 2010

Advisor: Ass.-Prof. Dipl.-Ing. Dr.techn. Peter Söser
Advisor: Dipl.-Ing. Bernd Janger

sensordynamics
Leading in micro and wireless sensor products

TU Graz

Abstract

This thesis describes the development of an advanced CPU Sub-System for a Low Frequency Transmitter IC. The integration of a powerful microcontroller offers the possibility to perform complex operations without the use of additional microcontrollers in the final application. The herein presented CPU Sub-System is based on an 8-bit 8051 microprocessor core including memories, I/O ports, diagnostic units and a memory control unit. Before the design was integrated into the Low Frequency Transmitter IC it had been tested on a Xilinx Spartan 3 FPGA. This approach allows testing the developed hardware as well as starting software development for the final ASIC at an early stage in the design process. The CPU Sub-System is conceptualised to be very flexible and configurable for future use. The system has been verified during all development phases by the execution of testcases in simulation, on the FPGA and finally on the ASIC.

Keywords: System-on-a-Chip, ASIC, Low Frequency Transmitter IC, FPGA, CPU Sub-System, 8051 microcontroller core, memory mapping

Kurzfassung

Diese Arbeit beschäftigt sich mit der Entwicklung eines CPU-Subsystems für einen Niederfrequenz Sende-IC. Die Verwendung eines leistungsstarken Mikrocontrollers erlaubt es, komplexe Berechnungen in der Anwendung durchzuführen, für die man ansonsten auf zusätzliche Bausteine angewiesen wäre. Das hier vorgestellte CPU-Subsystem basiert auf einem 8-bit 8051 Mikroprozessor inklusive Speichermodulen, I/O Ports, Diagnoseblöcke und einer Speichermanagementeinheit. Das System wurde in VHDL beschrieben und vor der Integration in den ASIC auf einem Xilinx Spartan 3 FPGA getestet. Durch die Implementierung in einem FPGA kann die entwickelte Schaltung sehr früh im Entwicklungsprozess getestet werden. Zusätzlich ermöglicht diese Vorgehensweise einen frühen Start der Softwareentwicklung für den ASIC auf realer Hardware. Das Design des CPU-Subsystems ist flexibel und konfigurierbar um zukünftigen Anforderungen gerecht zu werden. Das System wurde während aller Entwicklungsphasen durch Ausführen von Testfällen in Simulation, am FPGA und letztendlich auch am ASIC verifiziert.

Stichworte: System-on-a-Chip, ASIC, Niederfrequenz Sende IC, CPU-Subsystem, FPGA, 8051 Mikrocontroller, Speichermanagement

Statutory Declaration

I declare that I have authored this thesis independently, that I have not used other than the declared sources or resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

Place, Date	Signature

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen oder Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommene Stellen als solche kenntlich gemacht habe.

Ort, Datum	Unterschrift

Contents

List of Figures

List of Tables

List of Listings

List of Acronyms

AC	Alternating Current
ACC	Accumulator
ADC	Analog/Digital Converter
AHB	Advanced High-performance Bus
ALE	Address Latch Enable
AMBA	Advanced Microcontroller Bus Architecture
AMS	Analog and Mixed Signal
APB	Advanced Peripheral Bus
ASIC	Application Specific Integrated Circuit
ASK	Amplitude Shift Keying
ATPG	Automatic Test Pattern Generation
AVDD	Analog Supply Voltage
BIST	Built-In Self-Test
BPSK	Binary Phase Shift Keying
CISC	Complex Instruction Set Computing
CPU	Central Processing Unit
CVS	Concurrent Version System
DAC	Digital/Analog Converter
DC	Direct Current
DFT	Design for Test
DIP Switch	Dual In-Line Package Switch
DMA	Direct Memory Access
DMSP	Digital Measurement Signal Processing Unit
DoD	US Department of Defense
DPH	Data Pointer High Byte
DPL	Data Pointer Low Byte
DPTR	Data Pointer
DRC	Design Rule Check
DSP	Digital Signal Processing
DTMF	Dual Tone Multiple Frequency
DUT	Device Under Test

DVDD	Digital Supply Voltage
EASE	Evatronix Application Debugging Support Environment
EDA	Electronic Design Automation
EDP	Evatronix Debug Pod
EMU	Emulator Version
EPROM	Erasable Programmable Read-Only Memory
ERC	Electrical Rule Check
FPGA	Field Programmable Gate Array
GCC	GNU Compiler Collection
GDSII	Graphic Database System II
GPIO	General Purpose Input/Output
GUI	Graphical User Interface
HDL	Hardware Description Language
HVDD	Doorhandle Supply Voltage
IC	Integrated Circuit
IDATA	Internal Data Memory
IDE	Integrated Development Environment
IE	Interrupt Enable
IEEE	Institute of Electrical and Electronics Engineers
ILayer	Interconnection Layer
IP	Intellectual Property
IRAM	Internal Random Access Memory
IrDA	Infrared Data Association Interface
ISA	Instruction Set Architecture
ISR	Interrupt Service Routine
JLCC	JTAG Like Control Chain
JTAG	Joint Test Action Group
LF	Low Frequency
LIFO	Last In - First Out
LSB	Least Significant Bit
LVS	Layout Versus Schematic
MDU	Multiplication Division Unit
M-FET	Modulation FET

MSB	Most Significant Bit
MUX	Multiplexer
N-FET	Negative Channel FET
NMI	Non Maskable Interrupt
NVIC	Nested Vectored Interrupt Controller
OCDS	On-Chip Debugging Support
opcodes	Operation Codes
OS	Operating System
PAR	Program Address Register
PC	Program Counter
PCB	Printed Circuit Board
PCI Bus	Peripheral Component Interconnect Bus
PCON	Power Control
P-FET	Positive Channel FET
PROM	Platform FLASH
PSEN	Program Store Enable
PSK	Phase Shift Keying
PSRAM	Program Storage Random Access Memory
PSROM	Program Storage Read Only Memory
PSW	Program Status Word
PWM	Pulse Width Modulation
PXI	PCI eXtensions for Instrumentation
QPSK	Quadrature Phase Shift Keying
RAM	Random Access Memory
RCO	System RC-Oscillator
RCX	Parasitics Extraction
RISC	Reduced Instruction Set Computing
ROM	Read-Only Memory
RTL	Register Transfer Level
SBUF	Serial Buffer
SCON	Serial Control
SD	SensorDynamics
SDCC	Small Device C Compiler
SDF	Standard Delay Format
SDK	Software Development Kit
SFR	Special Function Register
SoC	System-on-a-Chip

SP	Stack Pointer
SPI	Serial Peripheral Interface
SPICE	Simulation Program with Integrated Circuit Emphasis
SVN	Subversion
SWI	Single Wire Interface
TAP	Test Access Port
TCL	Tool Command Language
TCON	Timer Control
3WI	Three Wire Interface
TMOD	Timer Mode
UART	Universal Asynchronous Receiver Transmitter
USB	Universal Serial Bus
UVDD	Weak Digital Supply Voltage
VDD	Supply Voltage
VHDL	Very High Speed Integrated Circuit Hardware Description Language
WIC	Wakeup Interrupt Controller
WRCO	Watchdog RC-Oscillator
XDATA	External Data Memory
XTAL	External Crystal

Acknowledgement

Work on this thesis would not have been finished without encouragement and support from many people. First of all, I wish to thank my advisors: Peter Söser and Bernd Janger. Without their patience and support this thesis would never have been possible.

Furthermore I would like to thank all my colleagues at SensorDynamics for having a great time. Especially Steven Dennis whose guidance and experience has influenced my way of thinking as an engineer. I am also indebted to Oliver Gerler - he did always provide valuable feedback to any question and he also took the effort to proofread this thesis.

Last but not least I wish to thank my parents, my girlfriend and my friends for supporting and guiding me throughout my studies.

René Allmeier
Graz, December 2010

Chapter 1

Introduction

This thesis deals with the development of an 8051 based CPU Sub-System for a Low Frequency (LF) Transmitter Integrated Circuit (IC). SensorDynamics (SD) developed the LF series for applications requiring high transmission reliability, low standby current and fast reaction time. Nowadays it is common to integrate microcontrollers to such a System-on-a-Chip (SoC) to control transmissions and to provide an interface to the environment. The behaviour of such systems can easily be changed or improved by modifying the embedded software. So the system can be adapted very quickly to implement new features or to improve existing functions. These LF Transmitter ICs are used in applications like remote sensing, as active wide range identification tags or in keyless entry/keyless go systems. Besides the interfaces to control up to five external antennas for data transmission also seven sensor input pins are available. For each antenna a diagnostic input is also available to check if the antennas are working correctly. To perform operations like antenna diagnostics in time it is important to implement an efficient algorithm, and to execute it on a fast microcontroller.

In previous projects developed by SD an 8051 microcontroller core has been integrated into those systems. Several applications have shown that the processing power of the used core is inadequate to compete with future requirements. Besides the computing power, testability and controllability of the executed program during development have also become important topics. For example, the recently used core did not offer the possibility to stop the execution of code at a certain point and to resume it afterwards. Those debugging features are important criteria for customers nowadays because they decrease code development time and improve fault diagnostics capability. So it has been decided for future projects like the LF Transmitter IC a more powerful microcontroller core has to be integrated within the IC.

The main aim of this thesis is the integration of a new microcontroller core into SensorDynamics' Intellectual Property (IP) structure, to develop a CPU Sub-System based on that core, and to verify its functionality. Verification of the design is performed by simulation and integration of the whole digital design into a Field Programmable Gate Array (FPGA).

In the following the thesis' outline is presented. First basic principles in SoC design and the features of the 8051 microcontroller are discussed. Its drawbacks are also outlined and a comparison with a newer core is provided. Subsequently the system requirements to the core are presented and the selection process between different options is outlined. In Chapter 3 a general overview about the system is provided which is thereafter refined to delineate the interaction between the CPU Sub-System and the other parts of the system. Afterwards the integration of the new core into the CPU Sub-System is explained in detail. Modules like the memory mapper and the parity check block are presented as well as the architecture and the design of the complete CPU Sub-System. Chapter 4 deals with the implementation details of the presented design. Also system integration topics like interrupt and register assignments are covered within this chapter. At the end of this chapter two implementation flows are presented: the first one which targets an FPGA and the second one describes the creation of a physical block for use in an Application Specific Integrated Circuit (ASIC). Chapter 5 presents the verification methods used to check if the design meets its requirements. Also a short overview about the evaluation process of the final ASIC is provided. The last chapter deals with the achieved results and suggestions for future work.

Chapter 2

Basic Principles and State of the Art

2.1 System-on-a-Chip

The technical advance in the last years within semiconductor industries made it possible to integrate very complex systems onto a single chip. Those systems are often referred to as System-on-a-Chip (SoC). Such a system generally combines analogue, digital and mixed signal blocks. Depending on the application, radio frequency or high voltage blocks are also integrated onto the same substrate. Most of those systems include a microcontroller or a control unit to process measured data or provide an interface for setup and data transmission. As shown in Figure 2.1 the complexity of ICs is increasing year by year. On the other hand shortening time to market is as necessary as reducing size to be competitive. New design philosophies have been introduced to handle more complexity in a shorter time. One of those philosophies is the "Design for Reuse" (also "Design and Reuse") paradigm.

In the early 1990's reuse became a very popular topic. Since the end of the last decade usage of virtual components (IP modules) is daily business for designers (cf. [19]). IP modules are previously developed hardware blocks which have been tested and/or used in products before. This approach offers various advantages:

- Cost saving (if the module is used several times)
- Shorter verification time
- Possibility to obtain blocks from IP suppliers
- Improved reliability for new products
- Modules are checked and improved with every use
- Existing hardware could be used as reference

On the other hand there are only a few disadvantages:

- Increased effort to implement modules in a flexible way
- Administrative effort to manage the IP pool

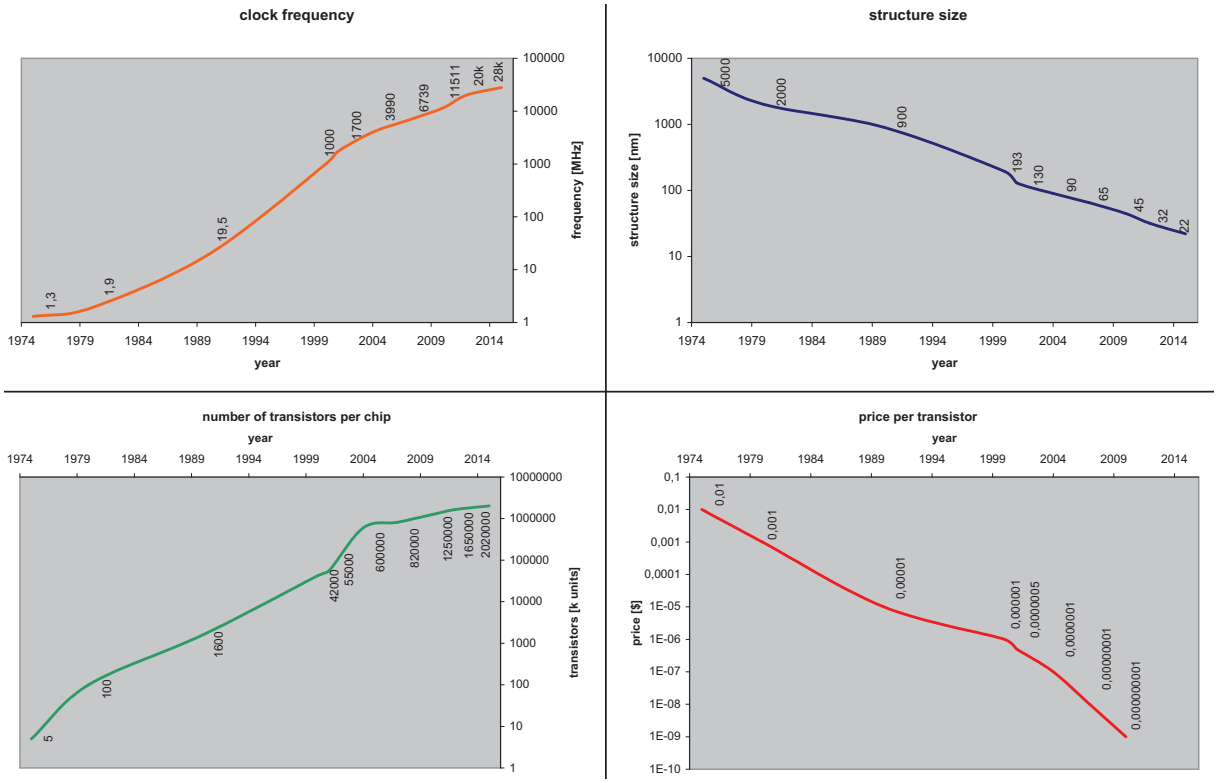

Figure 2.1: complexity of integrated circuits

The reuse of IP modules in different manufacturing technologies requires a hardware independent description of the circuit. To describe a digital circuit a Hardware Description Language (HDL) is used. The translation process between circuit description (algorithmic specification) and integrated circuit is done by software tools. Usually there are more levels of abstraction within a design. The translation from one abstraction level to another level is termed synthesis. To meet design constraints like timing, area and power consumption this process is computationally intensive and complex. Also the behaviour of the circuit needs to be verified and simulated. The combination of tools needed to create a design and to transform it into a fabricable electronic circuit is called design flow. This also includes verification at each abstraction level. In Figure 2.2 an example of a digital design flow is shown. The design flow is divided into two methodologies: frontend and backend methodology. Frontend summarizes all necessary actions to implement, create and simulate a Register Transfer Level (RTL) description of the design to implement. The result is a gate level netlist description of the circuit which is the entry point of backend. All processes needed to create a physical implementation from the gate level netlist are assigned to backend. After each important step in the design flow the circuit is simulated and the correct behaviour (e.g. timing) is checked. If the circuit does not pass this check a re-design has to be done. At the end of the backend technology checks are performed to verify the generated circuit meets also all technology restrictions.

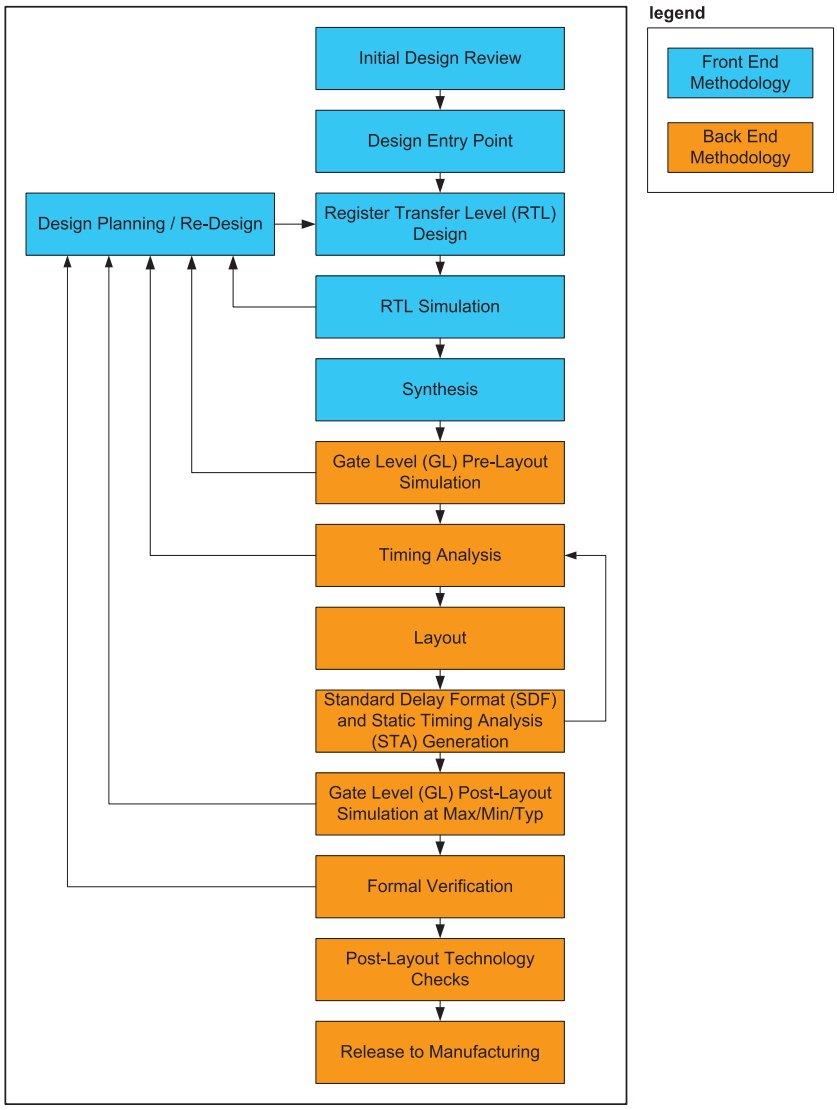

Figure 2.2: digital design flow

The following languages are very common in IC design:

- Very High Speed Integrated Circuit Hardware Description Language (VHDL), released in 1987 on the request of the US Department of Defense (DoD)
- Verilog, released in 1985, introduced by Gateway Design Automation

Those two languages are supported by many Electronic Design Automation (EDA) tools because the language definitions have been released early as Institute of Electrical and Electronics Engineers (IEEE) standard. VHDL was released in 1988 as *IEEE 1076-1988* and Verilog in 1996 as *IEEE 1364-1995*. Additional to the language specification there are also standards regulating waveform and vector exchange formats and synthesis. Aside from the widely-used languages mentioned above there are a few other HDLs which are not very popular because they are out-dated, limited in function or for special purposes.

To deal with new challenges like modeling the interrelation between hardware and software, formalising the specification of circuits and providing automatically generated testbenches new languages and methods have been developed. SystemC is one example which is very powerful. There are also compilers to translate circuit descriptions from SystemC to Verilog or VHDL like the Synopsys CoCentric® Design Compiler (cf. [21]).

HDLs dealing with analog and mixed signal descriptions are mostly named after their origin with Analog and Mixed Signal (AMS) as extension. Also Verilog and VHDL have been extended with AMS language elements. Those languages are referred to as Verilog-AMS and VHDL-AMS. In [16] a comparison of those two languages is presented by modeling the same system in each language. The VHDL-AMS extension originally was a strict superset (*IEEE 1076.1-1999*) of the above mentioned digital VHDL *IEEE 1076-1993* standard. The combination of those standards is colloquially termed as VHDL-AMS.

Nowadays tools on analogue and mixed signal synthesis are not very popular but the number of tools which take VHDL-AMS as input code to generate some type of analogue electronic implementation is increasing in recent years (cf. [2]). High-level analogue synthesis includes (according to [7]):

- Architecture generation
- Performance model generation
- Parameter optimization

It seems that synthesis of analogue and mixed signal circuits is at a similar stage as digital synthesis was around the year 1984. At the moment those languages are used to simplify and accelerate mixed signal simulations (cf. [15]). A comparison of simulation cycles between VHDL-AMS, Verilog-AMS and SystemC-AMS circuit descriptions has been performed in [12]. The results have been compared with a simulation done by Synopsys' Simulation Program with Integrated Circuit Emphasis (SPICE) simulator. It has been shown that simulations of models written in AMS languages are simulated faster than the SPICE models. However, for analogue synthesis the most important problem to address is identifying the type of language constructs that can be converted into electronic circuits.

In digital synthesis this process is less complex because the number of basic circuits (e.g. logic gates) and macro cells (e.g. memory blocks) is limited. Those basic circuits are delivered as digital libraries by ASIC manufacturers. This also ensures the basic blocks are optimised for the used manufacturing process. In [13] a Dual Tone Multiple Frequency (DTMF) decoder has been synthesised with a similar approach also in the analogue domain.

In general the modification of an existing circuit design itself is not necessary if a digital IP module is reused for another process. It has only to be re-synthesised with the correct digital library and constraints for the applicable production process. There are several ways to obtain IPs from external providers (cf. [17]):

Soft IP Modules are RTL or higher level descriptions. Their advantages are flexibility, portability and reusability. On the other hand timing and power characteristics are regulated by the setup of the synthesis tool and the target technology.

Hard IP Modules are already physically implemented blocks and highly optimised for an application in a specific process. The biggest disadvantage is the limited area of application followed by the lack of portability. But there are also benefits like predictable performance and area consumption. Usually Hard IP Modules are pre-qualified (tested on silicon).

Firm IP Modules are provided as parameterised circuit descriptions. So they can be optimised for specific design needs. Firm IP Modules are more flexible and portable than Hard IP Modules. They are also more predictable regarding timing and power characteristics than Soft IP Modules. Their drawback is the limited area of application.

The microcontroller integrated within this thesis has been obtained as s Soft IP Module. The range of microcontrollers used in SoC reaches from very cheap and simple designs like OKIs 4-bit microcontroller OLMS63K up to very powerful and complex 32-bit controllers like ARMs Cortex-M3. Since there is a wide range of microcontroller IPs it is very important to know the demands made to the microcontroller by the system. Even though it seems that powerful 32-bit controllers are getting more common in SoCs it has been decided to use an improved 8051 microcontroller core. This decision was also made with respect to reuse existing IP Modules designed for the previously used 8051 core. In Section 2.3 a comparison of the appropriate 8051 cores is shown. The next section provides a deeper look onto the 8051 architecture and its features.

2.2 The Microcontroller 8051

The 8051 was introduced in 1980 by Intel as one device in the MCS-51 family. At that time only two other members belonged to that family: The 8031 and the 8751. They are all based on an 8-bit Complex Instruction Set Computing (CISC) core with Harvard architecture. This means program storage and Random Access Memory (RAM) are separated. The only difference between the devices mentioned above is their type of program storage. The 8051 used a mask programmable internal Read-Only Memory (ROM), while the 8751 contained an internal Erasable Programmable Read-Only Memory (EPROM). For the 8031 the memory had to be connected externally. The instruction set of the MCU-51 family includes 255 Operation Codes (opcodes). In the original 8051 most instructions are executed within one or two machine cycles. Only divide and multiply operations take four machine cycles. A machine cycle (f_{mac}) lasts 12 clock cycles. The system clock (f_{osc}) is generated by the internal oscillator based on an external crystal.

Due to the licensing policies of Intel the 8051 architecture became very popular and more than 400 variants (cf. [20]) of the controller have been developed. The enhanced version of the 8051 was the 8052 microcontroller. Apart from a bigger Internal Random Access Memory (IRAM) (256 Bytes instead of 128 Bytes) and an additional timer/counter module the 8052 has the same features as the original 8051.

All versions of the MCS-51 family have the following common properties (cf. [22]):

- 8 bit Central Processing Unit (CPU)
- 128 Byte IRAM (also called Internal Data Memory (IDATA)); 8052: 256 Byte
- 16 bit address bus (up to 64 kB memory)
- Separated External Data Memory (XDATA) and Program Storage Read Only Memory (PSROM) (up to 64 kB each via separate control lines)
- Two timers/counters (8052: three timers/counters)
- 32 input/output (I/O) pins
- Integrated Universal Asynchronous Receiver Transmitter (UART) interface
- Two external interrupts with two priority levels
- Boolean processor for fast boolean operations
- Internal clock generation

Figure 2.3 shows a block diagram of the 8051 microcontroller. The 32 I/O pins are grouped into four ports (P0, P1, P2 and P3). P0 and P2 can be used as address and data lines for external memory. The address and data bus is shared between program and data storage. P0 serves as lower address bus as well as data bus. For this reason time multiplexing is used. This means the lower address byte output by P0 has to be stored into a latch, while the upper address byte is output directly from P2. The Address Latch Enable (ALE) pin is the control pin for the address latch. It indicates that the current value at P0 is part of the next address to access. If the address is applied to a program storage memory the Program Store Enable (PSEN) signal (active low) indicates the read has to be performed. If data memory is accessed the additional write and read strobes are provided by P3 (P3.6 write strobe, P3.7 read strobe).

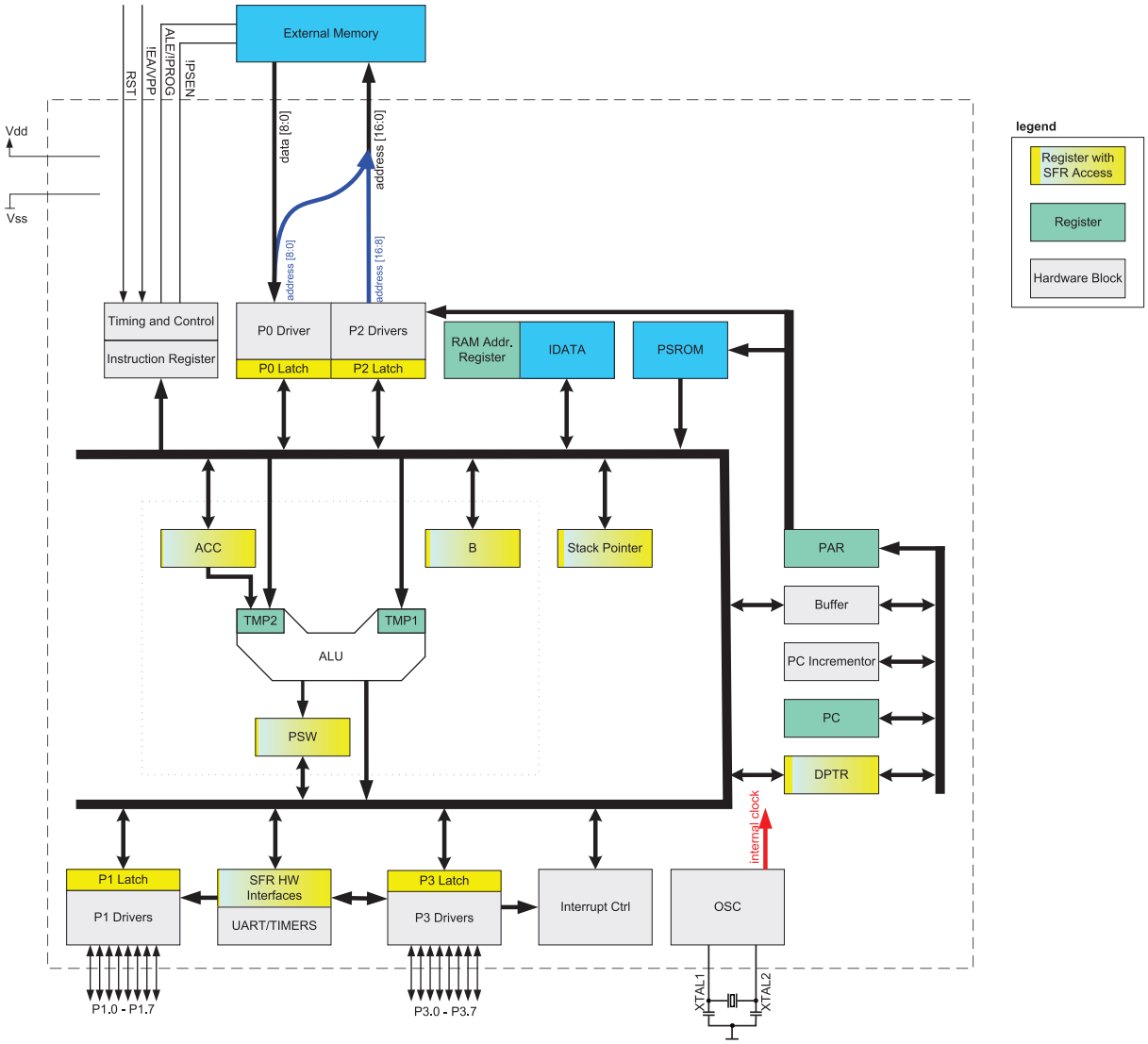

Figure 2.3: 8051 microcontroller block diagram (adapted from [8])

Some lines of P3 are also used for the serial interface (P3.0 receive/transmit data, P3.1 transmit data or clock), as external interrupt source (P3.2 for interrupt 0, P3.3 for interrupt 1) and as count input for Timer 0 (P3.4) and Timer 1 (P3.5). The switching between external data and program access is done automatically by evaluating the executed instruction in the timing and control unit. Since the number of pins in SoC is limited most of the integrated 8051 controllers do not provide the external memory interface to their users. This interface is used internally and the external memory is on chip. Accessing the external memory space is in general seven times slower than accessing the internal memory.

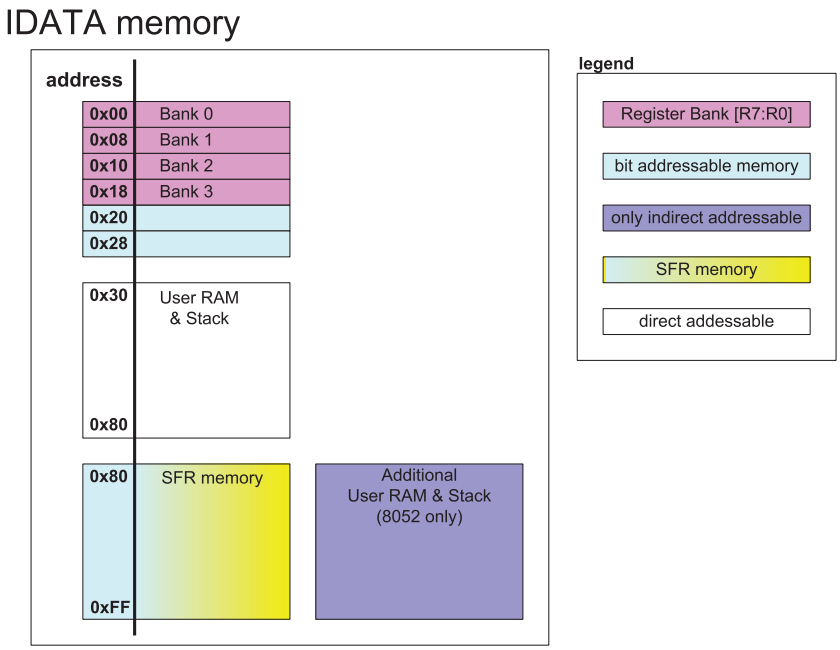

Figure 2.4: internal memory addressing scheme

Figure 2.4 shows the internal memory addressing scheme. The internal register banks are located between address *0x00* and *0x1F*. Each of the four register banks holds eight general purpose registers: R0 to R7. Those registers are used as intermediate storage when manipulating data or moving data between different memory locations. If an instruction stores a value in R0, the real address where R0 points to depends on the setting of the Program Status Word (PSW) Special Function Register (SFR). The bit addressable memory follows the registers with a size of 16 bytes (addresses: *0x20-0x2F*). Each bit has its own address (*0x00-0x7F*) where the Least Significant Bit (LSB) of each byte is the first in the right-most position. Those bits can be manipulated with the bit-oriented instructions of the 8051 (e.g. SETB, CLR).

User RAM and stack space are located between the bit addressable memory and the SFR memory space. The stack is a memory which provides a Last In - First Out (LIFO) functionality. Putting a variable onto the stack is called push, getting the last added element from the stack is called pop. The stack is used if functions are called to store the current working registers and the address to jump back after the function has been executed. The stack is controlled by the Stack Pointer (SP) SFR which holds the current end address of the stack. The SP is initialized on reset to *0x07* which points to the last register of register bank 0. So the stack memory space starts at address *0x08*. This means if one of the other register banks or the bit addressable memory space are used within the program the SP register has to be modified at startup. If the program is not written in assembler the compiler is aware of this behaviour. After the 80 bytes of user memory space the SFR memory space is placed. For the 8052 and the 8032 there are also 128 bytes of user memory space parallel to the SFR space. Accessing this user memory space is done via indirect addressing. This means R0 or R1 holds the address which points to the data.

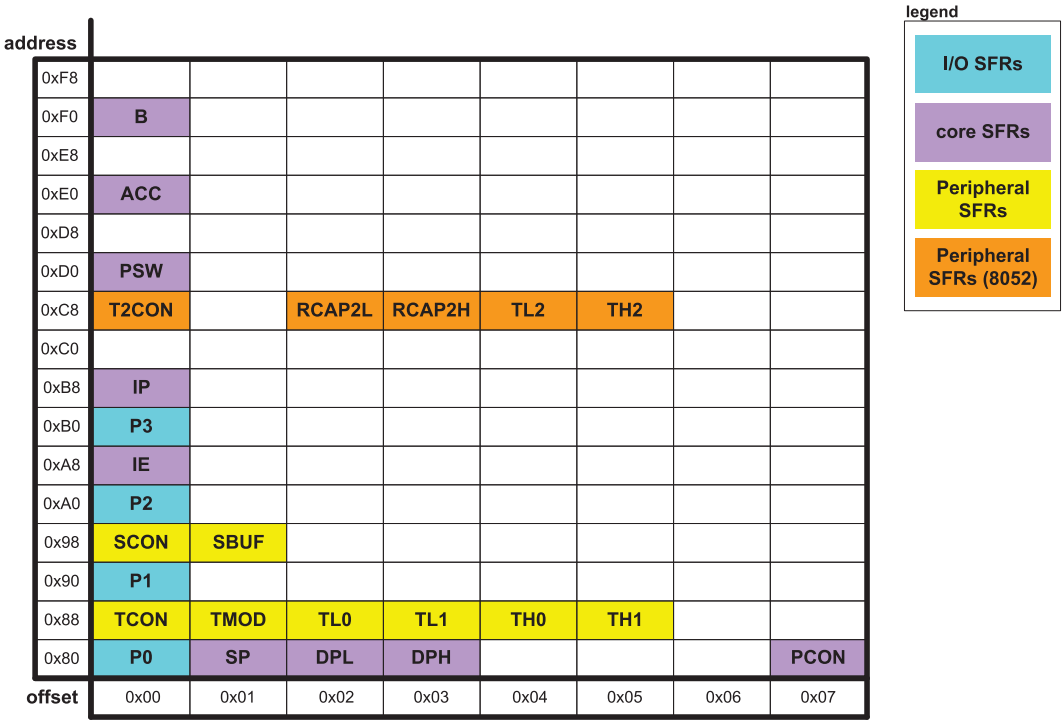

Figure 2.5: Special Function Register (SFR) address map

As its name suggests this memory space addresses special functions of the 8051 like setting the baud rate of the serial port or accessing the four I/O ports. The SFR is not part of the internal memory but is addressed like it would be part of it. In reality the registers addressed via the SFR memory space are distributed registers located in the hardware blocks which function they control. The registers are connected to the core via the SFR bus. This bus consists of the internal address bus, the internal data bus (output and input lines separated), the SFR write strobe and the SFR read strobe. So for 8051 IP cores it is possible to attach custom hardware via the SFR bus. Intels' 8051 used only 26 of the 128 available SFR registers. In Figure 2.5 the SFR map of the original 8051 is shown. The registers with addresses dividable by eight (e.g. 0x80, 0x88,...) are bit addressable. If the corresponding hardware is implemented (e.g. Timer 0) in general all vendors use the standard register (e.g. T0CON) to provide compatibility with existing software tools. Custom hardware is controlled via SFRs which are not used in the original core. Below a short description of each standard SFR register is presented:

Port 0 (P0): As mentioned above P0 is used to control the lower address byte and data bus for the external memory. If no external memory is used P0 could also be used as general input/output port. Port 0 is the only true bidirectional port. This means external pull-up resistors have to be applied if the port is used.

SP Register: Pointer to the last element of the actual stack memory.

Data Pointer (DPTR) Register: 16 bit width data pointer, which is accessed via Data Pointer Low Byte (DPL) and Data Pointer High Byte (DPH) SFR registers. The data pointer is used in operations accessing external memories.

Power Control (PCON) Register: Controls the power saving modes of the 8051. There are two modes: The idle and the power down mode. The Most Significant Bit (MSB) of this register doubles the baud rate of the serial interface if it is set.

Timer Control (TCON): Control SFR of the internal timers: Timer 0 and Timer 1. It also controls the handling of the two external interrupt sources.

Timer Mode (TMOD): This register serves to configure the two internal timers: They can be configured separately either as timer or as counter in several modes. In counter mode the falling edge of the dedicated timer input (on P3) increases the counter value. In timer mode the timer modules are counting with the internal clock. By setting the prescaler within the according bits of this register the clock frequency of the timer could be configured. For each timer two bits are configurable within this register setting the timers mode.

- Mode 0: Configures the timer as 13 bit timer.
- Mode 1: Configures the timer as 16 bit timer.
- Mode 2: The timer is used as 8 bit timer with auto load facility.
- Mode 3: The timer is configured as two independent 8 bit timers.

Timer 0 (TMR0): 16 bit register, divided into Timer 0 Low Byte (TL0) and Timer 0 High Byte (TH0) register. This register holds the actual counting value of Timer 0 depending on the mode. In mode 0 TL0 counts from 0 to 31. Then it increments TH0. In mode 1 the 16 bit counting value is hold in TH0 and TL0. In mode 2 the TL0 register holds the 8 bit counting value, while the TH0 holds the value to reload the timer. In mode 3 the TL0 register holds the 8 bit counting value of one 8 bit timer and TH0 holds the value of the other 8 bit timer.

Timer 1 (TMR1): 16 bit register, divided into Timer 1 Low Byte (TL1) and Timer 1 High Byte (TH1) register. This register provides the same functionality for Timer 1 as TMR0 for Timer 0.

Port 1 (P1): This port is used as general I/O port. Writing a one to a bit of this register sets the corresponding output to high, setting the same bit to zero drives it low. If the pin should be read it must be set to one. The port has internal pull-up resistors and does not provide any other functionality.

Serial Control (SCON): This register controls the behaviour of the serial interface. The serial interface usually uses one of the timers described above to establish the serial ports baud rate. The SCON register also holds the receive and transmit flags. Those are needed because there is only one interrupt for the serial interface connected to the CPU. If such an interrupt occurs the application has to evaluate the SCON register to check if a receive or transmit process has been finished. This register also holds two mode selection bits for the serial port. So four modes could be selected:

- Mode 0: Synchronous mode, 8 bit data, fixed baud rate ($\frac{f_{osc}}{12}$)
- Mode 1: Asynchronous mode, 8 bit data, variable baud rate

- Mode 2: Asynchronous mode, 9 bit data, fixed baud rate ($\frac{f_{osc}}{32}$ or $\frac{f_{osc}}{64}$)
- Mode 3: Asynchronous mode, 9 bit data, variable baud rate

Serial Buffer (SBUF): This register is the data register of the serial interface. Data received could be read by this register. If data is sent via the serial interface it has to be written to this register.

Port 2 (P2): Could be used like P1 as general I/O port. It is also used as higher address byte for external memory access. If external memory is used the port can not serve as general port any more. The port has internal pull-up resistors.

Interrupt Enable (IE): This SFR is used to enable or disable interrupts. The MSB of this register is a global interrupt enable bit. The lower bits are used as bit mask for the according interrupts. Writing a one to the according bit enables the interrupt, setting it to zero disables it. The register is organised as followed:

IE.0 External interrupt 0

IE.1 Timer 0 interrupt

IE.2 External interrupt 1

IE.3 Timer 1 interrupt

IE.4 Serial interface interrupt

IE.5 Timer 2 interrupts (8052 only)

Port 3 (P3): This port could be used like P1 as general I/O port. As mentioned above some of the port pins have a second function (e.g. external interrupt source). The port has internal pull-up resistors.

Interrupt Priority (IP) Register: The register defines the priority of the according interrupts. It is organised like the IE register. Setting the according bit to one defines the interrupt as higher priority interrupt, clearing the bit defines it as lower prior.

Timer 2 Control (T2CON): Control SFR of Timer 2 (8052 only). Timer 2 can be configured via this register for several modes:

- Auto reload mode: Timer 2 is reloaded with the values in the Timer 2 Capture Reload (RCAP2) register. Timer 2 is the only timer providing full 16-bit counting and pre-loading.
- Capture mode: If a falling edge occurs on the second pin of P1 (P1.1) the counting value is moved to the RCAP2 register.
- Baud rate generator mode: Timer 2 is used as baud rate generator for the serial interface.

Timer 2 (TMR2): 16 bit register, divided into Timer 2 Low Byte (TL2) and Timer 2 High Byte (TH2) register. This register holds the actual counting value of Timer 2.

PSW: This SFR contains a number of important bits: parity flag (even parity), overflow flag, register bank selection (2 bits), zero flag, auxiliary carry flag (subtraction, addition) and the carry flag (used by arithmetic and logical instructions).

Accumulator (ACC): The accumulator holds the source variable and the result is written back to this register. Some instructions can only be executed on this register (e.g. rotate).

B Register: This SFR is used like the ACC register in multiplication and division operations.

As shown in Figure 2.3 there are also some hardware registers which are not directly accessible by the user:

Program Address Register (PAR): This register addresses the ROM. If an address is accessed which is higher than the size of the internal ROM the code bytes are automatically fetched from external program memory. By setting the !EA (External Access) pin to high the microprocessor starts executing the program stored in the internal program memory. If it is tied low code from the external program memory is executed.

RAM Address Register (RAMADDR): This register is used to access the internal RAM.

Program Counter (PC): The 16-bit register holds the address of the next instruction to be executed. The PC always starts at address *0x0000*. Since some instructions are two or three bytes in length, the PC is not always incremented by one. In these cases the register is incremented by two or three. Accessing this register directly is not possible. It is automatically set to a new address if a jump instruction is executed.

The last item to be discussed here is the behaviour of the microcontroller if it is reset. To reset the device an active high pulse (duration: at least two machine cycles) has to be applied to the reset input of the microcontroller. This initialises following registers:

- PC is set to *0x0000*.
- SFR memory space is set to default values: P0 to P3: *0xFF*, SP: *0x07*, others to *0x00*.
- The user memory space is not initialised. This has to be done via software.

After releasing the reset line the 8051 starts executing the program at address *0x0000*. The next section deals with the extended versions of 8051 architectures and presents useful add-ons.

2.2.1 Current 8051 Implementations

Years of development lead the 8051 to higher clock frequencies, additional on-chip peripheral modules and more sophisticated instruction execution. At the moment fast versions of the 8051 execute one machine cycle in one clock cycle. This improvement on its own makes new versions of the controller several times faster than the original microprocessor. Cores like the R8051XC2 implement features that speed up the core by a factor of 12 in comparison to the original 8051 (depending on the executed instructions). In some derivates also the EPROM is replaced by a FLASH memory.

A rough overview of additional available hardware modules is given below:

- SPI-Port (Serial Peripheral Interface)
- CAN-BUS interface (Controller Area Network)
- Universal Serial Bus (USB)
- Inter Integrated Circuit Bus (I^2C)
- Infrared Data Association Interface (IrDA)
- Analog/Digital Converter (ADC) and Digital/Analog Converter (DAC)
- Enhanced interrupt module: adds more external interrupts and/or more interrupt priority levels
- Internal baud rate generator for the serial interface
- Watchdog timer: A timer which resets the chip if the timer is not cleared within a certain time
- Additional data pointers
- Extended address space (up to 16 MB of memory addressable)
- Arithmetic co-processors

There are also variants of the 8051 designed for low power applications. Also in modern wireless communication ICs many 8051 cores are used to provide a flexible and fast interface for the user. Also SensorDynamics has integrated an 8051 microprocessor to control the analog part of the former version of the LF Transmitter IC. Since the used core was the bottleneck in some applications it has been decided that its successor product should integrate a faster core. In former projects an 8051 IP core developed in 1998 at the Johannes Kepler University was used. Within this document it will be referred to as DDC8051. The next section analyses the requirements to be met by the new microcontroller core. Afterwards a comparison between suitable microcontrollers is done. The DDC8051 is used to provide a common base for the comparision with faster 8051 cores in in Section 2.3.

2.3 Comparison between different Microprocessor Cores

This section provides an overview about requirements to be met by the new microprocessor core. Later on a comparison between available cores for the LF Transmitter IC is made. Finally a detailed list comparing different features of the two selected possibilities and the currently used core is presented.

2.3.1 Requirements

Based on the output of projects using the DDC8051 a list of requirements has been generated:

- Possibility to reuse existing SD IPs
- Performance (perform calculations faster than the DDC8051)
- Power consumption
- Code compatibility to the DDC8051 (at C Level)
- Synchronous design
- Area size of the implementation
- Tool support (compilers, simulators, models)
- Debugging interfaces (hardware break points, debugging interfaces)
- Implementation and evaluation effort
- Overall costs (design cost, training, unit costs)
- Support
- Design already used in automotive products

A list of cores fulfilling most of the requirements above has been created.
Table 2.1 shows the supplier, the Instruction Set Architecture (ISA) and the features of the respective core. The architecture of the presented cores has not been considered - so not only 8051 architectures are compared.

Supplier	Core name	ISA	Architecture	Features
Evatronix LTD	R8051XC2	8 bit	8051	Single cycle execution, additional DPTR, co-processor (MDU), DPTR hardware arithmetics, 256 Byte IRAM, PSROM write capability, On-Chip Debugging Support (OCDS), up to 18 external interrupts with four priority levels
Oregano	MC8051	8 bit	8051	Single cycle execution, co-processor, up to 255 timers and serial interfaces
Digital Core Design	DP8051	8 bit	8051	Four clocks per machine cycle, co-processor, Serial Peripheral Interface (SPI), serial interfaces, power management module, one hardware breakpoint, six external interrupts
Handshake Solutions	HT80C51	8 bit	8051	Single cycle execution, up to eight hardware breakpoints, asynchronous low power design
Cambridge Consultants	XAP4a	16 bit	XAP	AMBA bus, JTAG debug interface, GNU Compiler Collection (GCC) based tool chain
ARM	Cortex-M0	3 2/16 bit	ARM	32 interrupts, context saving done in hardware, JTAG debugging interface, fast, low power
ARC	601	32/16 bit	ARC	fast, low power, small, no additional features
Cortus	APS3	32/16 bit	Cortus	JTAG debugging interface, fast memory access, bus bridges available, SystemC cycle accurate model
Beyond Semiconductor	BA22	32 bit	BA	Software controlled clock frequency, 32 external interrupt sources, power management unit
MIP32	m4k	32 bit	MIP	JTAG debug interface, software controlled clock divider, memory management unit, low power

Table 2.1: supplier and microcontroller key features

Although the 16/32 bit instruction set architectures are not code compatible with the 8051 they have been considered in table 2.1 to provide an overview of commonly used microcontrollers in SoC. Section 2.3.2 shows the advantages of 16/32 bit architectures in comparison to the 8051 architecture using the example of ARMs Cortex-M0.

It had been decided that the new 8051 core has to execute one machine cycle within one clock cycle to achieve a noticeable speed improvement. So the DP8051 from Digital Core Design was not an option any more. Since a synchronous design is a major criteria for the core also the HT80C51 from Handshake Solutions is not an alternative. Now only two cores are left: The R8051XC2 developed by Evatronix and Oreganos MC8051. A detailed comparison of those two cores with the DDC8051 as reference is presented in Section 2.3.3.

2.3.2 Comparing an 8051 Architecture with an ARM Cortex-M0

In general ARM is a 32-bit Reduced Instruction Set Computing (RISC) ISA developed by ARM Holdings. The company sells IP modules but does not produce microcontrollers with their IPs. ARM also provides a complete tool chain for their processors. The ARM Cortex family is based on the very popular ARM7 processor with improved power efficiency and speed. The ARM Cortex family is divided into three groups: Cortex-A for applications, Cortex-R for real-time requirements and the Cortex-M targets microcontroller. The lowest power and smallest ARM processor is the ARM Cortex-M0. It consumes only 85 μW/MHz and its area is only 12000 gates (numbers based on an ultra low power 0.18 μm process). In Figure 2.6 a basic block diagram of the ARM Cortex-M0 is shown.

The instruction set of the Cortex core contains only 56 instructions: the instruction set (named thumb) extended by the so called thumb2 instruction set. All ARM cores starting from ARM7T support the 16 bit thumb instruction set. All versions of the Cortex family additionally support the thumb2 instruction set extension. The smaller code bit width leads to a smaller program size which reduces the amount of necessary program memory. All registers and data paths within the Cortex-M0 are 32 bit. The processor is able to address 4 GB of memory. All instructions have a dedicated execution time:

- Data processing (e.g. sub) take one cycle
- Data transfers (e.g. load) take two cycles
- Branches take three cycles (if taken)

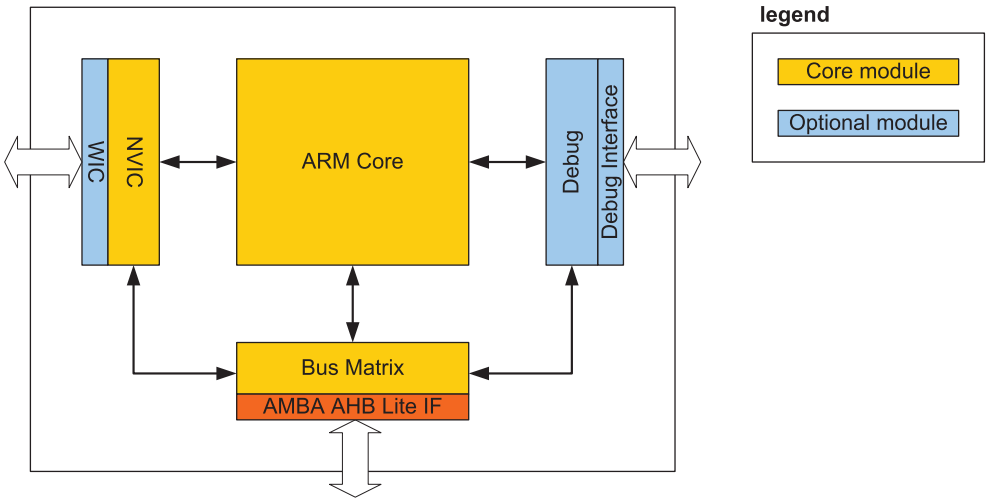

Figure 2.6: ARM Cortex-M0 block diagram (based on [1])

The most interesting features of the ARM Cortex-M0 core are listed below (cf. [1]):

- Integrated sleep modes
- Hardware multiplier (optional: single cycle multiplier)
- High-performance interrupt handling for time-critical applications via the Nested Vectored Interrupt Controller (NVIC) module
- Debug interface (serial or Joint Test Action Group (JTAG) interface)
- Selectable features: endianness, number of interrupts, number of breakpoints, halting debug support

Interrupts are processed by the ARM core as exceptions. ARM uses the term exception as an umbrella term for maskable interrupt sources, resets, Non Maskable Interrupt (NMI) sources, hard faults and the SysTick event which is caused by an additional timer. Also some exceptions for Operating System (OS) support are addressed by this term. If an exception occurs the processor automatically saves its state. The state is restored on exception exit with no instruction overhead. This means the user does not have to care about context saving for interrupts and the exception is handled with low (and fixed) latency of 16 cycles.

The NVIC module offers the possibility to apply up to 32 external interrupts with programmable priority levels. Also the level and pulse detection is programmable via this block. Also one NMI could be used by external peripherals. The NVIC registers are always little-endian independent of the core's selected endianness. The Wakeup Interrupt Controller (WIC) module could be obtained optionally. It is only connected to the NVIC and provides the possibility to enter a deep sleep mode so the power management unit within the core can power down most of the processors hardware. If the WIC receives an interrupt it causes the system to wake up and processes the interrupt after the context is restored. This implies increasing the interrupt latency time.

The bus matrix serves to coordinate the memory access of the processor core and the optional debug module. Memory access of the processor is always more highly prioritised than debugger access to ensure that debugging is non-intrusive. To interface peripherals on chip an Advanced Microcontroller Bus Architecture (AMBA) interface is available at the core. It implements the Advanced High-performance Bus (AHB) specification of this bus architecture. The debugger can either be interfaced via serial wire interface or JTAG interface. It connects to the processor slave port to provide full system-level debug access. Besides the observation of core registers execution breakpoints can also be set as well as data watchpoints.

Table 2.2 provides a comparison between ARMs Cortex-M0 architecture and the 8051 architecture. To represent the 8051 architecture the R8051XC2 of Evatronix LTD is used. The data presented in table 2.2 represent the cores without debug capability. The Evatronix core is assumed to provide 23 bit addresses. The ARM Cortex-M0 comes in its basic configuration without any additional hardware (e.g: power management unit).

Feature	ARM Cortex M0	R8051XC2
Architecture	Von Neumann	Harvard
ISA	RISC	CISC
Instruction width	16 bit	8 bit
Data width	32 bit	8 bit
Address width	32 bit	23 bit
Area	12k gates	8.5k gates
Power consumption	85 μW/MHz	80 μW/MHz
Performance (Dhrystone benchmark)	0.9 DMIPs/MHz	0.088 DMIPs/MHz
Memory size	4 GB	32 MB
Memory map	linear, divided into regions	distributed (IDATA, SFR, XDATA, PSROM ,...)
Peripherals	memory mapped	SFR mapped
External interrupts	32	13
Interrupt priorities	4	4
Interrupt context saving	automatically by hardware (16 cycles)	by software (48 cycles)
Sleep modes	sleep mode	Idle and stop mode

Table 2.2: comparison between Cortex-M0 and 8051

From table 2.2 it can be reasoned that the ARM Cortex-M0 provides a ten times higher calculation performance than the R8051XC2 from Evatronix. But in terms of area the 8051 performs better than the ARM. So it depends on the specific requirements of a project to determine which core fits better. For the LF Transmitter IC it has been decided to use an 8051 core to provide reusability of IP modules which are connected to the core via the SFR bus.

2.3.3 Comparison between DDC8051, R8051XC2 and MC8051

A detailed comparison between the two cores left with the DDC8051 as reference is given. Table 2.4 provides a comparison between single features for each core. The area comparison is based on the unit gates which describes the area a NAND gate occupies. This allows the area comparison to be done without knowledge of the fabrication process. The power consumption also depends on the used firmware/software as well as on the optimisations done by the synthesis tool. Also the library used for synthesis has a big impact to the performance of the created circuit. So the power consumption was estimated by the power report generated by the synthesis tool. The current drain of the DDC8051 was confirmed by basic measurements of an existing product.

Feature	DDC8051	MC8051	R8051XC2
ISA	8051 (8 bit)	8051 (8 bit)	8051 (8 bit)
External buses	IDATA, XDATA, SFR	IDATA, XDATA	IDATA, XDATA, SFR with wait states, synchronous and combinational external memory interface available (with wait states)
Area	5k gates	8.5k gates	7.5k gates (11.5k with OCDS)
Current consumption	19 μW/MHz	100 μW/MHz	80 μW/MHz
Performance (Dhrystone benchmark)	0.02 DMIPS/MHz	0.08 DMIPS/MHz	0.088 DMIPs/MHz
Number of clock cycles to execute one machine cycle	6	1	1
External Interrupts	6	2	13
Interrupt priorities	2	2	4
Maximum external memory size	2x 16 kB	2x 16 kB	2x 16 kB, (optionally 2x 16 MB)
Debug features	Keil In-System Debugger 51 (ISD51) and one breakpoint register (optionally)	Keil ISD51	JTAG OCDS, Keil ISD51
IDATA size	256 Byte	128 Byte	256 Byte
Additional options		Hardware divider/-multiplier, up to 255 timers and UARTs	Multiplication Division Unit (MDU), DPTR arithmetics, Direct Memory Access (DMA) unit, timers

Table 2.4: comparison between DDC8051, R8051XC2 and MC8051

The result of this comparison was the decision that the Evatronix R8051XC2 core will be used for future products. The modifications needed to fit the requirements listed in Section 2.3.1 for the Oregano core are too huge. Only 128 Bytes of IDATA memory are available. The SFR interface must be added to the cores top interface. Also a necessary hardware breakpoint for debugging would have to be designed in. The IP would have had to be extended to handle more external interrupts. In terms of energy saving power cosumption could also be manually reduced by 40% (as shown in [9]). In summary, adaption of the MC8051 to fit SensorDynamics' requirements would have caused too much effort.

Chapter 3

Design and Architecture

3.1 Design Methodologies

According to Webster's dictionary [5] design means "To create or execute in an artistic or highly skilled manner" as well as the "invention and disposition of the forms, parts, or details of something according to a plan.". In engineering this is the creation or invention of a system possessing a desired set of properties. In general those properties are defined by the system specification. To handle complex IC designs using a top-down approach is common. This means the system is defined on a high abstraction level (functionality of the IC) and is refined by splitting it into smaller blocks (e.g. powersupply block of the IC). Each sub-block itself is again split up into smaller fragments until the lowest abstraction level (transistor level) is reached. Based on this approach principle design methodologies have been introduced. In the following an overview of these methodologies is given as they are necessary to understand the design decisions afterwards. The list is based on the methodologies presented in [4]:

Design Environment Methodology: A design team has to deal with many files of different types in all project phases. There are source files like HDL files as well as simulation testbenches, constraint files, stimuli files and so on. The design environment ensures that multiple designers can share and modify design files without loosing integrity or consistency of the data. The following objectives have to be addressed by the design environment:

- Design data organisation (e.g. hierarchy of the design is modeled via a directory hierarchy)
- Source control (tools like Concurrent Version System (CVS) or Subversion (SVN))
- Automated processes (e.g. execution of tasks like simulation via Shell or Perl scripts)
- Revision control (could also be done via CVS or SVN)
- Project/issue tracking (e.g. Mantis, Bugzilla)

Design Capture Methodology: This methodology describes the partitioning of the system within a top-down design approach. Designers create system models and refine them after the current abstraction level has been validated. Therefore two major steps are necessary:

- System decomposition: Determination of the high-level system resources (e.g. buses, memories).
- Functional unit partitioning: Assignment of system functions to modules.

Within this approach models with different abstraction levels are implemented. Those models are organized hierarchically (starting with the highest abstraction level):

- System models: Defining the design concept or system intent.
- Implementation models: Models written in HDLs. These models can be implemented as behavioural model, functional model, structural model or gate level model.
- Gate level models: Created by the designer using a schematic editor.

This methodology ensures the system performs according to the operations defined on system level. Models at this abstraction layer are block diagrams, system specifications or analytic models (e.g. MATLAB® models).

Design for Test Methodology: Design for Test (DFT) describes methods making testing the design itself easier and faster. A good DFT strategy addresses testing at all levels of abstraction and provides a good test coverage of the design. Essential concepts of DFT are controllability and observability: Controllability is the ability to set or clear each single node within the design while observability constitutes the ability to observe every node. In the following the common test techniques for digital circuits are presented:

- Internal scan: This technique describes the queuing of all internal flip-flops or latches onto a scan chain. With an additional multiplexer for each element in the chain an alternative input can be selected. The chain works like a shift register if the test mode signal is asserted. This method allows complete access to all register elements and reduces the complexity of testing a complex sequential circuit by testing a large combinational circuit. The creation of test patterns by Automatic Test Pattern Generation (ATPG) tools is much easier for combinational circuits.
- Boundary scan: Boundary scan is the enhancement of the internal scan path method to support board level interconnect testing. The standard IEEE 1149.1 (commonly named JTAG) defines the mandatory architecture to control boundary scan chains. It defines the interface (Test Access Port (TAP)) and the boundary scan controller (TAP Controller).
- Test access collar: This technique describes the disconnection of a sub-block in the IC from the system to test the block on its own. Therefore the block interface is connected to certain I/O pins of the IC where the test patterns are applied and the outputs are evaluated.

- Built-In Self-Test (BIST): BIST describes a method to test memories. An internal scan chain to test memories would cause too much overhead. The result of this test would only be the information if it passes or fails. To provide BIST functionality a BIST controller has to be added to the design. This controller is connected to the data input, data output and address lines of the memory under test. After starting a BIST the memory is disconnected from the system and a special data pattern is written to it. Afterwards the memory is read and compared with the expected data. If the data is read correctly the test is passed and the memory is connected to the system again.
  BIST is also possible for ROM memories: At a certain point of the memory (e.g. the last two bytes in the memory) a checksum of the ROMs content is stored. After starting the test the BIST controller calculates the checksum of the memories content. Afterwards the result is compared with the stored checksum.
- IDDQ test: This method is based on measuring the supply current (I_{dd}) to find manufacturing faults. I_{ddq} describes the supply current in the quiescent state. This means the circuit is not switching and the inputs are held at static levels. In this state I_{ddq} is usually within nanoampere (nA) range. At the start of the test a certain vector is applied to put the Device Under Test (DUT) into a known state. After the switching currents have been settled I_{ddq} is measured. If the circuit is damaged the measured I_{ddq} is at least thousand times higher than the current measured for a unbroken circuit.
- Delay Fault Testing: This method is used to detect timing violations along a path. An input transition with a known output transition is applied and the time between applying the signal to receiving the expected answer is measured. Exceeding the specified time indicates manufacturing defects (e.g. increased metal resistance).

Design Verification Methodology: Design Verification describes the process verifying the design at each abstraction level versus requirements. Verification of a design is a very complex and time consuming process. A strategy has to be chosen which discovers all possible design errors. There are two major points to cover:

Validation of the system concept: This task has to be performed before implementation starts. It includes a detailed system analysis to verify that the design meets all requirements. Also the partitioning of the system into hardware and software is done within this design step as well as the rating and selection of algorithms used by the system.

Verification of the system implementation: From a general point of view the more accurate a simulation is the more time consuming it is (e.g. gate level simulation). Most of the time should be spent simulating at a high abstraction level (RTL). Changes there have the biggest impact on the systems performance. Very important points within this step are the development of a complete verification test plan, the usage of structured testbenches and the design of automated regression tests.

The impulse to automate handling of the points above lead to the development of so called formal verification tools. They are used to verify the equivalence of

circuit descriptions mathematically. Since these tools are not able to determine if the design meets the system requirements simulation is still a necessary task.
Also the usage of emulation and hardware prototypes is a way to test if the system or parts of it are working correctly before the system itself is produced. FPGAs make it nowadays very easy to do a kind of physical verification before the chip has been taped out (brought into production).

Most of the principles explained above will appear within the next chapters which are dealing with the system architecture, the implementation and verification of the developed blocks.

3.2 System Architecture

The system is an LF Base Station Antenna Driver IC mainly for the use in automotive key-less entry and key-less go applications. It integrates five interfaces to doorhandle sensors in addition to drivers and diagnostic interfaces for up to five LF antennas. Doorhandle sensors are integrated into the doorhandles of a car to check if a person wants to open a door. Those sensors are touch-sensitive capacitive sensors and communicate with the IC via a serial protocol. Multiple antennas are necessary to be able to determine the location of the receiver within the ignition key. In Figure 3.1 a basic block diagram of the microchip is shown.

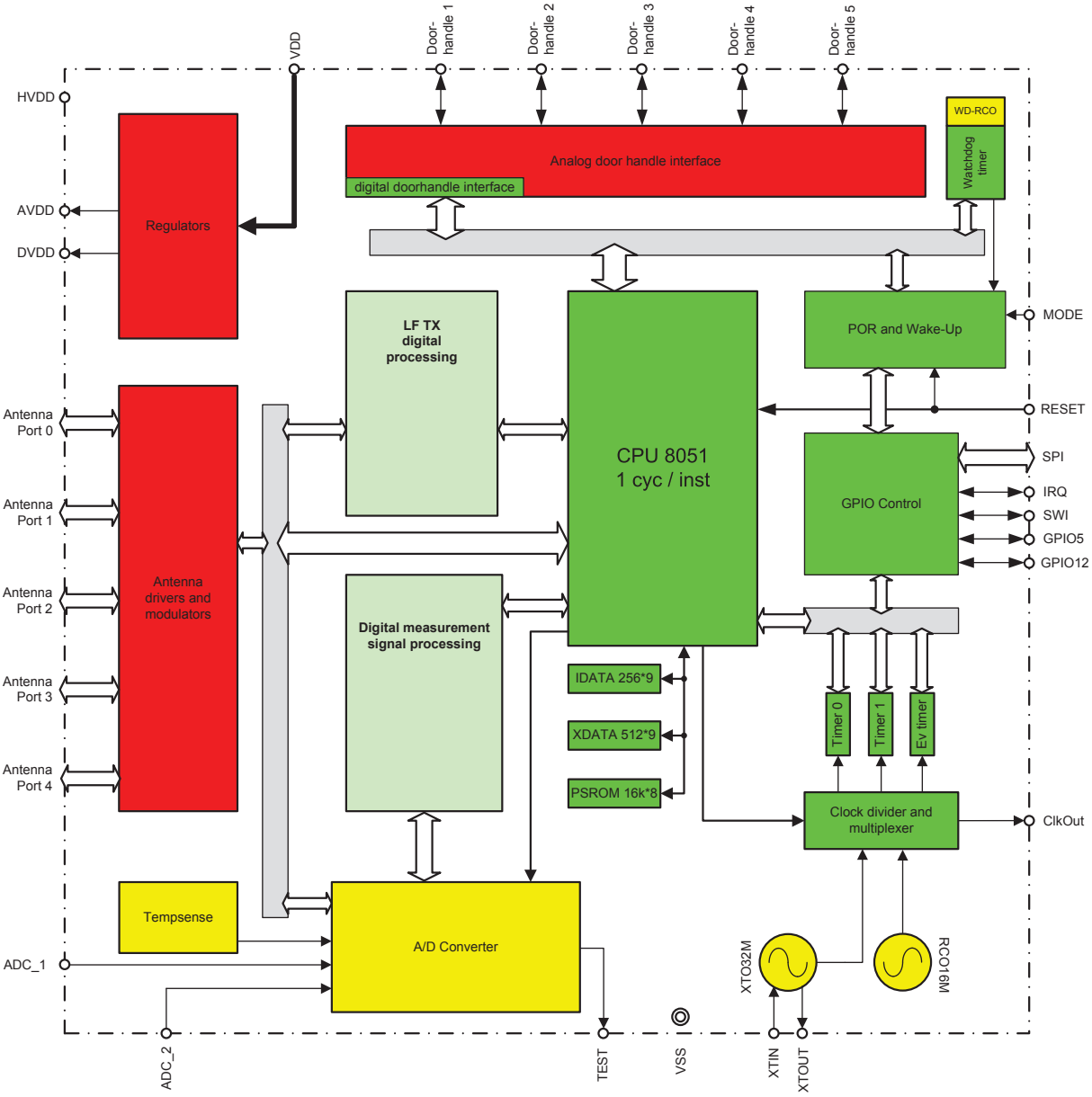

Figure 3.1: block diagram of the LF Transmitter IC adapted from [18]

The colors mark different power domains in the IC. The microchip is designed to operate on a Supply Voltage (VDD) up to 28 V. Blocks powered by this power domain

are colored red, as well as the blocks operating at the Doorhandle Supply Voltage (HVDD) which is limited to 16 V or to VDD if that is below that level. The yellow blocks are supplied by the Analog Supply Voltage (AVDD), the green ones by the Digital Supply Voltage (DVDD). Besides the power domains shown in Figure 3.1 there is also a Weak Digital Supply Voltage (UVDD) power domain. This power domain supplies the necessary digital parts in sleep mode (e.g. Power-On and Wake-Up Unit). In this mode the more powerful DVDD regulator which supplies the whole digital part can be switched off to save power. Besides the separation implicitly done via the different power domains the chip is also divided into two clock domains:

Watchdog RC-Oscillator (WRCO) clock domain: Oscillator working at 1 MHz which serves to provide a clock source for the watchdog timer as well as for the event timers of the doorhandle interfaces.

System clock domain: This is the main system clock for the digital part (sys_clk). There are two clock sources for this signal:

- System RC-Oscillator (RCO): Clock source running at 16 MHz. This signal is the clock source for the system if the external crystal is not yet ready or available.
- External Crystal (XTAL): System clock source running at 32 MHz. If the crystal frequency is stable the system automatically switches from RCO to XTAL.

Each antenna port consists of three pins to control three external FETs: Positive Channel FET (P-FET) labeled as P_x, Negative Channel FET (N-FET) marked as N_x and Modulation FET (M-FET) denoted as M_x. To provide an antenna diagnostic function each port provides also an additional input labeled DS_x. To reduce the number of pins, antenna port number four has an additional function: It serves as antenna port as well as JTAG debug interface for the OCDS of the Evatronix CPU core. The mapping between JTAG interface and antenna port is illustrated in Table 3.1. Since the antenna port interface partially works on VDD external level shifters to convert all levels to 3.3 V signals are necessary to connect the Debug-Pod.

JTAG signal	**Antenna port 4**
TDO	P_4
TDI	DS_4
TCLK	N_4
TMS	M_4

Table 3.1: mapping between JTAG interface and antenna port 4

The doorhandle interfaces are connected to the chip via the doorhandle pins. Doorhandle communication is done via a protocol operating in idle state at HVDD. The information is passed to the IC by a current loop with configurable thresholds. The built-in short protection switches off the doorhandle interface if a certain low time of the signal is exceeded.

All doorhandle blocks are connected to one doorhandle interrupt line and one doorhandle short interrupt line to communicate with the CPU. By evaluating the according status register using software it can be determined which doorhandle interface caused an interrupt.

The ADC_1 input pin serves as current measurement input for all antennas. The current is measured by the potential difference over a shunt resistor (100 mΩ). The pin ADC_2 serves as an additional sensing input pin.

The communication interfaces are the Single Wire Interface (SWI) and the Three Wire Interface (3WI). The SWI is a UART interface with a single data line operating in half duplex mode. As shown in the block diagram the interface works at VDD. The 3WI interface is a modified SPI interface using a single data line (DIO). Besides the data line a clock line (SCLK) and a chip select line (SCSE) are necessary. This interface works between 3.3 and 5 V. The 3WI is supplied by the SCSE pin which determines the high level of the interface. Besides the above mentioned interfaces there are four additional General Purpose Input/Output (GPIO) pins and five system pins:

GPIO5 works at 5 V. If this pin is configured as input and the corresponding function is enabled it serves as wake-up event for the system. In sleep mode this pin does not keep its output level. Internally a pull down resistor can be enabled.

GPIO12 is an open drain pin which is supposed to work at VDD. The output state of this pin keeps its level in sleep mode.

IRQ is an open drain output pin. It is used as synchronisation signal for the 3WI in the bootloader as well as in the final application.

ClkOut is also an open drain pin which works at 5 V. Additional to the programmable pull down functionality also internal clocks can be routed to this pin as well as a Pulse Width Modulation (PWM) output signal.

Mode is an 5 V input pin. A logic one at this pin loads the bootloader on start up, logic zero executes the program stored in Program Storage Random Access Memory (PSRAM).

Reset causes a system reset if a 5 V level is applied for a minimum of 100 µs.

Test is a pin used for internal test purposes.

XTin and XTout are used to connect the external 32 MHz crystal.

One of the main functions of the IC is the communication with an ignition key via LF wireless data transfer. The communication is based on a 21.858 kHz carrier frequency. Three types of modulation are supported: Amplitude Shift Keying (ASK), Binary Phase Shift Keying (BPSK) and Quadrature Phase Shift Keying (QPSK). By controlling an external antenna driver circuit a current causing a magnetic field is generated in the transmit resonant circuit. An amplitude or phase modulation propagates with the settling time of this circuit. This time depends on the quality factor Q. So the data rate C is limited by the transmit frequency f_{tx} and the quality factor Q_{tx} (Equation 3.1).

$$C \approx \frac{f_{tx}}{2 \cdot max(Q_{tx})} \tag{3.1}$$

To improve the data rate of the system a special modulation method is used which turns off the transmission current in its negative period. Phase Shift Keying (PSK) is performed by turning off the current in the negative period and turning it on after a certain delay. Therefore a zero crossing detector is required which is implemented in the Digital Signal Processing (DSP) part of the system. This part is named Digital Measurement Signal Processing Unit (DMSP) and controls with the antenna control block the antennas in automatic and semi-automatic mode. So the according interrupt signals depending on the current measured at the ADC_1 pin and the states of the transmit state machine are generated. The system supports three methods to generate a modulation:

Automatic modulation: In this mode the user only has to set up the modulation type (e.g. BPSK with 5.461 kbit s^{-1}) and write the data into the transmit data buffer. The transmission is started if the according bit is set. None of the generated interrupts have to be used - everything is handled automatically.

Semi-Automatic modulation: This mode partially uses the interrupts offered by the DSP block. The modulation or parts of it are done in software.

Manual modulation: In this mode everything is controlled manually. The carrier is generated by controlling the P-FET and N-FET control signals. The modulation is performed via switching the M-FET. All FET control signals are accessible via SFR registers. In this mode also higher carrier frequencies can be generated since switching the FETs is supposed to be implemented within the Interrupt Service Routine (ISR) of a system timer.

The DMSP is attached to the CPU via Advanced Peripheral Bus (APB), SFR and interrupt lines. The input of this part is the output bitstream of the Sigma-Delta ADC. There are two input paths to the ADC: the Alternating Current (AC) path and the Direct Current (DC) path. Each path owns a Multiplexer (MUX) (ACMUX, DCMUX) where different input signals can be selected. Via the DCMUX a number of internal voltage levels as well as the voltage level at the ADC_2 input can be selected. Also the voltage applied to the doorhandle inputs (divided by 10) can be multiplexed to the output with the DCMUX. The ACMUX is used to select either the signal applied on pin ADC_1 or the signal at pin ADC_2. If the AC path is selected two interrupts are generated in the signal analysis block:

Positive Zero Crossing Interrupt: This interrupt indicates that the signal crosses zero starting from a negative value to a positive.

Negative Zero Crossing Interrupt: This interrupt indicates that the signal crosses zero turning from a positive to a negative value.

The LF digital processing block generates the control signals of the P-, N- and M-FET depending on the internal PWM unit. This block generates interrupts based on the values of the PWM counter:

PWM min interrupt: If the PWM counter reaches zero.

PWM max interrupt: If the PWM counter reaches the set maximum.

MSynch interrupt: If the PWM counter value is equal to the duty cycle setting this interrupt signal is generated. The LF digital processing block ensures that the M-FET control signal is only switched in the negative period of the signal.

The digital part of the chip consists of the CPU, memories, peripherals and the DSP part. Section 3.2.1 deals with the design details of the digital top architecture.

3.2.1 Digital Top Architecture

The digital part of the system serves to generate and process the antenna signals as well as to provide an interface to control the system and to communicate with the world outside. As shown in Figure 3.1 the CPU is attached to a number of different memories. The first version of the IC is an Emulator Version (EMU). This means in addition to the PSROM there is also a PSRAM where a program could be downloaded to. In the PSROM a boot-loader resides which offers a communication protocol via the 3WI interface to download code into PSRAM. After downloading the code, the PSROM is mapped to the end of PSRAM and the code is fetched from there instead. To provide this kind of function a memory mapper has been developed within this project which is able to map different memories to the buses attached to the CPU. Section 3.2.2 deals with the detailed design of all developed blocks.

The digital top itself is an IP block (dig_top) where all digital sub-blocks are connected together. This block offers the interfaces to the analog part as well as to the padring which provides the off chip ports. Besides the hierarchical function within this IP also the low-level drivers are created and managed. The next section provides an overview about the hierarchical parts of this block.

3.2.1.1 Overview

As already mentioned there are two power domains in the digital part: The UVDD and the DVDD power domain. For this reason the digital part is divided into two blocks:

uvdd_power: This block contains the parts of the digital design which are active in sleep mode: the 28 bit watchdog timer and the power on and wake up unit. Also the system SFRs are located within this power domain. The following signals serve as wake up event if the system is in sleep mode:

- Reset pulse
- Doorhandle interface activity or short

- Watchdog timer overrun
- SWI event
- 3WI activity
- GPIO5 event

dvdd_power: This module contains all blocks operating in normal mode:

- CPU Sub-System (sd_cpu8051f_top) including configurable memories (with optional BIST and parity check blocks).
- CPU peripherals like timers, 3WI and SWI interfaces.
- Clock divider and multiplexer block.
- DSP part of the system: The DMSP and the driver pulse generator (ant_control).

To separate the two power domains an isolation layer has been introduced in the UVDD power domain. In Figure 3.2 the separation and the hierarchy of the digital part is shown.

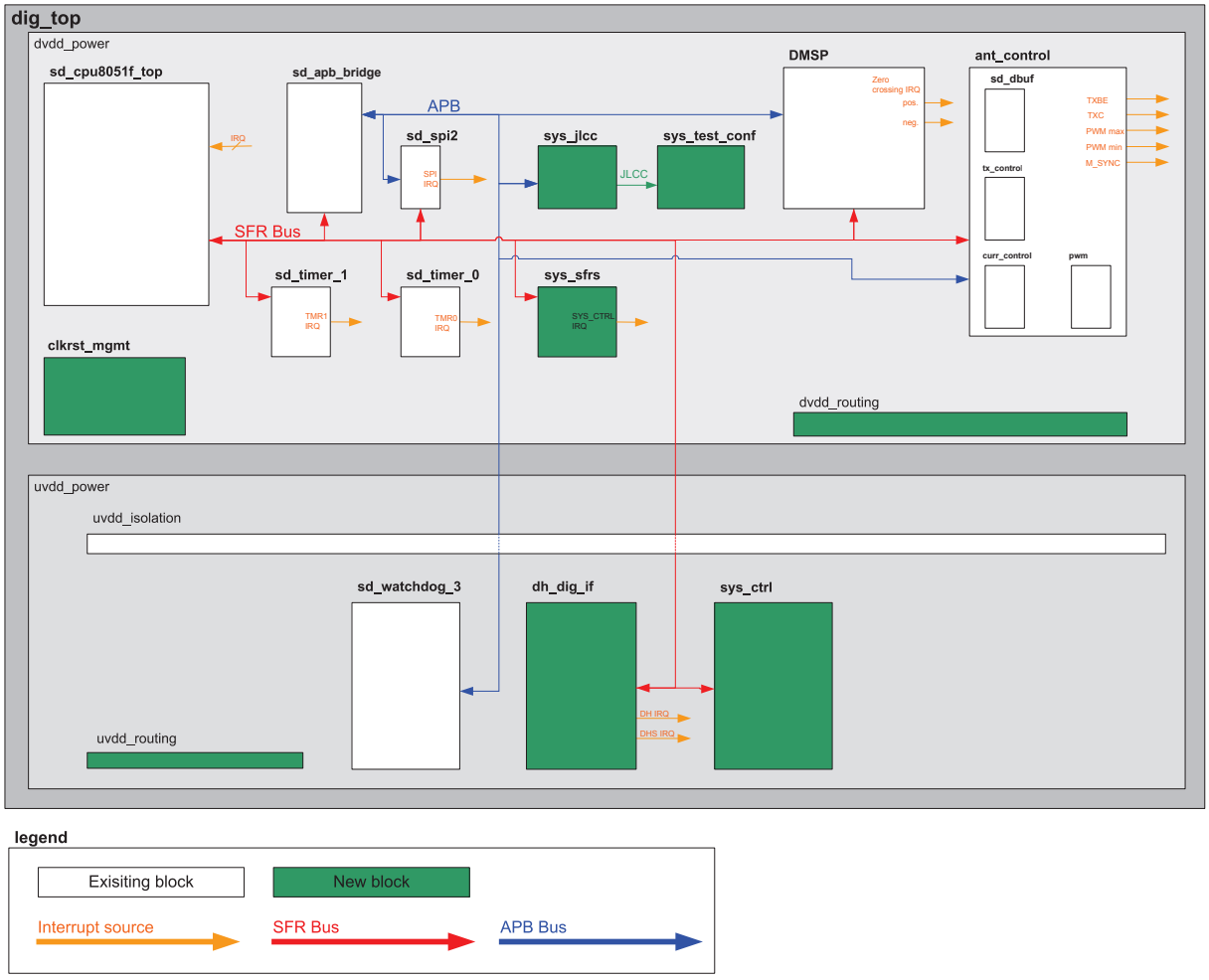

Figure 3.2: partitioning and hierarchy of the digital top IP

The next section deals with the details of the CPU Sub-System architecture and its design.

3.2.2 CPU Sub-System Architecture

The top level of the CPU Sub-System is the sd_cpu8051f_top IP. This block serves to configure features of the CPU Sub-System like used memories and their sizes. Each of the three CPU cores presented in Section 2.3.3 can be used within this block. To support this feature a CPU wrapper block has been developed (sd_cpu8051_wrapper) which is used to select one of those cores and to map the different I/O lines to a common interface. Since the timing of the cores is also different it has been decided to create additional CPU specific ILayers in the hierarchy block above (sd_cpu8051f_top).

As shown in Figure 3.3 this block integrates following sub-blocks:

- CPU wrapper block: The sd_cpu8051_wrapper block is used to provide a common interface for all of the selectable cores. The interfaces of the Evatronix core (R8051XC2) is used as interface template for this block.
- Memory mapper block: The sd_mem_mapper block serves to map different memories to the program memory bus of the core. To provide a correct switching between the memories the timing has to be considered. The memory mapper itself is controlled by the CPU wrapper block via P3. This port delays the switching command in the core by a number of cycles. Within this time setting the program counter to zero must be done. To ensure the execution of this mechanism is done in time the switching command is called by assembler code. In Section 4.5 an overview on the developed and modified firmware structure is provided.
- Memories: Each memory block can optionally be connected to a BIST block and a parity check block. The following memories can be used within the CPU Sub-System:
  - IDATA: internal RAM block
  - PSROM: read only program storage memory
  - PSRAM: RAM which could be used as alternative program storage memory
  - XDATA: external RAM
- FLASH and FLASH controller: To provide backwards compatibility the FLASH and its controller have also been integrated. Since the timing of these blocks is tied to the timing of the DDC8051 core this feature can not be used with any of the other cores without additional modifications.
- Additional I/O ports: Implemented with sd_extra_ports. This block is attached to the core via the SFR bus and offers additional 8 bit I/O ports.
- Cache block: The sd_8051_cache block implements a feature designed originally for the DDC8051: Via this block code could be downloaded directly to the core. Therefore a PSRAM (1 kB) block is used as cache and the code is received via a serial data line. If the cache functionality is not used the additional memory can be used as additional XDATA memory.

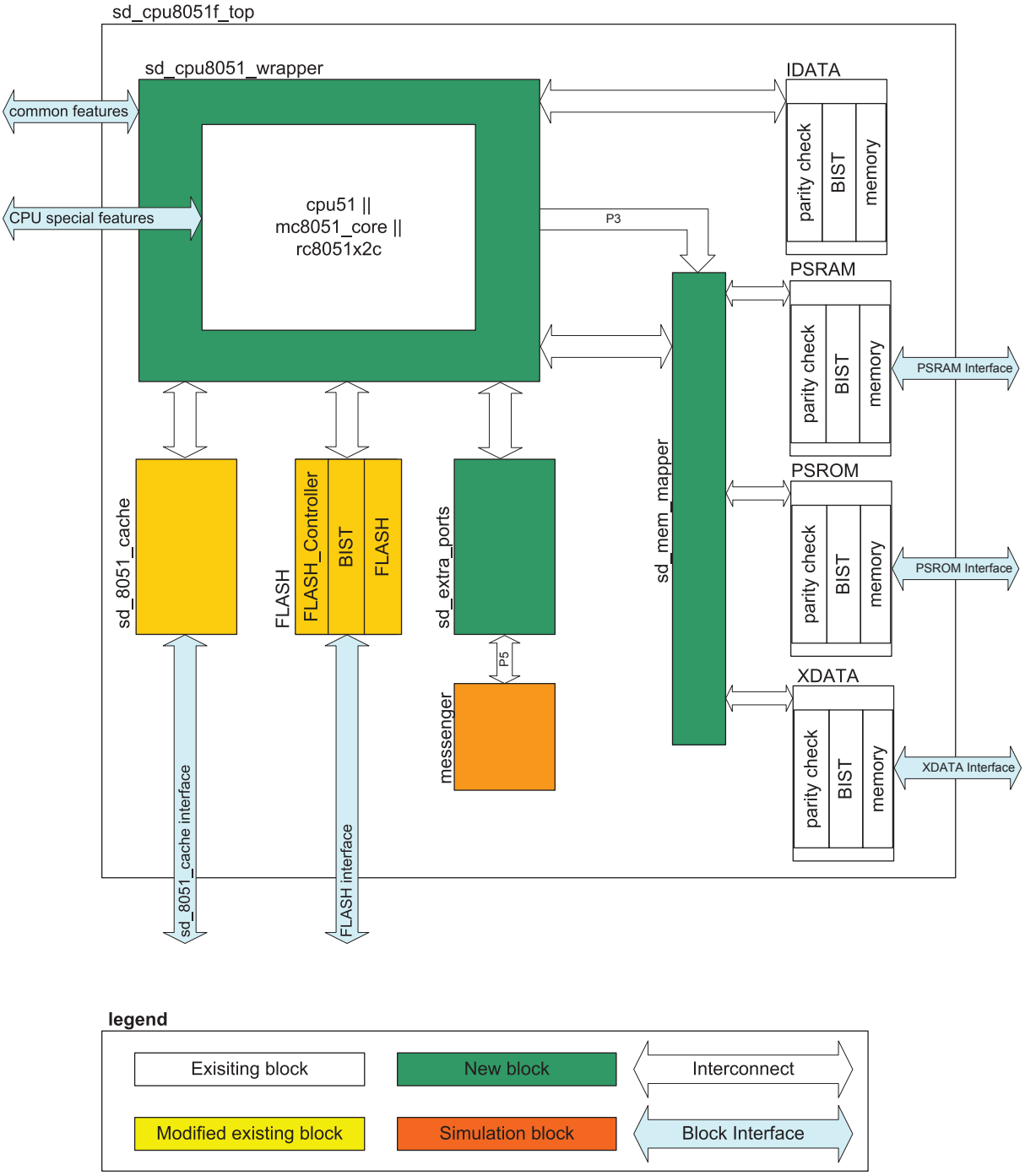

Figure 3.3: CPU Sub-System: sd_cpu8051f_top

The following interfaces are offered by the sd_cpu8051f_top IP:

- Interrupts (irq_n): The number of interrupts and interrupt priorities is different for each core (see Table 2.4).
- SFR Bus Interface: This interface is the same for all cores. Only the Evatronix core has an additional acknowledge input (sfr_ack) which is used to access slower blocks. If this feature is not used this input has to be tied high.

- External memory interfaces: If one of the following memories is not generated within the sd_cpu8051f_top block but its size is set bigger than zero the memory has to be connected externally. In this context external could mean on a higher hierarchy level or off-chip. For each memory a separate interface is provided:
  - PSROM interface,
  - PSRAM interface and
  - XDATA interface
- FLASH and FLASH controller interface: This interface allows to attach the FLASH memory externally if the FLASH controller is implemented. Also the FLASH test interface is attached to this interface.
- Cache interface: Provides the data lines needed to download code via the sd_8051_cache block.
- JTAG interface: This port is only used by the Evatronix R8051XC2 debugger. The other cores do not provide such a feature.
- Serial port interface: Used for UART. All cores offer at least one internal UART interface.
- Parallel port interface: Within this IP six ports are implemented: P0 to P5. Only P1 is supported by this block as I/O because P0 and P2 are used for external memory access, P3 is used to control the memory mapper block, P4 controls the parity check blocks and P5 is used to control the simulation messenger block.
- Additional ports: Some ports are CPU specific and provide functions like a RS485 interface (DDC8051) or a hold interface (R8051XC2) which could be used by an additional DMA controller.

The implementation details of the sd_cpu8051f_top and the sd_cpu8051_wrapper are presented in Section 4.1.

As shown above the architecture of the CPU Sub-System is very flexible and many features are selectable. Modern HDLs offer methods to define variables to control properties of a design. In VHDL such parameters are passed as generic elements to the design. The parameters can be used for example to define the bit width of a bus. In the design of the sd_cpu8051f_top IP and the sd_cpu8051_wrapper generics are used to provide following functions:

- Define the size of memories and address bus width
- Determine if blocks are implemented or not (via generate statements)
- Implementation of different signal paths depending on the value of a generic

As mentioned above the sd_cpu8051_wrapper block offers a common interface for all selectable cores. The core can be selected by defining a generic. Figure 3.4 presents the design integrated within this block. To offer the same features for all cores additional blocks are integrated.

The minimum peripheral setup for each core is:

- One serial interface: Oreganos MC8051 is able to generate and use up to 255 UART modules and Evatronix' R8051XC2 also has two serial interfaces integrated. Only for the DDC8051 an additional UART block has to be integrated.
- Four I/O ports: P0 to P3, where P3 offers the possibility to delay the signal by a number of cycles (memory mapping control interface). For the R8051XC2 this is integrated via a separate block.

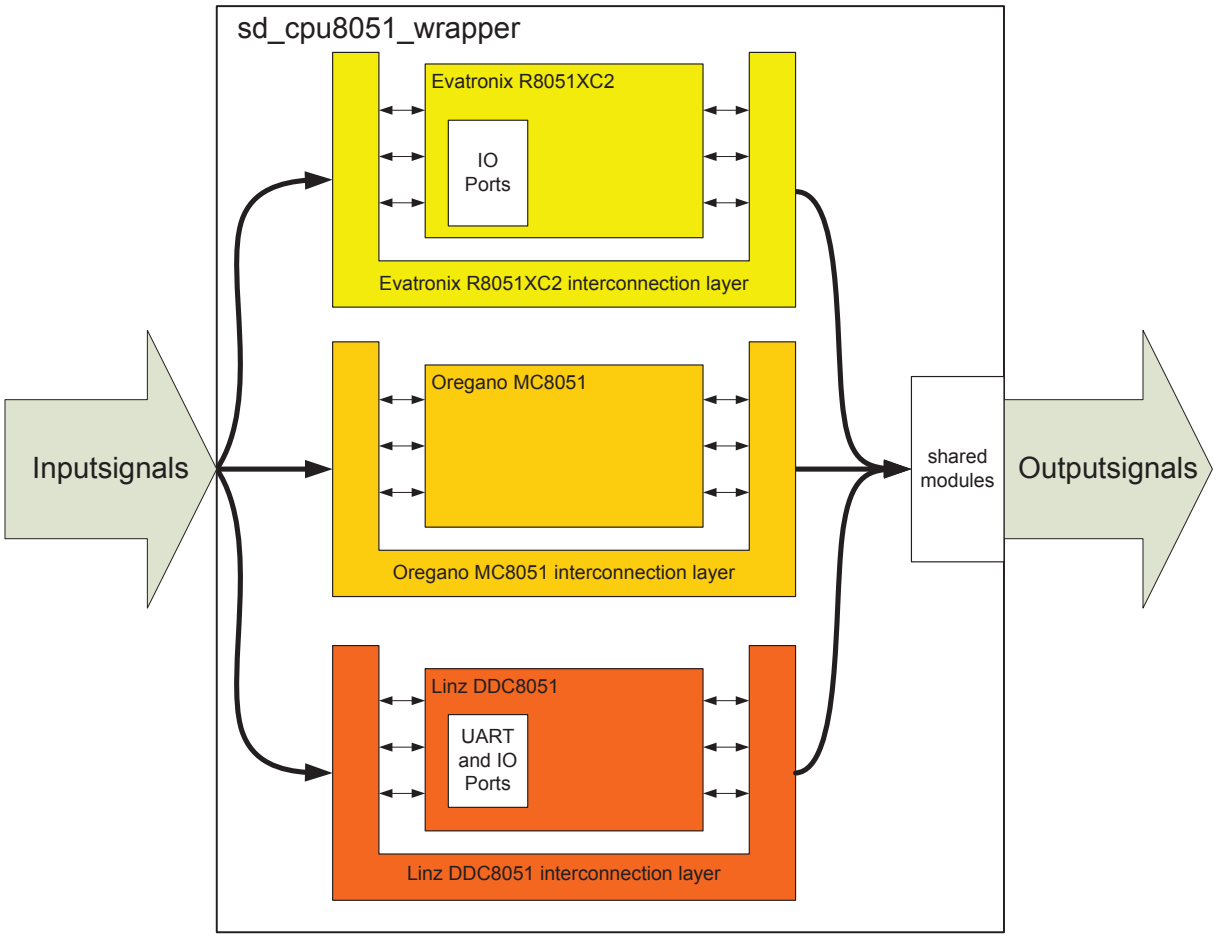

Figure 3.4: CPU core wrapper block

As shown in Figure 3.4 each core is isolated from the wrapper I/O lines via an Interconnection Layer (ILayer). This layer is only generated if the according core is chosen. It serves to define the state of wrapper signals which are not used by the selected core and to allow the conversion of signals. This means for example the memory enable signal of the DDC8051 is active low, but the memory enable signal of the R8051XC2 is active high. The inversion of this signal is done within the according ILayer. Shared modules are placed outside the interconnection layers and are used by all cores (e.g. delay of P3).

The wrapper block is the main item of the CPU Sub-System (sd_cpu8051f_top). The generic used in this block to select a core specific ILayer is the same generic which is passed to the wrapper to determine which core to use. Figure 3.5 illustrates the design of a peripheral block and its Interconnection Layers (ILayer).

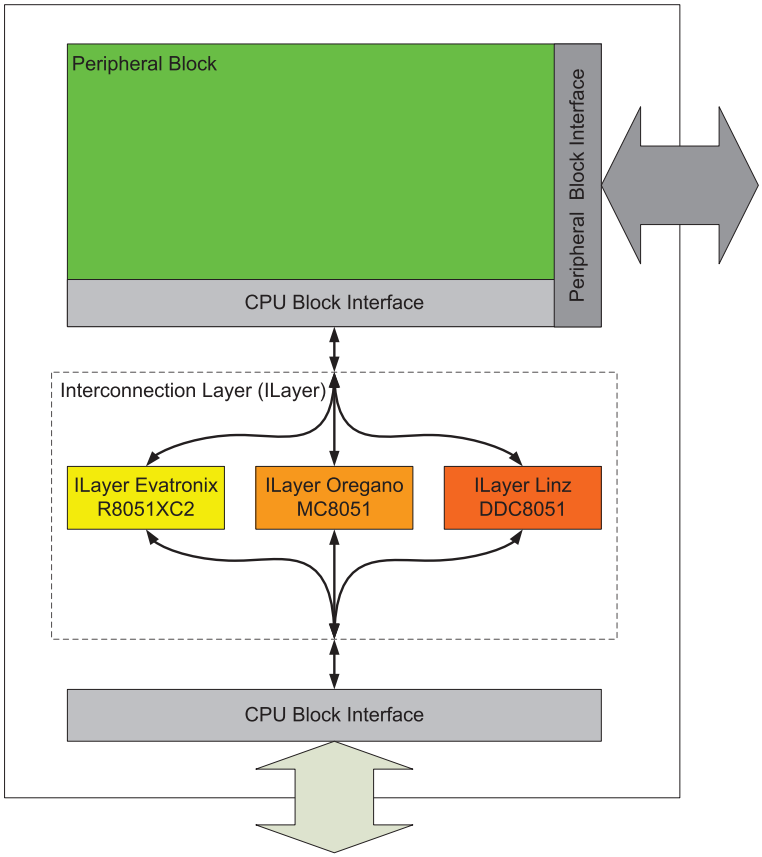

Figure 3.5: sub-block interface design in the sd_cpu8051f_top

This approach is necessary because the timing of the CPU cores differ and due to this additional modules or signals have to be generated for some blocks. This approach also offers additional advantages:

- If a new core is integrated into the wrapper none of the peripherals are connected to this core until the according layer is added for each peripheral block.
- A new peripheral block can only be attached to the cores for those the ILayer has been defined.

Disadvantages are the increased development time and the number of tests to ensure every feature is working properly in each possible configuration. Due to the time schedule the design has only been tested and verified with the Evatronix R8051XC2 (see Section 5).

As discussed above the EMU version of the ASIC supports two memory layouts illustrated in Figure 3.6:

Mode 0, Bootloader mode: In this mode the bootloader stored in the PSROM is executed. Via the 3WI interface a new application is downloaded to PSRAM.

Mode 1, Application mode: In this mode the application stored in PSRAM is executed. This only works if a program has been downloaded to the PSRAM. If a reset is applied to the IC the data stored in PSRAM is lost. Data in PSROM could be accessed by adding the offset of the PSRAM memory size to the address which should be read.

The block implementing this feature is the sd_mem_mapper block within the sd_cpu8051f_top IP. At the moment only the presented configurations are supported but the block is designed in a way that new configurations can be added very easily.

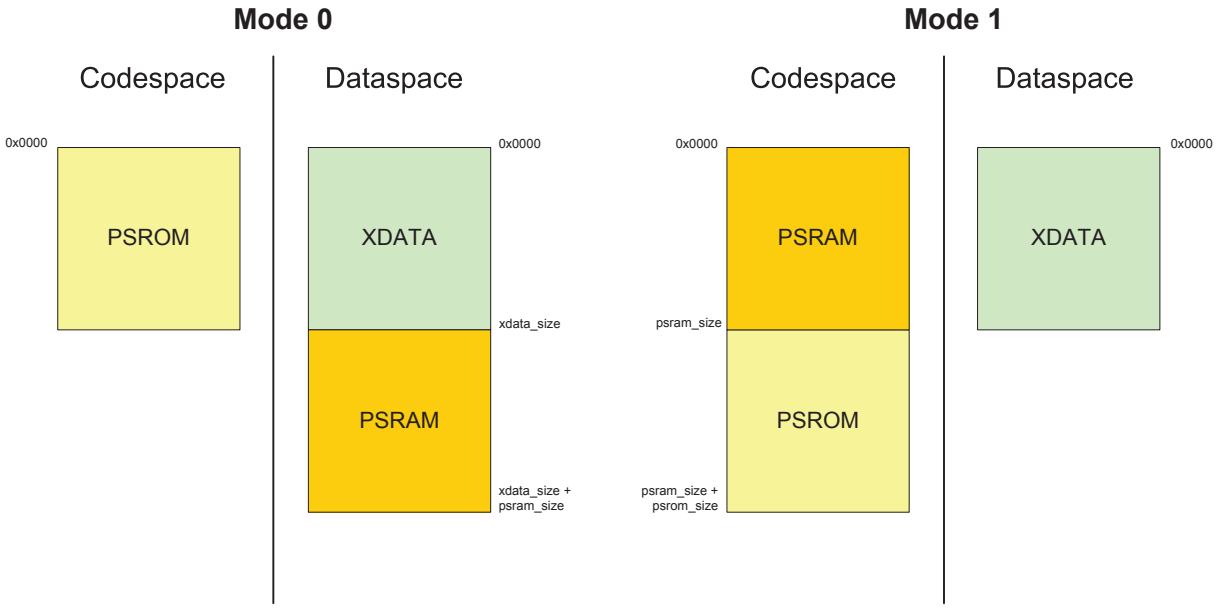

Figure 3.6: memory configurations

The sizes of the memories used are passed as generic parameters to the memory mapping block as well as the information whether the block itself should be generated or not. If it is decided not to integrate the block a bypass path is created which causes no overhead in synthesis. The size of the memories are used to define the maximum addressable space for program and data memory so that stacking two memories one after another without any empty space between is possible. This feature simplifies the setup of different compilers. The memory mapper also checks if the applied address is valid in the current configuration. If not a signal is generated (illegal address) which is connected to the system control block. Depending on the setup of this block (configurable by SFR) the signal either resets the digital part of the IC or causes a system interrupt.

All memories except the IDATA are connected to the CPU via the sd_mem_mapper block. The ILayers pictured above are also implemented for the memories control signals. Besides the memory unit itself also a BIST and a parity check block can be created (selectable by generic). In Figure 3.7 an example of a memory block with BIST and parity check block is shown.

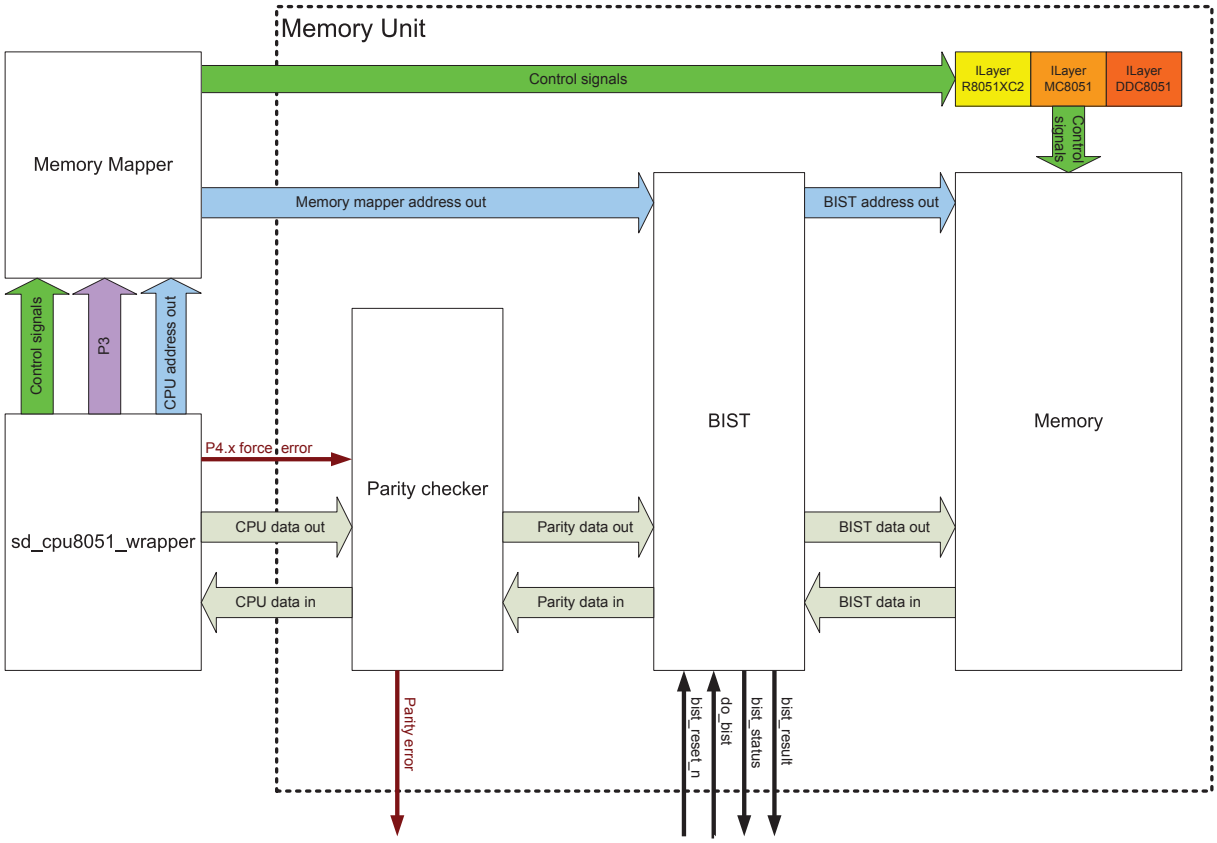

Figure 3.7: memory unit integration design

BIST and parity check blocks can also be created if there is no memory implemented within this hierarchy to provide those features for external memories as well. To test the detectability of errors each parity check block implements an error injection feature which is controlled by P4. For each BIST block there are four signals: reset_bist_n, do_bist, bist_status and bist_result. A BIST is performed if the do_bist signal is active. After BIST has finished the bist_status line is active and the bist_result line indicates if the test has passed or failed. The reset_bist_n signal serves to reset the BIST controller. If a BIST or parity block is not generated the according structure is replaced by a bypass structure which is removed during the synthesis process.

The next chapter deals with the basic implementation of the design shown above in VHDL as well as with the integration of the CPU Sub-System into the system. Also the integration of the digital part into an FPGA and the final synthesis of the design for the usage in the IC is discussed. The structure of the firmware and the necessary firmware drivers are also covered at the end of the following chapter.

Chapter 4

Implementation

4.1 Implementation of the CPU Sub-System

4.1.1 Implementation Basics

This section provides a short overview of the necessary basics of VHDL and the RTL synthesis design flow used to implement the design described in Section 3. A VHDL design entity consists of a file which models a hardware block. The file consists of a general part, an entity block and an architecture block. It is possible to use multiple architectures in one file but then also a configuration declaration block is necessary to determine which architecture to use.

The general part defines which libraries and packages are imported. The entity block is used to define the interface and the generics of the block. The architecture block is the part where the behaviour of the hardware is defined. In Listing 4.1 the code of a QUAD AND gate written VHDL is shown. At the top of the code the necessary IEEE standard packages are included. In those files (also written in VHDL) the necessary data types and operations are defined. Some tools are delivered with their own libraries providing similar functions which are more powerful than the IEEE functions. The problem with these libraries is the reduced portability if other tools are used. The first declaration in Listing 4.1 after including the standard libraries is including a custom library and its package. If a module is reused it has to be declared as component before it can be used (instantiated). The component definitions for a library (common represents an IP) are defined within a package. The code presented in Listing 4.1 in line 6 would make all components defined within the **exist_pkg** package available in the current source file. Besides the definition of components also constants, functions and type definitions can be defined within a package.

After including the libraries the definition of the entity follows. The entity declaration defines the black box properties of the design. As shown in the example a generic could be used to define the bus width of ports.
After the entity declaration the architecture of the design is defined. This is a very short example which only integrates a binary AND of each input signal (a, b) and puts the result to the according output (y).

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.numeric_std.all;
4
5  library exist_lib;
6  use exist_lib.exist_pkg.all;
7
8  entity module is
9      generic(
10       g_bus_size  : integer := 4
11     );
12     port(
13       a      : std_logic_vector(g_bus_size-1 downto 0);
14       b      : std_logic_vector(g_bus_size-1 downto 0);
15       y      : std_logic_vector(g_bus_size-1 downto 0)
16     );
17 end module;
18
19 architecture str of module is
20 begin
21   y <= a and b;
22 end architecture rtl;
```

Listing 4.1: sample VHDL code (QUAD AND Gate)

To support readability and maintainability of VHDL code following digital design rules have been defined in [14]:

- One file per entity with the same name. File names of packages have to end with '_pkg', libraries with '_lib'.
- Usage of the SensorDynamics' naming convention for VHDL code elements:
  - Generics have to start with 'g_'. Only generics of the type integer are allowed.
  - Constants have to be marked with 'c_' as prefix. They can be defined either in a package or before the begin statement of an architecture. Constants are for example used in packages to define addresses for an IP.
  - Instances are labeled with 'i_'. An instance is a particular implementation of an entity used in another architecture of a block. The interconnection between instances is done via lines called signals.
  - Processes should be named with 'p_' as first characters. Since everything in VHDL is executed in parallel processes provide an environment for sequential statements.
- Architectures should be named after their type: *str* for a structural, *beh* for a behavioural or *rtl* for an RTL architecture.
- Ports must be of type std_logic or std_logic_vector.

- Registers must use an asynchronous active low reset.

Besides the improvement of readability, the guidelines above serve also to improve the creation of synthesiseable code. Since VHDL is a very powerful language not every design which could be described can be synthesised. The IEEE standard 1076.6-04 describes the standard syntax and semantics for VHDL RTL synthesis. From a tool point of view the steps needed to create a real physical block from a design are divided into two major parts:

Front End: This term describes all processes needed to create and verify a gate level netlist from a system level description. The system level design could be done for example in MATLAB. The output of this model are the test vectors (e.g. stimuli files) which are reused to verify the RTL design done in VHDL or Verilog of the system or its blocks. The designed hardware is checked within a testbench using a simulator. After the design is verified at this abstraction level it is passed to synthesis and netlist generation. The netlist is usually the input for Back End.

Back End: All steps needed to create a physical implementation (layout) from the netlist are summarised as Back End. Usually following steps are needed to create a physical design of a netlist:

1. Floorplanning
2. Place and route
3. Clock tree synthesis
4. Layout
5. Design Rule Check (DRC), Electrical Rule Check (ERC) and Layout Versus Schematic (LVS) check
6. Parasitics Extraction (RCX) of the design for simulation

The steps described above are usually visualised as design flow (e.g. Figure 2.2). The tools used within the SD design flow are controlled by scripts. If a block within an IP is created only the correct file names and locations have to be added to the execution script. To perform an RTL level simulation of a design the following tasks are have to be executed by the script:

Compilation: The syntax and semantic of design files is checked. The Compiler analyses each design unit separately and stores it into the appropriate library.

Elaboration: Flattening the design. If instances in a design unit are used those elements are expanded recursively. The result of the elaboration is a flat design consisting of signals and processes.

Simulation: The simulator executes the extracted processes as a discrete event based simulation. This means a process is sensitive to its inputs - if an input changes an event is created and the process is evaluated. If an output signal of a process is different from the current value a new event is generated. The signal update and the process execution are two different phases of the simulation. The initialisation phase assigns the initial signal values, sets the current simulation time (T_c) to zero and calculates the first simulation cycle (T_n).

Afterwards the simulator performs the following simplified algorithm until simulation is finished:

1. Set the simulation time T_c to the next transaction time T_n.
2. Update all active signals with their new values and generate events if the according signal value changes.
3. Processes are executed if they are sensitive to changed signals or if the process is scheduled to resume at T_s. The process is evaluated until it suspends (if it suspends). It is important to note that the execution order is not defined. The processes generate the future values for the signals.
4. The next transaction time T_n is evaluated: It is set to the next time a signal is scheduled to be active or a process resumes. The next simulation cycle will be a delta cycle (δ) if T_n is equal to T_c. The delta cycle is necessary to simulate delay less models. Between two simulation cycles there is an arbitrary number of delta cycles.

To test modules usually a testbench written in VHDL is created which instantiates the block under test and applies signals based on stimuli data generated by a model. VHDL offers the possibility to access files to read stimuli data and to log result data. Often also the outputs of the block under test are evaluated immediately and the result of the test is a pass or fail information. The simulation setup used to verify the CPU Sub-System implemented within this thesis is presented in Chapter 5. The next sections deal with the implementation details of the CPU Sub-System.

4.1.2 CPU Wrapper

As presented in Chapter 3 the sd_cpu8051_wrapper provides an easy change between different 8051 CPU cores. The block is part of the sd_cpu8051f_top IP. At the moment three cores are supported: DDC8051, MC8051 and R8051XC2. The core is selectable by setting the generic g_select_cpu to one of the follwing values:

DDC8051: g_select_cpu = c_use_cpu8051_pi_core

MC8051: g_select_cpu = c_use_cpu8051_oregano_core

R8051XC2: g_select_cpu = c_use_cpu8051_evatronix_core

The constants to select the correct core are defined in the sd_cpu8051f_top_pkg package. Figure 4.1 shows the interface and the all available generics of the sd_cpu8051_wrapper block. The function of the used generics is illustrated in Table 4.2. For each of them a default value is defined. Only generics which are different to the default value have to be set if an instance of the sd_cpu8051_wrapper block is created.

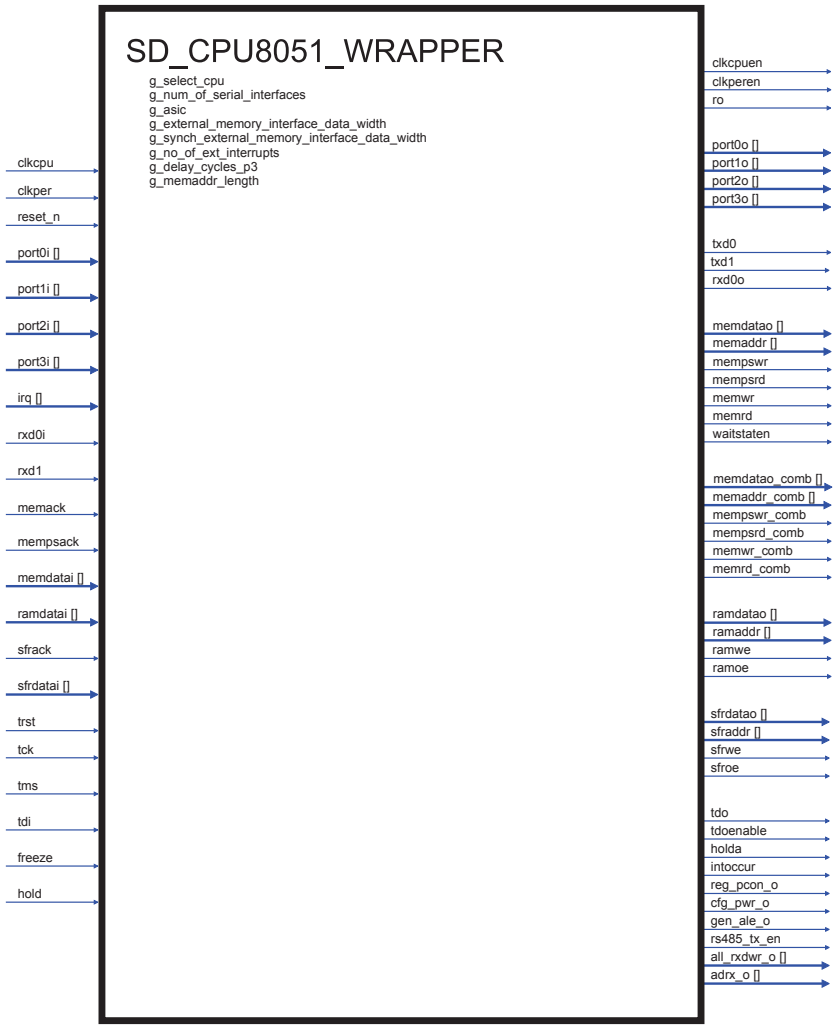

Figure 4.1: wrapper interface and generics

Generic	Description
g_select_cpu	Selects the core to be instantiated. Use one of the constants defined in the sd_cpu8051f_top_pkg to select the core.
g_num_of_serial_interfaces	Sets the number of serial interfaces to use. The R8051XC2 and the DDC8051 provide only one serial interface, the MC8051 supports up to 255.
g_asic	Generic to switch between FPGA implementation and ASIC implementation. This generic is for future use and has no influence on the current implementation.
g_no_of_ext_interrupts	Sets the number of external interrupts.
g_delay_cycles_p3	P3 is always implemented. This generic defines the number of delay cycles between assertion of a value to P3 and passing it to the output.

Generic	Description
g_memaddr_length	Defines the memory address width. DDC8051 and MC8051 only support 16 bits. The R8051XC2 supports up to 23 bits.
g_external_memory_interface_data_width	Combinational databus width is always 8 bit (R8051XC2 only). This generic is implemented to be prepared for future improvements.
g_synch_ext_memory_interface_data_width	Synchronous databus width is always 8 (used for all cores). This generic is implemented to be prepared for future improvements.

Table 4.2: wrapper generics

Incorrect generic settings are prohibited by assertion based checks. Those checks are only implemented if the appropriate structure is generated. Assertions are non synthesiseable VHDL constructs raising a simulation error if the according condition evaluates to false. For example if the R8051XC2 is used, the generic which selects the number of serial interfaces (g_num_of_serial_interfaces) has to be set to '1'. Any other value for this generic is prohibited if the Evatronix core is used. The code presented in Listing 4.2 checks if the correct value has been used. If an incorrect setup is done the string defined in the report statement is displayed. The severity statement selects if the simulation is stopped (severity level: error or failure) or continued (severity level: note or warning).

```
1 assert g_num_of_serial_interfaces = 1
2          report "[ERROR] [cpu8051_evatronix] number of serial
              interfaces must be 1, is:"&integer'image(
              g_num_of_serial_interfaces)
3          severity error;
```

Listing 4.2: assertion based generic check in VHDL

Table 4.4 shows the interface differences between the selectable cores. The wrapper in principle is based on the Evatronix R8051XC2 core interface. Additional to these ports also useful features of the other cores have been made available (e.g. PCON register interface of the DDC8051) if the appropriate core is implemented. If the core does not support the feature the appendant line is set to its inactive state. As an additional feature the freeze input pin has been added. This input is used as clock gating signal for all clock lines.

Wrapper	Dir.	#	R8051XC2	DDC8051	MC8051
Clock and reset signals					
clkcpu	in	1	clkcpu	zuluclka_i	clk
clkcpuen	out	1	clkcpuen	*not available*	*not available*
clkper	in	1	clkper	zuluclkb_i	*not available*
clkperen	out	1	clkperen	*not available*	*not available*
reset_n	in	1	reset	syncreset_i	reset
ro	out	1	ro	*not available*	*not available*
I/O Ports					
port0i	in	8	ext. block	ext. block	p0_i
port0o	out	8	ext. block	ext. block	p1_i
port1i	in	8	ext. block	ext. block	p2_i
port1o	out	8	ext. block	ext. block	p3_i
port2i	in	8	ext. block	ext. block	p0_o
port2o	out	8	ext. block	ext. block	p1_o
port3i	in	8	ext. block	ext. block	p2_o
port3o	out	8	ext. block	ext. block	p3_o
Interrupt lines					
irq	in	gen.	int0 to int12	irq_lines_i (6)	int0_i, int1_i (i)
Serial interfaces					
rxd0i	in	1	rxd0i	ext. block	all_rxd_i
rxd1	in	1	rxd1	ext. block	all_rxd_i
txd0	out	1	txd0	ext. block	all_txd_o
txd1	out	1	txd1	ext. block	all_rxdwr_o
rxd0	out	1	rxd0	ext. block	all_rxdwr_o
Memory interfaces					
mempsack	in	1	mempsack	*not available*	*not available*
memack	in	1	memack	*not available*	*not available*
memdatai	in	8	memdatai	ext_rddata_i	rom_data_i
memdatao	out	8	memdatao	Uses ext_wrdata	dataxo
memaddr	out	gen.	memaddr (n)	reg_extaddr_o (15)	rom_adr_o (15)
mempswr	out	1	mempswr	*not available*	*not available*
mempsrd	out	1	mempsrd	psen_o	*not available*
memwr	out	1	memwr	wrstrobe_extram_o	wrx_o
memrd	out	1	memrd	rdstrobe_extram_o	*not available*
waitstaten	out	1	waitstaten	*not available*	*not available*
Combinational memory interfaces					
memdatao_comb	out	8	memdatao_comb	*not available*	*not available*
memaddr_comn	out	gen.	memaddr_comn	*not available*	*not available*
mempswr_comb	out	1	mempswr_comb	*not available*	*not available*
mempsrd_comb	out	1	mempsrd_comb	*not available*	*not available*
memwr_comb	out	1	memwr_comb	*not available*	*not available*
memrd_comb	out	1	memrd_comb	*not available*	*not available*
IDATA memory interface					
ramdatai	in	8	ramdatai	int_rddata_i	ram_data_i

Wrapper	Dir.	#	R8051XC2	DDC8051	MC8051
ramdatao	out	8	ramdatao	reg_wrdata_o	ram_data_o
ramadr	out	8	ramadr	reg_intaddr_o	ram_adr_o (6)
ramwe	out	1	ramwe	wrstrobe_intram_o	ram_wr_o
ramoe	out	1	ramoe	rdstrobe_intram_o	ram_en_o
SFR interface					
sfrack	in	1	sfrack	*not available*	*not available*
sfrdatai	in	8	sfrdatai	sfr_rddata_i	*not available*
sfrdatao	out	8	sfrdatao	Uses reg_wrdata	*not available*
sfraddr	out	8	sfraddr (6)	Uses reg_intaddr	*not available*
sfrwe	out	1	sfrwe	wrstrobe_sfr_o	*not available*
sfroe	out	1	sfroe	rdstrobe_sfr_o	*not available*
OCDS interface					
trst	in	1	trst	*not available*	*not available*
tck	in	1	tck	*not available*	*not available*
tms	in	1	tms	*not available*	*not available*
tdi	in	1	tdi	*not available*	*not available*
tdo	out	1	tdo	*not available*	*not available*
tdoenable	out	1	tdoenable	*not available*	*not available*
Additional interfaces					
freeze	in	1	ext. clock gating	ext. clock gating	ext. clock gating
hold	in	1	hold	*not available*	*not available*
holda	out	1	holda	*not available*	*not available*
intoccour	out	1	intoccour	*not available*	*not available*
reg_pcon_o	out	8	*not available*	reg_pcon_o	*not available*
cfg_pwr_o	out	1	*not available*	cfg_pwr_o	*not available*
gen_ale_o	out	1	*not available*	gen_ale_o	*not available*
rs485_tx_en	out	1	*not available*	rs485_tx_en	*not available*
all_rxdwr_o	out	gen.	*not available*	*not available*	all_rxdwr_o
adrx_o	out	gen.	*not available*	*not available*	adrx_o

Table 4.4: wrapper mapping and interface overview

As described in Section 3.2.2 it is necessary to add additional blocks for each core to provide the minimum requested function set. The following list shows which blocks are created if the appropriate core is generated:

DDC8051 core: Besides the core IP (cpu51) only the sfrs_pi IP is used. This block adds four ports (P0, P1, P2, P3) and a serial interface to the core.

MC8051 core: No additional hardware blocks are necessary to use the MC8051. If the core is used, be aware that it has to be adapted to support a SFR interface and external interrupts.

R8051XC2 core: The serial interface is implemented by the core. Only the parallel I/O ports have to be added. Therefore the module sd_internal_ports has been created. This modules map SFR addresses to the according ports. The addresses of the ports are defined in the sfrs_config_pkg. Usually these registers are mapped to the addresses of the original 8051.

In addition to the blocks created on demand of the selected core a block to delay P3 is always implemented. The module is named sd_delay. In Figure 4.2 the implemented structure of the sd_cpu8051_wrapper is shown if the R8051XC2 core is selected.

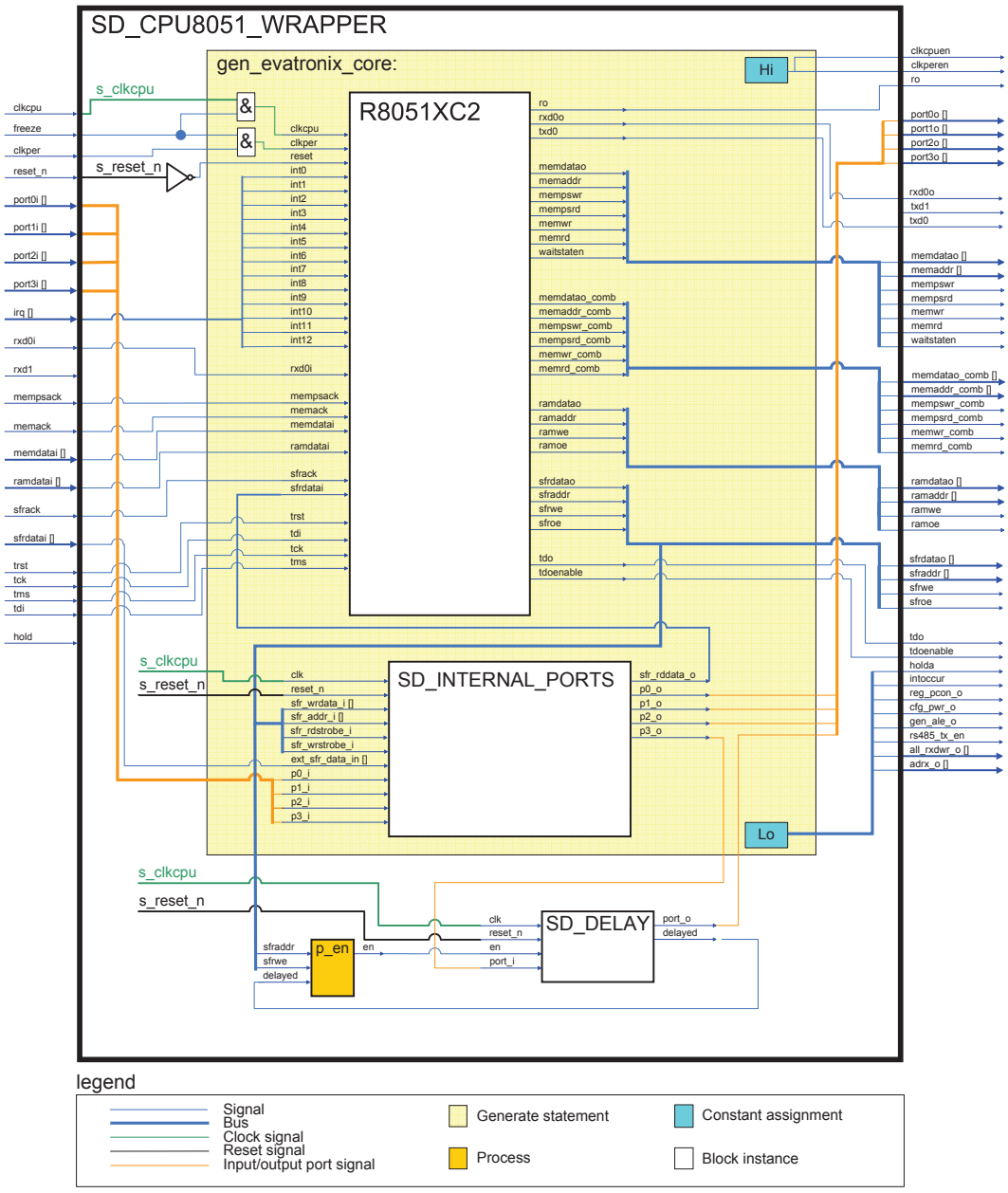

Figure 4.2: wrapper implementation with the Evatronix R8051XC2 core

Figure 4.2 also shows the connection of unused output signals to defined values (Hi - logic one, Lo - logic zero). The generate block is the element which differs for each core. In Listing 4.3 the VHDL code of the generate statement is presented. Generate

blocks can be used to generate instances, connections and signal assignments depending on a generic. Such generate blocks are also used in the CPU Sub-System IP to generate different ILayers.

```
1 gen_core: if (g_select_cpu=c_use_evatronix) generate
2     -- Put block instantiations, signal assignments
3     -- and/or connection mappings here
4 end generate gen_core;
```

Listing 4.3: VHDL generate statement

In Figure 4.2 a process block is shown which generates an enable signal (en) for the sd_delay block. This process enables the delay block if a value is written to P3. This is done by comparing the SFR address bus with the address of P3. If a write occurs, the enable signal of the sd_delay block is active. This signal starts the counter implemented within this block. If a certain value (selectable by g_delay_cycles_p3) is reached, the signal at port port_i is forwarded to the according output port port_o. Also the delayed signal becomes active and is passed back to the process. There the enable signal is set inactive. P3 is used to control the memory mapping block in the sd_cpu8051f_top. The next section illustrates the implementation details of this block.

4.1.3 CPU Sub-System

In Figure 4.3 the interface of the sd_cpu8051f_top is shown. The figure also illustrates the names of the available generics. Ports are labeled with an "[ ]" after their name. All other signals are single lines.

Generics in general are named after the function they are controlling. Since in this module there are many sub-blocks using the same generic names they are named after the according block and their function. XDATA and PSRAM for example use the same memory IP (sd_spram). The generic to define the memory address width (which defines indirectly its memory size: $2^{\text{address\_width}}$) is called ADDR_SIZE. Since this is not compliant with the SD design guidelines it has been decided to use generics containing the memory size. So the generic g_psram_size defines the size used for PSRAM, the g_xdata_size does the same for XDATA. The mapping between memory size (e.g. 4 kB) and address bus width is done via the function f_log2_z. This function is located in the sd_cpu8051f_top_pkg and performs a modified logarithm to the base two (log2). The algorithm was modified to return $x = 0$ if its parameter is 0: $x = log2(0)$. Table 4.6 provides an overview on the appropriate generics and their functions.

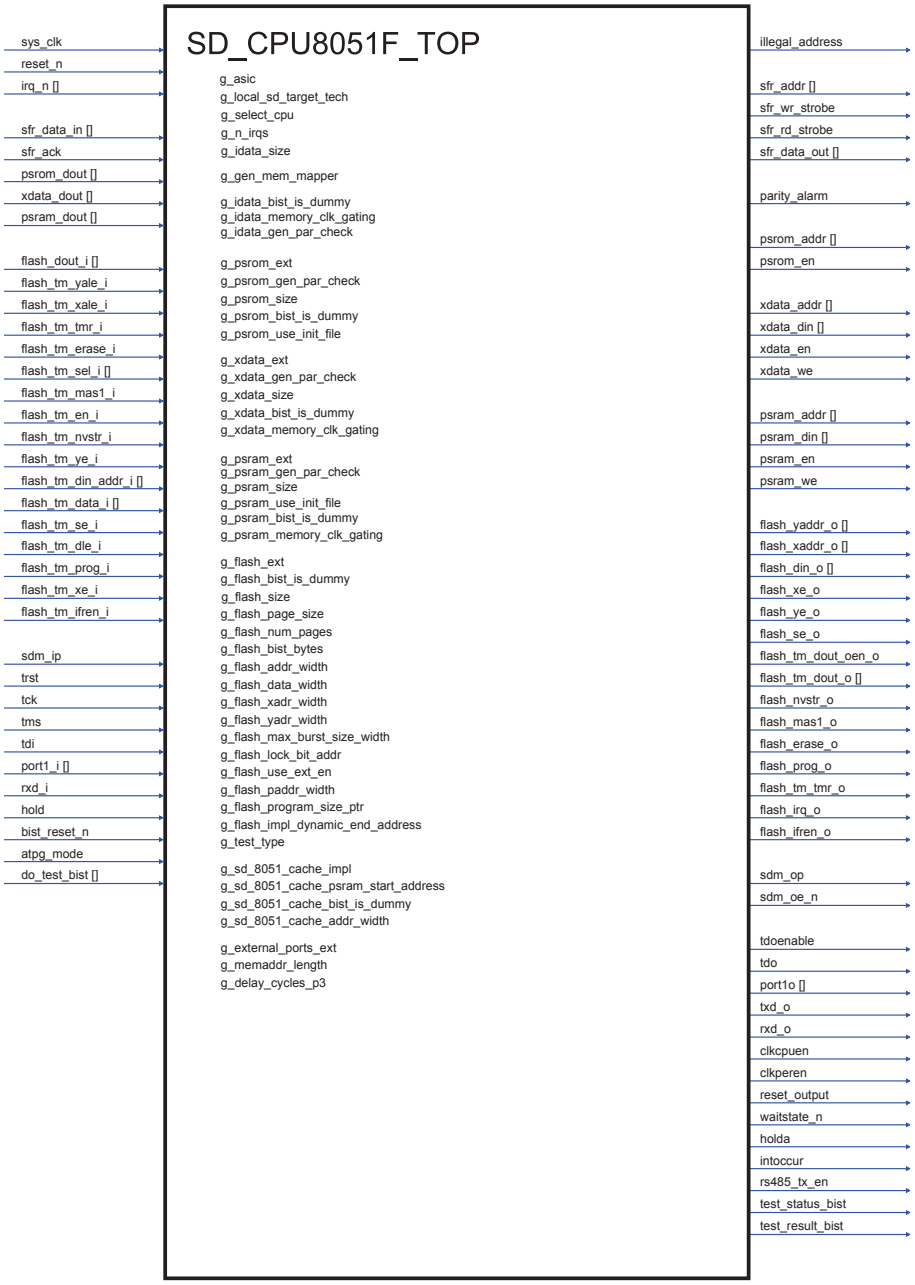

Figure 4.3: black box view of the CPU Sub-System

Generic	Description
Global generics	
g_asic	Generic to switch between FPGA implementation and ASIC implementation. This generic is for future use and has no influence on the current implementation.
g_local_sd_target_tech	Defines the target technology to use for memories.

Generic	Description
g_select_cpu	Select core: Use c_use_cpu8051_ constants from sd_cpu8051f_top_pkg. Switch between DDC8051, MC8051 or R8051XC2 core in the sd_cpu8051_wrapper.
g_n_irqs	Defines the number of external interrupts
g_idata_size	Defines the size of IDATA. DDC8051 and R8051XC2: 256 bytes, MC8051: 128 bytes.
g_gen_mem_mapper	If set ('1') the sd_mem_mapper is generated.
IDATA generics	
g_idata_bist_is_dummy	If zero the BIST module for IDATA is generated.
g_idata_memory_clk_gating	Defines if clock gating for IDATA should be implemented.
g_idata_gen_par_check	If set ('1') the parity check module for IDATA is implemented.
PSROM generics	
g_psrom_ext	If zero, PSROM is implemented within sd_cpu8051f_top, otherwise it is to be connected externally.
g_psrom_gen_par_check	If set to '1', the parity check module for PSROM is generated.
g_psrom_size	Defines the size of the implemented (or externally attached) PSROM.
g_psrom_bist_is_dummy	If zero, BIST for PSROM is generated.
g_psrom_use_init_file	If set, code.bin is used as PSROM content, otherwise nothing is loaded to PSROM. This is used by the simulation model of the memory.
XDATA generics	
g_xdata_ext	If zero, XDATA is implemented in sd_cpu8051f_top, otherwise it is to be connected externally.
g_xdata_gen_par_check	If set, parity check for XDATA is generated.
g_xdata_bist_is_dummy	If zero BIST for XDATA is generated, otherwise BIST is not generated.
g_xdata_size	Defines the size of the implemented (or externally attached) XDATA memory.
g_xdata_memory_clk_gating	If zero, clock gating for XDATA is not implemented.
PSRAM generics	
g_psram_ext	If zero, PSRAM is implemented in sd_cpu8051f_top, otherwise it is to be connected externally.
g_psram_gen_par_check	If set, parity check for PSRAM is generated.

Generic	Description
g_psram_size	Defines the size of the implemented (or externally attached) PSRAM.
g_psram_use_init_file	If set, the PSRAM init file (code_psram.bin) is loaded to PSRAM.
g_psram_bist_is_dummy	If zero, BIST for PSRAM is generated.
g_psram_memory_clk_gating	If zero, clock gating is disabled.
FLASH generics	
g_flash_addr_width	FLASH address bus with.
g_flash_bist_bytes	Number of BIST bytes.
g_flash_bist_is_dummy	If zero BIST for FLASH is generated.
g_flash_data_width	FLASH data bus width.
g_flash_ext	If '0', FLASH memory is implemented externally (but flash controller is created).
g_flash_impl_dynamic_end_address	If set dynamic BIST end address is implemented.
g_flash_lock_bit_addr	Defines the address of the lock bit.
g_flash_max_burst_size_width	Maximum BIST size width.
g_flash_num_pages	Number of pages in the FLASH.
g_flash_paddr_width	FLASH page address bus width.
g_flash_page_size	FLASH page size.
g_flash_program_size_ptr	Pointer to memory location of program size
g_flash_size	Defines the size of the FLASH (at the moment only 32kB + 256 Byte are supported).
g_flash_use_ext_en	External FLASH memory enable (implement FLASH controller).
g_flash_xadr_width	Flash x address bus width.
g_flash_yadr_width	Flash y address bus width.
g_test_type	Defines FLASH test type.
sd_8051_cache generics	
g_sd_8051_cache_impl	If set, the cache block is implemented.
g_sd_8051_cache_addr_width	Cache address width.
g_sd_8051_cache_bist_is_dummy	If zero BIST is generated.
g_sd_8051_cache_psram_start_address	PSRAM start address (PSRAM dimension).
Additional generics	
g_external_ports_ext	If set, the external ports (P4 / P5) are not created.
g_memaddr_length	Memory address bus size (default: 16).
g_delay_cycles_p3	Number of cycles to delay after switching memory configuration via P3.

Table 4.6: CPU Sub-System generics

Most of the generics presented are used to control generate statements or to define functionality used by memories (e.g. BIST). All memories except IDATA can be connected

externally without losing any functionality (parity check, BIST or memory mapping) provided by that IP. For each memory, that could be connected externally a separate interface exists. In Table 4.8 all ports of this block is presented.

Interface	Dir	#	Description
Clock and reset signals			
sys_clk	in	1	System clock input
reset_n	in	1	Asynchronous active low system reset
reset_output	out	1	Reset indicator (active if a reset has been applied)
clkcpuen	out	1	External control for clkcpu (if active apply clock)
clkperen	out	1	External control for clkper (if active apply clock)
External interrupts and error indicators			
irq_n	in	gen	Active low external interrupt bus. Some interrupt polarities may be configurable in the core
illegal_address	out	1	Indicates that an illegal address was accessed via the memory mapper
parity_alarm	out	8	Indicates parity errors
I/O ports			
port1i	in	8	Port 1 input bus
port1o	out	8	Port 1 output bus
Serial port interface			
rxd_i	in	1	Serial 0 receive data line
rxd_o	out	1	Serial 0 transmit data line
txd_o	out	1	Serial 0 receive clock output in mode
rs485_tx_en	out	1	RS485 transmit enable signal
SFR interface			
sfr_addr	out	8	SFR address bus (the highest bit is tied high)
sfr_data_in	in	8	SFR data bus input
sfr_data_out	out	8	SFR data bus output
sfr_rd_strobe	out	1	SFR read strobe (active high)
sfr_wr_strobe	out	1	SFR write strobe (active high)
sfr_ack	in	1	SFR acknowledge (active high, Evatronix only)
PSROM interface			
psrom_addr	out	gen	External PSROM address bus
psrom_dout	in	8	External PSROM data bus input
psrom_en	out	1	External PSROM enable signal (polarity depends on CPU sd_mem_mapper constant)
XDATA interface			
xdata_addr	out	gen	External XDATA address bus
xdata_din	out	8	External XDATA data bus output

Interface	Dir	#	Description
xdata_dout	in	8	External XDATA data bus input
xdata_en	out	1	External XDATA enable signal
xdata_we	out	1	External XDATA write enable signal
PSRAM interface			
psram_addr	out	gen	External PSRAM address bus
psram_din	out	1	External PSRAM data bus output
psram_dout	in	1	External PSRAM data bus input
psram_en	out	1	External PSRAM enable signal
psram_we	out	1	External PSRAM write enable signal
OCDS JTAG interface			
tck	in	1	JTAG Interface clock input
tdi	in	1	JTAG Interface test data input
tdo	out	1	JTAG Interface test data output
tdoenable	out	1	JTAG Interface test data output enable
tms	in	1	JTAG Interface test mode select input
trst	in	1	JTAG Interface reset
Test interface			
atpg_mode	in	1	ATPG mode control signal
bist_reset_n	in	1	BIST reset signal, active low
do_test_bist	in	8	BIST start signal
test_result_bist	out	8	BIST result value
test_status_bist	out	8	BIST status value
FLASH interface			
flash_din_o	out	8	External FLASH data bus output to FLASH memory
flash_dout_i	in	8	External FLASH data bus input from FLASH memory
flash_erase_o	out	1	External FLASH define erase cycle
flash_ifren_o	out	1	External FLASH information page access enable
flash_irq_o	out	1	Interrupt request, flash controller
flash_mas1_o	out	1	External FLASH define mass erase cycle
flash_nvstr_o	out	1	External FLASH define non volatile store cycle
flash_prog_o	out	1	External FLASH define program cycle
flash_se_o	out	1	External FLASH sense amplifier enable signal
flash_tm_data_i	in	8	FLASH Controller test mode input
flash_tm_din_addr_i	in	gen	FLASH Controller test mode input
flash_tm_dle_i	in	1	FLASH Controller test mode input
flash_tm_dout_o	out	8	FLASH Controller test mode input
flash_tm_dout_oen_o	out		FLASH Controller test mode input
flash_tm_en_i	in	1	External FLASH test multiplexer switch
flash_tm_erase_i	in	1	FLASH Controller test mode input

Interface	Dir	#	Description
flash_tm_ifren_i	in	1	FLASH Controller test mode input
flash_tm_mas1_i	in	1	FLASH Controller test mode input
flash_tm_nvstr_i	in	1	FLASH Controller test mode input
flash_tm_prog_i	in	1	FLASH Controller test mode input
flash_tm_se_i	in	1	FLASH Controller test mode input
flash_tm_sel_i	in	3	FLASH Controller test mode input
flash_tm_tmr_i	in	1	External FLASH interface test mode reset input
flash_tm_tmr_o	out	1	External FLASH interface test mode reset, for user mode '1'
flash_tm_xale_i	in	1	FLASH Controller test mode input
flash_tm_xe_i	in	1	FLASH Controller test mode input
flash_tm_yale_i	in	1	FLASH Controller test mode input
flash_tm_ye_i	in	1	FLASH Controller test mode input
flash_xaddr_o	out	gen	External FLASH x address input to select row
flash_xe_o	out	1	External FLASH x address enable signal
flash_yaddr_o	out	gen	External FLASH y address input to select a byte within a row
flash_ye_o	out	1	External FLASH y address enable signal
Cache interface			
sdm_ip	in	1	sd_8051_cache serial input data
sdm_oe_n	out	1	sd_8051_cache serial output data enable
sdm_op	out	1	sd_8051_cache serial output data
Optional DMA ports			
hold	in	1	Hold mode (R8051XC2 only - optional)
intoccur	out	1	IRQ occurred in hold mode (R8051XC2 only - optional)
holda	out	1	Hold mode acknowledge signal (R8051XC2 only - optional)
waitstate_n	out	1	Active low, when CPU performs a wait cycle (R8051XC2 only - optional)

Table 4.8: CPU Sub-System interface overview

The CPU Sub-System also uses generate statements if blocks are not created to tie the appropriate outputs to defined levels. This means for example that if no PSRAM is used that the according parity and BIST information bits are set to their inactive levels. The structure of the sd_cpu8051f_top is based on necessary instances (e.g. sd_cpu8051_wrapper) and optional generate blocks (e.g. gen_psram).

The following list explains the structure of the CPU Sub-System based on the existing instances (i_) and generate blocks (gen_):

i_sd_cpu8051_wrapper: This is the instance of the wrapper block described in Section 4.1.2. It is the main element of the CPU Sub-System.

i_messenger: The messenger block is a module used in simulation to pass messages from firmware to the simulation environment. Therefore the messenger is connected to the core via a parallel port (P5). The firmware implements a function called *messenger("STRING", MSG_INFO)* which writes the parameters to P5. The messenger block decodes the received data and generates an assertion with the received string. The second parameter sets the severity of the assert message.

i_sd_extra_ports: This block connects two additional parallel ports (P4 and P5) to the SFR bus. As described above, P5 is used to control the messenger block. P4 is used to control the force parity functionality of the memories with implemented parity check.

gen_idata: The IDATA memory is the only memory that can not be generated externally. The IDATA itself is an instance of the sd_spram block. This block is part of the sd_tech IP. The modules within this IP are used to provide the instantiation of the selected memory for a certain technology (e.g. FPGA or TSMC 0.18 μm). Besides the memory itself also the according BIST block (i_idata_sd_bist_sram) and the parity check block (i_idata_sd_parity_check) are created. Depending on the according generics (g_idata_gen_par_check, g_idata_bist_is_dummy) the blocks are functional or bypassed.

gen_mem_mapper: This generate statement instantiates the memory mapping block (i_sd_mem_mapper) if the according generic (g_gen_mem_mapper) is set to any other value than '0'. Otherwise this generate statement is not evaluated and the gen_no_mem_mapper statement is implemented. In this case only the generation of PSROM and XDATA is allowed.

gen_psrom: Generate statement which instantiates the selectable features for PSROM memory. The memory itself (sd_psrom) is a wrapper module in the sd_cpu8051f_top IP. It instantiates the correct sd_sprom block from the sd_tech IP with the selected memories size. At the moment only the following sizes are supported: 1 kB, 4 kB, 8 kB or 16 kB. The number of data bits is selectable (8 or 9) for each of them. It is also possible to assign a file to the model which could be loaded in simulation into the memory.

gen_xdata: Within this statement the memory (sd_spram) used as XDATA is instantiated. Also the according BIST and parity check blocks are instantiated. Depending on the according generics they are bypassed or implemented.

gen_psram: For the PSRAM also an instance of the sd_spram is created. Depending on the selected mode the memory mapper assigns the PSRAM to different memory locations. The creation of BIST and parity check blocks is handled as in the XDATA generate block. If the PSRAM contains a program it is not allowed to perform a BIST check because this would overwrite the memory contents. Also for this memory an initialisation file could be defined for simulation.

gen_flash: This statement adds a FLASH memory which can only be used if the DDC8051 is used. Within this generate block a BIST module is generated for the FLASH as well as the FLASH controller itself (sd_flash_control). If the memory is defined to be integrated within the sd_cpu8051f_top (g_flash_ext='0') an instance of sd_flash is created. At the moment only 32 kB + 256 Byte size is supported.

gen_cache: If the g_sd_8051_cache_impl generic is set this block creates an instance of the sd_8051_cache module. The module must be used with the DDC8051.

Finally a closer look at the implementation of the sd_mem_mapper block is provided. The memory mapper is configurable via generics for obvious configurations like memory sizes. There is also a generic (g_dont_assert_addressing_errors) which defines whether assertions have to be raised if an illegal address is applied. It is necessary to be able to switch off this behaviour because simulations are usually stopped if an illegal address error is detected. To test if those errors are recognised correctly the simulation must not stop.

An additional feature of the memory mapper is the conversion of memory enable signals for each attached memory. The CPU only provides a program storage memory enable (mem_ps_en) and a data storage memory enable (mem_en). In the configuration package of the memory mapper (sd_mem_mapper_config_pkg) constants defining the active and inactive levels for each memory are defined as well as the levels provided by the core. The final mapping between core enable and memory enable is implemented in the sd_mem_mapper block depending on the constants defined within the configuration package.

To provide linear addressing for gap-less arrangement of different memories two functions have been created:

Function f_mem_idx_cal: This function takes three input parameters (s_1, s_2 and $subtr$) and returns an integer. The result (ret) is calculated with the formula shown in Equation 4.1.

$$ret(s_1, s_2, subtr) = \begin{cases} \log_2(s_1 + s_2) - subtr, & \text{if } \log_2(s_1 + s_2) > substr \\ 0, & \text{else} \end{cases} \tag{4.1}$$

This function is used to calculate the upper border of address lines for two memories arranged after each other. The memory sizes are passed as parameter s_1 and s_2 to the function. The parameter $substr$ is set to 1 if the upper limit is used to define a bus width. Lets assume following configuration: s_1=1024 byte, s_2=512 byte and $substr$=1. The function calculates a logarithm to the basis two of 1024 + 512 and rounds the result up to an integer number (11). So we need 11 bits to address those two memories. The function returns 10 because the parameter $substr$ is set to 1. This is done because indexing in VHDL usually starts at zero ($11 \equiv 10...0$).

Function f_or_reduce: This function takes a std_logic_vector as input and ors the single elements. If one of the lines is active also the result (std_logic) is active.

The following example explains how the memory mapper works: Let's assume a system using 32 kB PSROM and 16 kB PSRAM. So the output of f_mem_idx_cal(32768, 16384, 1) is 15. We need to use 16 lines of the address bus. If we are in memory mode 1, PSRAM is mapped to the program memory space before PSROM. This means if we access any address above 16 kB the data should be loaded from PSROM. To address the PSRAM only $log_2(16384) = 14$ address lines are necessary. So control signals, data and address bus are routed to PSRAM if f_or_reduce(address(15 downto 14)) is inactive. If one of these lines becomes active those signals are multiplexed to PSROM instead. This is done in hardware so the firmware does not have to care about switching between memories (e.g. access of address 34231).

If an address exceeds the possible address space in the current configuration an illegal address error is reported. The active level for this signal is defined in the memory mapper configuration package. This signal is connected to the interface of the sd_cpu8051f_top module and has to be evaluated externally. The integration of the CPU Sub-System into the system is described in the next section.

4.2 CPU Sub-System Integration

This section describes the connection of the CPU Sub-System with the other parts of the system. As shown in Figure 3.2 parts of the system communicate with the CPU Sub-System with at least one of the following methods:

- SFR bus
- APB bus
- Interrupts

The APB bus interface is provided by the sd_apb_bridge IP which is controlled by SFRs. The APB interface is bus system with 32 bit data (read and write separated) and 16 bit address width. So the APB bridge occupies twelve SFR registers:

- One control register (OPCODE)
- One monitor register (MONCFG)
- Two address registers (PADDR)
- Four data read registers (PRDATA)
- Four data write registers (PWDATA)

Within the APB bridge there are drivers implemented which are used by the IP modules offering an APB interface. The relative addresses of those modules are defined in the according IP. In the system for each IP an offset is defined so that the addresses are unique within the system. For example the local address of the SPI IP of the RBR register is 0x0008. The address offset within the SPI IP for APB is defined to be 0x0200.

So the RBR can be accessed via APB by using address 0x0208. This method provides the necessary flexibility to reuse IP modules controlled by APB. To simplify handling those modules this functionality is abstracted by firmware macros.

The disadvantage of connecting peripherals via APB is the time consuming communication. Only the required SFR register access makes accessing an APB interfaced module seven times slower than accessing a module mapped directly to a SFR. So most of the peripherals are attached to the CPU Sub-System as SFRs. In Figure 4.4 the SFR mapping of the complete system is shown.

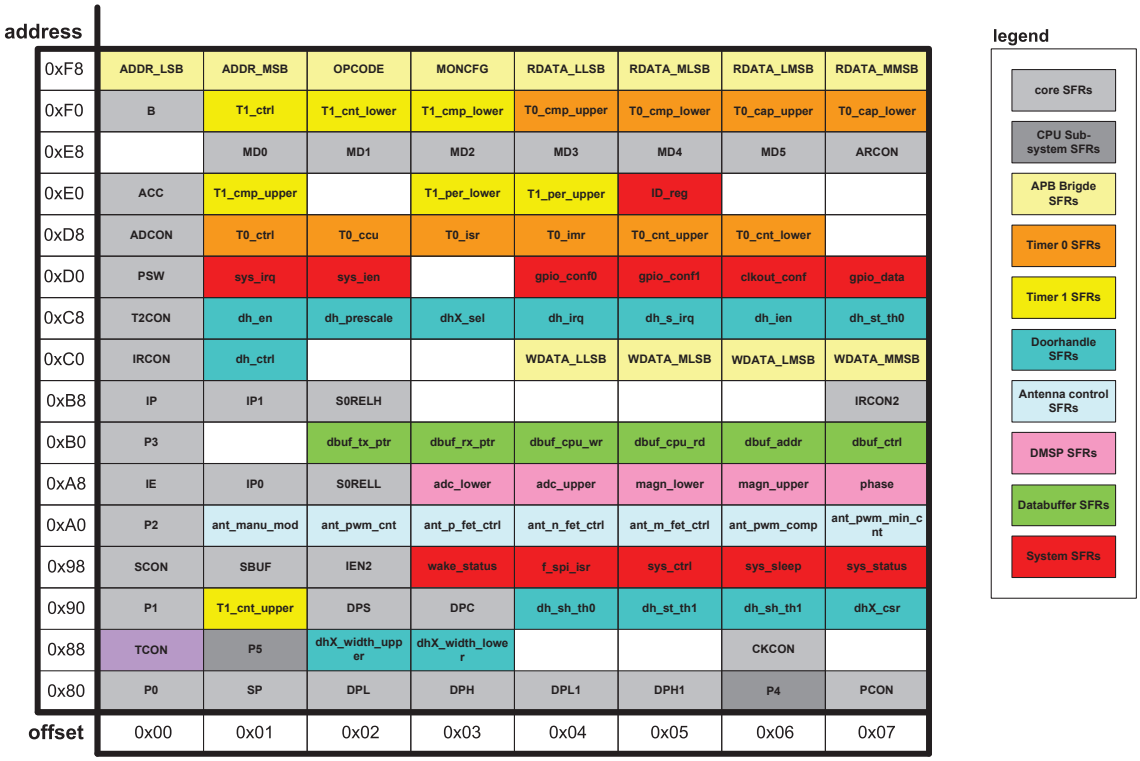

Figure 4.4: system SFR map

Because the R8051XC2 core has no integrated timer two timer IP modules have been attached to the system: Timer 0 and Timer 1. Timer 0 is a general purpose 16-bit counter/timer. It can be used as timer using the internal clock or as counter using and external clock. Additionally a capture and compare unit is available. This allows the capture of external events, and the generation of external events based on register compare values. Timer 1 is a general purpose 16-bit timer with integrated PWM unit. The timers are connected to the CPU Sub-System by the SFR bus. Additionally each timer is attached to the CPU via one interrupt line.

The R8051XC2 offers 13 interrupt lines with 4 interrupt priority levels. The priorities of the interrupts are defined by default and can be changed by writing the according SFR registers (IP0 and IP1). The interrupts are organised in six groups. Each group is allocated to two bits (one in IP0 and one in IP1). In Table 4.9 the default interrupt group assignment is shown.

Group	Highest priority <— Lowest priority			IP.# bit
Group 0	EXT_INT_0		EXT_INT_7	0
Group 1		EXT_INT_8	EXT_INT_2	1
Group 2	EXT_INT_1	EXT_INT_9	EXT_INT_3	2
Group 3		EXT_INT_10	EXT_INT_4	3
Group 4	SER_INT_0	EXT_INT_11	EXT_INT_5	4
Group 5		EXT_INT_12	EXT_INT_6	5

Table 4.9: Evatronix R8051XC2 interrupt arrangement

Group 0 is the group with the highest priority, group 6 the one with the lowest. Within a group the priority decreases from left to right. The assignment of system interrupts to the R8051XC2 interrupt lines shown in Table 4.10 was done in respect to the system requirements and use cases. The interrupts have been assigned in a way that in general no additional configuration of the IPx registers is necessary. The following interrupts exist within the system:

- Timer 0 Interrupt: Since this interrupt can have several sources the *T0_isr* has to be evaluated. The value of this register indicates if a timer event, a capture event or a compare event caused the interrupt. The counting value is stored in the SFRs: *T0_cnt_upper* and *T0_cnt_lower*.
- Timer 1 Interrupt: The source for this interrupt is Timer 1. There are no alternative functions within this unit to cause an interrupt.
- System Interrupt: This interrupt can have several sources. To evaluate which event caused the interrupt the *sys_irq* register has to be evaluated. All interrupt sources are maskable by clearing the according bit in the *sys_ien* register. Following events can cause a System Interrupt:
  - A rising edge at GPIO5
  - A parity error has been detected (alternatively this causes a system reset)
  - A rising edge at CLKOUT
  - An illegal address error has been detected (alternative: system reset)
  - A falling edge at GPIO12
  - An over temperature event occurred
  - A rising edge of the XTAL_RDY signal (this signal indicates that the system has switched to the external crystal)
- Doorhandle Interrupt: Indicates an event at the doorhandle interface. Source for this event could be the reception of data at the according doorhandle pin (S0 ... S4) or if the doorhandles are configured as timers a timer overrun of one of those timers. The register *dh_irq* indicates which doorhandle interface caused the interrupt. To read the value of the received pulse the according doorhandle has to be selected in the *dhX_sel* register and the value could be read from the registers *dhX_width_upper* and *dhX_width_lower*. If the mode of the according doorhandle should be changed the new mode has to be written to the *dhX_csr* SFR.

- Doorhandle Short Interrupt: If a pulse at a doorhandle pin exceeds the maximum width set up in register *dh_sh_th0* or *dh_sh_th1* the according module is disabled and a Doorhandle Short interrupt is generated.

- Negative Zerocrossing (PNZC) Interrupt: The DMSP block analyses the signal at pin ADC_1. This pin has an offset of 1.5 V. If a signal drops from a level higher than the offset value to a level below this threshold this interrupt is generated.

- Positive Zerocrossing (NPZC) Interrupt: This signal causes an interrupt if the ADC_1 offset is crossed from a lower level to a level above the offset.

- PWM Max Interrupt: This interrupt is caused by the PWM unit of the antenna control IP. The PWM unit is a counter which increases its count until its maximum value is reached. Then the counter is decreased until it reaches zero. If the maximum value is reached the PWMmax interrupt is generated. The value of the PWM counter can be read from the *ant_pwm_cnt* register. Via the *ant_pwm_comp* register the duty cycle of the P-FET and the N-FET signals is controlled. This is used to define the power used for the transmission.

- PWM Min Interrupt: If the PWM counter of the antenna control unit reaches zero this interrupt is internally processed. The interrupt increases a counter. Only if this counter is equal to the value set in the SFR *ant_pwm_min_cnt* the interrupt is passed to the CPU.

- M-Sync Interrupt: If in automatic modulation mode this interrupt is asserted whenever the value of the M-FET is updated. This ensures the M-FET is only switched in the negative half-wave of the generated waveform.

- TX Buffer Empty Interrupt: The transmit data buffer can only hold 64 Bytes. If a data packet exceeds this size the TX Buffer Empty Interrupt is used to indicate that there is only one byte left in the data buffer. So the remaining bytes of the packet to transmit can be written to the data buffer if this interrupt occurs. To fill the data buffer the packet data only has to be written to the *dbuf_cpu_wr*. If the autoincrement bit in the *dbuf_ctrl* register is set the address of the data buffer (stored in *dbuf_addr*) is incremented after each access. This address points to the location in the data buffer where the data is written to or read from (via *dbuf_cpu_rd*). The current write and read pointers are also available via SFR access: *dbuf_tx_ptr*, *dbuf_rx_ptr*.

- TX Complete Interrupt: This interrupt marks the end of an LF transmission in automatic transmission mode.

- 3WI Interrupt: This interrupt is caused by the SPI IP. To provide a fast reaction on events the interrupt and status register of this block could be accessed directly via the *f_spi_isr* SFR. Configuration of the IP and data exchange is done via APB register access.

- SWI Interrupt: The internal UART interface is used as SWI interface. The according interrupt line is directly attached to the core and indicates the occurrence of an SWI event.

Some interrupts shown above are generated by multiple sources (e.g. Timer 0). If this is the case there exists a second register to indicate which source caused the shared interrupt. In such a case not only the interrupt register within the CPU Sub-System has to be cleared but also the according interrupt status register of the module has to be cleared. In Table 4.10 the most important use cases requiring interrupts are shown.

	UC1: Automatic Modulation			**UC2: Manual Modulation**			**UC3: Semi-Manual Modulation 2**			**UC4: Semi-Manual Modulation**			
Interrupt	Latency	Likelihood	Priority	Latency	Likelihood	Priority	Latency	Likelihood	Priority	Latency	Likelihood	Priority	Sum
Timer 0				10	5	10	10	5	10				50
Timer 1				10	5	10	10	5	10				50
3WI	7	8	7	7	8	7	7	8	7	7	8	7	88
NPZC	1	5	1	10	5	10	10	1	10	1	5	1	60
PNZC	1	5	1	10	5	10	10	1	10	1	5	1	60
TX Buffer Empty	8	2	8	1	0	1	1	0	1	1	0	1	24
TX Complete	5	1	5	1	0	1	1	0	1	1	0	1	17
PWM Min	1	4	1	10	0	10	10	4	10	10	4	10	74
PWM Max	1	6	1	10	0	10	10	6	10	10	6	10	80
MSYNC	1	4	1	1	0	1	1	4	1	10	4	10	38
sys_irq													0
doorhandle short													0
doorhandle													0

Table 4.10: system use cases and interrupt priority assignment

According to these use cases the interrupts have been evaluated and assigned. The interrupts are rated by three criteria:

Latency: Defines the importance of reacting on this interrupt. A number between one and ten is assigned to this value. One means the timing is not important as long as the interrupt is handled. Ten means the interrupt has to be processed immediately.

Likelihood: This defines the probability of the occurrence of the respective interrupt in the presented use case. If an interrupt is not generated a value of zero is assigned to it. A value of ten indicates that the interrupt occours very often in this use case.

Priority: The value in this column defines if this interrupt is important for the current use case (10) or not (1).

The sum of the criteria for all use cases of an interrupt defines its default priority. It has been decided to assign the highest priority to the Timer 0 interrupt because this could be also used as system tick for an operating system (OS). To keep the flexibility of the system high the Timer 1 interrupt has been assigned to the next lower priority group. The other interrupts have been assigned appropriate to the values presented in Table 4.10. The doorhandle and system interrupts have not been considered in the table because they are not relevant for the analysed use cases. The final interrupt assignment is presented in Table 4.11 which allows the user to use the system in almost all use cases without changing interrupt priorities.

Group	Highest priority <— Lowest priority			IP.# bit
Group 0	Timer 0		3WI	0
Group 1		Timer 1	System	1
Group 2	PWM Max	PWM Min	MSYNC	2
Group 3		NPZC	PNZC	3
Group 4	SWI	TX Buffer Empty	Doorhandle Short	4
Group 5		TX Complete	Doorhandle Event	5

Table 4.11: interrupt assignment

If an interrupt occurs context depending registers are pushed to the stack by the core. The rest of the context saving has to be manually done by the user (in assembler) or the compiler does it (in C). Since there are different C compilers it may happen that the context is saved in different ways. So the time from interrupt indication to the first command executed in the interrupt service routine varies between the same code generated by different compilers. The second disadvantage of automatic context saving is that it is not always obvious which registers are necessary to save and which optimisations the compiler does. For example if in an ISR only a port is toggled no additional context saving may be necessary. But if a calculation is executed at least the according calculation registers have to be stored. If the compiler optimises the context saving for each interrupt separately the time between interrupt indication and first executed command differs for both routines. For real time applications it is always necessary to have a look at the generated assembler code to verify the ISR is executed in time. If an interrupt service routine can be interrupted by a higher priorised interrupt the interrupt priorities have to be set appropriately.

The integrated R8051XC2 is able to to nest up to four interrupt service routines. By disabling all interrupts with a higher priority (or all interrupts) it is possible to avoid the interruption of an ISR. Since some of the interrupts (NPZC, PNZC and MSYNC) rely on the output of the ADC they can not be tested on the FPGA. For this reason a digital stimuli unit has been added to the FPGA containing a pattern where number and timing of interrupts is known. So all interrupt sources could be tested and used in the FPGA implementation. The digital part of the system (with some modifications) has been synthesised and tested on a Xilinx Spartan 3 FPGA Starter Kit. The next section deals with FPGA implementation of the digital part.

4.3 FPGA Prototype

To target the FPGA a slightly simplified version of the digital part has been used. The following modifications have been made:

- Smaller XDATA memory: 256 Bytes instead of 512 Bytes
- Selectable serial interface: UART or SWI interface
- Watchdog is not connected to reset
- System Clock: 16 MHz instead of 32 MHz
- Simplified Clocking scheme
- Additional block: Simulation SINC to simulate waveforms converted by the ADC
- Mapping of P1 output to a seven-segment display
- Connection of the Dual In-Line Package Switch (DIP Switch) row to the input of P1 (7 bits) and to the MODE pin

The Simulation SINC is necessary to feed the DMSP with real data to generate the necessary interrupt signals for automodulation and to test the connection of those interrupts to the core. The output of this module is exactly the same as the output of the real one except the value of the amplitude. Due to the design of the CPU Sub-System and the limitations of the used FPGA it was not possible to synthesise a version working at 32 MHz system clock. The host system is connected to the Xilinx Spartan 3 FPGA Starter Kit board via a UART interface and Evatronix' OCDS Debug Pod.

Since the used FPGA is not able to store its configuration (the digital design) if it is not supplied there are two additional Platform FLASH (PROM) memories on board. They are connected to the FPGA by a JTAG interface. The FPGA is configured to load its configuration automatically from the PROMs on startup. The output of the design flow (shown in Figure 4.5) is a bitfile containing the digital design ready for the selected FPGA. The second part of the design flow (shown in Figure 4.6) is needed to add an application to this bit file. The resulting ".mcs" files can be downloaded directly to the according PROM memories using Xilinx®' Platform Cable and a tool called iMPACT.

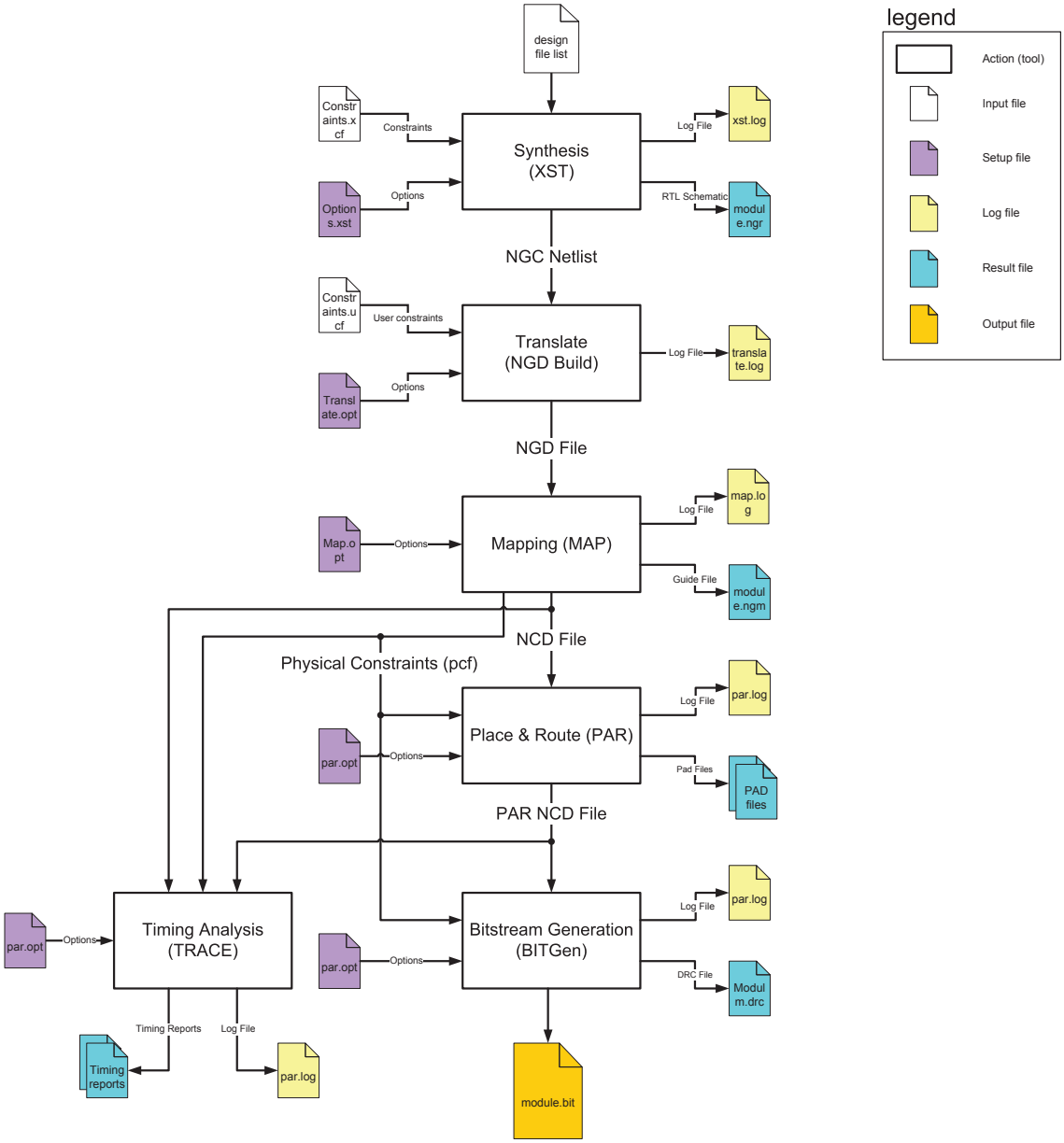

Figure 4.5: FPGA design flow

All tools targeting the FPGA are provided by the Xilinx® ISE® (Integrated Synthesis Environment). This tool suite on its own is able to generate the FPGA configuration files out of HDL sources. In the presented design flow only the tools needed to generate ".mcs"-files from an existing design are shown. With the ISE tool suite also several other tools are shipped which are not used by the presented design flow (e.g. FPGA editor).

A user constraint file (".ucf") is specified to determine the routing of signals from the design to FPGA pins. Some of the FPGA pins are already connected (e.g. to the seven-segment display) and can only be used for that purpose. The I/O pins are configured to work at 3.3 V levels. If the current into the pin is limited they are 5 V input tolerant. To generate output signals at this level additional hardware has to be used.

The principal advantage of testing a design in an FPGA is that every signal within the design can be observed by routing it to a pin. This mechanism was also used to verify the correct reading from PSROM by the core. The design has been simulated containing a certain program. The signals of data and address bus have been logged to a file. Afterwards the design has been downloaded to the FPGA containing the same program. Data and address bus have been routed to pins and recorded with a digital signal analyser. The created file has been compared with the log file written in simulation to verify the correctness of the read process.

In general an FPGA consists of three block types: Macrocells, Block RAM units and I/O cells. Block RAM units are combined to form PSRAM, PSROM, XDATA and IDATA memory for the design of the digital part. This means the program executed by the CPU is part of the FPGA configuration file. In Figure 4.6 the creation of a configuration file containing a program (hex-file) is shown.

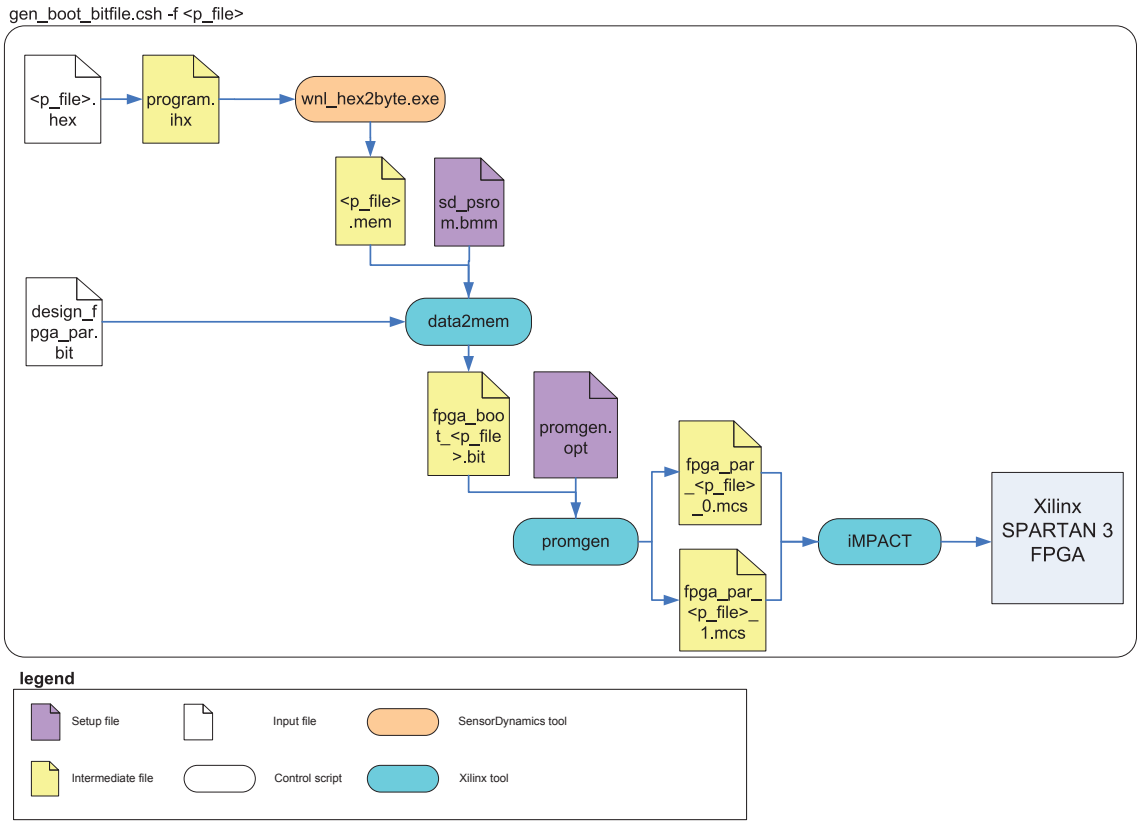

Figure 4.6: FPGA configuration file creation

The file *design_fpga_par.bit* file contains the digital design. The assignment of memories to FPGA Block RAM units is defined within the *sd_psrom.bmm* file. The creation and downloading process of the configuration files is time consuming. Since a bootloader for the ASIC had been developed anyway this bootloader has been integrated as PSROM content into the configuration file. So the digital part acts like the real ASIC on startup. On startup the bootloader checks if the MODE pin is high (selectable by a DIP Switch). If it is active the bootloader resumes, otherwise the bootloader switches to the program stored in PSRAM and starts its execution. If the bootloader resumes an application can

be downloaded via the bootloader host program (sd_boot). This program controls a USB-to-Serial interface to download the content of a ".hex"-file via 3WI interface. The data sent by the host program can also be stored into a file. This file can be used to simulate the functionality of the bootloader by a 3WI stimuli block (see Section 5.1). Also the execution of the downloaded program in simulation is possible.

On the FPGA the debug capability of the R8051XC2 could also be used. The debug environment is called Evatronix Application Debugging Support Environment (EASE) and includes the following modules (cf. [11]):

- Evatronix Debug Pod (EDP): Hardware USB-to-JTAG Interface connected via JTAG interface to the OCDS unit of the R8051XC2.
- Keil™ µVision® debug interface plug-in (EDIk51-3).
- EDIServer: Server software translating between EDIk51-3 and EDP.

The EDIServer runs on the computer connected via the EDP to the FPGA. The debugging itself is done by the Keil™ µVision® debug interface. The communication between Keil™ µVision® and the EDIServer is done via Ethernet. This is a useful feature if the debugging environment runs for example on a terminal server. In Figure 4.7 the debugging setup using EASE is shown.

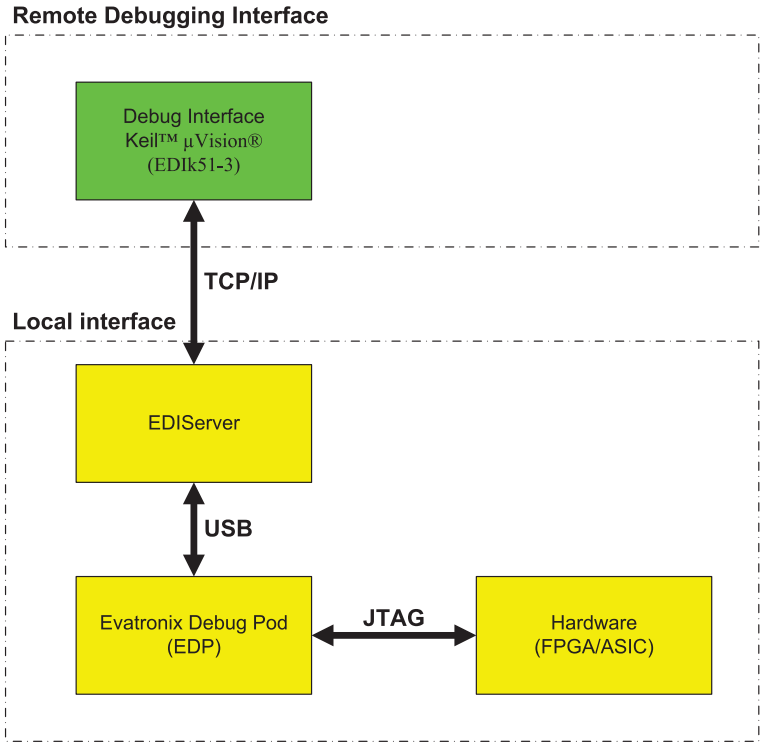

Figure 4.7: EASE setup (based on [11])

The debugging environment offers following debug features:

- Program execution control: "run", "halt" and "reset" functionality. Also breakpoints can be defined directly within the code displayed in the user interface. The

core has two hardware breakpoints which can be set during execution. This serves to implement features like the "run until" function. If this function is chosen the code is executed until the code line selected in the editor is reached. With hardware breakpoints also interrupt service routines can be stopped and debugged.

- Full access to IDATA, PSROM and XDATA memory spaces (includes also PSRAM).
- Partial access to internal processor registers (also the PC register).

With EASE debugging applications is easily operated. The decoupling between user interface and EDP via Ethernet offers various advantages if only one debugging setup is available for several developers. To use the debugging feature on the ASIC the IC has to be in the bootloader mode (MODE pin high) or the according bit in the *sys_ctrl* register has to be set. This is necessary to use antenna port 4 as debug interface.

4.4 ASIC Implementation

Similar to the FPGA design flow there is also an ASIC design flow to create a layout (physical implementation) of HDL files. In Figure 4.8 the design flow is shown.

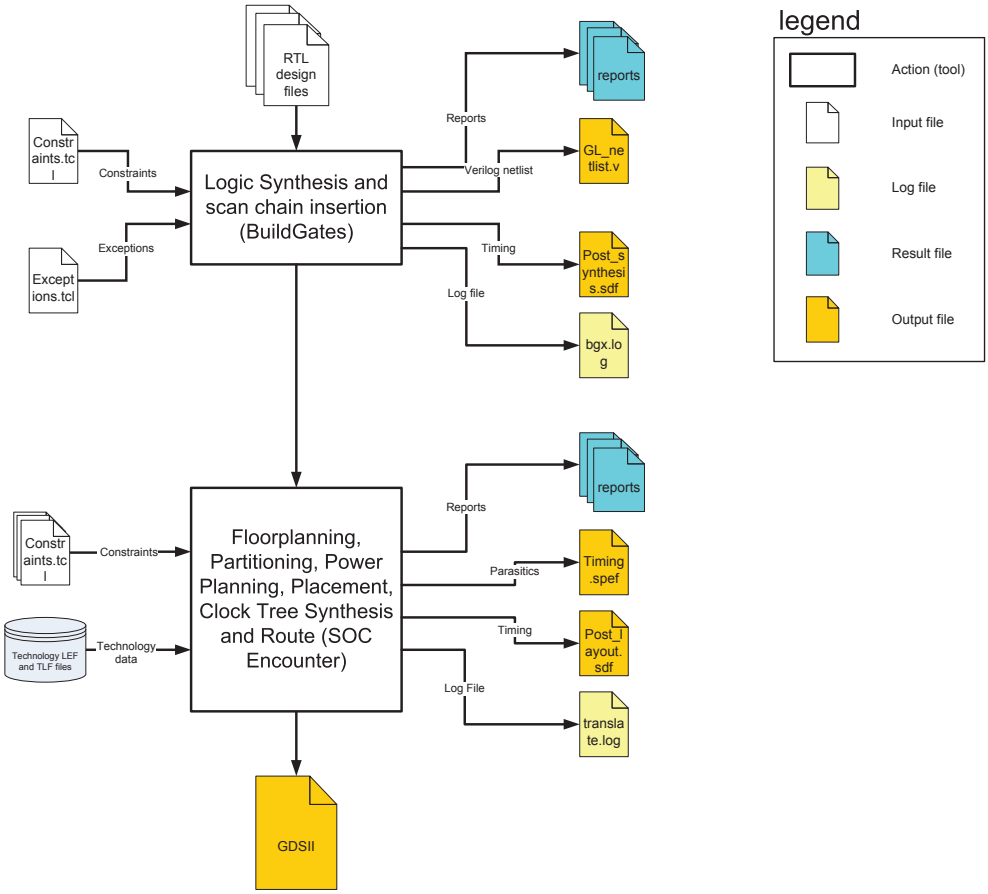

Figure 4.8: digital ASIC design flow

The output of the digital design flow is a Graphic Database System II (GDSII) file. It consists of geometric shapes and other information (e.g. labels) about the layout in

hierarchical form. So the design can be imported and checked in the analog design domain. Also the extracted netlist is imported so a LVS check of the whole system (digital and analog) can be performed. Before the LVS a DRC is done to check if all physical constraints are kept. In Figure 4.9 the layout of the whole digital part is shown.

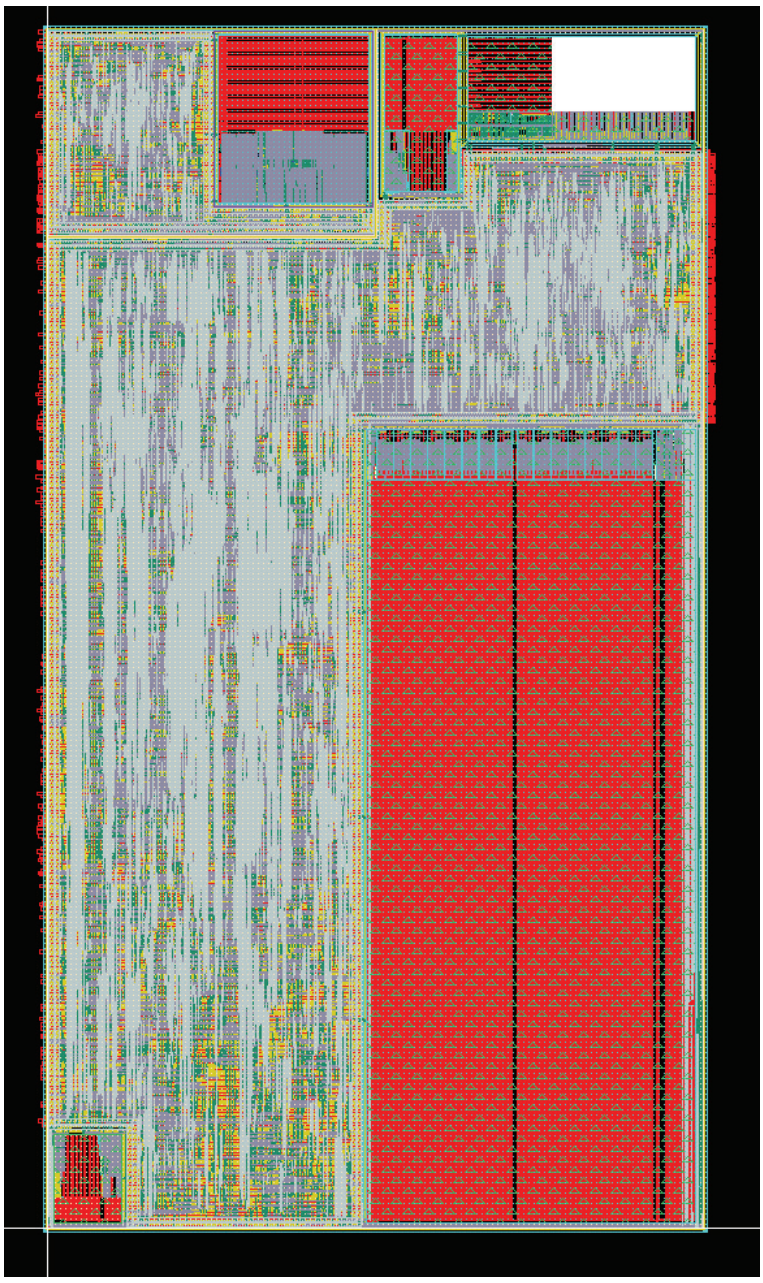

Figure 4.9: digital top IP layout

The back annotation of the developed design via the mapped netlist and the according timing information (Standard Delay Format (SDF) file) is very important to verify the correct functionality of the physical design. Since not every function of the implemented CPU Sub-System can be verified by using only hardware stimuli the next section provides a short introduction into the firmware architecture.

4.5 Software Development

In general low level drivers for SensorDynamics products are created as library. The library is shipped within a Software Development Kit (SDK) which also contains the necessary documentation. For the presented project two compilers are supported: The Keil™ C51 and the Small Device C Compiler (SDCC). For each compiler a separate library has to be generated.

The creation of the SDCC drivers library is controlled via Makefiles. The setup of the Keil™ μVision® project to generate the according library is done manually. All necessary driver files from the IP modules are linked automatically by a script to the library generation location. Since only one 8051 core has been used up to now the drivers structure did not offer the possibility to support different cores. Within the driver structure the most important file is the *sd_plattform_const.h*. In this file the most relevant definitions are set for the whole project. It also includes all necessary header files. In this file a certain keyword indicating the used core has to be defined (*USE_SD_core_IP*).

When compiling the driver files this keyword is evaluated and the correct statements and files for the defined core are used. To improve code compatibility between different cores a common header file defining the standard 8051 SFR registers has been introduced (*sd_8051_sfrs.h*). The core specific register defines (e.g. for the R8051XC2: *sd_evatronix_sfrs.h*) have been moved to the according IPs drivers directory. In Figure 4.10 the sources of the defines are shown.

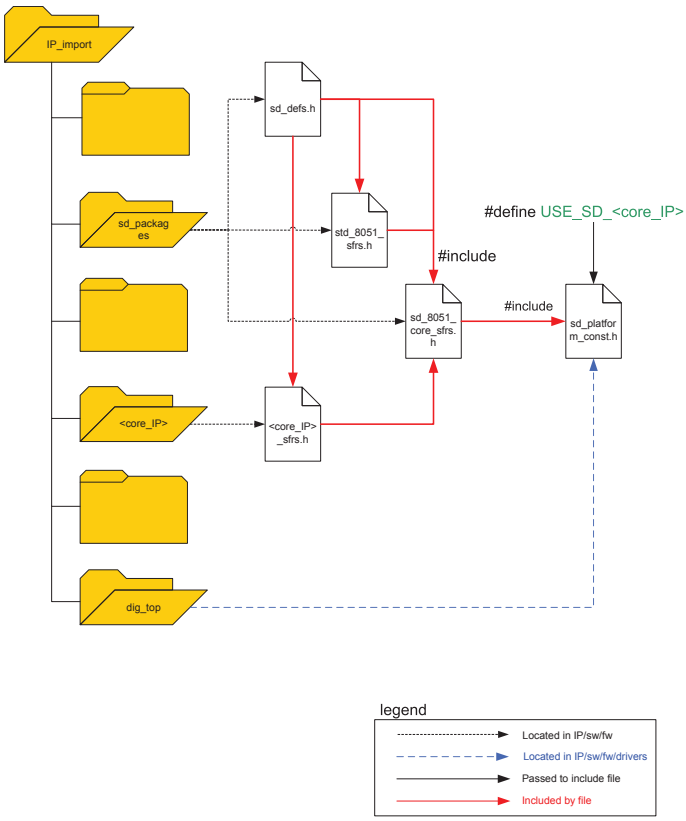

Figure 4.10: CPU core specific driver files source

In the *sd_plattform_const.h* compiler depended defines are also set up. Both compilers are able to generate executables from C-files. But there are differences in not standardised C-constructs like the definition of interrupt service routines or the macro which defines the usage of inline assembler within a source file. Each compiler defines a certain value which can be evaluated in the code. This feature is used to map compiler specific functions to general functions which are used in the code.
Besides the standard SFR register mappings also project depending registers and macros have to be introduced. Therefore the file *lftx_sfrs.h* exists. In this file the assignment of project dependent register names to SFR addresses is done. Also all driver header files from the sub-blocks are included here. This allows the same firmware macros to be reused in different projects if the according IP module is integrated into the design. If the user wants to implement software there are two different approaches depending on the environment used:

1. Keil™ μVision®: A project file with a standard development folder structure is created by the SDK. The user only has to add his files to the project and to define a name for the output file. After compilation of the source the hex-file can be downloaded. This approach is very simple but the user has to own a license.

2. SDCC: The user has to decide if he wants to use the compiler with an Integrated Development Environment (IDE) or via the command line. Alternatively he could also use Makefiles. The drawback of using SDCC is that the user has to define certain properties of the system (e.g. memory sizes) which could have a big impact on the systems performance. The advantage of this approach is that the environment is very flexible and free of charge.

For testing the digital part of the system firmware functions are also necessary. Therefore an approach using SDCC called via a script is used. Depending on the selected testcase the according C file is compiled and a hex-file is generated. This file is converted to a bin-file which is linked to the init file defined for the PSROM VHDL model. Testing the digital design is covered in detail by the next chapter.

Chapter 5

Verification

Test and verification are important topics in IC design. Testing covers all actions needed to measure the performance of an IC and to compare it with specification parameters or simulation results. Verification ensures the equivalence between different descriptions of the system, for example if the design matches its specification. Figure 5.1 shows the top-down design approach of an ASIC on the left side, the verification and implementation steps of the design are represented on the right side. The biggest advantage of a V-Model based approach is that discovering errors occours at the same level as they have been caused on the design side.

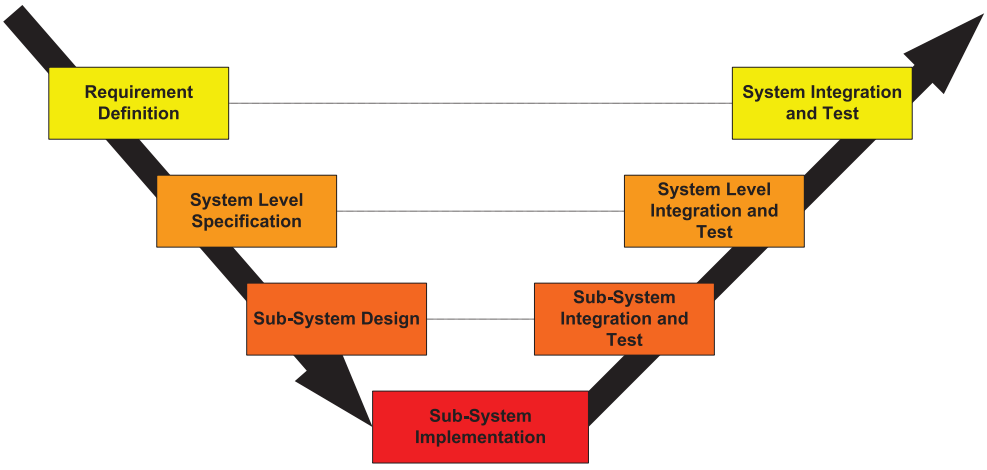

Figure 5.1: design verification V-Model

Verification techniques can be divided into two groups (cf. [10]):

Formal verification: These methods are also known as static verification. The equivalence between different models is proven mathematically.

Non-formal verification: This technique is also referred to as dynamic verification. Models are compared by applying a certain pattern (stimuli) and comparing the output of the designs. This approach needs a verification environment for each model (testbench and stimuli patterns).

Static verification is for example used to verify that the RTL model of the system behaves like the gate level netlist after synthesis or layout. Verification of the CPU Sub-System has been done by dynamic verification. The next section provides a closer look at the simulation topic.

5.1 Simulation

Within the SensorDynamics' Verification Flow the following simulation and test levels are defined to verify the IP is working as specified:

Sim0: A testbench checks internal signals of the block to verify that it works as intended. At the end of development the tests have to be self checking. This means the decision is done within the testbench if all specifications are met.

Sim1: A testbench is created which compares the IP with its high-level model. This testbench is intrinsically self checking.

Sim2: The developed IP is integrated into the system and tested via the according firmware drivers. The firmware tests are self checking based on data coherency.

Sim3: The system is mapped onto silicon or into an FPGA. The created firmware tests from Sim2 are reused to verify the functionality of the according IP and the system.

Sim0 and Sim1 are related only to the IP under test. Sim0 testbenches are also used to check complex sub-modules of the IP (e.g. sd_mem_mapper). Sim1 checks if the whole IP block works as intended. The testbenches at level Sim0 and Sim1 are located within the IP. Simulations performed on level Sim2 or executed tests on Sim3 are used to check if the IP block implements the desired functionality within the system. For simulations Cadence® Incisive (also known as NC Sim) is used.

Input for the simulator is a snapshot image generated by the elaborator from HDL libraries. This design flow allows the implementation of design and testbench in VHDL, Verilog or SystemC. Also mixing Verilog and VHDL is possible. The complete simulation process can be controlled by scripts written in Tool Command Language (TCL). Those scripts can also be used to force or observe signals within the design (or testbench). This feature makes it possible to apply external signals to a simulation (e.g. simulate APB register access of the CPU).

For system level verification a data structure is used which holds the according scripts and firmware files needed for each testcase. Besides the simulation of a testcase on its own it is also possible to run all testcases as regression tests. The output of the testcases is standardised, so all results can be summarised to obtain a pass/fail indication. For the CPU Sub-System the following modules have been tested at level Sim0:

- Module sd_cpu8051_wrapper: Two testbenches have been created for the wrapper module. A general testbench which is able to test each core via TCL stimuli and a special testbench using the Evatronix R8051XC2 core. This approach is necessary

to reuse existing testcases for the R8051XC2. With the above mentioned force and observe functions of the simulator the detectability of errors can also be tested (e.g. by error injection). For each test the results are written into a separate file. A summary containing the results of the executed tests is also generated.

- Module sd_parity_checker: The created testbench checks if the parity bit is calculated correctly and if the applied data is the same before and after the block. Besides checking if recognising data containing wrong parity bits works the error injection feature of this module to cause a parity error is also tested. The testbench is assertion based and shows as result a "[Success]" message if all tests are passed. If one or more tests result in wrong or unexpected values a "[Failed]" message is shown and the detected error is described.

- Module sd_mem_mapper: Since the functionality of this block is critical for the system a separate testbench has been created to verify this block. This testbench is described below as example for simulations at level Sim0 .

The VHDL testbench used to check the memory mapper module is shown in Figure 5.2. The stimulus and also the corresponding output compare values are generated internally.

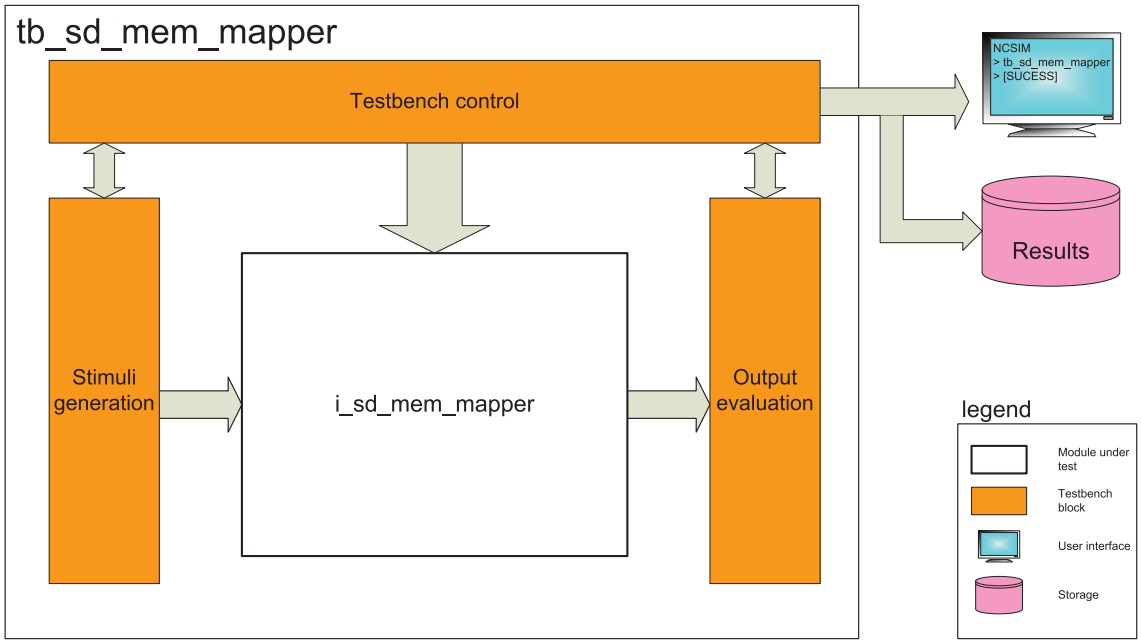

Figure 5.2: testbench of the sd_mem_mapper module

The testbench performs several tests for each configuration. Each of them shows information about its initial state. Printing messages is done via assertion statements. During the tests only errors are reported. After all tests are finished a global pass/fail information is shown. The stimulus block serves to provide values for address, data and control lines of the memories and the core according to the mode set up by the testbench control block. This block is configured by an array holding all necessary values to set up the module under test. All possible configurations for the actual memory mapper are again part of an array. This array is the main setup for the testbench control block. This easily allows adding of new configurations if the memory mapper block is extended.

The according outputs are calculated on the values stored within the array. If the calculated values do not fit the output of the module under test the according error is shown and the next test is executed. The following tests are performed by the testbench (for each configuration):

- Address range: Addresses the complete range of each memory type (program or data memory).
- Illegal address: Determines if invalid addresses are recognized correctly (program and data memory).
- Prohibited memory control signals: Processes running parallel to all other tests to check if only valid memory signals are applied at the same time (data memory enable signal and program memory enable signal are never allowed to be active at the same time).
- Configuration range: Memory setups which are valid (e.g. no PSRAM) but not allowed in certain configuration (e.g. Mode 1, PSRAM as program memory) are tested to verify if the sd_mem_mapper block rises the correct assertions.

With that kind of test setup all important modules within an IP are checked. Simulation at level Sim1 work similar at IP level. Since IP blocks cover more complex functions the stimuli and result data are not created within the testbench anymore. The data is rather imported from a high-level model. For example to verify the functionality of the SINC IP within the DSP part of the system a Matlab® model of the complete Sigma-Delta ADC is used. The output bitstream of the ADC is written to a file as well as the output of the SINC filter. The testbench used to verify the implementation of the SINC-filter imports the bitstream and applies it to the module under test. The output of the filter described in Matlab is compared with the output of the VHDL description. If the outputs are matching the test passes. This approach highly depends on the quality of the Matlab® model.

For the complete CPU Sub-System no high-level model exists but the Evatronix R8051XC2 IP is shipped with a behavioural model used for its testbench. In order to reuse this model and to verify the additional functions of the CPU Sub-System three Sim1 testbenches have been created:

tb_sd_cpu8051f_top: Basic testbench which can be used for all cores. Within this testbench only clock and reset signals are created. All cores can run a selected program loaded into the PSROM. Additional stimuli can be created via TCL scripts or using the command line of the simulator.

tb_sd_cpu8051f_top_eva: This testbench reuses some of the verification modules used in the testbench of the R8051XC2. Basically a behavioural model of the 8051 is instantiated with models of the according memories. The models of the memories are also connected to the sd_cpu8051f_top (all memories are connected externally). So both models are executing the same code. A comparator block compares the output of the model with the output of the CPU Sub-System. The signals which are compared by the comparator but not available at the top of the sd_cpu8051f_top

are analysed using the *nc_mirror* function of the simulator. This function is available in VHDL by including the *ncutils* package and connecting signals in the hierarchy with signals at another level. So also the correct internal behaviour can be verified. Also the assignment of values to signals in the hierarchy is possible (via the *nc_force* function). In Figure 5.3 the structure of this testbench is outlined. The functions mentioned above are also available in the command line of the simulator. This allows for example error injections to check if the testbench covers all possible faults.

tb_sd_cpu8051f_top_func: This testbench instantiates the module under test with the configuration used in the EMU version of the ASIC. At startup of the testbench BIST is started and the result is checked. Also an additional test module has been added which can be controlled via P1. The test module is connected to the interrupt lines of the sd_cpu8051f_top. By writing commands to port P1 single interrupt lines can be activated as well as all lines at the same time. This allows checking the interrupt functionality by firmware only. This testbench also checks the behaviour of the core if interrupts occur at the same time.

A different setup is necessary if the Evatronix testcases should be executed. Therefore two additional simulation execution scripts have been created (one for the wrapper module and one for the CPU Sub-System). The parameters to control those scripts are:

- *-testpath*: Sets the path to the according Evatronix testcase and result directory.
- *-input*: Defines the TCL-file which determines the tests to execute.

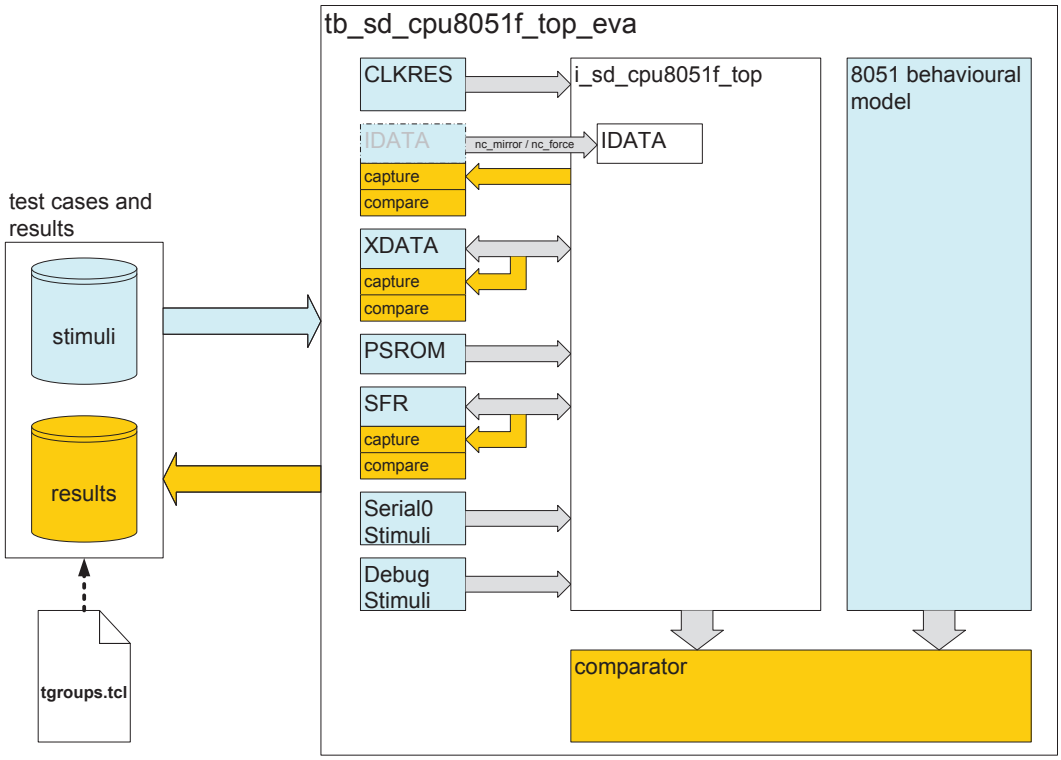

Figure 5.3: Sim1 testbench of the CPU Sub-System

Since all Sim0 and Sim1 level testcases are located within the IP the simulation execution script (run_sim.csh) has also been adapted to make the testbench to execute selectable. The following parameters can be passed to the script:

- *-rom /path/to/file*: The given path points to the file which should be loaded into PSROM (default: code.bin). Those files are created by a script from given hex-files.
- *-flash /path/to/file*: This path defines the file containing the FLASH memory content (default: sd_flash_init.mif).
- *-psram /path/to/file*: The selected path defines the file containing the content to be loaded into PSRAM (default: code_psram.bin).
- *-sim_mod "modulename"*: Defines the module to be simulated. This allows the selection of the testbench to use for all Sim1 and Sim2 level simulations described above.

Verification at level Sim1 ensures that the IP works as expected. They do not necessarily ensure the correct working of the IP within the system. Therefore simulations at system level (Sim2) and verification of an physical implementation (Sim3) have to be performed. Since Sim2 and Sim3 level verification operates on self-checking firmware tests it makes sense that the same data source is used for both levels. In Figure 5.4 the basic architecture of the Sim2 testbench written in Verilog is shown.

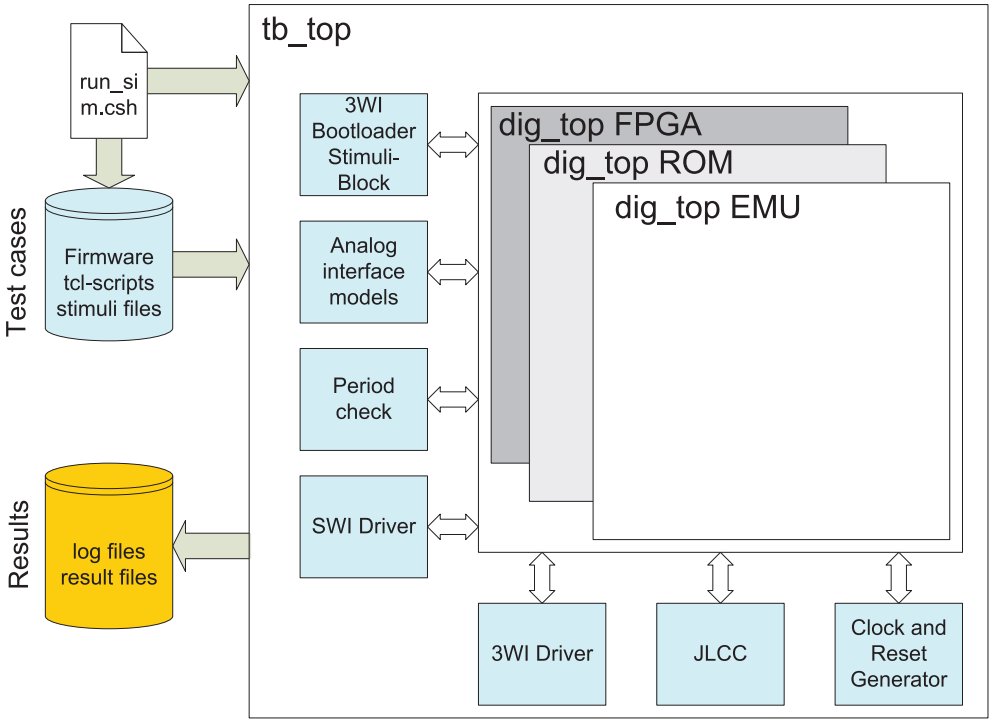

Figure 5.4: digital top Sim2 testbench

The testbench is written in Verilog because it easily allows to switch between different implementations of the design (FPGA, ROM version and EMU version). The stimuli blocks allow to perform different checks:

3WI Bootloader Stimuli-Block: This block can be used to virtually download software into PSRAM via the bootloader. It implements the same functionality as the USB-to-Serial converter used to download programs to the FPGA. This block was also used to verify the correct functionality of the bootloader firmware during development.

Analog Interface Models: This block covers all inputs and outputs of the analog part of the system. It creates stimuli patterns for example the output bitstream of the ADC. So the digital signal processing path of the DMSP can be tested. The associated firmware checks if the interrupts generated by the DMSP are recognized correctly.

Period Check: This block checks if the associated input (CLKOUT, IRQ, GPIO12, GPIO5 or MODE) toggles with the correct period. Toggling the pin is done via firmware.

SWI Driver: This module writes and reads certain patterns from the SWI. It is self-checking and shows pass or fail on exit. The module has to be enabled if the test should be performed.

3WI Driver: This block is also connected to the 3WI interface of the digital part. It is used to verify correct functionality of the interface. Again, this testmodule is self checking and has to be enabled.

JLCC: This block performs a JTAG Like Control Chain (JLCC) integrity check on the according digital JLCC chains and multiplexer. This module is self-checking and must be enabled.

Clock and Reset Generator: This module is used to provide clock and reset signals to the digital part.

Besides the test modules shown in Figure 5.4 no other stimuli modules are necessary. Everything else is checked in firmware with optional TCL-files. Therefore a hash table assigning a testcase name to a firmware source file and TCL-file (if necessary) exists. With the simulation execution script (*run_sim.csh*) the following choices can be made:

- Testcase to execute
- Design selection: FPGA-, EMU- or ROM version
- Simulation level: RTL, pre- or post-layout gate level simulation

A separate list holds the names of testcases to be executed if a regression test has to be performed. Therefore the simulations are started in batch mode (*run_sim_batch.csh*) and the output of each testcase is defined. So all results can be parsed and a global pass/fail indication can be made. The next section deals with running the testcases on an FPGA board.

5.2 FPGA Verification

As mentioned above testcases at level Sim2 are reused to test the FPGA implementation of the design. The external circuit within the simulation is emulated by forcing signals via TCL commands. Those signals have also to be applied at the according FPGA inputs. For example to start the bootloader the MODE pin must be held high. At level Sim2 this is simply done by defining *force MODE {1b'1'} after 0 ns* in the TCL script. On the FPGA board the mode pin has to be routed to a controllable input. For the FPGA design this input is routed to a DIP Switch which is able to apply a logic one to start the bootloader. To execute all testcases implemented for Sim2 the patterns defined by TCL commands have to be applied as signals to the respective ports of the FPGA.

Also the firmware used by the testcases has to be compiled again. This is necessary because the messenger subroutine has to be mapped from the simulation-only messenger block to the UART interface. So all messages can be received by a terminal program. Additional to the tests defined at level Sim2 some tests have been added to check if important functions are working correctly (e.g. 3WI interface). Some of the additionally developed testcases used outputs of the design (e.g. GPIO5, CLKOUT) to stimulate inputs (e.g. S0). So the ability of the design to measure the duration of doorhandle low pulses could be tested as well as the doorhandle short recognition. The results of those measurements have been sent via the SWI. Since the SWI operates at 3.3 V on the FPGA a UART-to-RS232 converter has to be used. For this reason a MAX3232 converter is used. Three versions of the FPGA design have been released during the development phase.

At the very beginning only the CPU Sub-System and its peripherals were integrated. This version was also used to check and verify the bootloader. Version two and three of the design included nearly the complete digital part (variations are listed in Section 4.3).

The FPGA design also served as development platform for the evaluation Graphical User Interface (GUI). This graphical user front end written in Labview® is used in combination with the evaluation firmware to access SFR and APB registers as well as JLCC chains. The firmware and the user interface communicate via the 3WI interface. It is also possible to add functions to the firmware. By reading the compiler functions mapping file they can be executed via the GUI by setting the function pointer to a certain address. Parameters and results of the executed functions are also controlled via the user interface. The evaluation GUI is used to check if register settings are working. If the applied setting is correct the register values are used to define the internal configuration of the ASIC for a certain testcase. For example with the user interface the analog antenna driver blocks can be enabled via JLCC. The according SFR (*ant_n_fet_ctrl*) is used to switch the N-FET on and off. At the according N_x pin the voltage can be measured. On the FPGA the high level of this pin would be 3.3 V, on the ASIC this parameter ranges from 4 to 5 V. This internal setup is used to implement automatically executed testcases for the ASIC. The parameters which have to be measured by those testcases are defined within the Design Verification Matrix (DV Matrix). The next section deals with the tests performed to verify the correct functionality of the ASIC.

5.3 ASIC Verification

Although FPGA and ASIC implement nearly the same digital part there are important differences between the designs:

- Interfaces are working on different levels
- The ASIC system clock runs at 32 MHz (external crystal required)
- Analog blocks are integrated: So they have to be configured correctly if the FPGA testcases are to be repeated
- Additional analog stimuli signals are needed to check all integrated functions

Due to those reasons an Evaluation Printed Circuit Board (PCB) is used to convert interface signals to lower levels (e.g. SWI from VDD to 5 V) and to provide the creation of analog signals. Evaluation describes the determination of fundamental electrical characteristics of an IC. The board is designed to be connected to a National Instruments PCI eXtensions for Instrumentation (PXI). This is a platform offering the possibility to measure and control analogue and digital signals via boards attached to the computer by the Peripheral Component Interconnect Bus (PCI Bus). Besides the connection to the PXI it is also possible to connect the PCB via a USB-to-Serial converter and the SWI interface to a computer. This is necessary to rerun existing testcases. Some of them have to be modified because the configuration of analog blocks had not been implemented before. This means for example if the testcase which checks if the recognition of a short to ground at a doorhandle pin works should be executed on the ASIC the appropriate analogue blocks have to be configured correctly. If for example the signal is not processed by the digital part because the analog part is switched off the testcase will not finish successfully. Configuring this part of the system is for example done via an additional JLCC configuration. The existing testcases can be modified because setting up the analog blocks does not affect the results in simulation or on the FPGA (as long as they only belong to analog blocks). Reusing testcases starting from the first simulations to the final ASIC ensures the specified functionality defined in the specification has been met by the implementation.

Besides checking if all existing testcases are working on the ASIC it is also necessary to evaluate the IC. It has to be checked if all important parameters of the circuit meet their specifications in all specified operating conditions. To do so a lot of signals have to be generated and measured. For this reason the Evaluation PCB is connected to the PXI. The measurements are controlled by the General Verification Platform Environment implemented in Labview®. The environment allows the execution of testcases (also implemented in Labview®) defined within the DV Matrix. As mentioned above all important parameters to be checked are defined within the DV Matrix. Also the tolerance of those parameters is specified there. So the measured values can be immediately compared with their specification and a pass/fail information is evaluated. The detailed results of the measurements are stored in a log file. The framework also offers the possibility to execute the testcases at defined temperatures. So all parameters are tested at the corners of the defined operating temperature of the ASIC (-40 to 85 °C) too.

Finally it is to say that verification and evaluation of an ASIC causes a lot of effort and usually consumes more time than its implementation. FPGA prototyping makes testing digital designs easier but also adds additional complexity to the design flow. The digital design has to be implemented in a portable way and a good knowledge of the implementation is necessary to avoid common design failures.

Chapter 6

Conclusion

This chapter presents achieved results, open items and provides suggestions for future work. The obtained 8051 core has been modified to fit into the SensorDynamics IP module structure. All necessary compile and simulation scripts have been adapted to allow the use of the new core without additional effort in all projects based on the existing design flow. Furthermore the CPU Sub-System developed herein has been integrated into the SensorDynamics' LF Transmitter IC. The engineering samples of the ASIC have already been produced and evaluated. It has been shown that the functionality proven by simulation and on the FPGA fits the final ASIC implementation. Also the bootloader and the evaluation firmware are working on the IC as expected although they have been developed and tested at the FPGA implementation of the design. The ASIC also passed all verification testcases created during the design phase and for the FPGA without major problems because the necessary signal level shifting has been considered in the design of the Evaluation PCB.

Due to the performance increase the Evatronix R8051XC2 is going to be used in future projects. It offers an easy way to gain performance with a minimum effort in changing existing firmware procedures and development flows. Although the integration of the CPU Sub-System for the presented project has been successfully finished there are some major points to cover if the design is reused in other projects. First of all it is important to support also FLASH memories. This means the FLASH controller has to be modified to access the memory within a single cycle so that it could by used by the R8051XC2. That fact has also to be considered for the cache block. The further improvement of the functionality of the memory mapper is also an interesting topic. Following additional memory configurations could be useful to extend the existing modes:

Mode 2, Different interrupt service routines: In this mode XDATA memory replaces the first part of the program memory. This allows on line mapping of interrupt vectors to different interrupt service routines in ROM versions of an IC.

Mode 3, Variable interrupt service routines: This mode allows the placement of PSRAM after the PSROM. If the interrupt vectors in PSROM are pointing to PSRAM addresses the interrupt service routines could be changed on line.

Mode 4, Program memory only: Map all memories to the program memory space: XDATA before PSROM followed by PSRAM. This has the advantage that interrupt service routines could be remapped (setting the target address in XDATA) or replaced (modifying the code in PSRAM).

An important topic is the verification of the functionality of the CPU Sub-System if the DDC8051 is selected. Due to the tight time schedule this task could not be performed during development. The main question is whether the core is relevant for future projects or not. If not, the wrapper generally should be replaced by the R8051XC2. Since this scenario has been considered at the very beginning of the work this would not cause that much effort. Also the implemented firmware functions and macros for the R8051XC2 are not complete and should be extended.

From a digital point of view, it has been demonstrated that a working and well performing IC is based on a detailed and sophisticated design as well as on the verification of the implemented hardware in simulation and on an FPGA.

Appendix A

Appendix

A.1 Standards

1076-87 IEEE 1076-1987 IEEE Standard VHDL Language References Manual Superseded by 1076-1993 Edition

1076-93 IEEE 1076-1993 IEEE Standard VHDL Language References Manual Superseded by 1076, 2000 Edition

1076-00 IEEE 1076-2000 IEEE Standard VHDL Language References Manual Superseded by 1076, 2002 Edition

1076-02 IEEE 1076-2002 IEEE Standard VHDL Language References Manual

1076.1-07 IEEE Std 1076.1-2007 - IEEE Standard VHDL Analog and Mixed-Signal Extensions

1076.2-96 IEEE 1076.2-1996 IEEE Standard VHDL Mathematical Packages

1076.6-04 IEEE 1076.6-2004 IEEE Standard for VHDL Register Transfer Level (RTL) Synthesis

1149.1-01 IEEE 1149.1-2001 - IEEE Standard Test Access Port and Boundary-Scan Architecture

Bibliography

[1] ARM. *Cortex-M0 Technical Reference Manual*, January 2009.

[2] G.D. Asensi, J.S Gomez-Diaz, J. Martinez-Alajarin, and R.R Merino. Synthesis on programmable analog devices from VHDL-AMS. *Electrotechnical Conference, 2006. MELECON 2006. IEEE Mediterranean*, 1, 2006.

[3] P. Ashenden. *The Designer's Guide to VHDL*. Morgan Kaufmann, 2006. ISBN 0120887851.

[4] C. Browy, G. Gullikson, and M. Indovina. A top-down approach to ic design. online, `http://www.indovina.us/~mai/a_top_down_approach_to_ic_design.pdf`, 1997.

[5] Houghton Mifflin Co. *Webster's II New College Dictionary*. Houghton Mifflin Harcourt, Boston, 2004. ISBN 0618396012.

[6] Evatronix Electronic Design Department. *R8051XC2 Design Specification*, April 2010.

[7] A. Doboli and R. Vemuri. Behavioral Modeling for High-Level Synthesis of Analog and Mixed-Signal Systems From VHDL-AMS. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22, 2003.

[8] Intel. *MCS-51 8-BIT CONTROL-ORIENTED MICROCOMPUTERS 8031/8051*, January 1980.

[9] F. Iozzi, S. Saponara, A. J. Morello, and L. Fanucci. 8051 cpu core optimization for low power at register transfer level. *2005 PhD Research in Microelectronics and Electronics - Volume II*, 1:178 – 181, 2005.

[10] W. S. Encinas Jr. and C. A. Dueas M. Functional Verification in 8-bit Microcontrollers: A Case Study . 2010.

[11] Evatronix LTD. *EASE-8051*, January 2010. Evatronix Application-debugging Support Environment for Evatronix 8051 Series Microcontrollers.

[12] R. Narayanan, N. Abbasi, M. Zaki, G. Al Sammane, and S. Tahar. On the Simulation Performance of Contemporary AMS Hardware Description Languages. *International Conference on Microelectronics, 2008.*, 1:361 – 364, 2008.

[13] A. Nunez-Aldana, N. Dhanwada, A. Dobola, S. Ganesan, and R. Vemura. A METHODOLOGY FOR BEHAVIORAL SYNTHESIS OF ANALOG SYSTEMS. *Symposium on Mixed-Signal Design, 1999. SSMSD '99.*, 1:162 – 167, 1999.

[14] S. Dennis O. Gerler, B. Janger. SensorDynamics AG - Digital Design Guide. SensorDynamics internal digital design guide, 2007.

[15] A.C. Parker. Automated Synthesis of Digital systems. *Design and Test of Computers, IEEE*, 1:75 – 81, 1984.

[16] F. Pecheux, C. Lallement, and A. Vachoux. VHDL-AMS and Verilog-AMS as alternative hardware description languages for efficient modeling of multidiscipline systems . *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24, 2005.

[17] R. Saleh, S. Wilton, S. Mirabbasi, A. Hu, M. Greenstreet, G. Lemieux, P. P. Pande, C. Grecu, and A. Ivanov. System-on-Chip: Reuse and Integration . *Proceedings of the IEEE*, 94, 2006.

[18] G. Schultes and O. Gerler. Circuit specification LF base station antenna driver. SensorDynamics internal chip specification, 2010.

[19] R. Seepold and N. M. Madrid. *Virtual Components Design and Reuse.* Springer, Berlin, 2000. ISBN 0792372611.

[20] Keil Software. *Cx51 Compiler, Optimizing C Compiler and Library Reference for Classic and Extended 8051 Microcontrollers*, September 2001.

[21] Synopsys. *CoCentric® SystemC™ Compiler - RTL User and Modeling Guide*, August 2001.

[22] S.K. Venkataram. *Comprehensive Advanced Microprocessor and Microcontroller.* Laxmi Publications, New Delhi, 2005. ISBN 8170083109.