

Chapter 4

Hardware Design of Individual Components

The previous chapters described the basic principles for the application channel coding. Before proceeding to advanced channel coding techniques and its possible hardware realization we will introduce in this chapter the basic steps for a hardware design. An entire receiver is large system and comprises many different functionalities. Combining all of them on a single die yields a so called System-on-a-Chip (SoC). The SoC design requires the knowledge from system specification down to hardware partitioning and refinement. However, every SoC is partitioned in smaller functional blocks which can then be developed individually on component level. This is especially true for the channel decoder which is just one single component in a larger system. The same hold for e.g. demodulator, source encoder or decoder and so on. The advantage of designing components individually is that typically the functionality is restricted and can be well described. In this chapter we first revise (Sect. 4.1) the design flow for a single component and show the different design constraints which are posed either by the communications domain or the hardware domain. Note, that the design flow shown here is no general hardware design flow. It is restricted to communications specific constraints with respect to the introduced base band processing components. Memories are an extremely important part for the entire SoC and for each individual component as well.

For every hardware designer an understanding of the data access patterns and their impact on the choice of the memory architecture, as well as the resulting area and power consumption is mandatory. The basic parameters for the instantiation of memories for the design of individual components are described in Sect. 4.2. The following Sect. 4.3 exemplary shows the design of a simple interleaver component, and how it is heavily influenced by the constraints of instantiated memories.

4.1 Design Flow

A generic design flow for individual component design is shown in Fig. 4.1. Every component like the discussed channel decoder in this manuscript is embedded in a larger system. The design flow for a full system is not explained here, the starting point is the isolated functionality like channel coding. Note that many of this small functional blocks exist in a system design which have to be extracted by a system engineer using the divide and conquer method. Every component design flow has input constraints and certain quality measures. Here, the flow starts at the algorithmic design level and ends in a refined model on the so called register transfer level (RTL). This design flow is tailored to components which are embedded in communications systems. For other applications different constraints especially for the quality of service will exist.

Typically we can distinguish between an algorithmic design space exploration and a hardware design space exploration. Different design requirements and quality assessments exist for different levels of the design.

4.1.1 Algorithmic Design Space Exploration

Quality of Service

The quality of service (QoS) is the expected error rate with respect to a given signal-to-noise ratio. This QoS is given by a communications standard or has to be defined by a system engineer. A defined error rate could be: at most 3 bits are erroneous out of 10,000 decoded bits. For every utilized algorithm we have to track the communications performance.

Communications Performance

At each level during the design process we have to ensure that the achieved communications performance meets the given QoS design requirements. The communications performance as an assessment of the quality means e.g. the measured frame error rate with respect to a certain noise level of a channel. Typically the communications performance cannot be evaluated analytically. Thus so called Monte Carlo simulations are performed: the entire transmission chain is model (e.g., using Matlab or C++) and the transmission of information is simulated until the observed communications performance is statistically stable. For every algorithmic transformations the impact on the communications performance has to be checked.

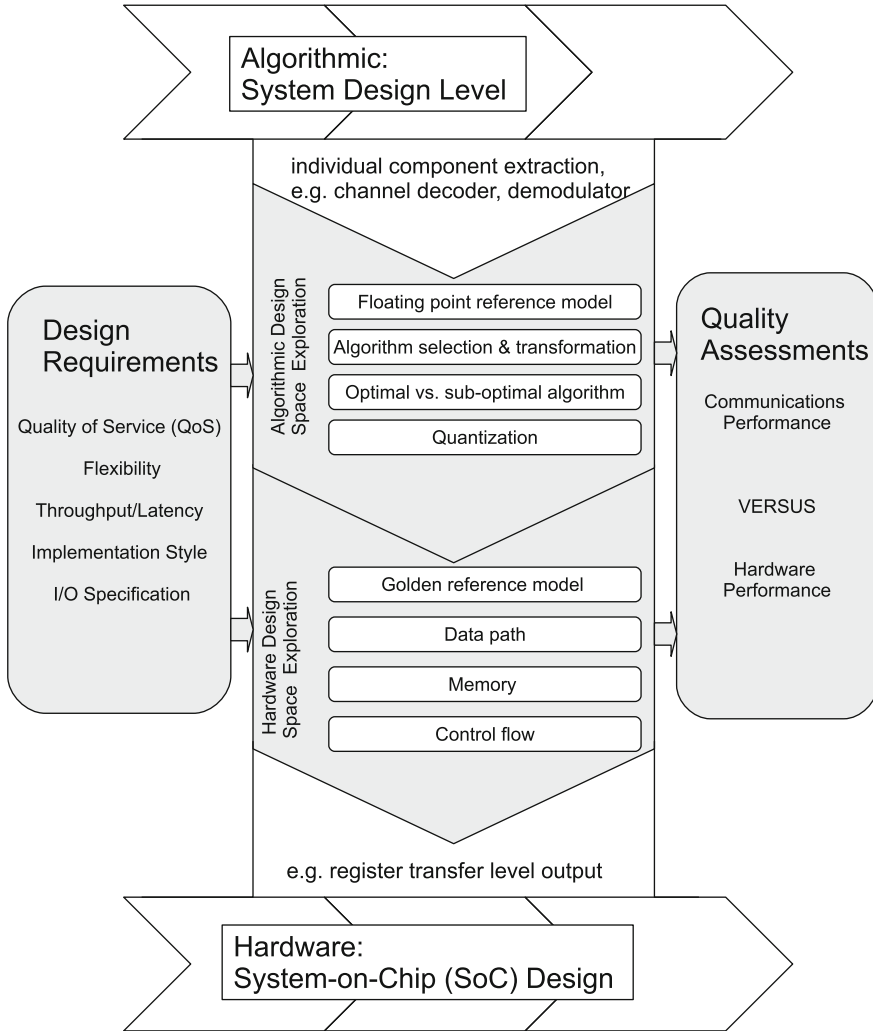


Fig. 4.1 Generic design flow: From component algorithmic extraction to register transfer level

Floating Point Reference Model

The floating point reference model is the time discrete model of the algorithm. It is not important in which language this model exist, e.g. C, C++, Matlab. Each value is represented by sufficient bits (e.g. 32 or 64 bits) to achieve the best achievable communications performance. In many cases this perfect model is too complex for an efficient hardware realization. Thus, sub-optimal algorithms have to be derived which can be implemented in hardware. The degradation of each algorithmic transformation has to be checked with respect to an optimal floating point reference model.

Algorithm Selection & Transformation

The floating point reference model defines one correct realization, often different basic algorithms exist to achieve the perfect performance as well. For example to decode a given channel code we can implement the algorithm in probability domain or in log-likelihood domain or we can change the basic flow of the algorithm e.g. depth first or breadth first algorithm. The algorithm type selection is the first important step towards a hardware realization. This decision step is for sure not a simple one, since a wrong decision at this point may lead to complex or cumbersome hardware realization. At this point the communications performance has to match that of the floating point reference model.

Optimal Versus Sub-optimal Algorithm

After defining the algorithm type the first approximations within the algorithms are introduced. Goal of this approximation is already to limit the complexity of the algorithm. This is a very important design step and requires already experience about a possible hardware complexity. This step is pretty often combined with the floating point to fixed point conversion.

Quantization

During the floating point to fixed point conversion many algorithmic manipulations take place. The communications performance has to be tracked whether it still fulfills the QoS requirements. This step is not a classical quantization step where the quantization noise can be measured, rather it in-cooperates many transformations with respect to sub-optimality of the algorithm. The overall effects can only be measured by Monte Carlo simulations by comparing the final performance degradation with respect to the floating point reference model.

4.1.2 Hardware Design Space Exploration

The fixed point model defines all bit widths of quantized data: input bits, output bits, and all internal values. However, the fixed point model does not define further implementation details. Many further exploration steps have to be performed until a register transfer model will be obtained. A hardware design can be decomposed into three major blocks the data path, control logic, and the memory structure. Different design requirements exist as well for the hardware design space exploration which are shortly discussed in the following.

Implementation Style

As mentioned with the term ‘implementation styles’ we refer to the choice of different hardware platforms and thus different possibilities with respect to flexibility, power consumption, design time, re-usability, and costs. Different possibilities for the implementation style were already shown in Fig. 1.2.

The choice of the implementation style is a trade-off between performance, power, programmability, and costs. It is typically made even before the beginning of the entire component design flow, since the platform will influence each design decision significantly. Furthermore, the clock frequency is given, or at least, which clock frequencies are available within a larger SoC.

Throughput and Latency Specification

The throughput and latency numbers for the entire communication system is defined in the communications standard. The latency of an individual component has to be derived from the overall system. Each component a dedicated respond time has to be allocated. Starting from the latency and a given clock frequency one can derive the required parallelism of an architecture. The goal is to meet the required latency constraints and the throughput design requirements. However, the interface has to be taken into account as well, as this can influence the final architecture.

Input and Output Interface Specification

The input and output (I/O) interface specification pose further constraints for the hardware designer. It is very important to define the interfaces early during the design space exploration since latency and throughput may be effected by the interface. The interface should be specified in the very beginning of the hardware design. However, in practice due to new requirements this definition changes frequently and thus poses a time consuming challenge for the component designer.

Golden Reference Model

The golden reference model is way more than a simple fixed point implementation. The fixed point model checks the statistical correctness with respect to e.g. communications performance. The golden reference model has to be statistically correct, however, it has to provide a bit accurate behavior with respect to a hardware model. Each internal variable should be modeled as well the correct bit width. Furthermore, often a so called cycle accurate model is required which gives detailed timing information, especially for the interface timing. The timing behavior is important for the system validation and for hardware debugging.

Data Path

The data path is the heart of the processing kernel at which most of the arithmetic operations are performed. Serial or parallel data paths may co-exist within a design. Their basic task is the evaluation of a function $output = f(input)$, where the input and output data may be scalar data or vectors. The parallelism of the data path depends on the required latency/throughput definition. For the design of the data path one has to take care to meet the target clock frequency constraint.

Memory Structure

Memories are important building block for a designer. Typically for component design only static random access memories are utilized which have fast access times. From system point of view, in communications systems up to 70% of the overall power consumption is due to the memories. Thus, it is of increasing importance that algorithm design takes the memory hierarchy into account. Memory hierarchy means the grouping and organizing of smaller blocks of memory. The memory access, i.e. reading and writing, has to be organized by a controller. Memories are explained in more detail in Sect. 4.2.

Control Flow

Passing the correct input data at the correct time to the data path is one task of a controller. Furthermore it has to organize the message transfer between different data paths and storage devices. The overall control flow of an individual component defines the sequence of individual task which are mandatory to obtain the desired functionality. The control can be either as a software running on a small micro-controller (CPU) or as a dedicated controller instance, depending on the complexity and the task of the controller.

Hardware Performance

The typical performance measures of a certain task with respect to its hardware realization are area and power. Area and power consumption have to be linked with the achieved throughput and with the achieved quality of service. Furthermore the flexibility of the implemented algorithm has to be taken into account. Thus, it is a multi-dimension performance measure with conflicting goals, e.g. lowest power consumption versus highest flexibility. The difficult task of communications performance versus VLSI performance will be addressed in Chap. 9.

4.2 SRAM Memories

Memories are important building blocks. The task of the memory is to store and provide information for different agents which may interpret or manipulate the information. There exist many different types of memories, and each type has different characteristics regarding functionality, accessibility, area efficiency, power efficiency, and of course implementation cost. We can distinguish two major types of memories, volatile memory and non-volatile memories. Volatile memories lose their information when the power is switched off, non-volatile not. Non-volatile memories like read only memories (ROM) or Flash memories are utilized to store long-term persistent data. Flash storage devices cannot be utilized for high performance applications with rapidly changing data content.

In every system-on-chip volatile memories are instantiated with its major storage type called random access memories (RAM). Random access means we can either store or read a data in any order, typically within a given maximum, deterministic time frame. There exist two major classes of random access memories. The static random access memories (SRAM) and the dynamic random access memory (DRAM). Table 4.1 compares the major differences between these two type of memories.

In summary of the table we can extract one major reasons why only SRAMs are possible for the design of an outer receiver. The mandatory bandwidth and a ‘truly random’ data access can only provided by this type of RAMs. Thus, in the following we only consider design aspects with respect to SRAMs.

Table 4.1 Major differences of SRAMs and DRAMs

	SRAM	DRAM
Technology process	Identical CMOS process with respect to logic and thus can fabricated on the same die	Dedicated process for DRAMs
Access time	Fast and constant access time, independent of the access pattern	Access time heavily depend on the history of access, thus only a worst case time can be specified
Access protocol	Simple access protocol, which means that no special controller is needed	Special controller mandatory
Area and size	SRAMs have typically a limited size ($\sim 8Mbyte$) due to cost and fabrication reasons. One cell to store a bit is typically composed of 6 to 10 transistors, depending on the specific SRAM type	One cell to store a bit is realized by utilizing the capacity of one transistor. Thus, the cell size is up to $\sim 10x$ smaller compared to that of a SRAM cell. The bit storage to area ratio of DRAMs are very efficient, at least for larger storage demands.
Power	SRAMs have a higher leakage current and dynamic power consumption due to logic technology compared to DRAMs	Very power efficient when large junks of data can be read

Typically we are only interested here in so called synchronous SRAMs, which means everything is triggered by a clock event. The opposite would be asynchronous SRAM where data input and output are triggered by the transition of the addresses.

Figure 4.2 shows the high level view of a SRAM which is mainly composed of a cell matrix, address port, data port, and command port (cmd) together with multiplexers and registers. The cell matrix is composed of individual bits in cells, while one storage cell is typically composed of 6 transistors building a bistable latching circuit. The bit cells are organized in an array, where the word width defines the cells in one row, and word depth defines the number of rows in this cell array. This high level view is one simplistic version of a so called single ported SRAM, with one address port and one data port. Typically for single ported SRAMs we can access exactly one word per clock cycle, either read one word or write one word, respectively. Of course there exist many different types of SRAMs which for example can read and write one data within one clock cycle. These SRAMs are called dual ported memories and will be discussed later. The bit pattern at the address port defines the row number in the cell matrix. The demultiplex after the address port is called address decoder which ensures the correct address in bits to row number. Note, that the number of addresses can be quite large, here indicated with 4096 rows. The address decoder and the mandatory wires to the corresponding cells can have

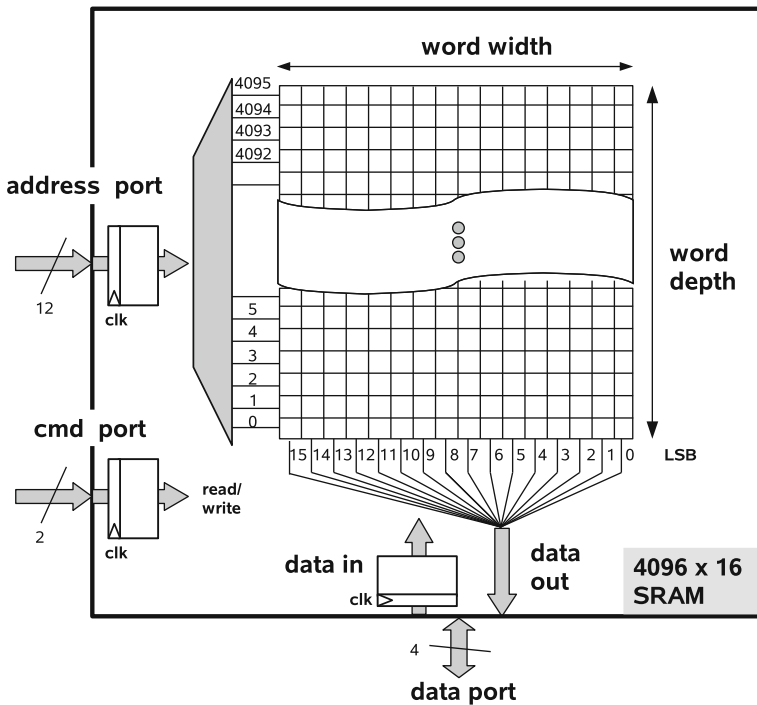


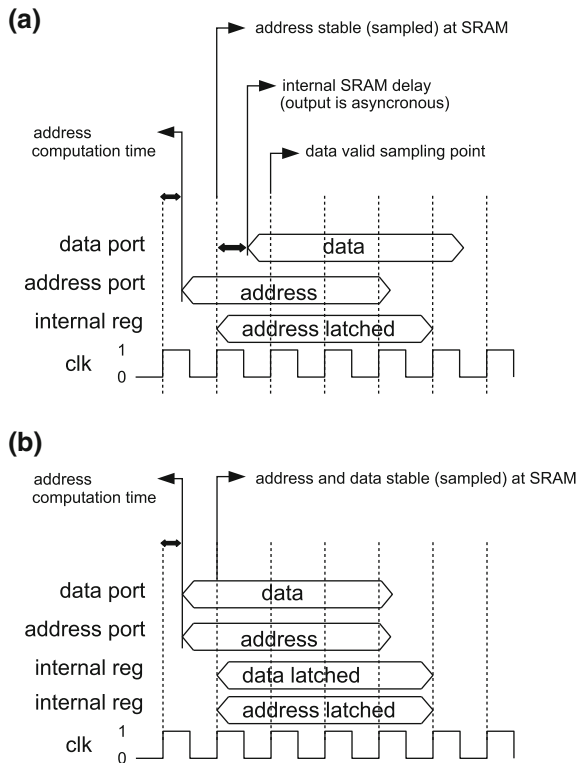
Fig. 4.2 High level view of a SRAM memory

a large influence on power of the overall SRAM which is explained in more detail in the next section. Via the command port we typically define the mode of access, either reading or writing. It is as well possible to put the memory in idle mode which means a reduced static power consumption.

The registers as indicated in Fig. 4.2 can best be understood when looking at the clock diagram for reading and writing the SRAMs. Figure 4.3a shows the timing diagram of a typical read access to a SRAM memory. The address data is passed to the input port of a SRAM memory. This address data is then latched in an internal register with the next rising clock edge. The data which can be read from the output port is then stable at the next clock cycle. We can read one new data at each clock cycle, while the corresponding address has to be passed one clock cycle before to the address port. This behavior is indicated in the high level SRAM view as shown in Fig. 4.2 which shows an input register at the address port but no output register for the data port.

Figure 4.3b shows the timing diagram of a typical SRAM writing access. The address and the corresponding data have to be passed at the same clock cycle to the SRAM ports. The address and data are internally latched and then the data is written, in the mean while the next data and address can be passed to the SRAM ports. This means that the SRAM accepts new data at each clock cycle.

Fig. 4.3 Reading and writing process of a SRAM memory. **a** SRAM read access. **b** SRAM write access



4.2.1 SRAM Design Space Exploration

As already mentioned the SRAM memories are one important building blocks for the design of individual components. The hardware designer has to care about the instantiation of the memory arrays. Many different SRAM can be instantiated with different shapes. The shape defines the word depth and word width. It is obvious that SRAM memories with different shapes result in different area numbers and power numbers. The designer would like to check what is the influence with respect to area and power when instantiating different types of SRAMs. What is the area benefit if my algorithm works with one bit less quantization? What is the expected power consumption if I have to access the memory every clock cycle? What is the difference in e.g. area when instantiating shapes of 512×8 versus 128×32 (both memories store the same amount of overall bits)? What is the maximum design frequency the memory can operate on?

To get answers on these question we have to explore the memory characteristics. Each manufacturer offers another possibility how to analyze SRAM data, which we denote in this manuscript as SRAM memory explorer. **Note that the following data and naming are artificial and not trailable to one specific manufacturer.** The SRAM memory explorer gives you the relevant design information for a given SRAM design, described by its characteristics under specified conditions. It gives here an example of SRAM output characteristics of a SRAM input characteristics request. In the following we will explain the input parameters and output parameters of a possible memory explorer. The input parameters define the SRAM for which we request the design information and its operating condition and are mandatory to narrow the search space.

Even the number of possible input parameters might be very large. For that reason we divide the input request in primary and secondary input parameters. The primary input parameters are mandatory to be defined by the designer, while for setting the secondary parameters a more precise knowledge of the internal memory structure is mandatory.

Primary Input Parameters

- Technology
 - Of course the feature size or process type is one of the first parameter with has to be specified. As shown in Fig. 1.1 the size of the technology node changes rapidly. Each technology node is coming typically with different SRAM types like
 - Low power: SRAMs are optimized with respect to low power consumption, typically for mobile applications we assume a so called low leakage technology.
 - High performance: SRAMs are optimized with respect to fast access times and thus high frequencies.
 - Regular: SRAMs which trades off low power and high performance which are of course typically counteracting parameters.

- **WordDepth**
The word depth parameter defines the number of words which can be stored in the memory.
- **WordWidth**
The word width parameter defines the number of bits per of each word.
- **Process**
For the process we have to distinguish between slow and fast. Due to process variation the switching time of transistor differs. Slow process defines e.g. the access time at -3σ of the process, while fast process at $+3\sigma$ respectively. An architecture has to be designed with respect to both conditions, slow and fast, since we do not know the exact resulting switching time of each individual fabricated chip. Thus, the critical path check should always be done on the slow process cycle time, while the so called hold time check has to be done on the fast process. The hold time specifies the time span a data has to be stable after a rising clock edge. This time span is mandatory to latch the data e.g. in a register.
- **Voltage**
Each memory is specified with different voltage levels (nominal, worst, best). The design which is done with respect to a nominal voltage e.g. 1. V has to work as well at the respective worst case voltage (-10%) of 0.9 V. Thus the cycle time has to be checked as well for worst case Voltage since this cycle time may be twice as large. Power is typically checked on nominal voltage level, however again worst case power consumption should be done on $+10\%$ voltage level.

Secondary Input Parameters

- **Power Off**
Some memory types allow to switch off the core cell array, where we have to specify if we would like to have a pin for such a power off mode or not. One has to remember since the SRAM is a volatile memory a power off mode will always delete the memory content.
- **Sleep Mode**
Again an additional pin is generated if a sleep mode or sometime retention mode is desired. A sleep mode switches off the periphery and thus reduces the leakage of the memory. The memory content will be preserved.
- **Write single bit**
Some memory types allow to write individual bits. These special property will increase the overall area since additional control logic will be mandatory to enable the addressing of each individual cell.

Primary Output Parameters

- **Area**
The area occupied by the specified memory is one of the key numbers, and typically given in mm^2 . This number is very accurate even for the final physical design.

Remember that for logic area after synthesis a significant overhead for place and route may occur (e.g. +30%), due to clock tree or additional design for test structures.

- Power read nominal ($P_{readnom}$)

This number defines the average power of a read under the assumption that half of the address bits are switching. All power numbers of a memory are specified by $\mu W/MHz$, i.e. the specified power consumption is normalized to the system frequency f . The power consumption of a memory results in

$$P = A_{ac} \cdot P_{readnom} \cdot f. \quad (4.1)$$

With $A_{ac} \in [0, \dots, 1]$ the access patten. $A_{ac} = 1$ means that the memory is accessed for a read at each clock cycle.

- Power write nominal ($P_{writenom}$)

This number defines the average power of a write under the assumption that half of the address bits are switching and half of the data bits are switching. The power consumption of writing a data is slightly larger than that of a reading access ($\leq +10\%$)

- Read access time (T_{acc})

The read access time specifies the (internal) time a memory requires to read a data. It gives you an indication about the amount of additional logic one can instantiate between the input/output memory port and the next register. Thus the critical path is composed of $T_{cyc} \geq T_{acc} + T_{logic}$.

- Cycle time (T_{cyc})

The cycle time determines mainly the maximum frequency of the design. For smaller design frequencies one access can be performed at each clock cycle.

Secondary Output Parameters

- Aspect Ratio

Defines the ratio between the geometries Height/Width.

- ColumnMux:

A memory array may be folded for good aspect ratio. In Fig.4.4 one possible example is shown. The memory is instantiated with parameters 4096×16 (rows \times columns). However the internal storage structure may be folded into a 2048×128 memory. However, to select 16 output bits form the 32 columns requires 16 times a 2:1 column multiplexer. Thus for this example the ‘Mux’ parameter would be $ColumnMux = 2$.

- Output Capacitance

This parameter defines the maximum capacitance which the outputs are able to drive.

- Width and Height

The physical width and height of the SRAM cell in mm.

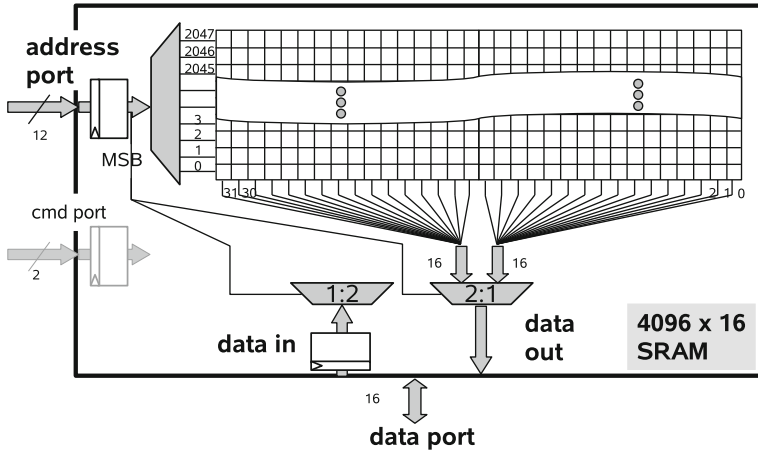


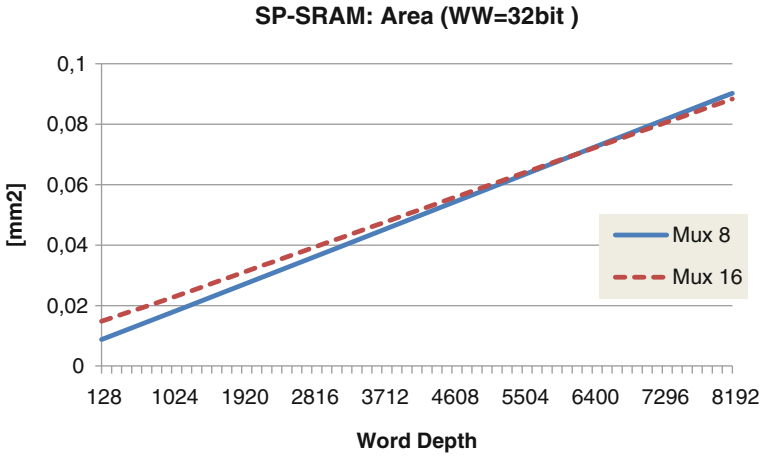
Fig. 4.4 High level view of a SRAM memory with ColumnMux=2.

- Density
The density is defined as Kbits/mm² and gives you an indicator about the area efficient of the memory.

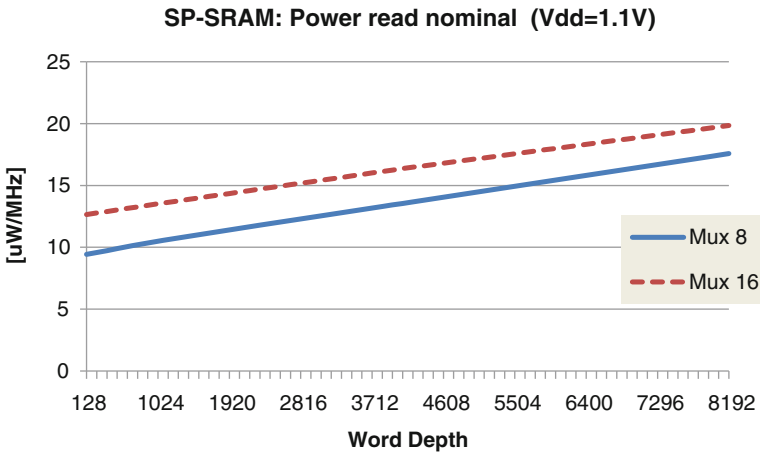
4.2.2 Exemplary SRAM Data: Area, Power, Cycle Time

In this section we show exemplary the results for area, power, and cycle time. The utilized technology for demonstration is a 40 nm low power technology with $V_{dd} = 1.1$ nominal voltage. Note, that the numbers are derived from an existing technology, however, the results are changed with respect to the absolute numbers and adjusted for educational purposes. Shown are different memory types like:

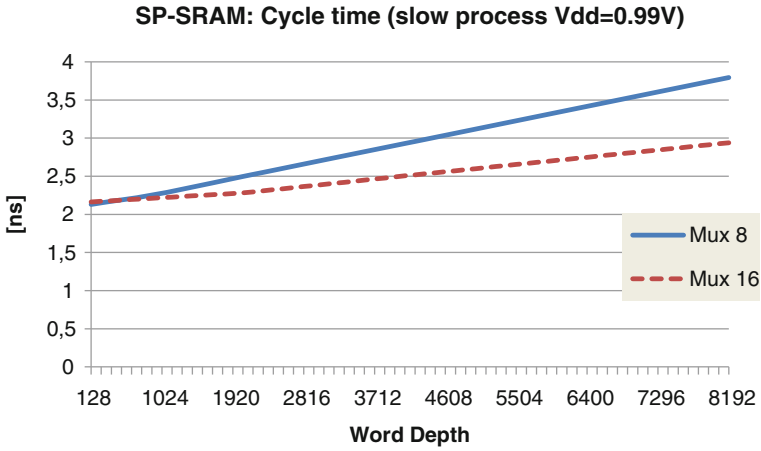
- **SP-SRAM**: single ported SRAM memories.
- **XS-SP-SRAM**: extra small single ported SRAM memories which are designed for a small storage requirement.
- **DP-SRAM**: dual-portted SRAM memories which enable the reading of two values and the possibility to write and read one value within a clock cycle.



The graph shows the area trends for a *WordWidth* of 32 bits and different *WordDepth* values, ranging from 128 addresses to 8192 addresses respectively. The two lines represents the area for two different *ColumnMux* factors. The area is given in mm^2 .

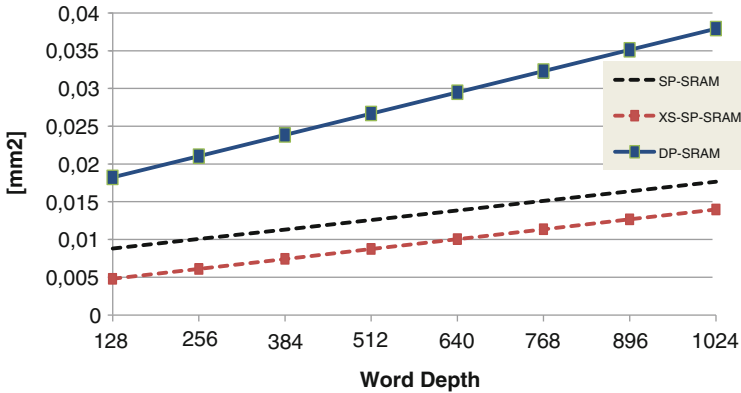


The figure shows the average power read operation ($P_{readnom}$) at which half of the addresses are switching. It is important to show the numbers for a nominal voltage case of $V_{dd} = 1.1\text{V}$. The two *ColumnMux* factors show at least up to 8192 addresses no break-even point. At least from perspective of power consumption one would decide in favor of $Mux = 8$.

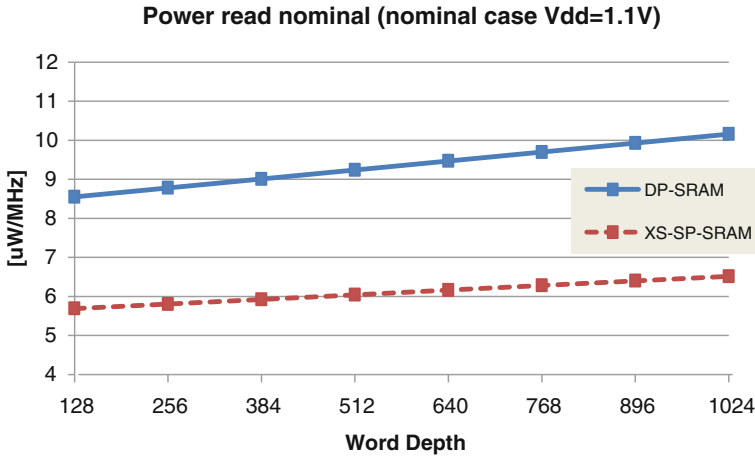


The graph shows the cycle time for the LPHDSPSRAM again with a word width of 32 bits. Attention the cycle time has to be checked for the worst case assumption, which is the slow case of $V_{dd} = 0.99\text{ V}$. The two lines represents again the area trends for two different ColumnMux factors. However, this time the larger Mux factor shows a etter cycle time behavior.

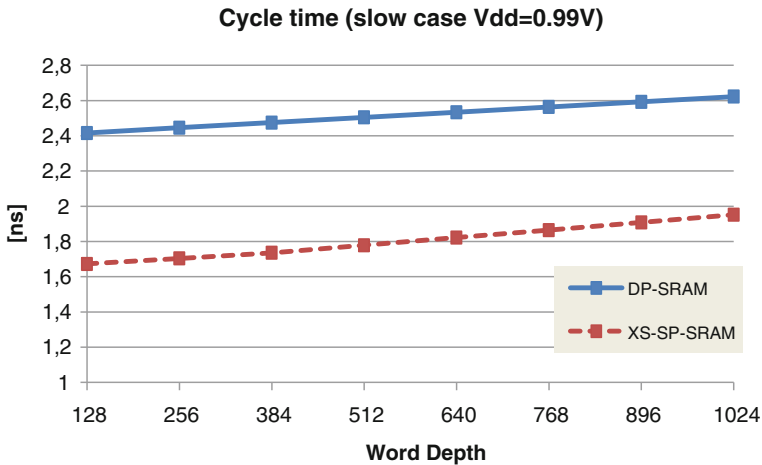
Area of XS-SP-SRAM vs. SP-SRAM vs. DP-SRAM all WW=32bit



The graph shows the area trend lines for three different memory types. The chosen *WordDepth* is quite small with a maximum of 1024. Not surprisingly, the special optimized memory for small storage demands XS-SP-SRAM shows the smallest area. The dual ported memory shows at least in this range a large area compared to both single ported SRAMs.



The figure shows Power read nominal case for the DP-SRAM and XS-SP-SRAM memories (WW = 32 bits, Mux = 4). Again the overhead of the power consumption for an access of a dual ported memory gives a strong argument to avoid dual ported memories. Note, that these memories here allow a full simultaneous write/read of two values. Thus the entire control logic, I/Os, and internal routing has to be doubled.



The graph shows the cycle time for the XS-SP-SRAM and DP-SRAMs again with a word width of 32 bits. The Mux factor was chosen to obtain the best possible cycle time for both cases. The cycle time overhead for the dual ported SRAMs can be significant. When using dual ported SRAMs, special care has to be put on the cycle constraint given by the system constraints.

Summary SRAM Exploration

Figures 4.5 and 4.6 show the area and the power trend for the three different memory types. Both figures show the result for the slow process with $V_{dd} = 0.99\text{ V}$, thus the best case power consumption. However, the intention of the figures is to show the relative area and power and the area and power trends for different memory types with respect to bit width and word depth. For every memory type three different, typical word width $WW \in \{8, 16, 32\}$ bit are given. One important fact can be seen, that each memory type is designed for a specific range of word depth. Especially the SP-SRAMs are designed for very large number of words. This can be seen when comparing the slopes of the extra small memories and the large single ported memories. Between ~ 1024 and ~ 2048 the area slopes will cross. Attention in the figures: no MuxNumbers are given, thus, it is just an indication of sizes and trends. For the designer we can summarize the most important issues for working with the memory explorer or a similar tool.

- Different nominal cases for the memory exist, e.g. $V_{dd}(nom) = 1.0\text{ V}$ or $V_{dd}(nom) = 1.1\text{ V}$.
- Derived from the nominal case a slow and fast case exist with $V_{dd}(slow) = 0.9 \cdot V_{dd}(nom)$ and $V_{dd}(fast) = 1.1 \cdot V_{dd}(nom)$.
- The cycle time has to be checked for its worst case assumption, which is $V_{dd}(slow)$.

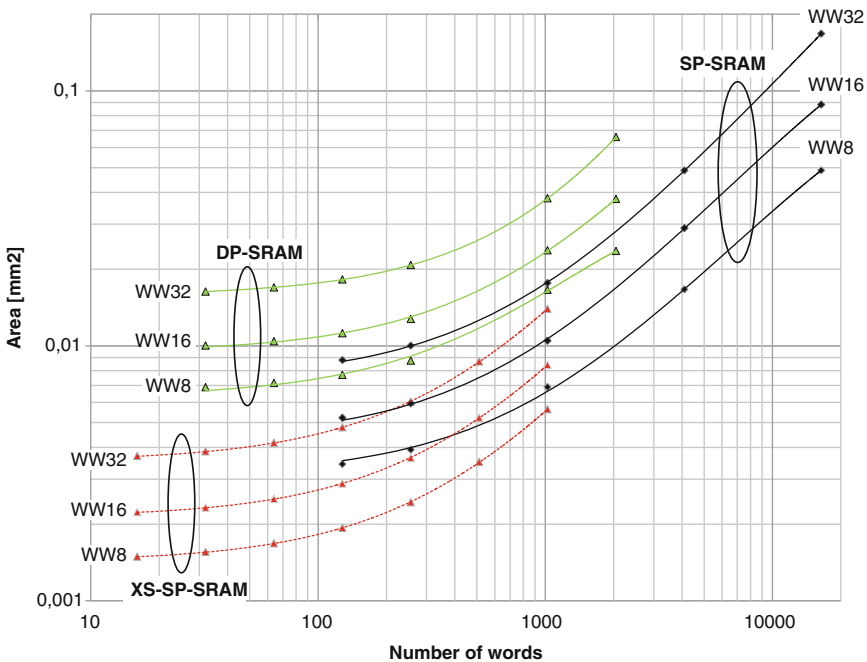


Fig. 4.5 Example of the area for different memories types, all in 40nm technology

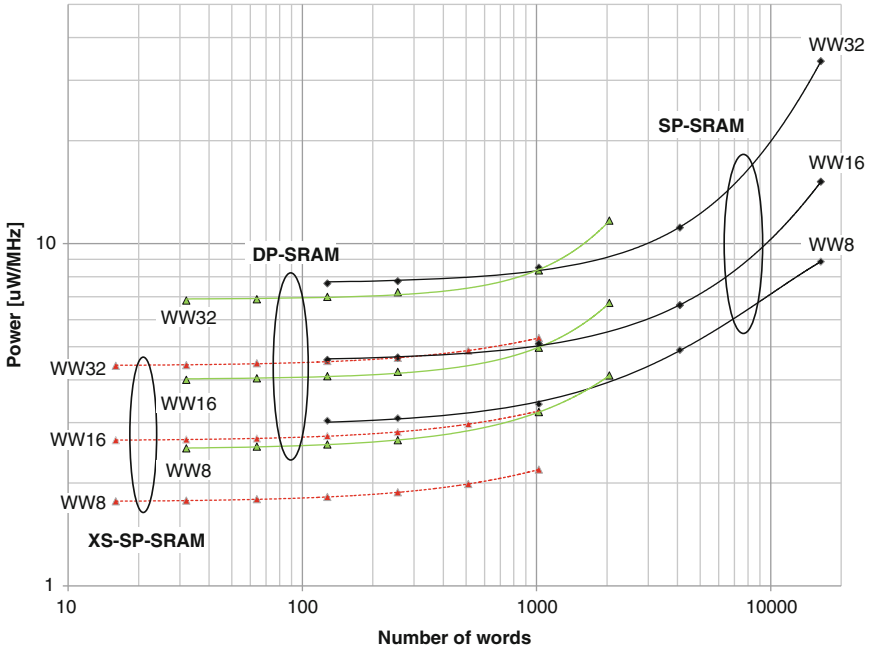


Fig. 4.6 Example of the power for different memories types, all in 40 nm technology

- The power consumption should be investigated for the nominal case $V_{dd}(nom)$.
- The ColumnMux factor defines the internal structure of the memory. Different ColumnMux factors result in different, sometimes contradicting, area, power and cycle time numbers.
- When possible, the instantiation of dual ported memories should be avoided, since area and power are larger than for single-ported memories with identical storage capabilities.

Which SRAM memories are instantiated during the design of a component, and whether it is possible to avoid the instantiation of dual ported memories, depends on the access pattern of the application. Access pattern defines the data access in time and space (location) and depends of the functionality we would like to implement. One example with a difficult access pattern is presented in the next section.

4.2.3 Importance of Memory Exploration

We have seen in the previous section that we can instantiate different types of memories. For the design of a digital baseband receiver exploring the different options gets more and more important. In future designs it is expected that the size of memories with respect to the overall chip area will increase further. For example Fig. 4.7

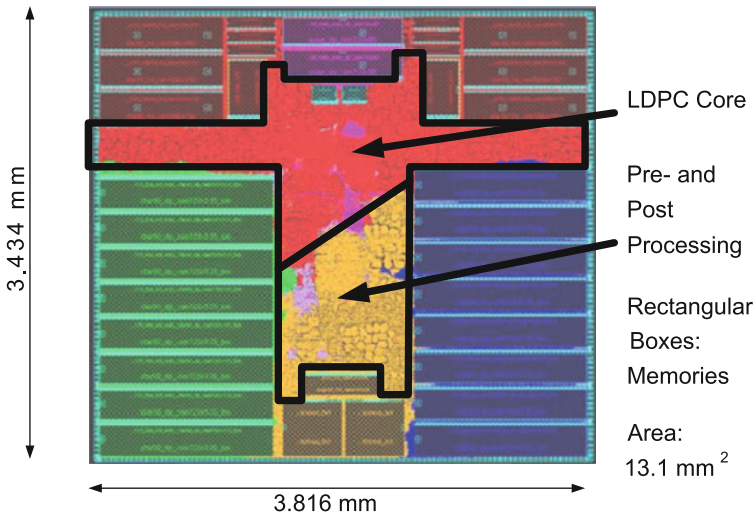


Fig. 4.7 Chip photo after place&route of product DVB-S2 receiver chip, 70% of the overall area is determined by instantiated memories.

shows the final place and route (*P&R*) layout of a low-density parity-check (LDPC) decoder which was designed for a DVB-S2 receiver product.

LDPC codes are explained in Chap. 7 while the entire design of this particular decoder is described in [1]. Here, we only consider the memories, which determines 70% of entire area. The instantiated memories are indicated by the boxes. In this design we have to store in total ~ 2 Mbits of data. 960 bits are read and written in each clock cycle. An SRAM featuring a bitwidth of 960 bits does not exist as a monolithic building block. Thus, multiple memories have to be instantiated to enable the access of 960 bits per clock cycle. The designer now has the possibility to choose a possible fragmentation to achieve the required memory access bandwidth. A so-called memory hierarchy is introduced. Note, that the term memory hierarchy is often used in large systems and defines the organization of memories of different types (DRAM, SRAM) in which each storage element may have a different response time. Here, we use the term memory hierarchy to emulate a large SRAM memory array for an application while the internal structure is fragmented. This is indicated in Fig. 4.8 for two different setups to emulate the required SRAM shape of 2048×960 . Either we instantiate 15 SRAMs of shape 2048×64 or we could even instantiate 240 SRAMs of shape 256×32 . Both possibilities are shown in Table 4.2. Given are the corresponding data in terms of area and power of a single instance and the overall expected numbers, respectively. In case of a high fragmentation the extra small memories are assumed since these are optimized for the instantiated shape. Assuming the low fragmentation case the data for area and power correspond to standard single-ported SRAMs. The table shows clearly the trade-off a designer has to be aware of. The power is optimized for the case of the highly fragmented instantiation, while the area is optimized for the low fragmentation variant.

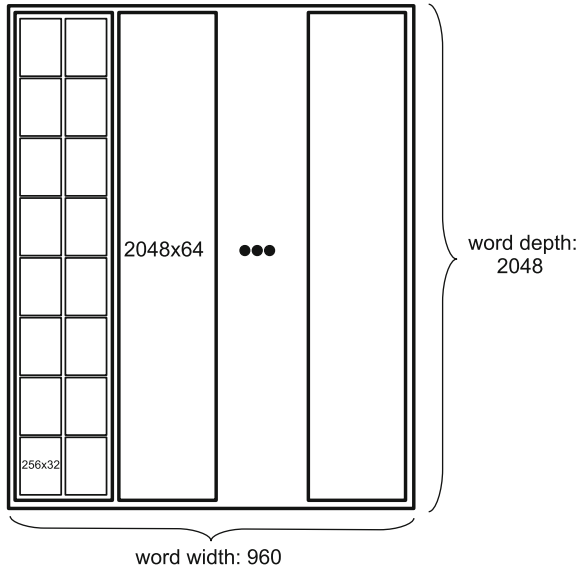


Fig. 4.8 Two different memory hierarchies to enable the access of words with 960 bits

Table 4.2 Two different memory organizations to enable an access of 960 bits per clock cycle.

Case memory type	High fragmentation XS-SP-SRAM	Low fragmentation SP-SRAM
Shape	256×32	2048×64
Single area	0.006 mm^2	0.05 mm^2
Average Power (Preadnom+Preadwrite)/2	$5.8 \mu\text{W}/\text{MHz}$	$22 \mu\text{W}/\text{MHz}$
Number of instances	240	15
Total number of stored bits	1.96 Mbits	1.96 Mbits
Total area estimate	1.44 mm^2	0.75 mm^2
Total power estimate	52.2 mW	99 mW

The total power estimate assumes here a frequency of $f = 300 \text{ MHz}$ by evaluating Eq. 4.1. The power and area numbers given here are only estimates. $P\&R$ will influence these results again since the data have to be routed to the corresponding memories. Thus, the area and the power consumption will be higher in both cases. The large influence of the $P\&R$ is further highlighted in the design example presented in Sect. 7.5.

In the example here we did not assume any constraints about how to access the 960 bits. It is of course a huge difference if these 960 bits have a regular access pattern or a random access pattern. In all cases the designer has to design the controller to ensure the correct addressing across instances of memories. One example which often requires a high fragmentation due to its difficult access pattern is presented in the next section.

4.3 Individual Component: Interleaver

In this section we will show exemplary the considerations a designer has to make when designing an individual component, in this case an interleaver. We will see that its design, at least for higher throughputs, can be difficult due to requirements to the memory architecture.

4.3.1 Interleaver Types

In communication systems interleavers are used in many different components. For wireless transmission systems, typically, block-based interleavers are used, which change the location of a bit or symbol within a block, where the term block means either a codeword, multiple codewords grouped to a frame, or only a part of a codeword. Thus, we have to distinguish between:

- inter-frame interleaving: interleaving across multiple frames,
- intra-frame interleaving: changes the position of bits within a codeword, as in the case of bit-interleaved coded modulation as shown in Fig. 2.1, and
- channel code interleaving: the channel code interleaving is the bit permutation within a part of the codeword to ensure randomness of the code. This is applied in e.g. turbo codes or LDPC codes, see Chaps. 6 and 7.

The task of any interleaver is to break up dependencies between adjacent positions within a data stream. For example, the inter-frame interleaving ensures that burst errors are spread via multiple frames. Burst errors are errors on consecutive positions within a transmission stream, which may occur when transmitting via fading channels. An interleaver spreads these uncertain locations over a larger distance.

An interleaver uses a bijective function which maps the indices of an input sequence to changed indices of an output sequence. An interleaver table Π is one realization of this index mapping, i.e. it defines the one to one positional mapping from input position to output position of a given interleaver. Typically i defines the index in the interleaved block when we speak about $\Pi(i)$. Thus the interleaved data sequence can be derived by indirect addressing:

$$\mathbf{x}(\Pi(i)) = \mathbf{x}'(i) \quad (4.2)$$

\mathbf{x} refers to the data vector in the original order, the vector \mathbf{x}' is the interleaved vector at the output of the interleaver. It is also possible to define a direct addressing. However, to obtain the same output sequence of \mathbf{x}' the inverse interleaver table has to be derived, which is typically indicated as Π^{-1} .

$$\mathbf{x}(i) = \mathbf{x}'(\Pi(i)^{-1}) \quad (4.3)$$

Many different possibilities exist to generate an interleaver table for a specific block. Which one to use depends mainly on the application. In the following we will present four different methods to generate an interleaver table. Three of these interleavers are actually used in today's communications standards. Figure 4.9 plots the output position over the input position for four different interleavers operating on blocks, all with a block length of 80 bits. The x-axis shows the input position, the y-axis the corresponding output position.

Figure 4.9a reflects the output of a so called random interleaver. The random interleaving shows no special structure and is typically not utilized in communications systems. Figure 4.9b shows the pattern for a classical block interleaver. Note that the name block interleaver denotes just a special type of a block-based interleaver, but

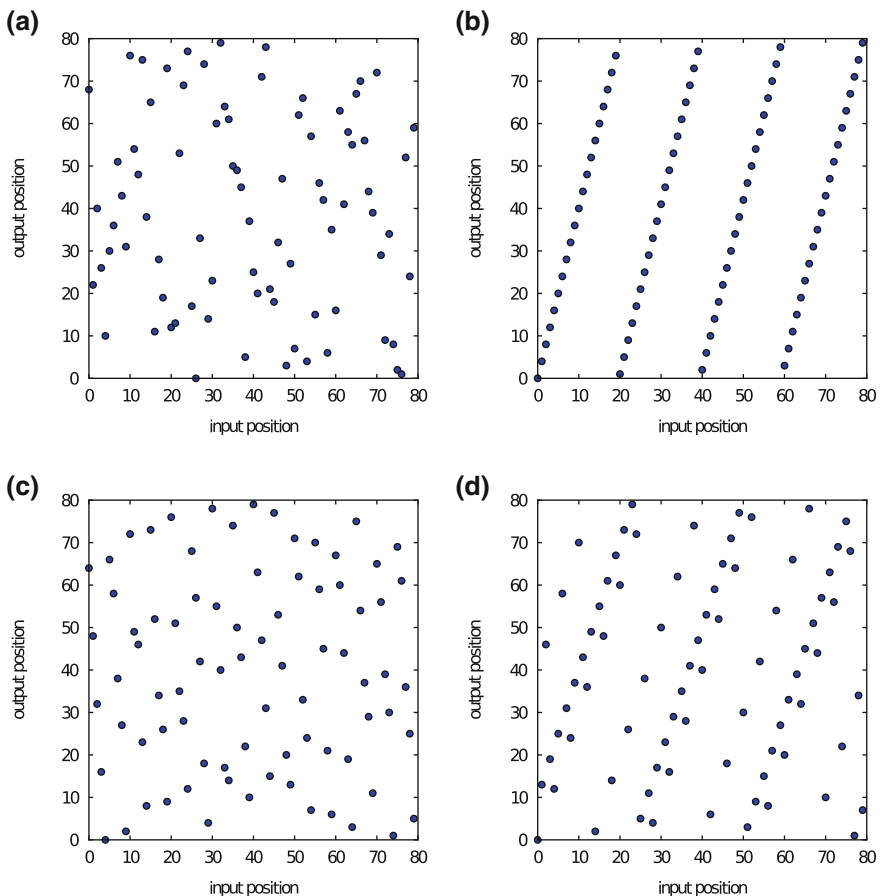


Fig. 4.9 Illustration of the permutation pattern of 4 different interleavers, all with block length of $N = 80$: **a** random interleaver, **b** block interleaver, **c** UMTS channel code interleaver, and **d** LTE channel code interleaver

these terms are not to be confused. The typical procedure of a block interleaver is to write the data stream in an array in a column by column fashion. The output stream is generated by reading the content row by row. In Fig. 4.9 we utilized an array with $C_1 = 20$ rows and $C_2 = 4$ columns. The generation of the interleaver table Π can be described with

$$\Pi(i) = (i \bmod C_1) \cdot C_2 + \left\lfloor \frac{i}{C_1} \right\rfloor \quad (4.4)$$

C_1 and C_2 are the two dimensions of the block interleaver with the overall block size of $C_1 \cdot C_2$. Block interleavers are used in communication standards very often, since they provide a simple and often effective permutation.

In addition there exist a so called cyclic block interleaver. The cyclic block interleaver needs a further vector for its description, \mathbf{I}^{offset} . The offset vector has C_2 entries, one for each column. The values define the start index at which each column is filled. For the interleaver plotted in Fig. 4.9b this offset vector would be

$$\mathbf{I}^{offset} = [1 \ 1 \ 1 \ 1]. \quad (4.5)$$

Here, all columns are written starting from the very first position top to bottom.

Plot (c) shows the interleaver which is instantiated within the UMTS channel code encoder (turbo encoder). The interleaver construction is called permuted block interleaving and is based on a classical block interleaver. Again, the input block is written to an array row by row. However, before the reading step, the position of the columns and rows are permuted as well. With this additional row and column permutation stage a quasi random interleaver pattern is obtained, even so, with a deterministic procedure to calculate this interleaver pattern within an application.

In UMTS a different interleaver table has to be generated for each block length ranging from 40 to 5114 bits. Thus, the granularity of this interleaver generator is 1 bit. Despite its deterministic generation procedure, for a hardware realization of the turbo decoder it turned out that the specified procedure to generate the interleaver tables has two disadvantages. Especially for high throughput applications, where spatial parallelism is required two problems have to be solved:

- The generation of the interleaver tables is relatively complicated and may require many clock cycles. Producing multiple indices of the interleaver table in one clock cycle is possible but cumbersome.
- The second problem is more difficult to solve. Parallel processing requires parallel interleaving of data. Memory access conflicts may occur, which have to be resolved, see Sect. 4.3.2.

Due to these two problems a new interleaver type was defined in the newer LTE standard. Figure 4.9d shows the permutation pattern of the LTE channel code interleaver (turbo encoder). It has a very simple construction rule to generate the interleaver table:

$$\Pi(i) = \left(f_1 \cdot i + f_2 \cdot i^2 \right) \bmod K \quad (4.6)$$

Table 4.3 Overview of interleaver types the block size and granularity their characteristics

Standard	Interleaver	Classes	Smallest block	Largest block	Granularity in bits
LTE [3]	TTI frame	Block interleaver		150k	8
	Sub-block	Block interleaver	120	18k	8
	Channel code	ARP interleaver [4]	40	6144	8
UMTS [5]	TTI frame	Block interleaver		150k	8
	Sub-block	Block interleaver	120	15k	8
	Channel code	Block permutation	40	5114	1
DVB-S2	Intra block	Block interleaver	16.4k	64.8k	360
[6]	Channel code	Quasi-cyclic	80k	240k	360

f_1 and f_2 are interleaver parameters defined in the standard and K is the block length. The granularity is 4, 8, or even 16 for larger block sizes. This interleaver type is called quadratic polynomial permutation (QPP) interleaver [2].

In summary we can say that different interleaver types are utilized in communications standards. The interleaver has a major impact on the resulting communications performance while different interleaving types within a transmission scheme are mandatory. For modern communication standards the choice of the appropriate interleaver type is often determined by the resulting communications performance and the efficiency of possible hardware realizations of that interleaver.

Table 4.3 shows the different interleaves which are utilized within one communication standard. For example the LTE standard features an interleaver for the transport transmission interval (TTI) frame which interleaves up to 1,50,000 positions. Then, an interleaver stage for the coded codeword is utilized with a maximum block length of 18,000 indices, furthermore the channel coding itself has an interleaver inside. Here, the block sizes for the interleaving ranges from 40 to 6144 bits. The granularity in the table defines the step width between to block sizes. The LTE turbo code interleaver has a granularity of 8 and for larger block sizes even 16, the UMTS turbo code interleaver has a granularity of one. Thus a hardware realization for the UMTS channel code interleaving has to realize all possible block sizes between 40 and 5114 bits. Thus, it is of importance if we can compute the interleaver tables in hardware or whether to compute the corresponding interleaver tables offline. Depending on the throughput requirements different challenges for the hardware realization exists.

4.3.2 Hardware Realization

This section discusses the different possibilities which can be applied to the storage of data in an interleaved order. First we will derive a serial architecture for interleaving, followed by an architecture which interleaves P data in parallel. Assuming a serial architecture, we always store a data block in original order in one memory

while the data with changed indices position is stored in a second memory. Furthermore, we assume that the interleaver tables are stored as well in memories. For the serial interleaver architecture we can identify three major methods to the interleaving/deinterleaving.

- **Interleaving by reading** (Fig. 4.10a)

The memory which stores the interleaver table is read sequentially. The output data of this memory is used as address for data memory A. The retrieved data is then written to data memory B at consecutive addresses. Thus, we read the content DATA A in an interleaved manner, which is denoted as interleaving by reading.

- **Interleaving by writing** (Fig. 4.10b)

The data memory A and the deinterleaver memory are read sequentially. The output data of the deinterleaver memory is used as address for data memory B. Thus, the retrieved data is written in an interleaved manner to data memory B. The content of the address memory is different to the first case and is called deinterleaver table.

- **Interleaving in two stages** (Fig. 4.11)

The interleaving is performed in two stages. Stage one writes the data in an interleaved manner to an intermediate data storage (DATA TMP). Interleaver Table 4.1

Fig. 4.10 One stage interleaving: Interleaving from memory A to memory B upon read (*top*), upon write (*bottom*)

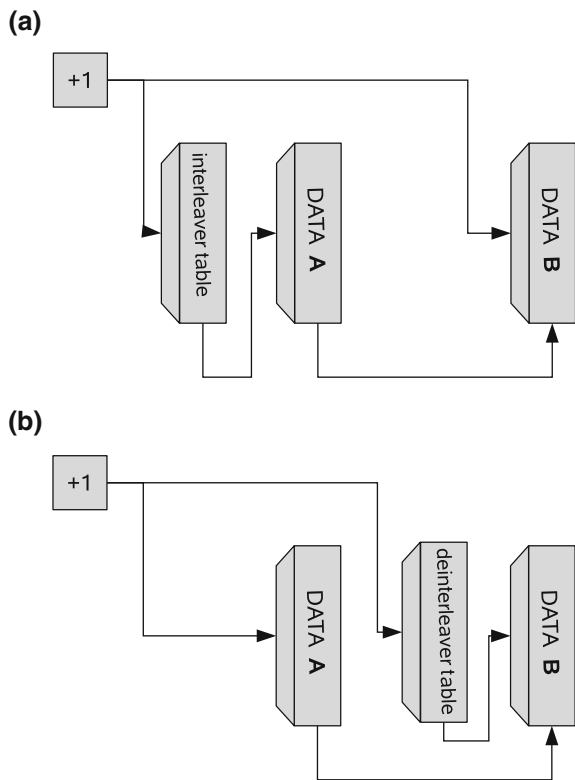
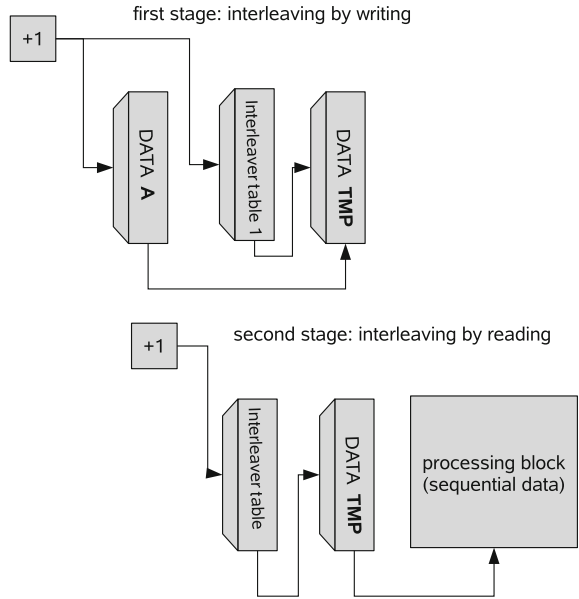


Fig. 4.11 Two stage interleaving: the content of address Tables 4.1 and 4.2 are different.



does not hold the final interleaving table. The final interleaved data sequence is obtained after the second stage, which is here indicated as an input to a processing block. The second stage performs an interleaving by reading. The content of interleaver Tables 4.1 and 4.2 are different and have to be derived from the overall interleaver table.

Within all presented serial architectures we have instantiated memories for the interleaver tables. However, when it is possible to generate the interleaver tables on the fly we can replace these memories by an appropriate logic block. On the fly means we have to provide the correct target address for reading or writing at each clock cycle.

Parallel Interleaving

So far we have presented only possibilities for a sequential interleaving, which means we need N clock cycles to interleave N values. Increasing throughput demands requires a parallel interleaving. Multiple data have to be processed and, consequently, also interleaved in one clock cycle. Figure 4.12 shows the major problem of parallel interleaving. On the right side the address table with the corresponding interleaving address is shown. The data flow for the architecture is top down. 4 single ported SRAM memories are instantiated in front of a parallel processing unit. The data A,B,C,...,P are stored in the four input memories, while the numbering next to each memory indicates the original address labeling of the interleaver table. The parallel

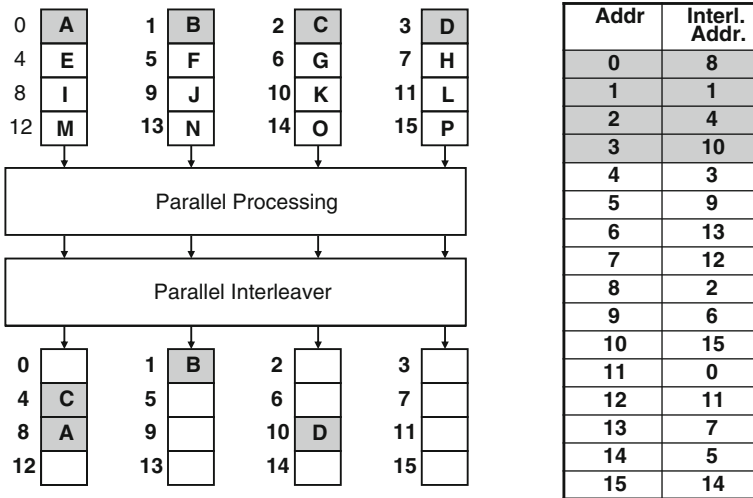


Fig. 4.12 Parallel Interleaving and memory access conflict for the very first processing step

processing unit accepts now four values per clock cycle, i.e., in the first clock cycle it processes data A,B,C,D. After the processing we would like to write the four output data in an interleaved order. However this is not possible within one clock cycle since data A and C have to be written to the same output memory. We call this a memory access conflict. Figure 4.13 shows the data after all data are written. In this small example at all four clock cycles a memory access conflict occurs.

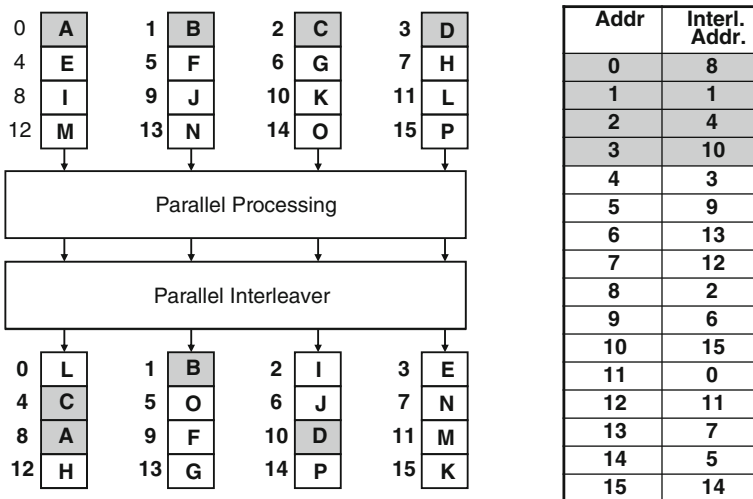


Fig. 4.13 Parallel Interleaving and memory access after all values are processed

One of the most elegant method to resolve the parallel interleaving problem is two utilize a two stage interleaving process. As shown in Fig. 4.11 the two stage interleaver requires two interleaver tables. Each stage interleaves P data in parallel without memory access conflicts. The trick is the intelligent pre-computation of the two appropriate interleaver tables. In [7] it was shown that it is always possible to find a two stage procedure to interleave P values without memory access problems.

Currently, there is no deterministic algorithm known which can calculate the two address tables on the fly, thus a pre-processing to determine the two interleaver tables has to be performed Typically in practical applications all interleaver patterns are pre-calculated and stored in external memories. Assuming a new block size the corresponding interleaver tables have to be loaded to corresponding interleaver table memories. This parallel interleaving problem occurs also for example in UMTS turbo decoding which features 5000 different block length and thus as many varying interleaver pattern. The storage of the pre-computed interleaver patterns is quite large and exceeds the storage demands of the turbo decoder architecture itself.

Joint Algorithm-Hardware Design

Interleavers are defined in many communication standards. Since the throughput demands of nearly all standards steadily increases we have to implement the interleaving sooner or later as well in a parallel manner. As seen, this poses problems for the implementation. However, it is possible to design an interleaver which provides parallel processing without any memory access conflicts. For that we have to consider jointly constraints from a possible hardware realization and constraints for the algorithm in this case an appropriate permutation pattern.

The idea how to achieve this is quite elegant and often denoted as **joint algorithm-hardware design**. Figure 4.14 shows the first step for designing an interleaver table which allows a conflict free parallel interleaving. We have seen that in the first clock cycle there was an access conflict. We would like to write data A and C to the same memory. The idea now is to replace the original destination address of data C by a new address which allows a conflict free storage. For the first clock cycle in this example we have here 4 possible address which have no conflicts. In theory we could choose any of these for possibilities, nevertheless, since the interleaving has to achieve a certain goal with respect to an application, additional constraints may exist. For example a clustering of addresses is not allowed, see Fig. 4.9a in the case of a random interleaving. A clustering of address indicates that e.g. a burst error would not be spread across the block. The interleaver table is now filled step-by-step, always preventing a possible conflict at each construction step. The entire 'conflict free' interleaver pattern can be obtained by Heuristics as explained, or as well by algebraic methods [4]. The interleaver design method, however, has always a parallelism level in mind, at which memory access conflicts can be prevented. Thus the joint algorithm-hardware design requires knowledge of both domains. The hardware gives us constraints with respect to an initial architecture template and defines also

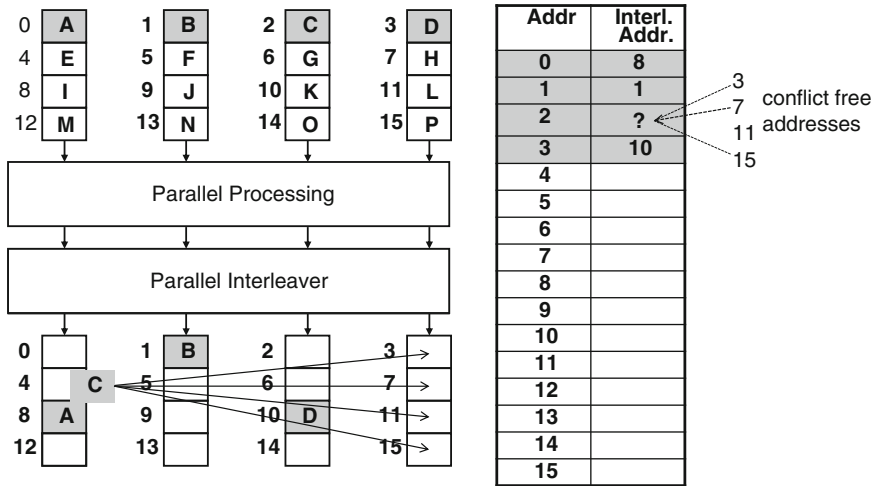


Fig. 4.14 Designing an interleaving table which allows a parallel interleaving without memory access conflicts

as well a target parallelism, while the algorithm or application has constraints with respect to functionality.

The idea of joint algorithm-hardware design was already applied in the design of communications standards. The (turbo code) interleavers of the LTE standard are designed with respect to hardware knowhow. The interleavers can be implemented for parallel processing without occurring memory access conflicts. In the case of LTE turbo code a parallelism of the decoder architecture is considered with $P = 4$, $P = 8$, or $P = 16$ respectively.

References

1. Mller, S., Schreger, M., Kabutz, M., Alles, M., Kienle, F., Wehn, N.: A novel LDPC decoder for DVB-S2 IP. In: Proc. DATE '09. Design, Automation. Test in Europe Conference. Exhibition, pp. 1308–1313 (2009)
2. Sun, J., Takeshita, O.Y.: Interleavers for turbo codes using permutation polynomials over integer rings. IEEE Trans. Inf. Theory **51**(1), 101–119 (2005). doi:10.1109/TIT.2004.839478
3. Third Generation Partnership Project: 3GPP TS 36.212 V8.5.0; 3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Evolved Universal Terrestrial Radio Access (E-UTRA); Multiplexing and channel coding (Release 8) (2008). www.3gpp.org
4. Nimbalkar, A., Blankenship, Y., Classon, B., Blankenship, T.K.: ARP and QPP interleavers for LTE Turbo coding. In: Proceedings of the IEEE Wireless Communications and Networking Conference WCNC 2008, pp. 1032–1037 (2008). doi:10.1109/WCNC.2008.187
5. Third Generation Partnership Project: 3GPP TS 25.212 V1.0.0; 3rd Generation Partnership Project (3GPP); Technical Specification Group (TSG) Radio Access Network (RAN); Working Group 1 (WG1); Multiplexing and channel coding (FDD) (1999). www.3gpp.org

6. European Telecommunications Standards Institute (ETSI): Digital Video Broadcasting (DVB) Second generation framing structure, channel coding and modulation systems for Broadcasting, Interactive Services, News Gathering and other broadband satellite applications; TM 2860r1 DVBS2-74r8. www.dvb.org
7. Tarable, A., Benedetto, S.: Mapping interleaving laws to parallel turbo decoder architectures. *IEEE Commun. Lett.* **8**(3), 162–164 (2004)