

**ORGANISATION EUROPÉENNE POUR LA RECHERCHE NUCLÉAIRE
CERN EUROPEAN ORGANIZATION FOR NUCLEAR RESEARCH**

CERN Accelerator School
Digital Signal Processing

Sigtuna, Sweden
31 May – 9 June 2007

Proceedings
Editor: D. Brandt

GENEVA
2008

Abstract

These proceedings present the lectures given at the twenty-first specialized course organized by the CERN Accelerator School (CAS), the topic being Digital Signal Processing. The course was held in Sigtuna, Sweden, from 31 May–9 June 2007. This is the first time this topic has been selected for a specialized course. Taking into account the number of related applications currently in use in accelerators around the world, it was recognized that such a topic should definitively be incorporated into the CAS series of specialized courses. The specific aim of the course was to introduce the participants to the use and programming of Digital Signal Processors (DSPs) and Field Programmable Gate Arrays (FPGAs) evaluation boards. The course consisted of lectures in the mornings covering fundamental background knowledge in mathematics, controls theory, design tools, programming hardware platforms, and implementation details. In the afternoons the students split into two groups with people working in pairs. One group worked on problem solving using a PC-based workstation connected to a DSP evaluation board while the second group performed the same exercise on PCs connected to a FPGA evaluation board. Half-way through the school, those working on DSPs moved to FPGAs and vice versa. The problems to be solved were of increasing difficulty and all of them were directly related to real and practical realizations currently in use in the accelerator field.



Foreword

The aim of the CERN Accelerator School (CAS) to collect, preserve and disseminate the knowledge accumulated in the world's accelerator laboratories applies not only to general accelerator physics, but also to related sub-systems, equipment, and technologies. This wider aim is achieved by means of specialized courses. For 2007, the topic of the course was Digital Signal Processing (DSP) and was held at the SigtunahÅjden Hotel, Sigtuna, Sweden, from 31 May–9 June 2007. The course was organized in collaboration with the Uppsala University, Sweden.

This was the first time Digital Signal Processing was used as a topic for a CAS specialized course. However, when considering the number of related applications currently in use in the field of accelerators, it was felt that it was the right time to address this specific topic. The organization of such a course was a real challenge in itself, since it implied finding and transporting computers to Sweden, negotiating the availability of software licences for the duration of the course, negotiating the acquisition of DSP and FPGA evaluation boards, and preparing 50 computers to be compatible with the proposed exercises. It is a pleasure to express my thanks to CERN, the companies Synplicity, Mathworks and Xilinx, as well as the Texas Instruments Worldwide University Program, and the Xilinx University Program for their kind support, without which the course would not have been made possible.

To ensure that the required material was made available was certainly a mandatory pre-requisite, but the elaboration of the course and, in particular, the hands-on courses in the afternoon sessions were no less challenging. I would particularly like to acknowledge the outstanding contribution of our CERN colleagues Hermann Schmickler, Maria-Elena Angoletta and Javier Serrano for their contacts with the relevant industry and the high-quality afternoon courses they prepared and supervised, John Evans for his hard work in obtaining the software licences and Edwige Bournonville for the perfect preparation of the computers as well as their maintenance during the course. Similarly, I would like to thank Leif Gustafsson and Paweł Marciniewski (Uppsala University) for their highly appreciated contribution as tutors during the afternoon courses.

It is also important to thank the Local Organizing Committee (Professor Tord Ekelof, Dr. Volker Ziemann and Ms. Inger Ericson) for obtaining financial support from the Swedish Research Council (Vetenskapsrådet) to help CAS with the organization of the course.

As always, the backing of the CERN management, the guidance of the CAS Advisory and Programme Committees, the attention to detail of the Local Organizing Committee and the management and staff of the SigtunahÅjden Hotel ensured that the school was held under optimum conditions.

Very special thanks must go to the lecturers for the enormous task of preparing, presenting, and writing up their topics.

Finally, the enthusiasm of the participants who came from more than 23 different countries around the world was convincing proof of the usefulness and success of the course.

It is my pleasure and privilege to thank most sincerely all those persons who helped me to make the course a success, including Barbara Strasser, Suzanne von Wartburg, and the team of the CERN Scientific Text Processing Service for their dedication and commitment to the production of this document.

Daniel Brandt
CERN Accelerator School

PROGRAMME
Digital Signal Processing (DSP)
31 May – 9 June 2007, Sigtuna, Sweden

Time	Friday 1 June	Saturday 2 June	Sunday 3 June	Monday 4 June	Tuesday 5 June	Wednesday 6 June	Thursday 7 June	Friday 8 June	Saturday 9 June
08:30	Types of Accelerators and Specific Needs I		Math I	Math II	Math III	Math IV	Real Time Control of Beam Parameters I	Real Time Control of Beam Parameters II	Buses to airport
09:30	T. Shea	M. Hoffmann	M. Hoffmann	M. Hoffmann	M. Hoffmann	M. Hoffmann	M. Dehler	M. Dehler	M. Dehler
09:30	Types of Accelerators and Specific Needs II	Control Theory I	Control Theory II	Control Theory III		RF Application I	RF Application II	Controls Integration II	
10:30	T. Shea	S. Simrock	S. Simrock	S. Simrock	S. Simrock	X	T. Schilcher	T. Schilcher	T. Shea
	C O F F E E	B R E A K				C O F F E E	B R E A K		
11:00	High Level Modeling Tools I	Introduction to DSP I	Introduction to DSP II	Introduction to DSP III	Introduction to DSP III	C	Transverse Feedbacks	Controls Integration I	BI Application II
12:00	J. Evans	M. Angloletta	M. Angloletta	M. Angloletta	M. Angloletta	U	M. Lonza	T. Shea	U. Raich
12:00	High Level Modeling Tools II	Introduction to FPGA I	Introduction to FPGA II	Introduction to FPGA III		R	Longitudinal Feedbacks	BI Application I	Outlook or Seminar
13:00	J. Evans	J. Serrano	J. Serrano	J. Serrano	J. Serrano	S	M. Lonza	U. Raich	H. Schmickler
	L U N C H					L U N C H			
14:30	Introduction to Afternoon Courses	From Analog to Digital I	From Analog to Digital II	From Analog to Digital III	From Analog to Digital III	I	Parallel Lab course (DSP/FPGA) Group B/A	Parallel Lab course (DSP/FPGA) Group B/A	Parallel Lab course (DSP/FPGA) Group B/A
15:30	H. Schmickler	J. Belleman	J. Belleman	J. Belleman	J. Belleman	O			
15:30	Parallel Lab course (DSP/FPGA) Group A/B	Parallel Lab course (DSP/FPGA) Group A/B	Parallel Lab course (DSP/FPGA) Group A/B	Parallel Lab course (DSP/FPGA) Group A/B	Parallel Lab course (DSP/FPGA) Group A/B	N	Parallel Lab course (DSP/FPGA) Group B/A	Parallel Lab course (DSP/FPGA) Group B/A	Discussion Session
16:30		C O F F E E	B R E A K				C O F F E E	B R E A K	D. Brandt
17:00	Parallel Lab course (DSP/FPGA) Group A/B	Parallel Lab course (DSP/FPGA) Group A/B	Parallel Lab course (DSP/FPGA) Group A/B	Parallel Lab course (DSP/FPGA) Group A/B	Parallel Lab course (DSP/FPGA) Group B/A		Parallel Lab course (DSP/FPGA) Group B/A	Parallel Lab course (DSP/FPGA) Group B/A	Parallel Lab course (DSP/FPGA) Group B/A
18:00	Welcome Drink	DINNER	DINNER	DINNER	DINNER		DINNER	DINNER	DINNER

Contents

Foreword	
<i>D. Brandt</i>	v
High-level modelling languages	
<i>J. Evans</i>	1
Digital signal processing mathematics	
<i>M. Hoffmann</i>	11
Control theory	
<i>S. Simrock</i>	73
From analog to digital	
<i>J. Belleman</i>	131
Digital signal processor fundamentals and system design	
<i>M.E. Angoletta</i>	167
Introduction to FPGA design	
<i>J. Serrano</i>	231
RF applications in digital signal processing	
<i>T. Schilcher</i>	249
Multi-bunch feedback systems	
<i>M. Lonza</i>	285
Real time control of beam parameters	
<i>M. Dehler</i>	331
Beam diagnostics	
<i>U. Raich</i>	361
Control system integration	
<i>T.J. Shea</i>	385
LHC technological challenges: use of digital signal processors in the power converters for the LHC particle accelerator	
<i>H. Schmickler</i>	411
List of Participants	429

High-level modelling languages

J. Evans
CERN, Geneva, Switzerland

Abstract

This paper gives an introduction to the latest developments in modern electronic design methodology. It will give a brief history of the evolution of design software in an attempt to explain the seemingly haphazard development up to the present-day situation.

1 Introduction

This paper gives an introduction to the latest developments in modern electronic design methodology. It will give a brief history of the evolution of design software in an attempt to explain the seemingly haphazard development of these tools to become what is now known as the Electronic Design Automation industry. The emphasis is to provide a common background to all school attendees with specific references made to the tools used during the week.

The frenetic pace of advance in the electronics industry is not expected to change in the near future. According to the research firm Gartner, there is no slowdown in the expected revenue growth for the Field Programmable Gate Array, Application-Specific Integrated Circuit and Application-Specific Standard Product markets for the next few years [1]. FPGAs look set to maintain the highest compound annual growth rate (almost 20%) as more and more applications adopt this solution due to their increasing functionality and decreasing costs and the often prohibitive development costs for ASICs and ASSPs. (ASSPs are essentially ASICs dedicated to a specific application market and sold to more than one user.) The total 2007 revenue from three families is expected to reach almost 100 billion US dollars.

Modern-day electronic design tools have evolved for a very good reason. The increasing size, speed, and complexity of these new products and the increasing pressure on bringing them to market quickly means that design targets would be impossible to reach without help from software programs.

Tool development has been continuous over the last thirty years. What traditionally was based on the two main axes of analog or digital designs has now been expanded to include software additions and we shall see how these previously distinct areas are becoming increasingly blurred and merging.

1.1 A brief history

Before electronic design automation, PCBs were designed by hand and manually laid out. Perhaps more surprisingly, this was also true for IC design. The first programs developed were drafting software that included some digitizing capabilities with different companies focusing on specific areas of the market. Calma was a major player in the area of IC manufacture (its GDSII format still exists) while other essentially similar products targeted the PCB sector. While invaluable in laying out a board or IC, there was still no link between the design process and the eventual layout. Various large companies developed their own in-house software and methodology solutions and some of these teams were eventually spun out as separate companies in the beginning of the 1980s. Various pioneers of the EDA industry date from this period: Mentor Graphics (original parent company Tektronix), Daisy Systems (Intel) and the forerunner of Cadence Design Systems, Valid Logic Systems (spawned from Hewlett Packard and the Lawrence Livermore Laboratory).

The need to simulate the transistors being laid out for all ICs was one of the inspirations for the development of the Simulation Program with Integrated Circuit Emphasis (SPICE) software at the University of California, Berkeley. The main driving forces behind the original SPICE program are credited to Larry Nagel working under the supervision of Donald Pederson. The work was based on the CANCER program and was one of the many such projects funded by the United States Department of Defense. Pederson insisted that the newly written SPICE be far enough removed from the original CANCER such that any distribution restrictions could be removed and that the program be put in the public domain.

The first public presentation of SPICE1 at a conference was in 1973 [2]. This was a relatively limited-feature program based on nodal analysis (leading to problems in describing some circuit elements, particularly inductors) and fixed time-step transient analysis (this aspect will be discussed in more detail later in this paper). SPICE2 was introduced in 1975 and offered improvements such as more inherent circuit elements, variable time-step analysis, and solutions based on modified nodal analysis. These were significant advances and ultimately led to the program's widespread adoption. SPICE2 was further developed until 1983 with the introduction of SPICE 2g6 which was the last FORTRAN version of the program. SPICE3 was written in C and was introduced in 1989.

The insistence on SPICE's open-source development (and famously, program availability for the cost of a magnetic tape) led to its becoming widely distributed and used as the basis for countless other simulators in the academic, industrial, and commercial worlds. Well-known commercial versions still existing today include HSPICE (now owned by Synopsys) and PSpice (currently owned by Cadence Design Systems).

Most 'digital' design tools also being developed during the same period were largely based on schematic-driven design entry. One of the next major advances was the development of text-based Hardware Description Languages (HDLs) describing design behaviour at the Register Transfer Level. RTL describes the operation of synchronous digital circuits where signals are treated before being clocked into hardware registers. HDLs allowing circuits to be described using RTL provide a much higher level of abstraction than at transistor or at gate level. Since the strengths and advantages of this level of abstraction were realized, tools were subsequently developed to allow direct and automatic gate synthesis from the HDL circuit description. This was a major step forward in the EDA industry and allowed a huge advance in design complexity and reduced time to market.

Two major HDLs became prominent during the mid-80s.

Verilog is a proprietary language defined by Gateway Design Automation. This was an industry changer and is still the prominent language in certain design areas. Written by Phil Moorby in around a year, it quickly became the de facto standard for ASIC design along with the Verilog-XL simulator. Its impact was reinforced when Synopsys introduced software permitting direct gate-level synthesis from the Verilog netlist. This allowed a direct line from design description to simulation and layout in one integrated flow. Gateway was eventually acquired by Cadence Design Systems during 1989. With the increasing market penetration of VHDL (described below), Cadence decided to make the language generally available through the Open Verilog International organization. The language was later submitted to the IEEE for standardization and this was eventually accorded as IEEE Standard 1364-1995. Subsequent developments of Verilog have culminated with the release of SystemVerilog.

The Very-High-Speed Integrated Circuits Hardware Description Language (commonly abbreviated to VHDL) was developed for the United States Department of Defense. It was becoming impossible to reproduce reliably the functionality of technically obsolete equipment as its specifications were insufficiently documented. VHDL was conceived as an attempt to overcome this limitation by defining a language that could formally and completely describe an object's specification. It was based on the existing ADA language syntax so as to benefit from already well-proven concepts. As with Verilog, the idea of being immediately able to simulate and synthesize the

specification was conceptually attractive and appropriate logic simulators and gate synthesis tools were subsequently developed.

Unfortunately, the advent of these two similar but competing languages led to a ‘Dark Ages’ period in the EDA world where it was unclear which language would prevail. VHDL was touted as the new ‘standard’ language in the industry but the already existing base of Verilog users guaranteed that Verilog remained and continues to be widely used.

The mid-80s also saw the introduction of Field Programmable Gate Array circuits. Originally developed by a Xilinx co-founder, the FPGA was based on complex programmable logic devices but offered more design flexibility at the expense of more difficult design effort and timing uncertainties. This type of circuit could be described by the existing HDLs along with the newly written synthesis programs. As noted in the paper’s opening section, FPGAs have become an important part of the market and seem set to at least maintain their share in the next few years.

1.2 Mixed-signal modelling

Modern circuitry often includes both analog and digital circuitry. The most obvious example of this is in a mobile phone where there is a radio-frequency front-end involving amplifiers and mixers before the downshifted signal is treated using digital signal processing. To simulate this complete circuit, some form of mixed-signal simulation is needed. In the extreme case, all circuit elements could be modelled and simulated at transistor level in SPICE but this would soon lead to unacceptably long simulation times for any design of complexity. Also, SPICE simulates a circuit by resolving Kirchhoff’s current and voltage laws and as such does not understand any signals that are non-conservative, e.g., signal flow. The program also supposes continuous time, so does not support the idea of discrete-event simulation. These are a few of the problems facing mixed-domain simulation solutions.

Digital-only simulators can easily perform many simulations very quickly. There is a time clock when all events are scheduled to happen, all circuits can be described as a list of time-scheduled queues, and all events are predictable.

A modern SPICE simulator faces a completely different set of problems. Knowing the value of a function at some point in time, it has to estimate the function at a future time. It does this by calculating an approximation to an analytical solution at discrete time points using numeric integration. One method would be to use the forward Euler method: from a given point, the next point is estimated from the slope of this known initial point:

$$y_{n+1} = y_n + hf(t_n, y_n). \quad (1)$$

This explicit method is mathematically very efficient but can, however, lead to stability problems. The backward Euler method is instead often used:

$$y_{n+1} = y_n + hf(t_{n+1}, y_{n+1}). \quad (2)$$

This implicit method is computationally more demanding but leads to more stable results. One method of calculating the value is to expand the Taylor series around the original point. For completeness, we mention that SPICE simulators often use the Runge–Kutta algebraic approximation to the Taylor series to evaluate this new value. This method is commonly used in other commercial programs such as the Simulink simulator that will be used during this school.

To reduce simulation times, SPICE uses a variable time-step during analysis. During periods of fast signal change, more calculations are needed to produce what is deemed as tolerably accurate

results. During periods of slow signal change, the time-step can be larger while still generating acceptable solutions. SPICE determines the ‘goodness’ of its results at each time-step from an evaluation of the resolved node voltages and currents:

$$|V(n) - V(n-1)| < VLIMIT = V(n) * Reltol + Vntol . \quad (3)$$

$$|I(n) - I(n-1)| < ILIMIT = I(n) * Reltol + Abstol . \quad (4)$$

If these inequalities are not satisfied, SPICE will reduce its time-step and again resolve the equations to produce another approximation (this recursive action is sometimes called ‘rollback’). If the simulator’s time-step can eventually not be further reduced to produce an acceptable solution, the solver will fail and signal that it has failed to converge. Experienced SPICE users will recognise not only the message but also the parameters used in Eqs. (3) and (4). The values for *Reltol*, *Vntol* and *Abstol* can be adjusted (with care) from within the simulator to overcome such convergence problems.

The preceding paragraph highlights the problem of performing mixed-signal simulations — digital simulation is done using fixed, well-defined time-steps while analog simulation uses variable time-steps and getting both simulators working together is not trivial for efficient mixed-signal simulations. The situation is complicated by rollback where it is possible that a re-calculation of the SPICE values can affect what were previously correct digital values. Some techniques have been used to overcome these problems:

- Lockstepping the simulators such that the two engines are locked together in time throughout the simulation. This means that the simulation time is dependent on the smallest time-step and can lead to very inefficient use if the analog circuit is non-trivial.
- Using ‘backplanes’ between simulators where each simulator connects to a virtual software interfacing bus.
- Other proprietary techniques, e.g., the Calaveras algorithm. This allows one simulator to run ahead of the other. If the simulators subsequently determine that evaluations in one domain would have affected the other, the affected simulator ‘rewinds’ and repeats simulation with the new data. The technique can often lead to the discarding of large amounts of data which again reduces efficiency.

Despite these difficulties, mixed-domain simulation solutions are available and indeed, both the Verilog and VHDL languages have been extended to include ‘Analog and Mixed-Signal’ capabilities (Verilog-AMS, VHDL-AMS).

1.3 System-level design

The necessity for mixed-signal simulation led to the development and use of such solutions. However, modern designs can also involve a large amount of circuitry along with a large amount of embedded software (the mobile phone is again an example). Traditionally, hardware and software have been developed concurrently but distinctly from a single specification. This methodology has several drawbacks at different levels. Some of them follow:

- The translation of the specification requirements to the hardware and software branches is done separately and manually.
- What is being tested in the hardware and software implementations is what has been understood as the design intent after this translation — this might not be the same as the original specification.

- There is no obvious link between the hardware and software.

These limitations have been appreciated for many years and have inspired the development of *Electronic System Level* tools. Unfortunately, ESL design does not have a well-defined meaning but most people seem to agree that the concept makes possible:

- fast exploration of the solution space before detailed simulation (top-down design)
- easy change of the level of model sophistication
- co-simulation of hardware and software system behaviour

One other great advantage of a system level design would be if the original specification were executable. This is conceptually a very attractive idea — the specification itself would act as the model that can be used in the design and by other tools for simulation, verification and eventual synthesis. This obviously removes one important drawback in the traditional flow where what is built is what was perceived as the intention from the specification and not the specification itself.

The flexibility and availability of a language such as C++ would seem to make it a good basis as a system level design language. However, C++ is based on sequential programming so it is unsuited to modelling concurrent processes and also lacks the notion of time — both notions being essential to model successfully at the hardware level. Other essential features for system modelling such as hardware data types and definitions for signals and ports are also missing.

Attempts to remove these limitations have been made in SystemC. SystemC defines a set of libraries for C++ containing the constructs and core language elements needed to perform hardware modelling in C++. This allows hardware to be described using C++ syntax and to be compiled into an executable that will behave as the modelled hardware when run. The required SystemC libraries have been made generally available by the Open SystemC Initiative. OSCI also makes available a reference simulator, although arguably more advanced commercial versions are available. EDA vendors also offer mixed-language possibilities allowing SystemC models to be simulated with previously defined Verilog or VHDL models. This improves performance as SystemC simulation speed at RTL level is not yet at the level of other HDL simulators.

Different levels of model abstraction can be described in SystemC (though is not limited to SystemC) through Transaction Level Modelling. TLM enables modelling at a far higher level of abstraction than RTL. Using TLM, the communication between modules is separated, and can be developed independently from the description of the functionality of the modules. Briefly, channels are used to model communication between components and transaction requests occur by calling interface functions to these channels. Taking the example of a design's system bus, this would be represented by such a channel defined as part of a SystemC interface class. Transaction requests then take place through these channel models through this interface. This makes it easy for the designer to explore different bus architecture possibilities as long as all models interact with the different bus representations through the common interface. The use of this common interface also allows models to be easily refined to include more functional and timing detail. The different levels of refinement are often called views and which one was used during simulation depends on the level of detail required at that design stage — the more detailed the model, the longer the simulation time. Unfortunately, there is no standard TLM nomenclature but one set of definitions is

Functional View – The design focuses on algorithm development and not implementation. Timing is not a concern.

Programmer's View – The hardware models are functionally correct but timed only with sufficient detail to allow use by the software developers.

Architect's View – The level at which architectural exploration and optimization is conducted. Exploration is again done using approximate timing.

Verification View – The hardware models are refined to include timing detail and are cycle-accurate.

Figure 1 graphically represents the different available possibilities for TLM refinements (at the time of writing of this paper, the TLM-2 Draft was under discussion by OSCI to try and define at which abstract-level each view should be modelled).

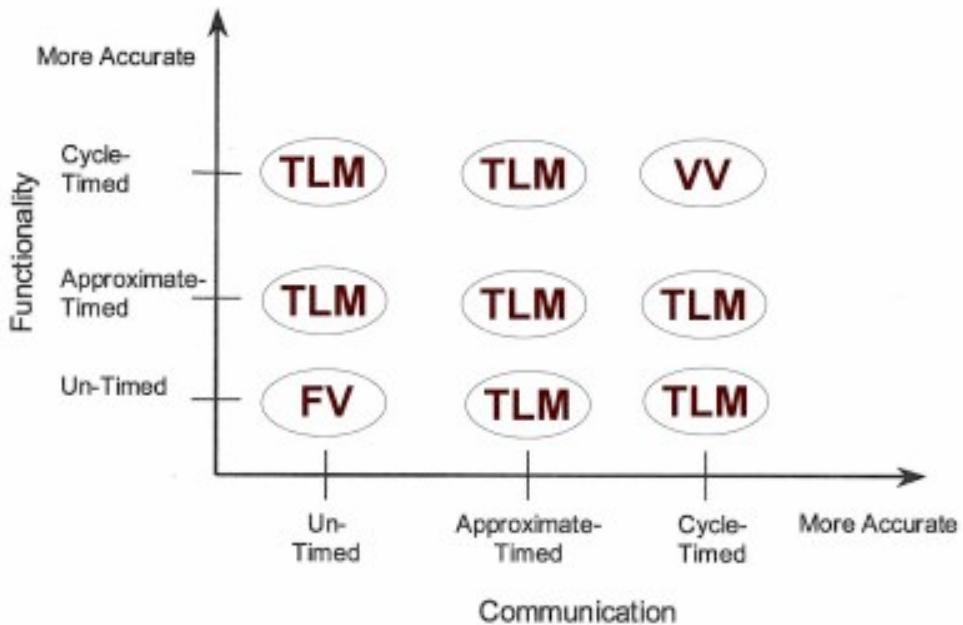


Fig. 1: Abstraction level for transaction level modelling

MATLAB is another program that is widely used and can be considered as an ESL language. Typically used for DSP and control algorithm developments, it has suffered as there was no easy method of translating proprietary MATLAB or Simulink code into equivalent RTL. However, recently introduced toolboxes and applications have eased this problem, making for a more complete design flow. Third-party applications (such as Xilinx's SysGen as used in this school) have also been developed that allow direct implementation of The Mathworks Simulink designs into FPGAs.

Other existing languages used with different levels of penetration include Rosetta, Esterel, Specification and Description Language, XML and UML.

1.4 Design verification

The ultimate aim of verification in the case of electronic design is to check if a completed design conforms to the original specification. The overall verification procedure should answer the following questions:

- What is the design intent?
- What does the design actually do?
- Do the two things match?
- How good is the testing?

With increased design complexity, a substantial part of the overall design effort is involved with circuit verification. Some estimates place it as high as 70% and large companies often have separate verification teams that are involved only with this aspect of the project. These verification teams often work separately from their design colleagues as they are essentially involved at different levels of testing. The verification engineers work at ‘black-box’ level. They are interested only with verifying behaviour against the original specification and are unconcerned with the actual design implementation. Design engineers work at the ‘white-box’ level; they understand fully the inner workings of the design and test at a more detailed level. While seemingly an inefficient way to operate, this working method can help to highlight any differences in the interpretation of the original specification.

Simulation has traditionally been the most common method used to test a design but this method suffers from some disadvantages:

- Setting up a testbench to verify each desired functionality can be time-consuming and a non-trivial matter.
- Writing a full testbench to cover all possible states for large designs is practically impossible.
- No indication of how well the design has been tested.

To help overcome these problems, special verification languages such as Vera and Property Specification Language have been developed to allow easier and fuller design testing, often using constructs called *properties* and *assertions*.

A *property* is a Boolean expression defined in the syntax of the HDL used to describe the model and can also include some extra temporal information.

An *assertion* is used to ask a verification language to evaluate a given property.

The following is a simple PSL property example.

```
property Nooverflow is always (Mydata < 256);
```

We have defined a property *Nooverflow* that states that *Mydata* must be always less than 256 (*always* is a temporal keyword in this example). We can then include an assertion in our code to ask that the simulator verify that this is true:

```
assert Nooverflow;
```

Properties and assertions can be used during simulation to check for specific behaviour and are a very valuable addition to the available tools and techniques. However, verifying assertions adds simulation overhead so care should be taken to use them at an appropriate level.

Generally, the more different the test performed during simulation, the better the verification coverage. Verification tools can be set up to create automatically a set of testbenches to improve the coverage while relieving the designer of the task. A random seed is used as an input to the automation process and a testbench created from this arbitrary value. Properties and assertions can then be used in this context to check if this newly created testbench is valid or not.

Figure 2 shows an example using SystemC (and its associated verification capabilities) where the *addr* variable is limited between 20 and 50, and data has to be *addr+10*.

```

#include "scv.h"

struct bus_constraint : public scv_constraint base {
    scv_smart_ptr<sc_unit<32>> addr;
    scv_smart_ptr<sc_uint<32>> data;
    scv_smart_ptr<bool> r_wn;

    SCV_CONSTRAINTCTOR(bus_constraint) {
        SCV_CONSTRAINT( (addr() > 20 && addr () < 50 );
        SCV_CONSTRAINT( data() < ( addr() + 10 ) );
    }
};

```

Fig. 2: Example of constrained values using SystemC verification libraries

Constrained random data generation can help enormously in setting up and running a large number of simulation tests. However, it is impossible to know which addresses and data values are generated until the test is run. Recording the actual values created helps to provide verification coverage metrics. Assertions can also be included to determine if a particular test verified a given design feature, and to what extent. This information can again be recorded to help provide a more complete coverage metric. The gathered data can then be used to determine if all cases have been covered and whether any more tests should be run to complete the testing. This overall procedure is known as coverage-driven verification.

While CDV can help to cover more possible simulation cases, it is still not an exhaustive test. A technique called Formal Model Checking can be employed to guarantee full coverage.

FMC performs a mathematical analysis of all the possible states of a design to determine if any violate the included properties or assertions. This technique is a full, rigorous, formal analysis so there is no need to write a testbench. While it would be theoretically possible to use FMC to verify fully the functionality of any arbitrarily complex design, it is typically used for sub-blocks of a design and used in conjunction with simulation. If FMC has been used to prove that a given sub-block behaves correctly for a set of inputs and the sub-block is only driven by inputs in this range during simulation, the sub-block can be assumed to function correctly in the complete system. For completeness, it should be noted that in some cases, the same assertion used in a simulation run and in a formal analysis verification can return different results, e.g., it will pass in one test and signal a failure in the other analysis. Tool vendors overcome this problem either by supplying carefully defined assertions that are known to be consistent in both cases or by cross-checking the assertions by running both analyses.

An interesting variation using both FMC and simulation is known as Dynamic Formal Verification (DFV). This is particularly useful for known ‘dangerous’ cases, e.g., FIFO is full. For example, using constrained driven verification only, FIFO_full might be tested but some other condition at the same time could cause a bug (it would be unlikely that both conditions are present using CDV alone). DFV recognises that this hazardous situation has occurred and then instigates a formal analysis on all states leading up to and beyond this time to identify any possible problems. This helps to target verification time on potentially dangerous situations.

Recording the outputs of the analyses above is used as an input to provide a functional coverage metric. This can be used to determine when we have performed ‘enough’ tests. Another metric used to assess verification completeness is Code Coverage. This is invariably measured as it is conceptually very simple and essentially reports how much of the HDL code has been exercised during testing. This is a rather simple test but can guarantee that no dead code is synthesized. While easy to gain a set of metrics for the coverage, it can unfortunately return an artificially high pass-rate, for example, exercising a circuit’s ‘reset’ can cause many lines of code to be accessed. These lines will therefore be deemed as having passed though there is no guarantee that they will be exercised again during the verification process.

Code and Functional coverage tools should be considered as complementary in the verification procedure.

2 Conclusion and future developments

We have seen how tools have evolved to solve the immediate problems faced by design engineers. We have also seen how the gradual merging between analog and digital hardware and software designing is gradually taking place. Industry has also recognized the difficulties faced to verify sufficiently the very large designs currently being manufactured.

Specialized design point-tools such as MATLAB have become important in some niche areas. They have traditionally been used as algorithm development tools only, but their capabilities have recently been extended to allow direct FPGA implementation from the model description.

SystemC has been mooted as an industry-standard solution covering all of the above design aspects. It provides a good system level design solution and, since its introduction, has been extended to include verification capabilities using SCV. It has a strength in that it is founded on C++, a well-known and widely-used language. However, simulation at hardware level is much faster using one of the standard HDL languages and their associated simulators. Hardware engineers also are not necessarily familiar with C++ and this might contribute to the fact that SystemC uptake is currently not as extensive as was once forecast.

The verification bottleneck has led to the development of specialized languages focusing on this problem. These have become rather well used (due to necessity) but they again tend to suffer from the fact that their syntax is different from the underlying HDLs.

Verilog and VHDL are continually being revised to address the new problems faced by the industry.

VHDL is currently being considered to include stronger verification capabilities.

Verilog has been extended to SystemVerilog which became an IEEE standard in 2005. SystemVerilog combines not only Verilog language updates (including stronger typing as found in VHDL) but also includes system-level and verification capabilities. It is significant in that it is the industry’s first unified hardware design language — for the first time a common HDL can be used to cover all aspects of the design flow from description to verification.

This could lead the language to become the de facto industry-standard HDL but, as ever, this remains to be seen.

Acknowledgement

Extensive use has been made of the Internet and various texts to prepare this paper but this article is presented as original work. Any errors or omissions of acknowledgements are entirely accidental and, should any have occurred, full apologies are extended. Various well-known commercial tools have been mentioned in this article and their trademarks and any other copyright rights are acknowledged.

The author would like to thank Synplify, The Mathworks, and Xilinx companies for graciously offering their software for use during this school and for their help in installing the programs.

References

- [1] <http://electronicdesign.com/Articles/Print.cfm?AD=1&ArticleID=14442>.
- [2] L.W. Nagel and D.O. Pederson, SPICE (Simulation Program with Integrated Circuit Emphasis), Memorandum No. ERL-M382, University of California, Berkeley, Apr. 1973.

Digital signal processing mathematics

M. Hoffmann

DESY, Hamburg, Germany

Abstract

Modern digital signal processing makes use of a variety of mathematical techniques. These techniques are used to design and understand efficient filters for data processing and control. In an accelerator environment, these techniques often include statistics, one-dimensional and multidimensional transformations, and complex function theory. The basic mathematical concepts are presented in four sessions including a treatment of the harmonic oscillator, a topic that is necessary for the afternoon exercise sessions.

1 Introduction

Digital signal processing requires the study of *signals* in a digital representation and the methods to interpret and utilize these signals. Together with analog signal processing, it composes the more general modern methodology of signal processing. Although the mathematics that are needed to understand most of the digital signal processing concepts were developed a long time ago, digital signal processing is still a relatively new methodology. Many digital signal processing concepts were derived from the analog signal processing field, so you will find a lot of similarities between the digital and analog signal processing. Nevertheless, some new techniques have been necessitated by digital signal processing, hence, the mathematical concepts treated here have been developed in that direction. The strength of digital signal processing currently lies in the frequency regimes of audio signal processing, control engineering, digital image processing, and speech processing. Radar signal processing and communications signal processing are two other subfields. Last but not least, the digital world has entered the field of accelerator technology. Because of its flexibility, digital signal processing and control is superior to analog processing or control in many growing areas.

Around 1990, diagnostic devices in accelerators began to utilize digital signal processing, e.g., for spectral analysis. Since then, the processing speed of the hardware [mostly standard computers and digital signal processors (DSPs)] has increased very quickly, such that now *fast* RF control is now possible. In the future, direct sampling and processing of all RF signals (up to a few GHz) will be possible, and many analog control circuits will be replaced by digital ones.

The design of digital signal processing systems without a basic mathematical understanding of the signals and its properties is hardly possible. Mathematics and physics of the underlying processes need to be understood, modelled, and finally controlled. To be able to perform these tasks, some knowledge of trigonometric functions, complex numbers, complex analysis, linear algebra, and statistical methods is required. The reader may look them up in his undergraduate textbooks if necessary.

The first session covers the following topics: the dynamics of the harmonic oscillator and signal theory. Here we try to describe what a signal is, how a digital signal is obtained, and what its quality parameters, accuracy, noise, and precision are. We introduce *causal time invariant linear systems* and discuss certain fundamental special functions or signals.

In the second session we are going to go into more detail and introduce the very fundamental concept of convolution, which is the basis of all digital filter implementations. We are going to treat the Fourier transformation and finally the Laplace transformation, which are also useful for treating analog signals.

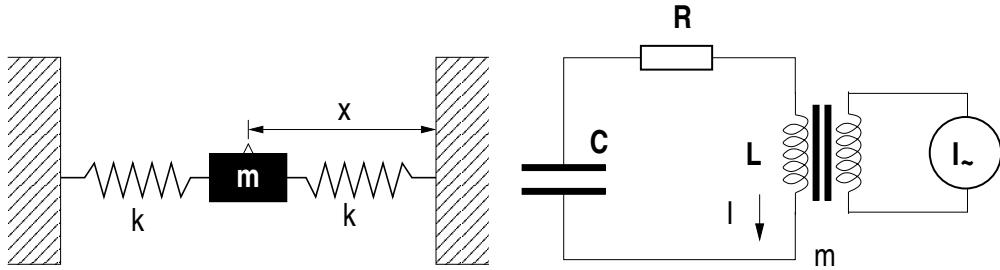


Fig. 1: Principle of a physical pendulum (left) and of an electrical oscillator

The third session will make use of the concepts developed for analog signals as they are applied to digital signals. It will cover digital filters and the very fundamental concept and tool of the z -transformation, which is *the* basis of filter design.

The fourth and last session will cover more specialized techniques, like the Kalman filter and the concept of wavelets. Since each of these topics opens its own field of mathematics, we can just peek at the surface to get an idea of its power and what it is about.

2 Oscillators

One very fundamental system (out of not so many others) in physics and engineering is the harmonic oscillator. It is still simple and linear and shows various behaviours like damped oscillations, resonance, bandpass or band-reject characteristics. The harmonic oscillator is, therefore, discussed in many examples, and also in this lecture, the harmonic oscillator is used as a work system for the afternoon lab-course.

2.1 What you need to know about...

We are going to write down the fundamental differential equation of all harmonic oscillators, then solve the equation for the steady-state condition. The dynamic behaviour of an oscillator is also interesting by itself, but the mathematical treatment is out of the scope of this lecture. Common oscillators appear in mechanics and electronics, or both. A good example, where both oscillators play a big role, is the accelerating cavity of a (superconducting) linac. Here we are going to look at the electrical oscillator and the mechanical pendulum (see Fig. 1).

2.1.1 The electrical oscillator

An R-L-C circuit is an electrical circuit consisting of a resistor (R), an inductor (L), and a capacitor (C), connected in series or in parallel (see Fig. 1, right).

Any voltage or current in the circuit can be described by a second-order linear differential equation like this one (here a voltage balance is evaluated):

$$\begin{aligned}
 RI + L\dot{I} + \frac{Q}{C} &= mI_\sim \\
 \Leftrightarrow \quad \ddot{I} + \frac{R}{L}\dot{I} + \frac{1}{LC}I &= KI_\sim . \tag{1}
 \end{aligned}$$

2.1.2 Mechanical oscillator

A mechanical oscillator is a pendulum like the one shown in Fig. 1 (left). If you look at the forces which apply to the mass m you get the following differential equation:

$$m\ddot{x} + \kappa\dot{x} + kx = F(t)$$

$$\Leftrightarrow \ddot{x} + \frac{k}{m}\dot{x} + \frac{\kappa}{m}x = \frac{1}{m}F(t) . \quad (2)$$

This is also a second-order linear differential equation.

2.1.3 The universal differential equation

If you now look at the two differential equations (1) and (2) you can make them look similar if you bring them into the following form (assuming periodic excitations in both cases):

$$\boxed{\ddot{x} + 2\beta\dot{x} + \omega_0^2x = Te^{i(\omega_{\sim}t+\xi)}}, \quad (3)$$

where T is the excitation amplitude, ω_{\sim} the frequency of the excitation, ξ the relative phase of the excitation compared to the phase of the oscillation of the system (whose absolute phase is set to zero),

$$\beta = \frac{R}{2L} \quad \text{or} \quad \frac{k}{2m}$$

is the term which describes the dissipation which will lead to a damping of the oscillator and

$$\omega_0 = \frac{1}{\sqrt{LC}} \quad \text{or} \quad \sqrt{\frac{\kappa}{m}}$$

gives you the eigenfrequency of the resonance of the system.

Also one very often uses the so-called *Q-value*

$$Q = \frac{\omega_0}{2\beta} \quad (4)$$

which is a measure for the energy dissipation. The higher the *Q*-value, the less the dissipation, the narrower the resonance, and the higher the amplitude in the case of resonance.

2.2 Solving the DGL

For solving the second-order differential equation (3), we first do the following ansatz:

$$\begin{aligned} x(t) &= Ae^{i(\omega t+\phi)} \\ \dot{x}(t) &= i\omega A e^{i(\omega t+\phi)} \\ \ddot{x}(t) &= -\omega^2 A e^{i(\omega t+\phi)} . \end{aligned}$$

By inserting this into (3) we get the so-called *characteristic equation*:

$$\begin{aligned} -\omega^2 A e^{i(\omega t+\phi)} + 2i\omega\beta A e^{i(\omega t+\phi)} + \omega_0^2 A e^{i(\omega t+\phi)} &= T e^{i(\omega_{\sim}t+\xi)} \\ \Leftrightarrow -\omega^2 + 2i\omega\beta + \omega_0^2 &= \frac{T}{A} e^{i((\omega_{\sim}-\omega)t+(\xi-\phi))} . \end{aligned}$$

In the following, we want to look only at the special solution $\omega \stackrel{!}{=} \omega_{\sim}$ (o.B.d.A $\xi = 0$), because we are only interested in the steady state, for which we already know that the pendulum will take over the

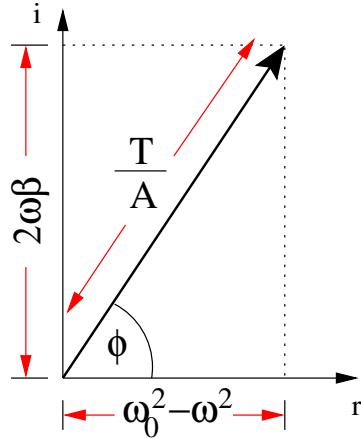


Fig. 2: Graphical explanation of the characteristic equation in the complex plane

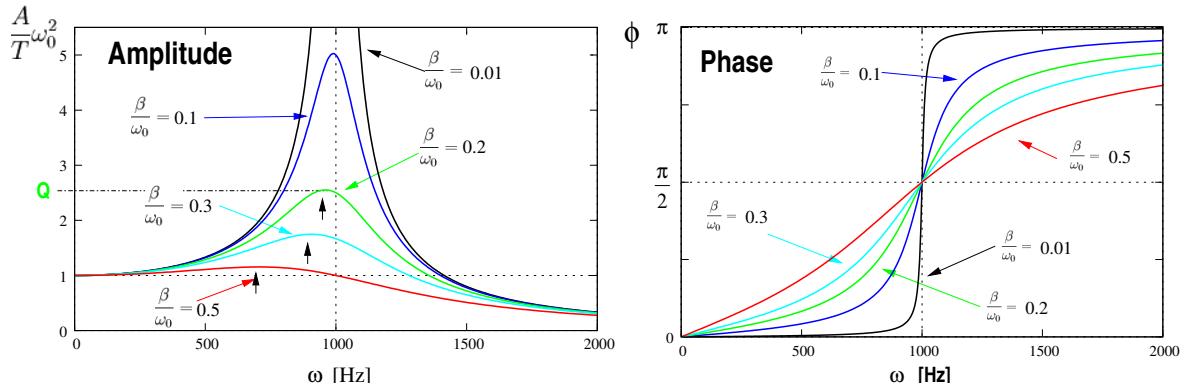


Fig. 3: Amplitude and phase of the excited harmonic oscillator in steady state

excitation frequency. Since we are only interested in the phase difference of the oscillator with respect to the excitation force, we can set $\xi = 0$.

In this (steady) state, we can look up the solution from a graphic (see Fig. 2). We get one equation for the amplitude

$$\begin{aligned} \left(\frac{T}{A}\right)^2 &= (\omega_0^2 - \omega^2)^2 + (2\omega\beta)^2 \\ \Leftrightarrow A &= T \frac{1}{\sqrt{(\omega_0^2 - \omega^2)^2 + 4\omega^2\beta^2}} \end{aligned}$$

and another for the phase

$$\tan(\phi) = \frac{2\omega\beta}{\omega_0^2 - \omega^2}$$

of the solution $x(t)$.

Both formulas are visualized in Fig. 3 as a function of the excitation frequency ω . Amplitude and phase can also be viewed as a complex vector moving in the complex plane with changing frequency. This plot is shown in Fig. 4. You should notice that the Q -value gets a graphical explanation here. It is linked to the bandwidth $\omega_{1/2}$ of the resonance by

$$\omega_{1/2} = \beta = \frac{\omega_0}{2Q},$$

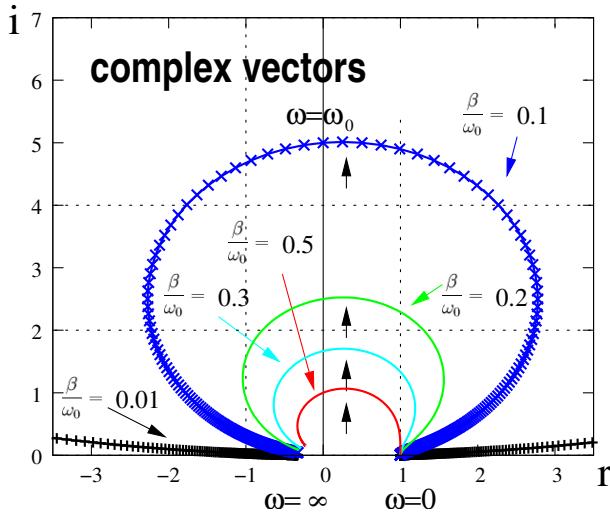


Fig. 4: Complex vector of the harmonic oscillator moving with frequency for different Q values

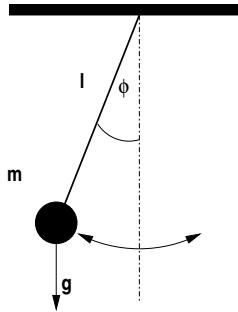


Fig. 5: The gravity pendulum. A mass m oscillates in the gravity field.

and this also gives

$$Q = \frac{\omega_0}{2\beta} = \left[\left| \frac{A}{T} \right| \omega^2 \right]_{\omega=\omega_0},$$

a relation to the height of the resonance peak.

2.3 Non-linear oscillators

Besides the still simple harmonic oscillator described above, which is a linear oscillator, many real oscillators are non-linear or at least linear only in approximation. We are going to discuss two examples of simple looking non-linear oscillators. First the *mathematical pendulum*, which is linear in good approximation for small amplitudes, and a yo-yo-like oscillator which is non-linear even for small oscillations.

2.3.1 The mathematical pendulum

The differential equation which represents the approximate motion of the simple gravity pendulum shown in Fig. 5 is

$$ml\ddot{\phi} + \kappa\dot{\phi} - mg \sin(\phi) = F(t),$$

where κ is the dissipation term (coming from friction from the air).

The problem with this equation is that it is unintegrable. But for small oscillation amplitudes, one can approximate: $\sin(\phi) = \phi$ and treat it as the harmonic, linear mechanical pendulum described in the

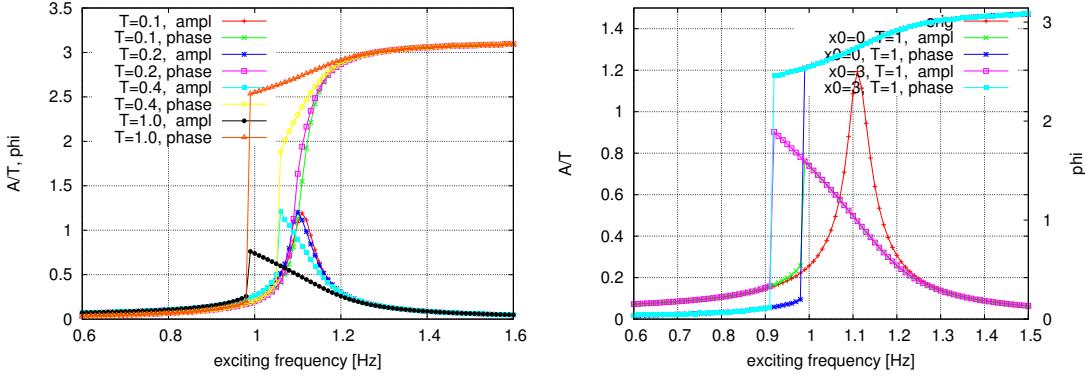
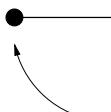


Fig. 6: Simulated behaviour of the mathematical pendulum

previous section. But what if we have large amplitudes like

like  ? or even a rotation of the pendulum


Well, this system is unbounded (rotation can occur instead of oscillation) and so the behaviour is obviously amplitude dependent. We especially expect the resonance frequency to be a function of the oscillation amplitude, $\omega = F(A)$. At least, we can still assume $\omega = \omega_\sim$ for the steady state solution; this means that the system will follow the excitation frequency after some time.

Figure 6 shows the simulated behaviour of the mathematical pendulum in the steady state. You can see the single resonance peak, which for small amplitudes looks very similar to the one seen in Fig. 3. For larger amplitudes, however, this peak is more and more bent to the left. When the peak hangs over¹, a jump occurs at an amplitude-dependent excitation frequency, where the system can oscillate with a small amplitude and then suddenly with a large amplitude. To make things even worse, the decision about which amplitude is taken by the system depends on the amplitude the system already has. Figure 6 (right) shows that the jump occurs at different frequencies, dependent on the amplitude x_0 at the beginning of the simulation.

Last but not least, *coupled* systems of that type may have a very complicated dynamic behaviour and may easily become chaotic.

2.3.2 The yo-yo

Another strongly non-linear oscillator is the one known as the yo-yo and which is in principle identical to the system shown in Fig. 7.

The differential equation of this system expresses like:

$$\frac{m}{\cos(\alpha)} \ddot{x} + \kappa \dot{x} - \text{sgn}(x) \cdot mg \sin(\alpha) = F(t) ,$$

¹A similar emergence can be observed for superconducting cavities: Lorentz force detuning.

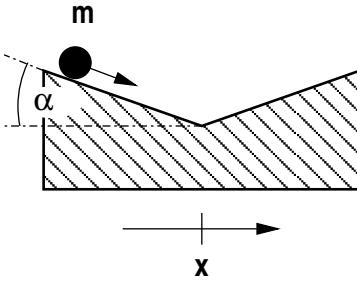


Fig. 7: The yo-yo. A mass m on the inclined plane. For simplicity, the rotation of the ball is not considered here.

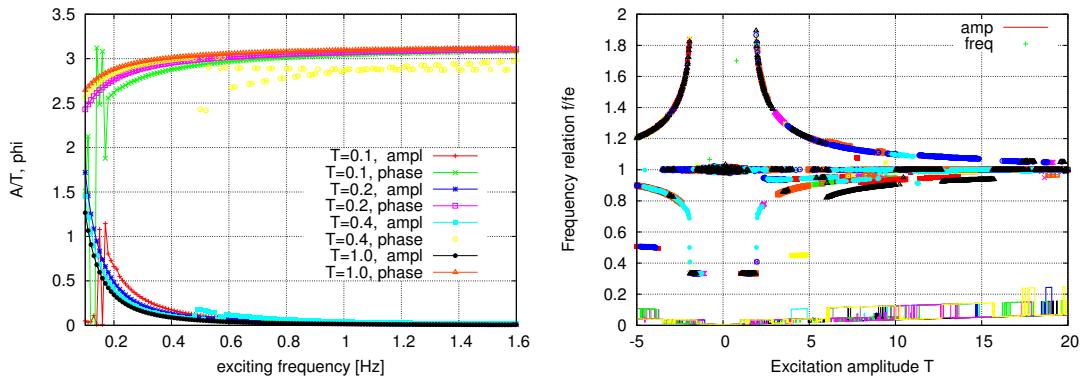


Fig. 8: Simulated frequency response of the yo-yo for different excitation frequencies and amplitudes (left). On the right you can see different oscillation modes of this system depending on the excitation amplitude for different excitation frequencies. The system responds with different oscillation frequencies in an unpredictable manner.

where

$$\operatorname{sgn}(x) := \begin{cases} \frac{x}{|x|} & x \neq 0 \\ 0 & x = 0 \end{cases}.$$

Now let us answer the questions: Is there a resonance? And if so, what is the resonance frequency?

Obviously, the resonance frequency here would also be highly amplitude-dependent ($\omega_0 \stackrel{!}{=} f(A)$) because it takes longer for the ball to roll down the inclined plane if it starts with a bigger amplitude. But if we look at the simulated frequency response with different excitation amplitudes (see Fig. 8) it looks like there is a resonance at 0 Hz!?

Looking closer at the situation one finds that the oscillation frequency can differ from the excitation frequency: $\omega \neq \omega_0$. Figure 8 (right) shows all possible oscillation frequencies (in relation to the excitation frequency) with different starting amplitudes x_0 (colours) under excitation with different amplitudes. The system responds with oscillations in an unpredictable manner.

Now you know why linear systems are so nice and relatively easy to deal with.

3 Signal theory

The fundamental concepts we want to deal with for digital signal processing are *signals* and *systems*. In this section we want to develop the mathematical understanding of a signal in general, and more specifically look at the *digital* signals.

3.1 Signals

The signal $s(t)$ which is produced by a measurement device can be seen as a real, time-varying property (a function of time). The property represents physical observables like voltage, current, temperature, etc. Its instant power is defined as $s^2(t)$ (all proportional constants are set to one²).

The signal under investigation should be an *energy signal*, which is

$$\int_{-\infty}^{\infty} s^2(t) dt < \infty . \quad (5)$$

This requires that the total energy content of that signal be finite. Most of the elementary functions (e.g., $\sin()$, $\cos()$, $\text{rect}()$, ...) are not energy signals, because they ideally are infinitely long, and the integral (5) does not converge. In this case one can treat them as *power signals*, which requires

$$\lim_{T \rightarrow \infty} \int_{-T/2}^{T/2} s^2(t) dt < \infty . \quad (6)$$

(The energy of the signal is finite for any given time interval.) Obviously $\sin()$ and $\cos()$ are signals which fullfil the relation (6).

Now, what is a physical signal that we are likely to see? Well, wherever the signal comes from, whatever sensor is used to measure whatever quantity, in the end — if it is measured electrically — we usually get a voltage as a function of time $U(t)$ as (input) signal. This signal can be *discrete* or *continuous, analog or digital, causal or non-causal*. We shall discuss these terms later.

From the mathematical point of view we have the following understanding/definitions:

- Time: $t \in \mathbb{R}$ (sometimes $\in \mathbb{R}_0^+$)
- Amplitude: $s(t) \in \mathbb{R}$ (usually a voltage $U(t)$)
- Power: $s^2(t) \in \mathbb{R}_0^+$ (constants are renormed to 1)

Since the goal of *digital signal processing* is usually to measure or filter continuous, real-world analog signals, the first step is usually to convert the signal from an analog to a digital form by using an *analog-to-digital converter*. Often the required output is another analog signal, so a *digital-to-analog converter* is also required.

The algorithms for signal processing are usually performed using specialized electronics, which either make use of specialized microprocessors called *digital signal processors* (DSPs) or they process signals in real time with purpose-designed *application-specific integrated circuits* (ASICs). When flexibility and rapid development are more important than unit costs at high volume, digital signal processing algorithms may also be implemented using *field-programmable gate arrays* (FPGAs).

Signal domains

Signals are usually studied in one of the following domains:

1. time domain (one-dimensional signals),
2. spatial domain (multidimensional signals),
3. frequency domain,
4. autocorrelation domain, and

²For example, the power considering a voltage measurement would be $P = U^2/R$, considering a current measurement $P = I^2 R$, so we can set $R := 1$ and get the relations $P = U^2$ or $P = I^2$.

5. wavelet domains.

We choose the domain in which to process a signal by making an informed guess (or by trying different possibilities) as to which domain best represents the essential characteristics of the signal. A sequence of samples from a measuring device produces a time or spatial domain representation, whereas a discrete Fourier transform produces the frequency domain information, the frequency spectrum. Autocorrelation is defined as the cross-correlation of the signal with itself over varying intervals of time or space. Wavelets open various possibilities to create localized bases for decompositions of the signal. All these topics will be covered in the following sections. First we are going to look at how one can obtain a (digital) signal and what quantities define its quality. Then we are going to look at special fundamental signals and linear systems which transform these signals.

Discrete-time signals

Discrete-time signals may be *inherently discrete-time* (e.g., turn-by-turn beam position at one monitor) or may have originated from the sampling of a continuous-time signal (**digitization**). Sampled-data signals are assumed to have been sampled at periodic intervals T . The sampling rate must be sufficiently high to extract all the information in the continuous-time signal, otherwise **aliasing** occurs. We shall discuss issues relating to amplitude **quantization**, but, in general, we assume that discrete-time signals are continuously valued.

3.2 Digitization

The digitization process makes out of an analog signal $s(t)$ a series of *samples*

$$s(t) \longrightarrow s_n := s[n] := s(nT) \quad n \in \mathbb{Z} (\text{ sometimes } \in \mathbb{N}_0)$$

by choosing discrete sampling intervals $t \longrightarrow nT$ where T is the period.

The sampling process has two effects:

1. time discretization (sampling frequency) $T = 1/f_s$ and
2. quantization (AD conversion, integer/float).

The second effect must not be neglected, although in some cases there is no special problem with this if you can use a high enough number of bits for the digitization. Modern fast ADCs have 8, 14 or 16 bits resolution. High-precision ADCs exist with 20 or even more effective bits, but they are usually much slower. Figure 9 illustrates the digitization process.

Dithering

Because the number of bits of ADCs is a cost issue, there is a technique called *dithering* which is frequently used to improve the (amplitude) resolution of the digitization process. Surprisingly, it makes use of noise which is added to the (analog) input signal. The trick is that you can subtract the noise later from the digital values, assuming you know the exact characteristics of the noise, or even better, you produce it digitally using a DAC, and therefore know the value of each noise sample. This technique is illustrated in Fig. 10.

3.3 Causal and non-causal signals

A signal is *causal* if (at any time) only the *present* and *past* values of that signal are known.

given $x[t_n]$ where $t_0 := \text{present}$, $n < 0 : \text{future}$, $n > 0 : \text{past}$

So if $x[t_n] = 0 \quad \forall n < 0$ the *system* under investigation is causal.

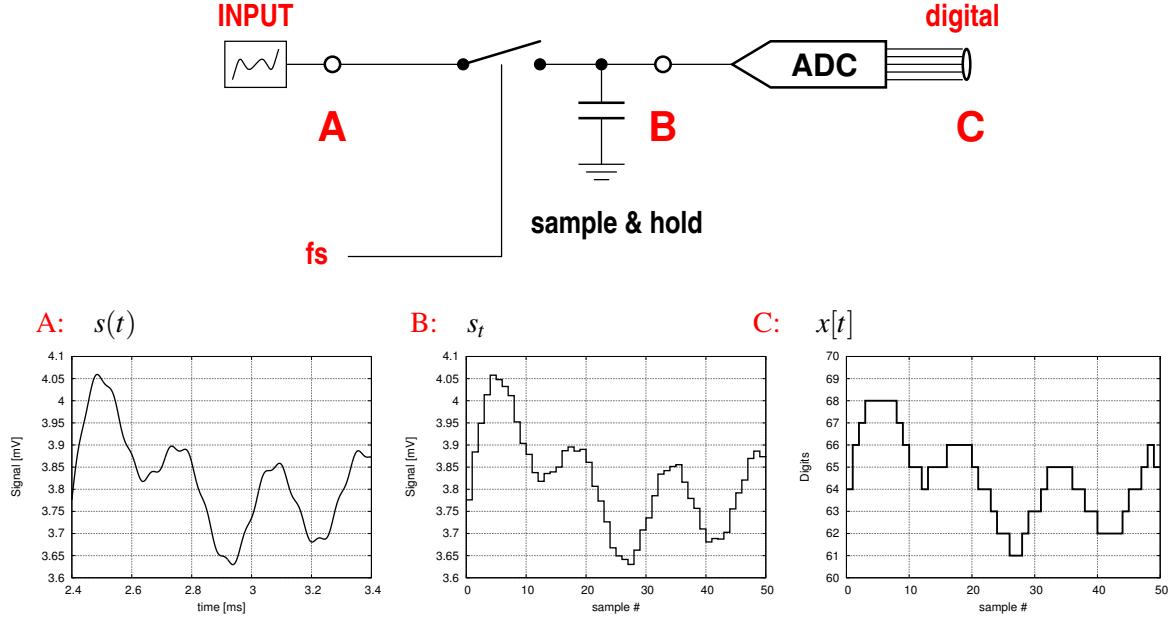


Fig. 9: The digitization process is done in two steps: First, samples are taken from the analog input signal (A). The time discretization is done with the sampling frequency f_s . The voltage is stored in a sample-and-hold device (B) (a simple capacitor can do). Finally the voltage across the capacitor is converted into a digital number (C), usually represented by n bits of digital logic signals. The digital representation of the input signal is not perfect (as can be seen on the bottom plots) as it has a limited resolution in both time and amplitude.

The only situation where you may encounter non-causal signals or non-causal algorithms is under the following circumstances: Say, a whole chunk of data has been recorded (this can be the whole pulse train in a repetitive process or the trace of a pulse of an RF system). Now you want to calculate a prediction for the next measurement period from the last period's data. From some viewpoint, this data is seen as a non-causal signal: If you process the data sample by sample, you always have access to the whole dataset, which means you can also calculate with samples before the sample actually processes. You can thereby make use of non-causal algorithms, because from this algorithm's perspective your data also contains the future. But from the outside view, it is clear that it does not really contain the future, because the whole chunk of data has been taken in the past and is now processed (with a big delay). A measurement can not take information from the future! Classically, nature or physical reality has been considered to be a causal system.

3.3.1 Discrete-time frequency units

In the discrete world, you deal with numbers or digits instead of voltage, with sample number instead of time, and so we ask what is the discrete unit of frequency? Let us go straightforward starting with an analog signal:

$$x(t) = A \cdot \cos(\omega t) =: A \cdot \cos(2\pi f_c t),$$

sampling at intervals $T = \frac{1}{f_s} = \frac{2\pi}{\omega_s}$ leads to

$$\begin{aligned} \Rightarrow x[n] &= A \cdot \cos(\omega n T) \\ &= A \cdot \cos(n \frac{\omega}{f_s}) = A \cdot \cos(n \frac{2\pi\omega}{\omega_s}) \\ &=: A \cdot \cos(\omega_d n), \end{aligned}$$

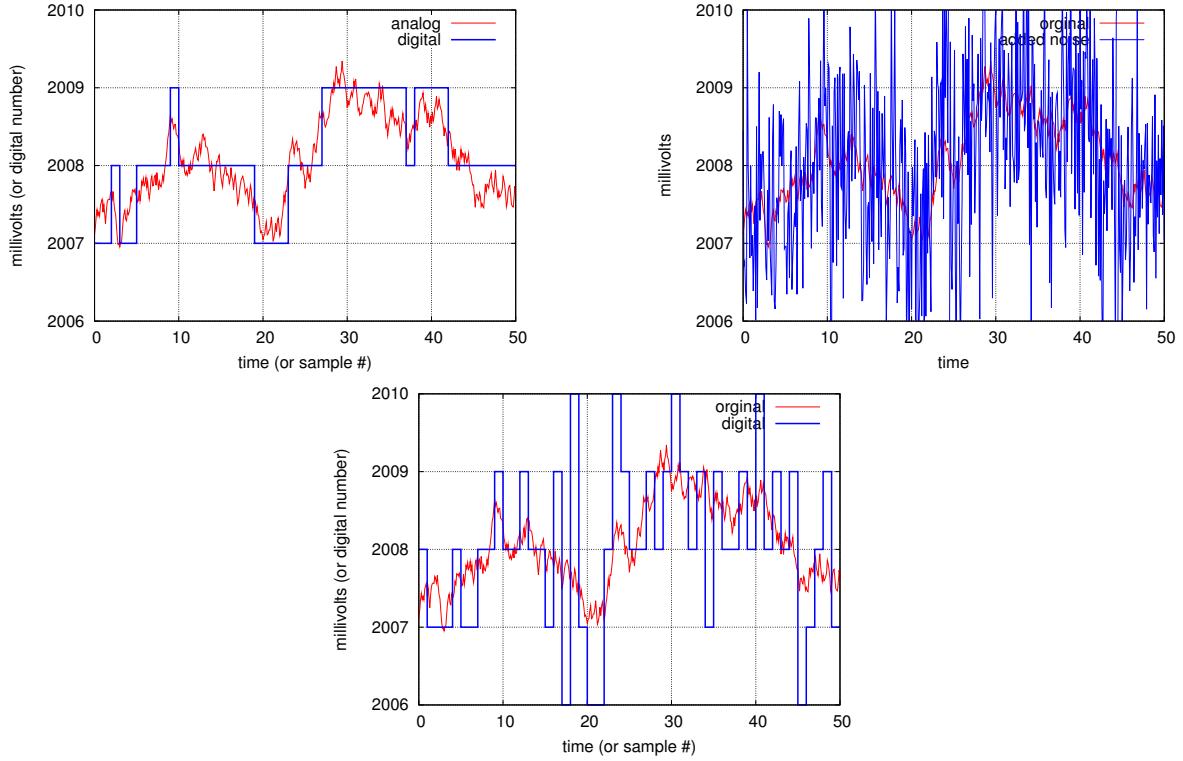


Fig. 10: The *dithering* technique makes use of (random) noise which is added to the analog signal. If this noise is later removed from the digital signal (e.g. using a digital low pass filter or statistics) the accuracy of the digital values can be improved. The best method would be the **subtractive dither**: produce the ‘random’ noise by a DAC and subtract the known numbers later.

where

$$\boxed{\omega_d = \frac{2\pi\omega}{\omega_s} = \omega T} \quad (7)$$

is the *discrete time frequency*. The units of the discrete-time frequency ω_d are **radians per sample** with a range of

$$-\pi < \omega_d \leq \pi \quad \text{or} \quad 0 \leq \omega_d < 2\pi .$$

3.4 The sampling theorem

Proper sampling means that you can exactly reconstruct the analog signal from the samples. *Exactly* here means that you can extract the ‘key information’ of the signal out of the samples. One basic key information is the frequency of a signal. Figure 11 shows different examples of proper and not proper sampling. If the sampling frequency is too low compared with the frequency of the signal, a signal reconstruction is not possible anymore. The artefacts which occur here are called *aliasing*.

To express a condition, when a signal is properly sampled, a sampling theorem can be formulated. This theorem is also known as *the Nyquist/Shannon theorem*. It was published in 1940 and points out one of the most basic limitations of the sampling in digital signal processing.

Given $f_s \triangleq$ sampling rate:

“A continuous signal can be properly sampled if it does not contain frequency components above

$$f_{\text{crit}} = \frac{f_s}{2} , \quad \text{the so-called } \underline{\text{Nyquist frequency}} .$$

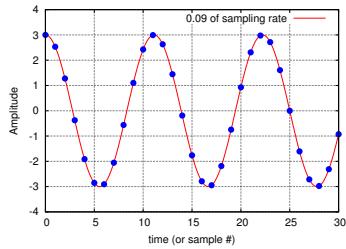
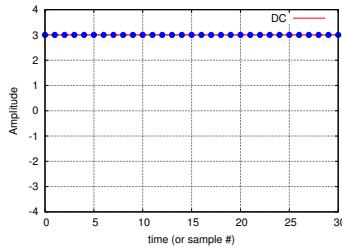
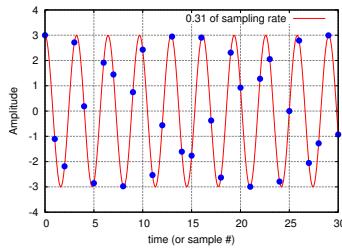
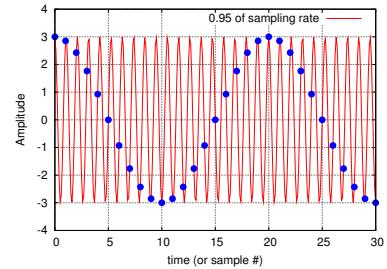
Proper:**Still proper:****Not proper:****"aliasing"**

Fig. 11: Different examples of proper and not proper sampling. If the sampling frequency is too low compared with the frequency of the signal, a signal reconstruction is not possible anymore.

Frequency components which are larger than this critical frequency ($f > f_{\text{crit}}$) are **aliased** to a mirror frequency $f^* = f_{\text{crit}} - f$.

The sampling theorem has consequences on the choice of the sampling frequency you should use to sample your signal of interest. The digital signal cannot contain frequencies $f > f_{\text{crit}}$. Frequencies greater than f_{crit} will add up to the signal components which are still properly sampled. This results in information loss at the lower frequency components because their signal amplitudes and phases are affected. So except for special cases (see undersampling and down-conversion) you need

1. a proper choice of sampling rate and
2. an *anti-aliasing filter* to limit the input signal spectrum.

Otherwise your signal will be affected by aliasing (see Fig. 12).

3.4.1 Mathematical explanation of aliasing

Consider a continuous-time sinusoid $x(t) = \sin(2\pi ft + \phi)$. Sampling at intervals T results in a discrete-time sequence

$$x[n] = \sin(2\pi fTn + \phi) = \sin(\omega_d n + \phi) .$$

Since the sequence is unaffected by the addition of any integer multiple of 2π , we can write

$$x[n] = \sin(2\pi fTn \pm 2\pi m + \phi) = \sin(2\pi T(f \pm \frac{m}{T})n + \phi) .$$

Replacing $\frac{1}{T}$ by f_s and picking only integers $m = kn$ we get

$$x[n] = \sin(2\pi T(f \pm kf_s)n + \phi) .$$

This means: when sampling at f_s , we cannot distinguish between f and $f \pm kf_s$ by the sampled data, where k is an integer.

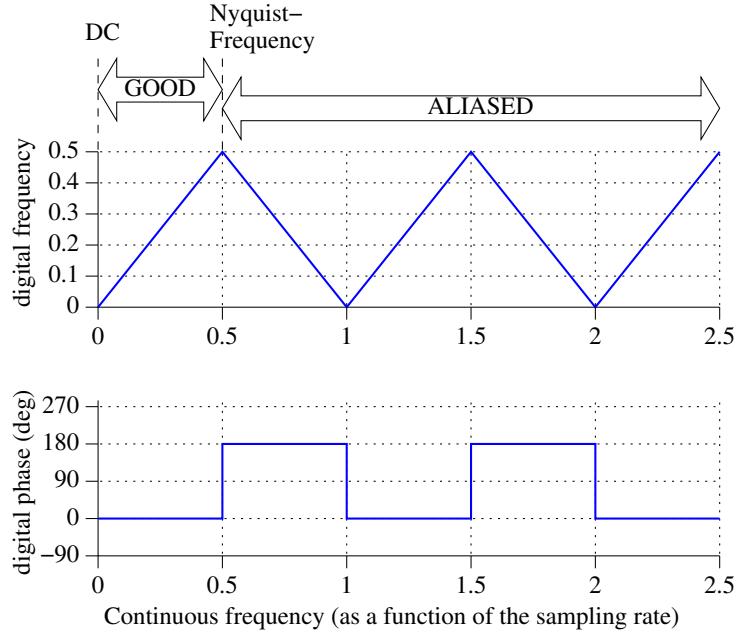


Fig. 12: Mapping of the analog frequency components of a continuous signal to the digital frequencies. There is a good area where the frequencies can be properly reconstructed and several so-called Nyquist bands where the digital frequency is different. Also the phase jumps from one Nyquist band to the other.

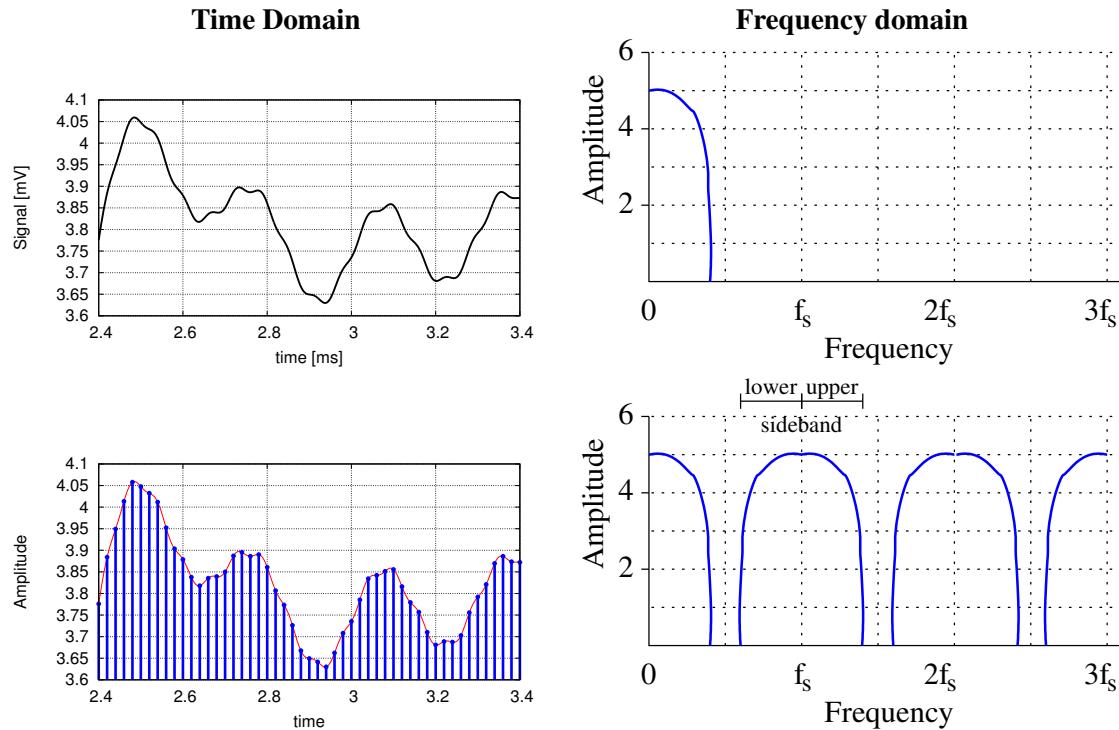


Fig. 13: Aliasing example. In frequency domain the continuous signal has a limited spectrum. The sampled signal can be seen as a pulse train of sharp (δ -)pulses which are modulated with the input signal. So the resulting spectrum gets side-bands which correspond to the Nyquist bands seen from inside the digital system. By the way: the same applies if you want to convert a digital signal back to analog.

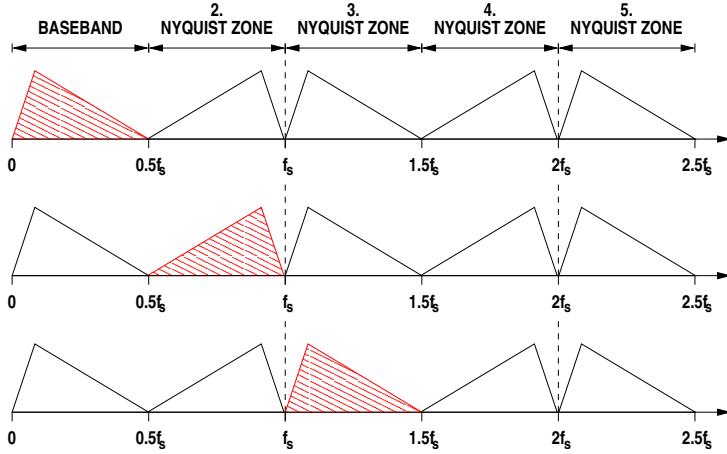


Fig. 14: Principle of undersampling

The aliasing can also be seen the other way round: Given a continuous signal with a limited spectrum (see Fig. 13). After sampling we cannot distinguish if we originally had a continuous and smooth signal or a signal consisting of a pulse train of sharp (δ -)pulses which are modulated corresponding to the input signal. Such a signal has side-bands which correspond to the Nyquist bands seen from inside the digital system. The same principle applies if you want to convert a digital signal back to analog.

This concept can be further generalized: Consider the sampling process as a **time-domain multiplication** of the continuous-time signal $x_c(t)$ with a sampling function $p(t)$, which is a periodic impulse function (Dirac comb). The frequency-domain representation of the sampled data signal is the **convolution** of the frequency domain representation of the two signals, resulting in the situation seen in Fig. 13. If you do not understand this by now, never mind. We shall discuss the concept of convolution in more detail later.

3.4.2 Undersampling

Last but not least, I want to mention a technique called *undersampling*, *harmonic sampling* or sometimes also called *digital demodulation* or *downconversion*. If your signal is modulated onto a carrier frequency and the spectral band of the signal is limited around this carrier, then you may take advantage of the ‘aliasing’. By choosing a sampling frequency which is lower than the carrier but synchronized with it (this means it is exactly a fraction of the carrier), you are able to demodulate the signal. This can be done with the spectrum of the signal lying in any Nyquist zone given by the sampling frequency (see Fig. 14). Just keep in mind that the spectral components may be reversed and also the phase of the signal can be shifted by 180° depending on the choice of the zone. And also — of course — any other spectral components which leak into the neighboring zones need to be filtered out.

3.5 Analog signal reconstruction

As mentioned before, similar problems, like aliasing for analog-to-digital conversion (ADC), also apply to **Digital-to-Analog Conversion** (DAC)! Usually, no impulse train is generated by a DAC, but a **zero-order hold** is applied. This modifies the output amplitude spectrum by multiplication of the spectrum of the impulse train with

$$H(f) = |\text{sinc}(\frac{f}{f_s})| := \left| \frac{\sin(\pi f/f_s)}{\pi f/f_s} \right|,$$

which can be seen as a convolution of an impulse train with a rectangular pulse. The functions are illustrated in Fig. 15.

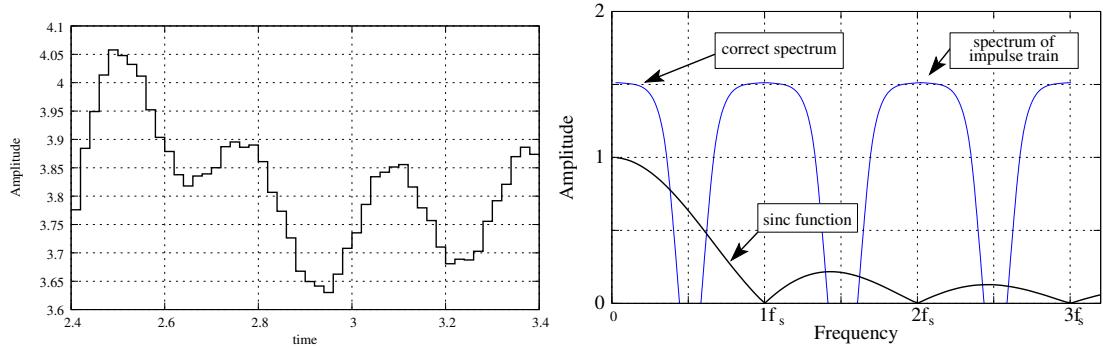


Fig. 15: Frequency response of the zero-order hold (right) which is applied at the DAC and generates the step function (left)

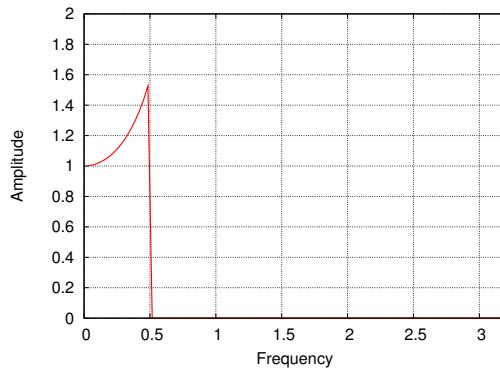
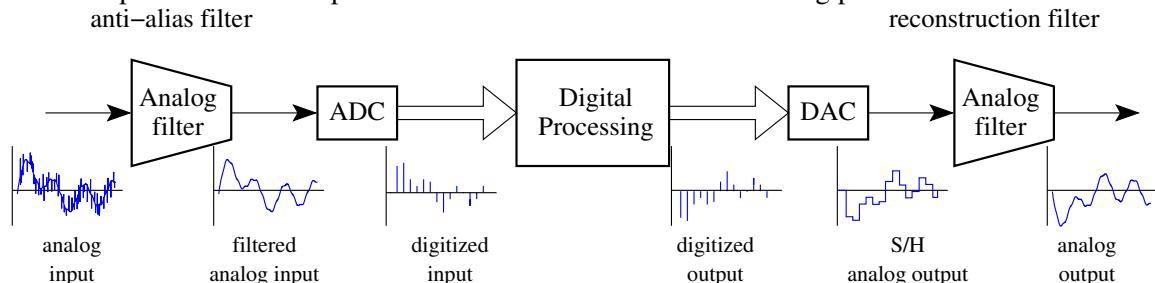


Fig. 16: Transfer function of the (ideal) reconstruction filter for a DAC with zero-order hold

As you can imagine, this behaviour appears to be unpleasant because now, not only components of the higher order sidebands of the impulse train spectrum are produced on the output (though attenuated by $H(f)$), but also the original spectrum (the baseband) is shaped by it. To overcome this ‘feature’, a *reconstruction filter* is used. The reconstruction filter should remove all frequencies above one half of f_s (an analog filter will be necessary, which is sometimes already built into commercial DSPs), and boost the frequencies by the reciprocal of the zero-order-hold’s effect ($\frac{1}{\text{sinc}(\cdot)}$). This can be done within the digital process itself! The transfer function of the (ideal) reconstruction filter is shown in Fig. 16.

3.6 Anti-aliasing techniques

Putting it all together, digital signal processing needs additional care concerning the sampling and reconstruction processes. The steps needed are summarized in the following picture:



To design your digital signal processing system, you need to know about (analog) *filter design*, the

characteristics of anti-aliasing and reconstruction filters, and about *limitations* of signal processing like bandwidth and noise of the analog parts and, for the digital parts, sampling frequency and quantization.

4 Noise

The terms *error* and *noise* are closely related. Noise is some fluctuation on the input signal which can come from different sources, can have different spectral components and in many cases (except for the dithering methods) is unwanted. It can cover the information you want to extract from the signal and needs to be suppressed with more or less advanced techniques. Usually, some of the noise components can hardly be avoided and, therefore, we shall have to deal with it. Noise on the signal can cause an *error*. But there are also errors which do not come from noise. We therefore distinguish between *systematic* (deterministic) errors on the one hand and *unsystematic* (statistical) errors (or *noise*) on the other hand. We are going to take a closer look at this distinction.

Systematic error \longleftrightarrow **accuracy** comes from characteristics of the measurement device (ADC/DAC: *offset*, *gain*, *linearity-errors*). It can be improved by improvements of the apparatus, like calibration. The only limits here come from the practical usefulness and from quantum mechanics, which keeps you from measuring certain quantities with absolute accuracy.

Statistical error comes from unforeseen random fluctuations, stochastics, and noise. It is impossible to avoid them completely, but it is possible to estimate the extent and it can be reduced through statistical methods (averaging), multiple repetitive measurements etc. This determines the **precision** of the measurement.

Note that the definition is context dependent: The accuracy of 100 devices can be a matter of precision! Imagine that you measure the same property with 100 different devices where each device has a slightly different systematic (calibration) error. The results can now be distributed in much the same way as they are with a statistical measurement error — and so they can be treated as statistical errors, in this case, and you might want to use the statistical methods described in the following section.

The distinction above leads to the terms *accuracy* and *precision*, which we shall define in the following sections. Besides this, we want to deal with the basic concepts of statistics which include

- random variables and noise (e.g., white noise, which has an equal distribution, Gaussian noise, which has a Gaussian distribution, and $1/f$ or pink noise, which is $1/f$ distributed),
- the mean and the standard deviation, variance, and
- the normal or Gaussian distribution.

4.1 Basic statistics

4.1.1 Mean and standard deviation

Assuming that we do N measurements of a quantity which result in a series of measurement values x_i , the *mean* (or average) over N samples can be calculated as:

$$\hat{x} := \frac{1}{N} \sum_{i=0}^{N-1} x_i .$$

The *variance* σ^2 (σ itself is called *standard deviation*) is a measure of the ‘power of fluctuations’ of the set of N samples. It is a direct measure of the precision of the signal.

$$\sigma^2 := \frac{1}{N-1} \sum_{i=0}^{N-1} (x_i - \hat{x})^2 . \quad (8)$$

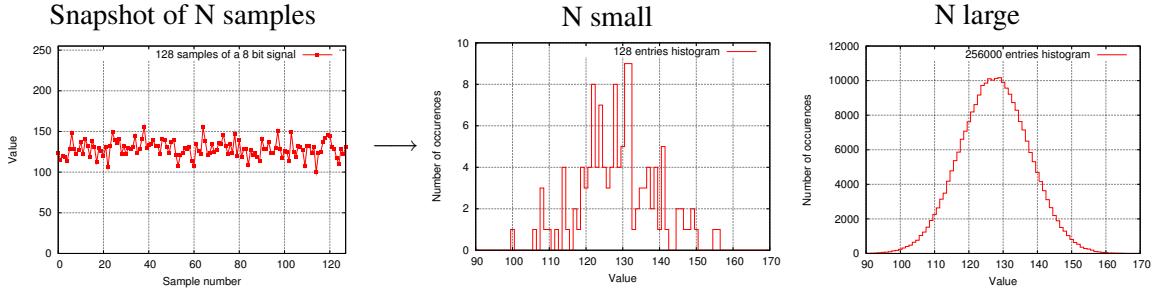


Fig. 17: Creating a histogram from a snapshot of samples

Equation (8) can also be written in the following form:

$$\sigma_N^2 = \frac{1}{N-1} \left[\underbrace{\sum_{i=0}^{N-1} x_i^2}_{\text{sum of squares}} - \frac{1}{N} \underbrace{\left(\sum_{i=0}^{N-1} x_i \right)^2}_{\text{sum}^2} \right],$$

which is useful, if you want to calculate running statistics ‘on the fly’.

There are also quantities which are derived from the mean and the variance like

$$\text{The Signal to Noise Ratio (SNR): } \text{SNR} = \frac{\hat{x}^2}{\sigma^2}, \quad (9)$$

$$\text{the Coefficient of Variation (CV): } \text{CV} = \frac{\sigma}{\hat{x}} \cdot 100\% \quad \text{and} \quad (10)$$

$$\text{the Root Mean Square (RMS): } x_{\text{rms}} := \sqrt{\frac{1}{N} \sum_{i=0}^{N-1} x_i^2}. \quad (11)$$

The latter is a measure of the ‘Power of fluctuations plus power of DC component’.

4.1.2 Histograms and the probability density distribution

A common way to reduce the amount that must be processed is to use *histograms*. A snapshot of N samples is summed up in (M) bins (see Fig. 17). Each bin now contains the number of occurrences of a certain value (or range of values) H_i and the mean and variance can now be calculated using this histogram:

$$N = \sum_{i=0}^{M-1} H_i, \\ \hat{x} := \frac{1}{N} \sum_{i=0}^{M-1} i \cdot H_i, \\ \sigma^2 := \frac{1}{N-1} \sum_{i=0}^{M-1} (i - \hat{x})^2 H_i.$$

As you already saw in Fig. 17, with a large number of samples the histogram becomes smooth and it will converge in the limit $N \rightarrow \infty$ to a distribution which is called the *probability mass function*. This is an approximation of the (continuous) *probability density distribution*. This is illustrated in Fig. 18. In this case, the fluctuations of the samples have a Gaussian distribution. Examples of probability mass functions and probability density distributions of common waveforms are shown in Fig. 19.

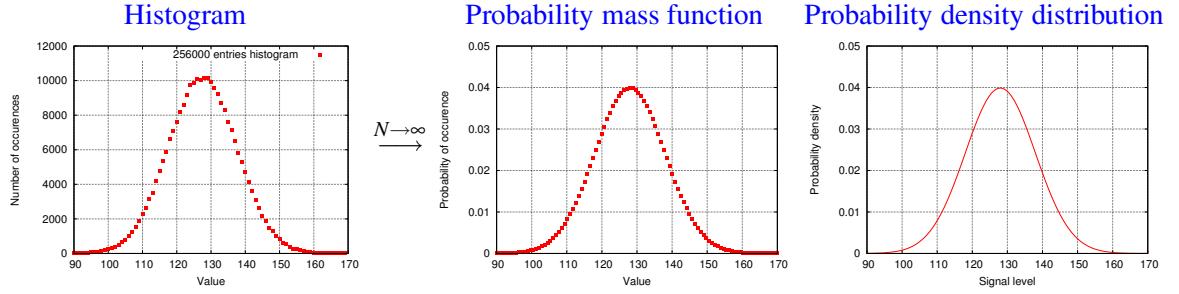


Fig. 18: Histogram, probability mass function, and probability density distribution

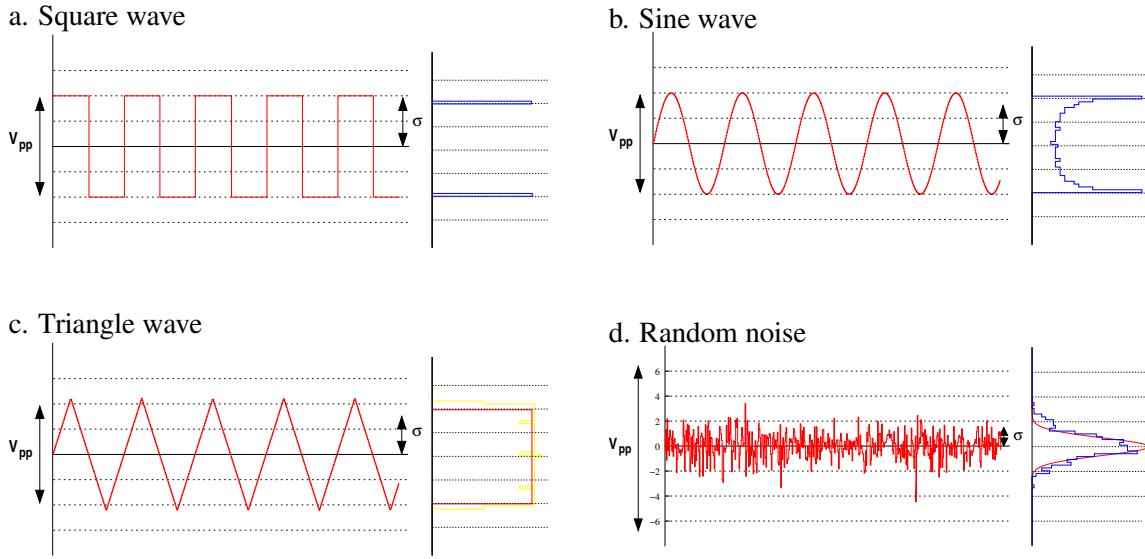


Fig. 19: Probability mass functions and probability density distributions of common waveforms

4.1.3 The normal distribution

The best known and most common distribution is the *normal distribution* that has the form of a Gauss function:

$$P(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x - \hat{x})^2}{2\sigma^2}}.$$

The Gauss formula is illustrated in Fig. 20. Note that the probability density is normalized, so that the integrated density is the overall probability. This should, of course, be equal to one:

$$\int_{-\infty}^{+\infty} P(x) dx = 1.$$

Now what is this good for? Imagine that we have N samples of a measured quantity. Then we can define the

$$\text{typical error: } \Delta A = \frac{\sigma_N}{\sqrt{N}}.$$

Here σ_N is an estimate of the standard deviation of the [underlying process](#) over N samples (e.g., extracted from the histogram). This is the best information about the underlying process you can extract out of the

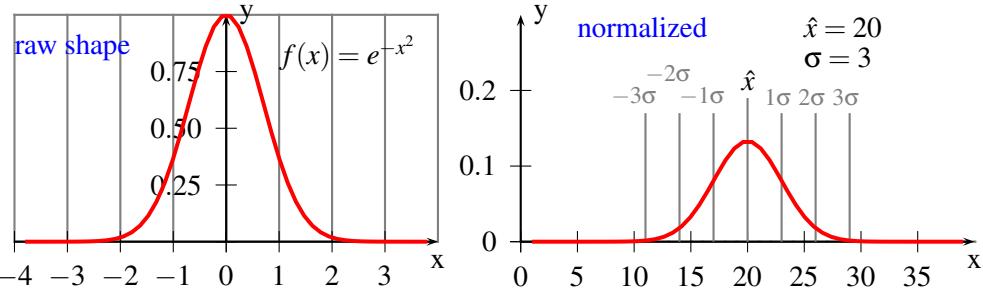


Fig. 20: The raw shape and the normalized shape of the Gauss function. The area of one standard deviation $\pm \sigma$ integrates to 68.3%, the area of $\pm 2\sigma$ to 95.4%.

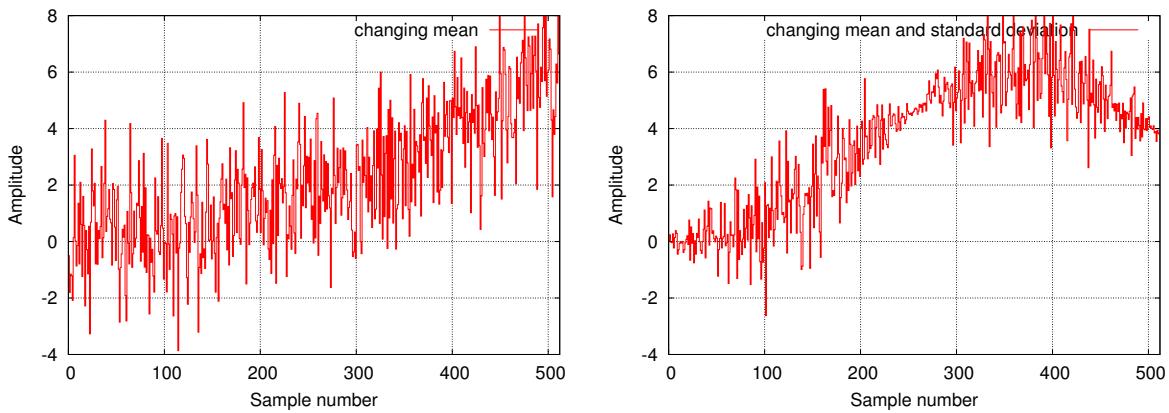


Fig. 21: A signal with changing mean and standard deviation

sampled signal. In practice, that means that the more samples you take, the smaller the typical error ΔA is. But this can only be done if the underlying quantity does not change during the time the samples were taken. In reality, the quantity and also its fluctuations may change, as in Fig. 21, and it is a real issue to select the proper and useful number of samples to calculate the mean and standard deviation σ_N to get a good approximation of what the real process may look like. There is no such thing as an *instant error*; the probability density function cannot be measured, it can only be approximated by collecting a large number of samples.

4.2 The central limit theorem

Why does a normal distribution occur so frequently? Why are most processes and most signals normally distributed? Why is it always a good assumption that the probability density distribution of an arbitrary measurement is Gaussian, and we know everything we can get about the underlying process if we know the measurement value A and its typical error ΔA ?

This is the consequence of the *central limit theorem* which says:

The sum of independent random numbers (of any distribution) becomes Gaussian distributed.

The practical importance of the central limit theorem is that the normal distribution can be used as an approximation of some other distributions. Whether these approximations are sufficiently accurate depends on the application for which they are needed and the rate of convergence to the normal distribution. It is typically the case that such approximations are less accurate in the tails of the distribution.

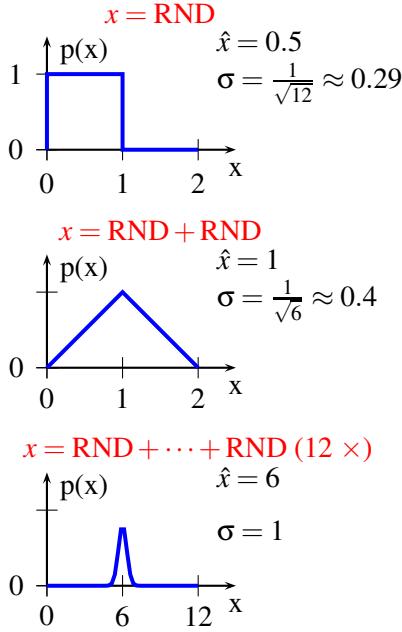


Fig. 22: Consequence of the central limit theorem: Summing up more and more equally distributed random numbers will result to good approximation in a Gaussian distributed random variable

It should now be clear why most of your measurements may be Gaussian distributed. This is simply because the measurement process is a very complicated one with many different and independent error sources which all together contribute to the final measurement value. They do so without caring about the details of their mechanisms — as long as there are enough contributors, the result will be approximately Gaussian.

There is also a practical application of the theorem in computing. Suppose you need to generate numbers which have a Gaussian distribution. The task is quite easy; you just have to have a function which generates any kind of (pseudo-) random numbers and then sum up enough of them.

Here is an example: first generate *white noise* using a function which produces equally distributed random numbers between zero and one $\text{RND} := [0; 1[$. This is often implemented in the form of a pseudo random generator which calculates

$$\text{RND} = (as + b) \bmod c ,$$

where s is the *seed* and a, b and c are appropriately chosen constants. The new random number is used as a seed for the next calculation and so on.

The distribution of this function is shown in Fig 22, top. If you now add two such random numbers, the result will have a distribution as shown in the figure in the centre. After adding 12 random numbers you already get a very good approximation of a Gaussian distribution with a standard deviation of $\sigma = 1$ and a mean value of $\hat{x} = 6$. If you subtract 6 from this sum, you are done. But do not really implement it like this, because there is a simpler formula which only uses 2 random variables and will also do a good job ($\hat{x} = 0$, $\sigma = 1$):

$$x = \sqrt{-2 \log_{10}(\text{RND}_1)} \cdot \cos(2\pi \text{RND}_2) .$$

4.3 Accuracy and precision

Having understood the probability density distribution of a series of measurement samples, it is now straightforward to define precision and accuracy. Figure 23 illustrates the difference.

To summarize:

- *accuracy* is a measure of *calibration*,
- *precision* is a measure of *statistics*.

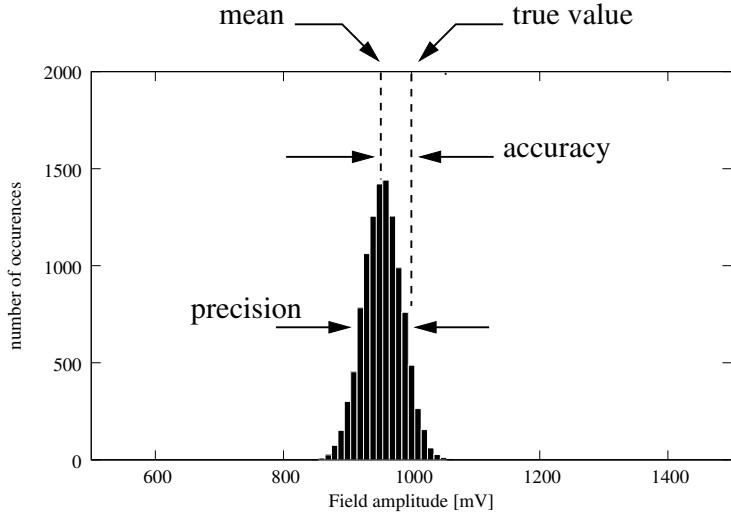


Fig. 23: The difference between accuracy and precision: Accuracy is the difference between the true value and the mean of the underlying process that generated the data. Precision is the spread of the values coming from fluctuations, noise and any other statistical error. It is specified by the standard deviation or the signal noise ratio.

4.3.1 Signal-to-noise ratio

Because it is a very common term in engineering, let us define the *signal-to-noise ratio* which is a measure of the relative error of a signal. From the statistical mathematics point of view we already defined it in Eq. (9). But maybe you are more familiar with the following definitions which deal with the power P and the amplitude A of a signal. In these terms, the signal-to-noise ratio is the power ratio, averaged over a certain bandwidth of the power spectrum $p(v)$:

$$\text{SNR} := \frac{\bar{P}_{\text{signal}}}{\bar{P}_{\text{noise}}} = \left(\frac{\hat{A}_{\text{signal,rms}}}{\hat{A}_{\text{noise,rms}}} \right)^2 ,$$

$$\bar{P} := \int_{\text{BW}} p(v) dv .$$

Quantities which come from ratios are very often — for practical reasons (you avoid multiplication and division) — expressed in decibels, a logarithmic pseudo-unit:

$$\begin{aligned} \text{SNR(dB)} &:= 10 \log_{10} \left(\frac{\bar{P}_{\text{signal}}}{\bar{P}_{\text{noise}}} \right) = 20 \log_{10} \left(\frac{\hat{A}_{\text{signal,rms}}}{\hat{A}_{\text{noise,rms}}} \right)^2 \\ &= P_{\text{signal}}[\text{dBm}] - P_{\text{noise}}[\text{dBm}] . \end{aligned}$$

A similar ‘unit’ is used if you talk about the carrier as reference: [SNR(dB)] = dBc (=‘dB below carrier’), and so you can also define a CNR = *carrier-to-noise ratio*.

4.4 Error sources in digital systems

From the digital processing, the digitization, and the analog reconstruction of the signals, there are various sources of errors:

1. **Systematic errors:** Most importantly, ADC and DAC distortions: e.g. offset, gain and linearity errors. These types of errors can be corrected for through calibration.
2. **Stochastic errors:** quantization noise, quantization distortions, as well as aperture and sampling errors (clock jitter effects).
3. **Intrinsic errors:** DAC-transition errors and glitches. They are random, unpredictable, and sometimes systematic, but it is hard to correct the source of these errors, and so they need to be filtered.

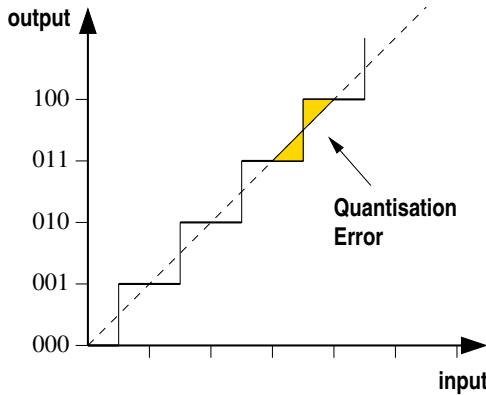


Fig. 24: Transfer function of an ADC. The quantization noise comes from the difference between the continuous (analog) input signal level and the signal level represented by the digital number produced by the ADC. Because the ADC has a finite resolution, this error can be no more than $\pm \frac{1}{2}$ of the step height.

The systematic errors can in principle be corrected for through calibration, and this is also the recommended way to treat them wherever possible. The intrinsic errors are hard to detect, may cause spurious effects and therefore make life really bad. If they bother you, a complete system analysis and probably a rework of some components may be required to cure them. There is (nearly) no way to overcome them with some sort of data processing. Therefore we focus here on the stochastic errors, because the way we treat them with data processing determines the quality of the results. At least, we can improve the situation by use of sophisticated algorithms which, in fact, can be implemented in the digital processing system more easily than in an analog system.

4.4.1 Quantization noise

The transfer function of an analog-to-digital converter (ADC) is shown in Fig. 24. The quantization noise comes from the difference between the continuous (analog) input signal level and the signal level represented by the digital number produced by the ADC. Because the ADC has a finite resolution, this error can be no more than $\pm \frac{1}{2}$ of the step height (least significant bit resolution $|A| < 0.5\text{LSB}$). The RMS error of the quantization noise is

$$\text{RMS}(\Delta A) \approx \sqrt{12} \text{ LSB} .$$

Although this error is not really independent of the input value, from the digital side it actually is, because there is no control when the least significant bit flips. It is, therefore, best to treat this error as a (quantization) noise source.

For a full-scale $\sin()$ signal, the signal-to-noise ratio coming from the quantization noise is

$$\text{SNR} = 6.02n + 1.76\text{dB} + 10 \log \left(\frac{f_s}{2\text{BW}} \right) . \quad (12)$$

As you can see, it increases with lower BW. This means that doubling the sampling frequency increases the SNR by 3dB (at the same signal bandwidth). This is effectively used with so-called ‘oversampling’ schemes. Oversampling is just a term describing the fact that with a sampling frequency that is much higher than would be required by the Nyquist criterium, you can compensate for the quantization noise caused by a low ADC bit resolution. Especially for 1-bit ADCs, this is a major issue.

In Eq. (12), it is assumed that the noise is equally distributed over the full bandwidth. This is often not the case! Instead, the noise is often *correlated* with the input signal! The lower the signal, the more correlation. In the case of strong correlation, the noise is concentrated at the various harmonics of the input signal; this is exactly where you do not want them. Dithering and a broad input signal spectrum randomizes the quantization noise.

Nevertheless, this simple quantization noise is not the only cause of errors in the analog-to-digital conversion process. There are two common, related effects: *missing codes* and *code transition noise*. These effects are intrinsic to the particular ADC chip in use. Some binary codes will simply not be produced because of ADC malfunction as a consequence of the hardware architecture and internal algorithm responsible for the conversion process. Especially for ADCs with many bits, this is an issue. Last but not least, the ADC may show code transition noise; this means that the output oscillates between two steps if the input voltage is within a critical range even if the input voltage is constant.

5 Linear systems

You now know some of the main consequences, advantages, and limitations of using digitized signals. You know how to deal with aliasing, downsampling, and analog signal reconstruction. You know the concepts of noise and the basic mathematical tools to deal with it.

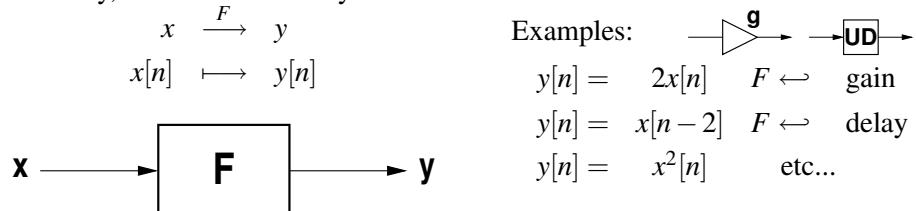
Next, we are going to look more closely at the **systems** which transform the (digital) signals. Of course, there are *analog* systems as well as *digital* ones. But, since there are not many conceptual differences, we can focus mainly on the digital ones. The analogy to analog system concepts will be drawn from whenever useful.

We are also going to use different notations in parallel: besides the mathematical notation, we show the rather symbolic expressions commonly used in engineering fields. In contrast to the mathematical notation, which is slightly different for analog systems (e.g. $y(t) = 2x(t)$) and digital systems (e.g. $y[n] = 2x[n]$), the latter does not make a formal difference here. Both concepts and notations are in use in different books on the field. They are, however, easy to understand, so you will quickly become familiar with both notations.

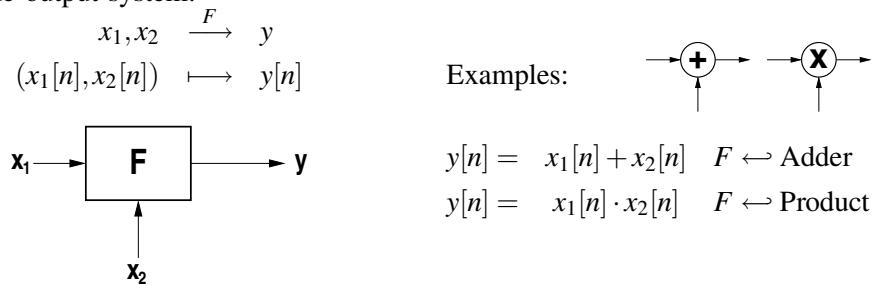
5.1 Discrete-time systems

A **system** receives one or more **inputs** and generates one or more **outputs** dependent on the inputs. We distinguish between three kinds of systems:

1. **MIMO** (Multiple-Input-Multiple-Output) systems; these are the most general.
2. **SISO** (Single-Input-Single-Output) systems; such are many of the elementary systems, e.g. gain and the unit delay, and of course many combinations:



3. and **MISO** (Multiple-Input-Single-Output) systems; here the adder is the most popular double-input-single-output system:



Besides this, there is also a way to split signals. This produces a generic Single-Input-Double-Output system.

Starting from elementary systems, the concept of *superposition* allows us to combine systems to create more complex systems of nearly any kind.

5.2 Superposition

Systems may be of any complexity. It is, therefore, convenient to look at them as a composition of simpler components. If we restrict ourselves to the class of *linear systems*, it is possible to first decompose the input signals and then process them with simple systems. In the end, the result will be synthesised by superposition for prediction of the output. In this way, we can split up the problems into many pieces of simpler complexity, and even use only a few fundamental systems. Without the concept of decomposition and linear systems, we would be forced to examine the individual characteristics of many unrelated systems, but with this approach, we can focus on the traits of the linear system category as a whole.

Although most real systems found in nature are not linear, most of them can be well approximated with a linear system, at least for some limited range of ‘small’ input signal amplitudes.

5.3 Causal, linear, time-invariant systems

Systems under investigation in this lecture should therefore be *linear*, *causal*, and *time invariant*. We shall see what this means in detail.

5.3.1 Linearity

Given system F with $F(x_1[n]) = y_1[n]$ and $F(x_2[n]) = y_2[n]$, then F is said to be *linear* if

$$F(x_1[n] + x_2[n]) = F(x_1[n]) + F(x_2[n]),$$

(it follows that $F(x[n] + x[n]) = F(2x[n]) = 2F(x[n])$), and for two linear systems F_1 and F_2

$$F_1(F_2(x[n])) = F_2(F_1(x[n])).$$

5.3.2 Time-invariance

(also ‘shift-invariance’) Given F with $F(x[n]) =: y[n]$ is considered *time-invariant* if

$$F(x[n - k]) = y[n - k] \quad \forall k \in \mathbb{N}.$$

5.3.3 Causality

The system is *causal* if the output(s) (and internal states) depend only on the *present and past* input and output values.

$$\text{Causal: } y[n] = x[n] + 3x[n - 1] - 2x[n - 2]$$

$$\text{Non-causal: } y[n] = \textcolor{red}{x[n + 1] + 3x[n] + 2x[n - 1]}.$$

In the latter case the system Y produces its output y by using an input value of the input signal x which is ahead of time (or the currently processed time step n).

5.3.4 Examples

Which of the following systems are **linear** and/or **time-invariant** and/or **causal**?

- (1) $y[n] = Ax[n] + Bx[n - 2]$ ***l,ti,c***
- (2) $y[n] = x[2n]$ ***l***
- (3) $y[n] = x^2[n]$ ***ti,c***

- (4) $y[n] = -2x[-n]$ ***l,ti***
- (5) $y[n] = Ax[n - 3] + C$ ***ti,c***
- (6) $y[n] = x[2n + 1]$ ***l***

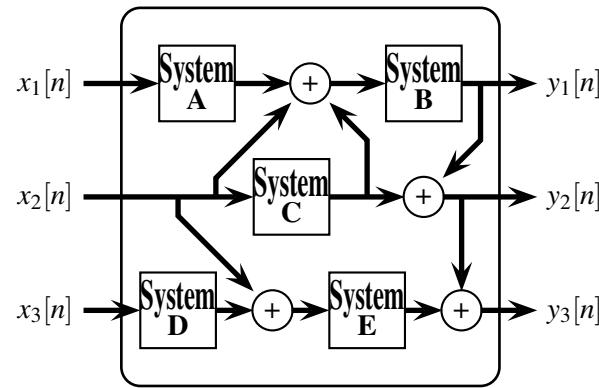
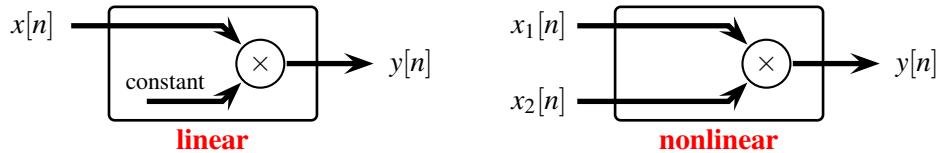


Fig. 25: A linear MIMO system composed of linear SISO systems and adders

5.4 Linearity of MIMO and MISO systems

Any MIMO system will be linear if it is composed of linear systems and signal additions, like in the example in Fig. 25.

However, multiplication is not always linear ...



5.5 Decompositions

An important consequence of the linearity of systems is that there exist algorithms for different ways of decomposing the input signal. The spectral analysis is based on this, so one can say the concept of decomposition is really fundamental. The simplest decompositions are

- Pulse decomposition

$$x[n] = x_0[n] + x_1[n] + x_2[n] + x_3[n] + \dots + x_{11}[n] + \dots$$

- Step decompositions

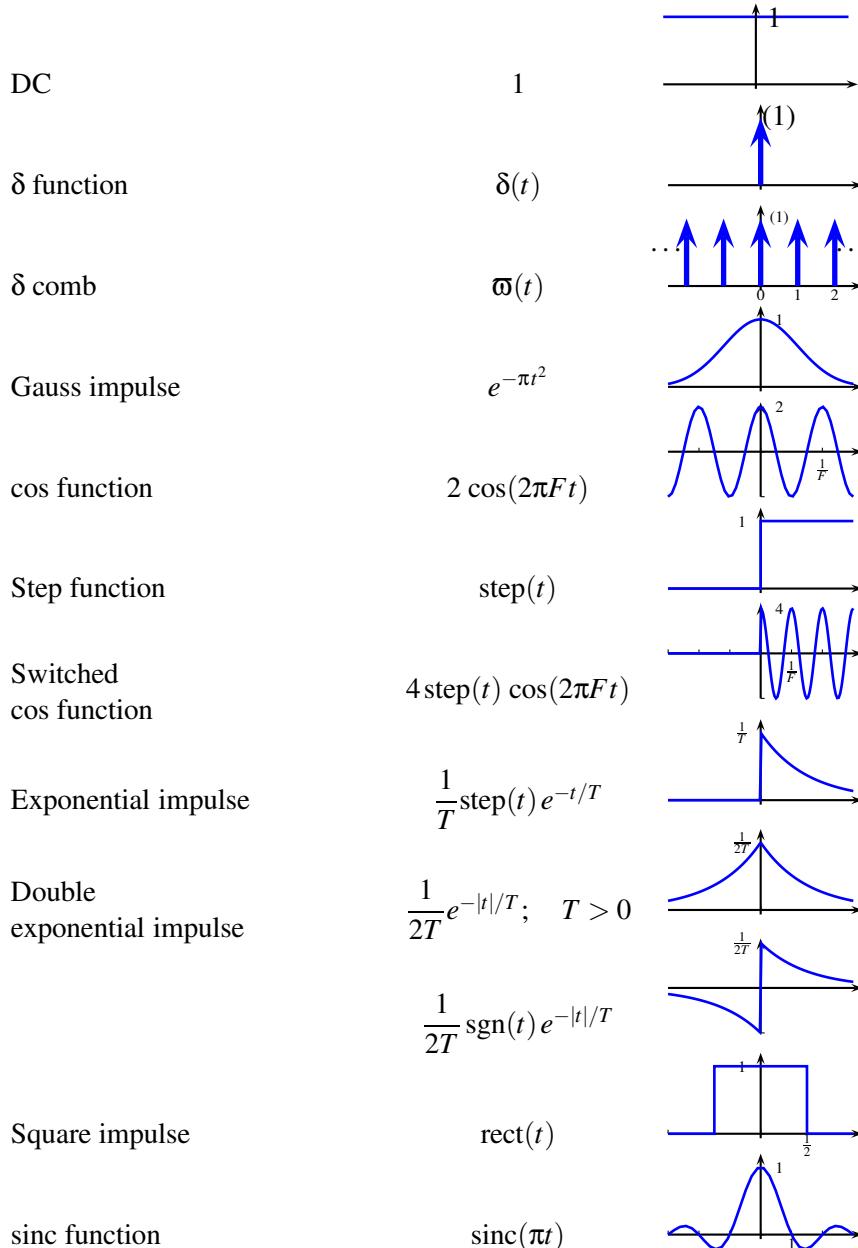
$$x[n] = x_0[n] + x_1[n] + x_2[n] + x_3[n] + \dots + x_{11}[n] + \dots$$

- Fourier decomposition

$$\begin{aligned} x[n] &= x_{c0}[n] + x_{c1}[n] + x_{c2}[n] + x_{c3}[n] + \dots + x_{c6}[n] + x_{c7}[n] \\ N = 16 &+ x_{s0}[n] + x_{s1}[n] + x_{s2}[n] + x_{s3}[n] + \dots + x_{s6}[n] + x_{s7}[n] \end{aligned}$$

- and many others.

Later, we shall make extensive use of special decompositions and also convolutions (which is the opposite process). Their applications are in the Fourier transformation, the Laplace and z -transformation, wavelets and filters.

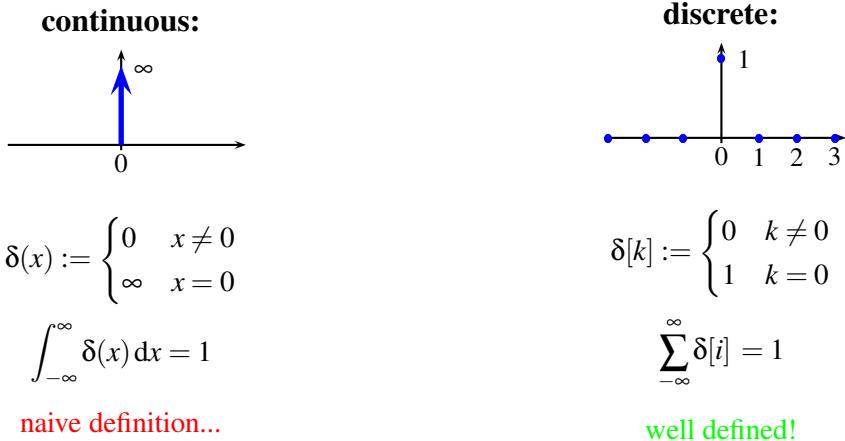
**Fig. 26:** Common waveforms

6 Special functions

In this very short section I wish to introduce you to some very common functions and signal forms shown in Fig. 26. Special focus will be put on the δ -function (or better: the δ -distribution; but in practice the difference does not play a big role). Other common waveforms are shown in the figure.

6.1 The δ -function

The δ -function can be defined for continuous and for discrete signals:



The continuous δ -function is not well defined that way. This is because its nature is that of a **distribution**. One important and required property of the δ -function cannot be seen this way: it is normalized (like the Gaussian distribution) so that

$$\int_{-\infty}^{\infty} \delta(x) dx = 1 .$$

The definition above can be improved if you look at the δ -function as the limit of a series of functions. Some popular definitions include

sinc functions:

$$\delta(x) = \lim_{\kappa \rightarrow \infty} \frac{\sin(\kappa x)}{\pi x}$$

Gauss functions:

$$\delta(x) = \lim_{\varepsilon \rightarrow 0} \frac{1}{\sqrt{\pi \varepsilon}} e^{-\frac{x^2}{\varepsilon}}$$

Lorentz functions:

$$\delta(x) = \frac{1}{\pi} \lim_{\varepsilon \rightarrow 0} \frac{\varepsilon}{x^2 + \varepsilon^2}$$

rectangles:

$$\delta(x) = \lim_{\varepsilon \rightarrow 0} \frac{1}{2\varepsilon} r_\varepsilon(x) \quad ; \quad r_\varepsilon(x) := \begin{cases} 0 & |x| \geq \varepsilon \\ 1 & |x| < \varepsilon \end{cases}$$

Also a complex (Fresnel) definition is possible:

$$\delta(z) = \lim_{\alpha \rightarrow \infty} \sqrt{\frac{\alpha}{i\pi}} e^{i\alpha z^2} .$$

More important than the correct definition are the calculation rules of the δ -function, which can be applied independently of its definition, whether you use ∞ or the limits of series. The most important ones are given here:

Continuous convolution rule:

$$\int_{-\infty}^{\infty} f(x) \delta(x - x_0) dx = f(x_0)$$

Discrete convolution rule:

$$\sum_{i=-\infty}^{\infty} f[i] \delta[i - n] = f[n]$$

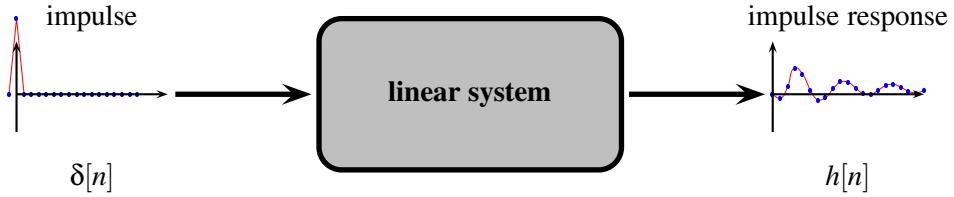


Fig. 27: The concept of impulse response

Fourier transform:

$$\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} \delta(t) e^{-i\omega t} dt = \frac{1}{\sqrt{2\pi}}$$

Laplace transform:

$$\int_0^{\infty} \delta(t-a) e^{-st} dt = e^{-as}$$

Scaling rule:

$$\delta(\alpha x) = \frac{\delta(x)}{|\alpha|}$$

Another popular pseudo function is the so-called *Dirac comb*, which is a combination of an infinite number of equally shifted δ -functions:

$$C(x) = \sum_{k \in \mathbb{Z}} \delta(x - k) .$$

7 Convolution

As already mentioned before, decomposition and convolution are the fundamental operations of linear systems. Here we are going to look more closely at the concept of convolution because the technique is the basis of all digital filters. The specialized digital signal processors always have special and ready-made instructions built in to support this operation.

7.1 The impulse response

The *impulse response* of a linear system is its response to a δ -pulse on its input. For digital systems, the discrete δ pulse, which is a unit pulse here, is applied and a sufficient number of samples $h[n]$ of the output of the system are recorded (see Fig. 27).

This is like ringing a bell with a hammer. The hammer produces a δ like excitation and after that the bell rings for a while; this is its impulse response. The way in which it rings is very characteristic for that bell; it contains, for example, all its eigenfrequencies, each of which decays with some characteristic time constant. What you cannot hear are the phases of the spectrum. The impulse response $h[n]$ is the fingerprint of the system. If two linear systems have the same impulse response, then they are identical. This means that all possible information about the system can be found in its impulse response. One can say the impulse response $h[n]$ is the system. Now, let us look at it from a mathematical point of view:

For an arbitrary input signal written in the form

$$x[n] := \sum_{i=0}^{N-1} x_n \delta[n-i]$$

we can now immediately write down the output of the system if we know its impulse response:

$$y[n] = \sum_{i=0}^{N-1} x_n h[n-i] .$$

This arises because the system is linear and so the sum stays a sum and the product with a scalar (x_n) transforms to a product with a scalar. Only the response to the δ -function needs to be known, but this is just the impulse response! Try to really understand this fundamental fact, recapitulate the linearity criteria if necessary and make it clear to yourself what $x_n \delta[n-i]$ means. The features you should remember are

- $h[n]$ has all information to process the output of the system for any input signal !
- $h[n]$ is called **filter kernel** of the system (and can be measured by impulse response).
- The system is '**causal**' if $h[i] = 0 \quad \forall i < 0$.
- The output for any input signal $x[n]$ is

$$y[n] = x[n] * h[n] ,$$

where $*$ is the **convolution operator**. The mathematical definition follows.

7.2 Convolution

Given two functions $f, g : D \rightarrow \mathbb{C}$, where $D \subseteq \mathbb{R}$, the **convolution** of f with g , written $f * g$ and defines as the integral of the product of f with a mirrored and shifted version of g :

$$(f * g)(t) := \int_D f(\tau)g(t-\tau) d\tau .$$

The domain D can be extended either by periodic assumption or by zero, so that $g(t-\tau)$ is always defined.

Given $f, g : D \rightarrow \mathbb{C}$, where $D \subseteq \mathbb{Z}$, the **discrete convolution** can be defined in a similar way by the sum:

$$(f * g)[n] := \sum_{k \in D} f[k]g[n-k]$$

Two examples of discrete convolutions are shown in Fig. 28 and Fig. 29. As you can see, it is very simple to realize digital filters with this technique by choosing the appropriate filter kernels. You may ask where the filter kernels come from. Well, this is the topic of filter design where a practical formalism can be used which we briefly discuss in the section about the z -transform.

7.3 Calculating with convolution

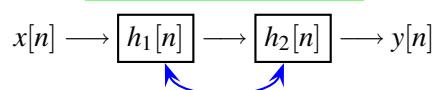
7.3.1 Commutative property

$$x[n] * y[n] = y[n] * x[n]$$

The commutative property of convolution tells you that the result will be the same if you exchange the input signal with the filter kernel (whatever sense this makes). It makes more sense if you look at the

7.3.2 Associative property

$$(a * b) * c = a * (b * c)$$



This feature allows you to rearrange systems which are in series in different and arbitrary orders. It does not matter if you first pass a differentiator and then a low-pass or vice versa. The result will be the same.

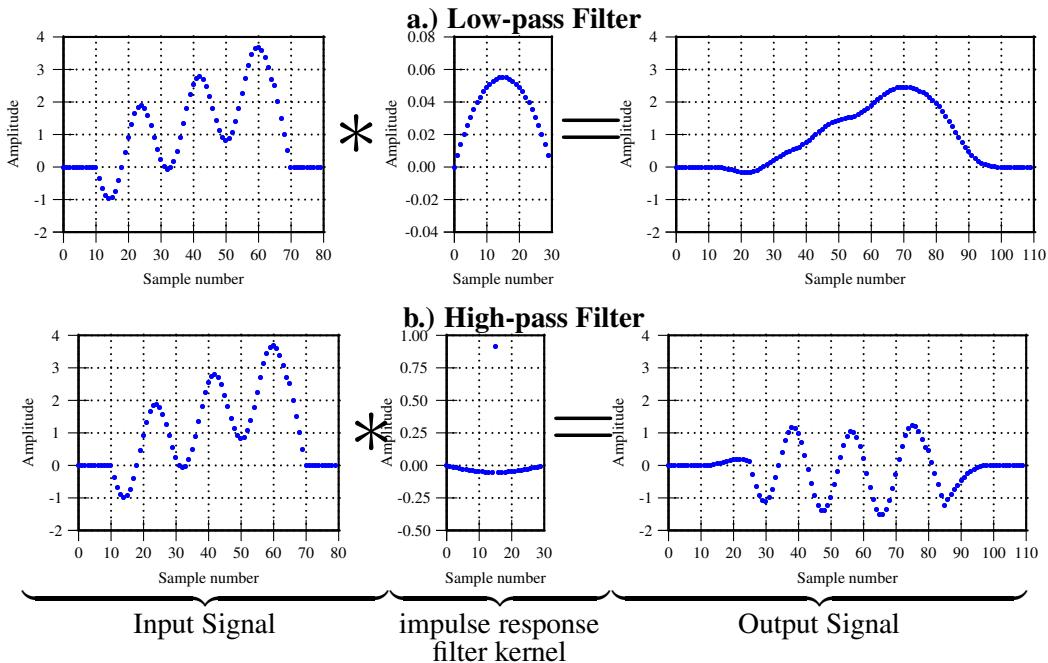


Fig. 28: Realization of a low-pass and a high-pass filter with convolution. The input signal is convoluted with an appropriate filter kernel and the result is the output signal.

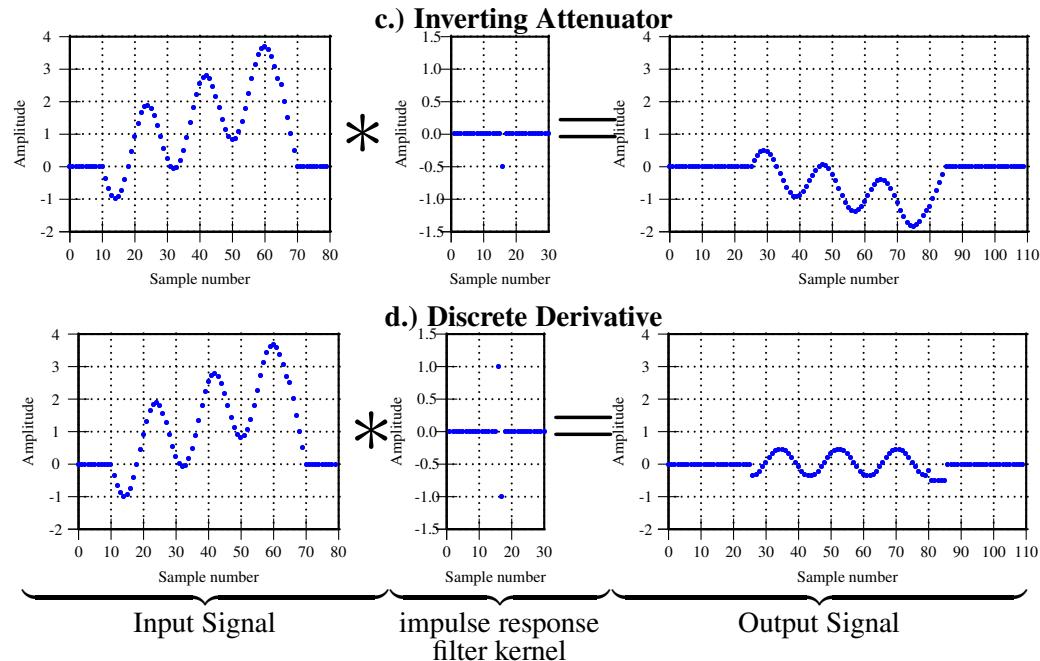


Fig. 29: Realization of a digital attenuator and calculating the derivative of an input signal

7.3.3 Basic kernels

Identity:

$$x[n] * \delta[n] = x[n]$$

Scaling:

$$x[n] * k \cdot \delta[n] = kx[n]$$

Shift:

$$x[n] * \delta[n - a] = x[n - a]$$

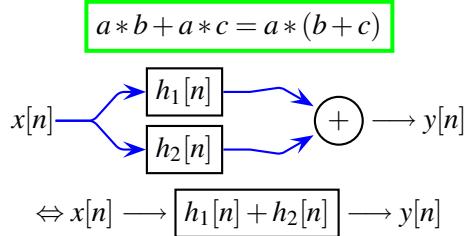
Integrator:

$$h[n] = \begin{cases} 1 & n \geq 0 \\ 0 & n < 0 \end{cases}$$

Differentiator:

$$h[n] = \delta[n] - \delta[n - 1]$$

7.3.4 Distributive property



From the distributive property, it follows that parallel systems whose output is added can be treated in the way that you add the systems (add its impulse response and then treat it as one system).

7.3.5 Exercise

Given $x[n]$ a ‘pulse-like’ signal ($x[n] = 0$ for small and large n), what is the result of

$$x[n] * x[n] * x[n] * \dots * x[n] = ?$$

Well, remember the **central limit theorem**. The result will be approximately Gaussian with $\sigma = \sigma_x \cdot \sqrt{m}$ and shifted in time, due to the latency which comes from the fact that the pulse $x[n]$ lies in the positive range of n .

7.4 Correlation functions

One useful application of the convolution, which is essentially the convolution of one signal with itself, is the *correlation function*. The *cross-correlation* is a measure of similarity of two signals, commonly used to find features in an unknown signal by comparing it to a known one. It is a function of the relative time between the signals and has applications in pattern recognition. A high value of the cross-correlation function for a given lag of time indicates a high similarity of the two signals at this time lag. In an *auto-correlation*, which is the cross-correlation of a signal with itself, there will always be at least one peak at a lag of zero.

7.4.1 Cross-correlation

Given two functions $f, g : D \rightarrow \mathbb{C}$, where $D \subseteq \mathbb{R}$, the **cross correlation** of f with g :

$$(f \circ g)(t) := K \int_D f(\tau) g(t + \tau) d\tau .$$

The cross-correlation is similar in nature to the convolution of two functions. Whereas convolution involves reversing a signal, then shifting it and multiplying it by another signal, correlation only involves shifting it and multiplying (no reversing).

7.4.2 Auto-correlation

$$A_g(t) := g \circ g = K \int_D g(\tau)g(t+\tau)d\tau$$

. The auto-correlation can be used to detect a known waveform in a noisy background, e.g., echoes of a signal. This can also be used to detect periodicities in a very noisy signal. The auto-correlation function of a periodic signal is also a periodic signal with the same period (but the phase information is lost). Because white noise at one time is completely independent of white noise at a different time, the auto-correlation function of white noise is a δ pulse at zero. So, for the analysis of periodicities, you just look at the auto-correlation function for bigger time lags and ignore the values around zero, because this area contains only the information about the strength of the noise contribution.

7.4.3 Discrete correlation

For discrete systems and signals we use the discrete version of the correlation integral: Given $f, g : D \rightarrow \mathbb{C}$, where $D \subseteq \mathbb{Z}$, the **discrete correlation**:

$$(f \circ g)[n] := \alpha \sum_{k \in D} f[k]g[n+k] ,$$

which is identical to

$$f[n] \circ g[n] = f[n] * g[n] .$$

8 Fourier transform

The *Fourier transform* is a linear operator that maps complex functions to other complex functions. It decomposes a function into a continuous spectrum of its frequency components, and the inverse transform synthesizes a function from its spectrum of frequency components. The Fourier transform of a signal $x(t)$ can be thought of as that signal in the *frequency domain* $X(\omega)$.

time domain	\longrightarrow	frequency domain
$x(t)$		$X(\omega)$

Information is often hidden in the **spectrum** of a signal. Figure 30 shows common waveforms and its Fourier transforms. Also looking at the *transfer function* of a system shows its frequency response. The Fourier transform is, therefore, a commonly used tool. As you will see later, a discretized version of the Fourier transform exists which is the *Discrete Fourier Transform*.

Given $f : D \rightarrow \mathbb{C}$, where $D \subseteq \mathbb{R}$, the **Fourier transformation** of f is:

$$F(\omega) := \int_D f(t) e^{-i\omega t} dt$$

and the **inverse Fourier transformation**

$$f(t) := \int_{-\infty}^{\infty} F(\omega) e^{+i\omega t} d\omega .$$

The Fourier transform can also be expressed using the convolution

$$F(\omega) = [f(t) * e^{i\omega t}]_{t=0} .$$

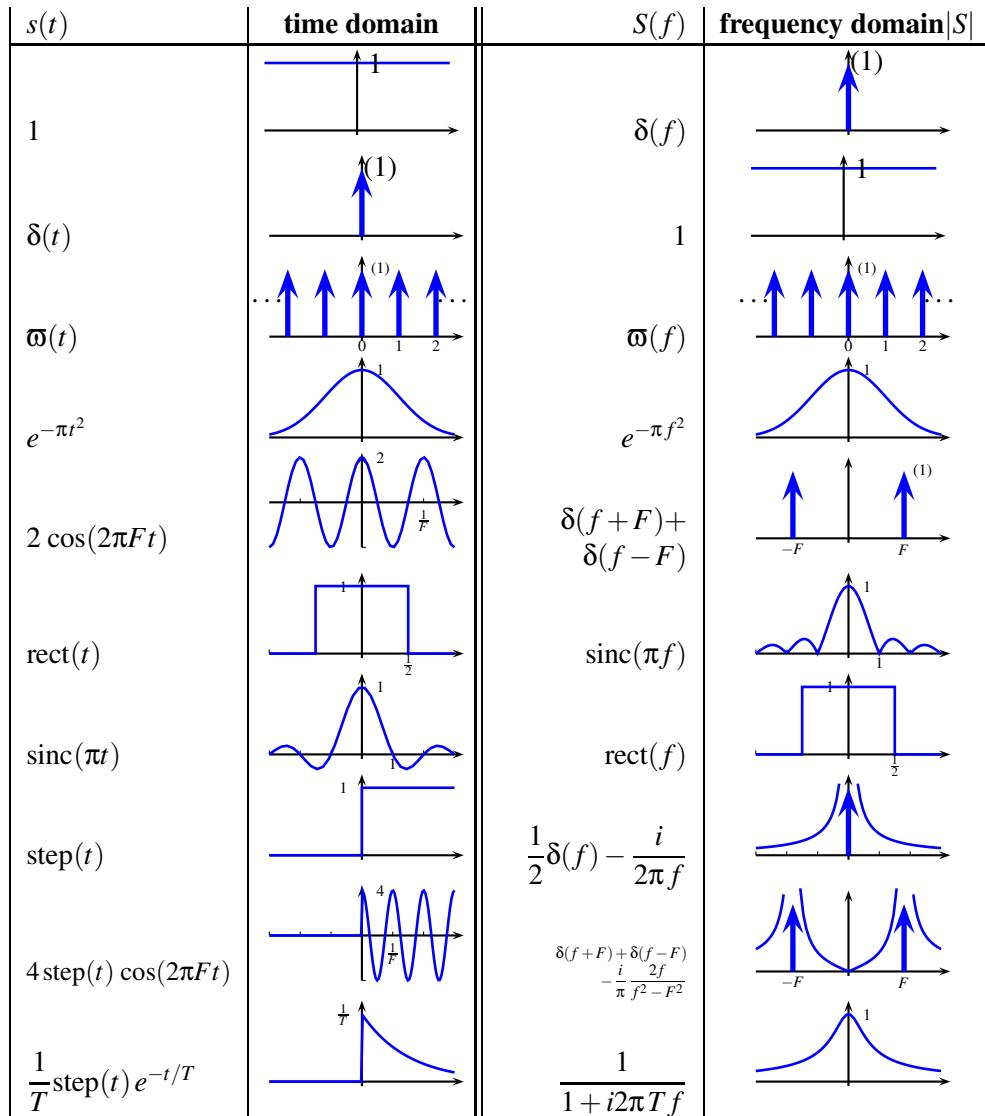


Fig. 30: Fourier transformation examples of common waveforms

8.1 Calculation with Fourier transforms

For a *real* input, the transformation produces a *complex* spectrum which is symmetrical:

$$X(\omega) = X^*(-\omega)$$

complex conjugate

The Fourier transform of a cos-like signal will be purely real, and the Fourier transform of a sin-like signal will be purely imaginary. If you apply the Fourier transform twice, you get the time-reversed input signal $x(t) \xrightarrow{\text{FT}} X(\omega) \xrightarrow{\text{FT}} x(-t)$. In the following, most important calculation rules are summarized:

Symmetry: $\text{FT}^2\{x(t)\} = x(-t)$

Linearity: $\text{FT}\{c_1 x_1(t) + c_2 x_2(t)\} = c_1 X_1(\omega) + c_2 X_2(\omega)$

Scaling:

$$\mathbf{FT}\{x(\lambda t)\} = \frac{1}{|\lambda|} X\left(\frac{\omega}{\lambda}\right)$$

Convolution:

$$\mathbf{FT}\{x_1(t) * x_2(t)\} = X_1(\omega) \cdot X_2(\omega) \quad ; \quad \mathbf{FT}\{x_1(t) \cdot x_2(t)\} = X_1(\omega) * X_2(\omega) \quad (13)$$

Integration:

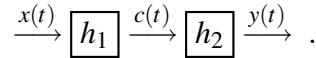
$$\mathbf{FT}\left\{\int_{-\infty}^t h(\tau) d\tau\right\} = \frac{1}{i\omega} X(\omega) + \frac{1}{4\pi} \underbrace{\left[\int_{-\infty}^{\infty} h(\tau) d\tau\right]}_{\text{DC offset}} \delta(\omega) \quad (14)$$

Time-shift:

$$\mathbf{FT}\{x(t + t_0)\} = e^{i\omega t_0} X(\omega)$$

8.2 The transfer function

Consider the following signal path consisting of two linear systems with impulse responses h_1 and h_2 :



The output signal will be the convolution of the input signal with each of the impulse response vectors

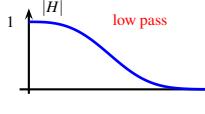
$$y(t) = x(t) * h_1 * h_2. \quad (15)$$

If we now look at the spectrum of the output signal $Y(\omega)$ by Fourier transforming Eq. (15), we get

$$\Rightarrow Y(\omega) = X(\omega) \cdot H_1(\omega) \cdot H_2(\omega).$$

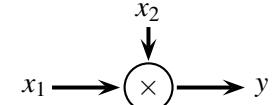
Transfer Functions

Here we made use of the calculation rule (13), that the Fourier transform of a convolution of two signals is the product of the Fourier transforms of each signal. In this way, we are going to call the Fourier transforms of the impulse responses *transfer functions*. The transfer function also completely describes a (linear) system; it contains as much information as the impulse response (or kernel) of the system. It is a very handy concept because it describes how the spectrum of a signal is modified by a system. The transfer function is a complex function, so it not only gives the amplitude relation $|H(\omega)|$ of a system's output relative to its input, but also the phase relations. The absolute value of the transfer function can tell you immediately what kind of filter characteristic the system has. For example, a function like $|H| =$



behaves like a low-pass.

It is now also very easy to tell what the output spectrum of a multiplier will be:



$$y(t) = x_1(t) \cdot x_2(t)$$

$$\Rightarrow Y(\omega) = X_1(\omega) * X_2(\omega).$$

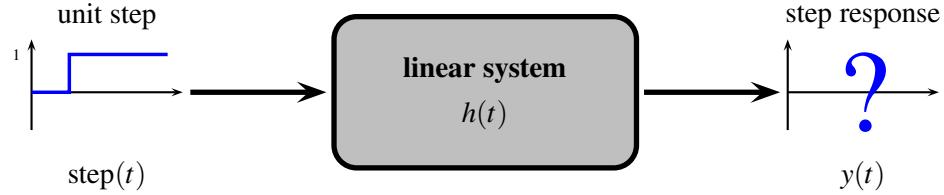


Fig. 31: What is the step response of a dynamic system?

It is the convolution of the two input spectra. In the special case, where one input signal consists only of a single frequency peak, the spectrum of the second input will be moved to this frequency. So a multiplier (sometimes also called a *mixer*) can be used to shift spectra. Exercise: what does the resulting spectrum look like if you have a single frequency on each of the two inputs? Which frequency components will be present? Do not forget the negative frequencies!

8.3 Step response

Earlier in this lecture we defined the impulse response of a system. This was a way to extract the essential information of that system. But this is not the only way to do it. An equivalent method uses the *step response* instead. The step response is the response of a system to a unity step. Unity step means, the input changes instantly from 0 to unity value (1). The system will react on this excitation showing its step response (see Fig. 31). It also contains all the information of the system, and can also be used as a fingerprint, exactly the same as the impulse response. There are rather practical reasons why one might prefer to look at the step response: Knowing the step response of a system gives information on the dynamic stability of such a system and on its ability to reach a stationary state, starting from another state.

Showing the equivalence to the impulse response is now an easy task with the convolution rule (13) and the integration rule (14) of the Fourier calculus:

$$\begin{aligned}
 y(t) &= \text{step}(t) * h(t) = ? \\
 Y(\omega) &= \underbrace{\left(\frac{1}{i\omega} + \frac{1}{4\pi} \delta(\omega) \right)}_{\text{from table}} \cdot H(\omega) = \underbrace{\frac{H(\omega)}{i\omega}}_{\text{low-pass}} + \underbrace{\frac{1}{4\pi} \delta(\omega) H(\omega)}_{\text{DC offset}} \\
 y(t) &= \boxed{\int_{-\infty}^t h(\tau) d\tau} \xrightarrow{\text{FT}^{-1}}
 \end{aligned}$$

The step response is the integral over time of the impulse response.

8.3.1 Correlation revisited

Coming back to correlation, what does the spectrum of the correlation function tell us?

auto-correlation:

$$s(t) * s(-t) \xrightarrow{\text{FT}} S(\omega) \cdot S^*(\omega) = |S(\omega)|^2$$

Energy spectrum

The spectrum of the auto-correlation function of a signal $s(t)$ is identical to its *energy spectrum*. The information about phase (or time/shift/location) is lost, so one can say that the auto-correlation function is *time invariant*.

cross-correlation:

$$s(t) * g(-t) \xrightarrow{\text{FT}} S(\omega) \cdot G^*(\omega)$$

Here, the *real* part of the spectrum of the cross-correlation of two signals tells us about parts which are *similar*, and the *imaginary* part of the spectrum tells us about parts which are *not correlated*.

9 Laplace transform

You have seen how handy the Fourier transformation can be in describing (linear) systems with $h(t)$. But a Fourier transform is not always defined:

- for example, $x(t) = e^{-t}$ has an infinite frequency spectrum $X(\omega) > 0$ everywhere;
- for example, $x(t) = e^t$ is unbounded and can not even be represented;
- for example, $\text{step}(t) \rightarrow$ infinite frequency spectrum;
- a lot of δ -functions appear, etc.

To handle this, we decompose these functions, not only into a set of ordinary cosine and sine functions, but we also use exponential functions and exponentially damped or growing sine and cosine functions. It is not so complicated to do. We just substitute the frequency term $i\omega$ by a general complex number p . You can look at this as introducing a complex frequency

$$p = \sigma + i\omega ,$$

where ω is the known real frequency and σ is a (also real) damping term. The functions to deal with now become

$$f(t) = e^{-pt} = e^{-\sigma t} \cdot e^{-i\omega t} .$$

Instead of the Fourier transform we now introduce a more general transform, called the *Laplace transform*: Given $s : \mathbb{R}^+ \rightarrow \mathbb{R}$, the **Laplace transformation** of s is:

$$s(t) \xrightarrow{\text{L}} S(p) := \int_0^\infty s(t) e^{-pt} dt$$

and the **inverse transformation**

$$S(p) \xrightarrow{\text{L}^{-1}} s(t) := \begin{cases} \frac{1}{2\pi i} \int_{\sigma-i\infty}^{\sigma+i\infty} S(p) e^{pt} dp & \text{for } t \geq 0 \\ 0 & \text{for } t < 0 . \end{cases}$$

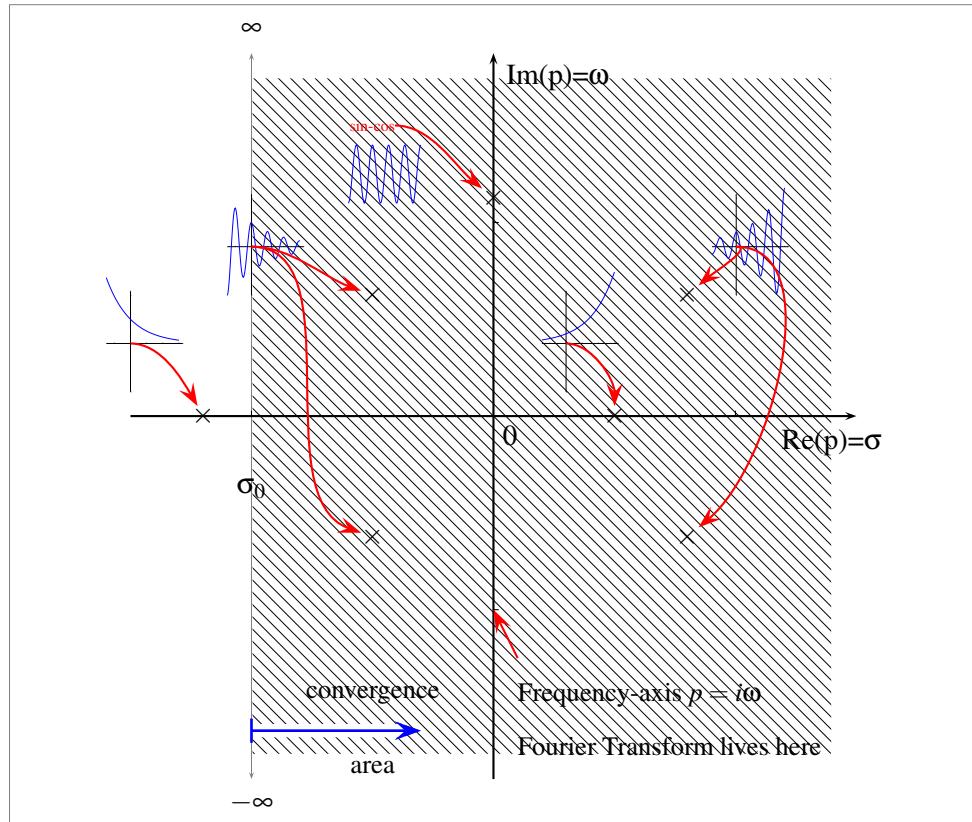
Remember:

- $s(t)$ is real and $s(t) = 0$ for $t < 0$.
- $S(p)$ is a complex function: $\mathbb{C} \xrightarrow{S} \mathbb{C}$ (in contrast to the Fourier transformation, where $\mathbb{R} \xrightarrow{F} \mathbb{C}$).

We shall come back to the inverse Laplace transform later in Section 9.5.

9.1 Region of convergence

As we mentioned before, the Fourier transformation was not always defined, even a simple sine wave produced a δ -peak in its spectrum which causes a problem. So does the Laplace transform always exist? The Answer is: no, but there is an area $\text{Re}(p) > \sigma_0$; $\sigma_0 \in \mathbb{R}$ where it exists (*region of convergence*). This means $\exists M < \infty : |s(t)| \leq M e^{\sigma_0 t}$. If $\sigma_0 < 0$, the Fourier transform also exists.


 Fig. 32: The p plane

To see what that means in practice, it is useful to visualize the functions and numbers we deal with in a diagram. This is possible if we look at the complex plane shown in Fig. 32, called the p plane. Different points on the plane correspond to different types of base functions as shown. The ordinary sine and cosine functions are also present and live on the imaginary axis. If the imaginary axis lies inside of the convergence area, then also the Fourier transform exists, and you will get it if you go along the imaginary axis. In addition, you also get spectral values for other regions of the plane.

What is this good for? Well, it is good for solving differential equations, especially those for analog filters. Let us see how this works. Formally, we do exactly the same as we did with the Fourier transform:

$$x \longrightarrow \boxed{\text{System}} \longrightarrow y$$

$$\overset{L}{\downarrow} \quad \cdot \quad \overset{L}{\downarrow} \quad = \quad \overset{L^{-1}}{\downarrow} \quad \overset{L}{\downarrow} \quad Y$$

$$X \quad \cdot \quad H = Y$$

We will have the concept of transfer functions slightly extended onto the whole p plane, but the concept stays the same. So we may get answers to questions like: What filter do I need for getting a specific output $y(t)$? Or we can compose the system out of subsystems by multiplying the transfer functions of each of them, etc. This implies that we will have nearly the same or similar calculation rules as for the Fourier transformation, and indeed that is exactly true.

Calculating with the Laplace transform

As before, we have similar calculation rules to the ones for the Fourier transform. In addition, the integration and differentiation operations also can be transformed.

$$x(t) \xrightarrow{\mathbf{L}} X(p) \xrightarrow{\mathbf{L}^{-1}} x(t)$$

Linearity:

$$\mathbf{L}\{c_1x_1(t) + c_2x_2(t)\} = c_1X_1(p) + c_2X_2(p)$$

Scaling:

$$\mathbf{L}\{x(\lambda t)\} = \frac{1}{|\lambda|}X\left(\frac{p}{\lambda}\right); \quad \lambda > 0$$

Time-shift:

$$\mathbf{L}\{x(t - t_0)\} = e^{-pt_0}S(p) \quad ; \quad \mathbf{L}^{-1}\{X(p + p_0)\} = e^{-p_0t}s(t); \quad t_0 > 0$$

Convolution:

$$\mathbf{L}\{x_1(t) * x_2(t)\} = X_1(p) \cdot X_2(p) \quad ; \quad \mathbf{L}^{-1}\{X_1(p) * X_2(p)\} = x_1(t) \cdot x_2(t)$$

Integration:

$$\begin{aligned} \mathbf{L}\left\{\int_0^\infty s(\tau)d\tau\right\} &= \frac{S(p)}{p} \\ \mathbf{L}^{-1}\left\{\int_p^\infty S(p')dp'\right\} &= \frac{s(t)}{t} \end{aligned}$$

Differentiation:

$$\mathbf{L}\left\{\frac{d^n}{dt^n}s(t)\right\} = p^nS(p) \quad \text{if } \frac{d^k s}{dt^k}|_{t=0} = 0 \quad \forall k < n$$

Laplace transformation examples

$$\mathbf{L}\{\text{step}(t)\} = \frac{1}{p} \quad ; \quad \sigma > 0$$

$$\mathbf{L}\{\delta(t)\} = \mathbf{L}\left\{\frac{d}{dt}\text{step}(t)\right\} = \frac{p}{p} = 1$$

$$\mathbf{L}\{\text{step}(t - t_0)\} = \frac{e^{-t_0 p}}{p} \quad ; \quad t_0, \sigma > 0$$

$$\mathbf{L}\{e^{p_0 t} \text{step}(t - t_0)\} = \frac{1}{p - p_0} \quad ; \sigma > \text{Re}(p_0)$$

$$\mathbf{L}\{\cos(\omega_0 t) \text{step}(t)\} = \frac{1}{2} \left(\frac{1}{p - i\omega_0} + \frac{1}{p + i\omega_0} \right) = \frac{p}{p^2 + \omega_0^2}$$

$$\mathbf{L}\{\sin(\omega_0 t) \text{step}(t)\} = \frac{1}{2i} \left(\frac{1}{p - i\omega_0} - \frac{1}{p + i\omega_0} \right) = \frac{\omega_0}{p^2 + \omega_0^2} \quad ; \quad \sigma > 0$$

$$\mathbf{L}\{t^n \text{step}(t)\} = \frac{n!}{p^{n+1}} \quad ; \quad \sigma > 0, n \in \mathbb{N}_0$$

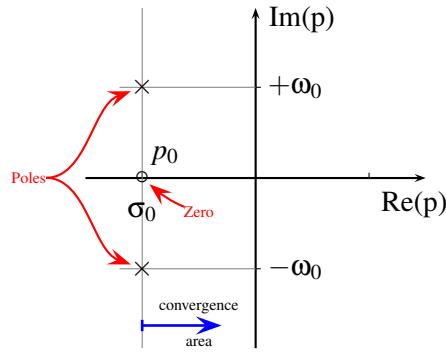


Fig. 33: Two poles and one zero in the p plane for the complex spectrum of a damped oscillation

9.2 Poles and zeros

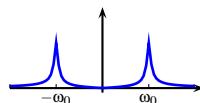
Now, we are interested in regions where the Laplace transformation does not exist. Such functions were also of interest when looking at the Fourier transformation. Remember: the spectrum of a sine function has two poles. These were expressed by δ -functions, but in practice this means that there are unphysical infinities and spectral peaks with zero bandwidth. Not nice. Because the Fourier spectrum is included in the Laplace spectrum, we also expect such poles here. Clearly they cannot be inside the convergence region, but anyway they are of interest. We shall soon see why. Other areas of interest are where the spectral components are exactly zero. That means, signals of these frequencies are completely rejected by the system. Zeros are of interest because they have the potential to cancel out poles. This is of major interest because system instabilities (which usually means that they have poles and the output signal will grow ad infinitum, mostly also oscillating) can be cured if another system with an appropriate pole is put in series (or in parallel) to the unstable one. But first, let us have a closer look at poles in the p plane. Remember that the real spectrum of the signals live only on the imaginary axis. We shall see how a pole (or a zero) near this axis will influence the spectrum.

First we are going to analyse the Laplace transform of a causal (all values for $t < 0$ are zero) damped oscillation:

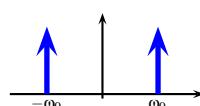
$$\mathbf{L}\{\cos(\omega_0 t) \exp(p_0 t) \text{step}(t)\} = \frac{p - p_0}{(p - p_0)^2 + \omega_0^2}, \quad (16)$$

where ω_0 is a real number and p_0 should be real but $p_0 < 0$. After having calculated the Laplace transform of this function (using the calculation rules given above), one can read from Eq. (16) that there is one zero at p_0 and two poles $p_0 \pm \omega_0$ (see Fig. 33). Since no pole lies to the right of p_0 , the region of convergence is $\sigma - \text{Re}(p_0) > 0$ ($\sigma_0 = \text{Re}(p_0)$).

Because the imaginary axis is inside the region of convergence, a Fourier transform also exists. It looks like



and you can see the resonance. If the poles were on the i -axis, a δ function would be necessary for expressing the spectrum:



9.3 Laplace transform of linear systems

To be more general, we now have a look at arbitrary linear systems. The general differential equation for such analog filters is

$$y(t) = \sum_{k=0}^M a_k \frac{d^k}{dt^k} x(t) + \sum_{k=1}^N b_k \frac{d^k}{dt^k} y(t) \quad (17)$$

Coefficients from the filter components (resistors, capacitors, inductivities, ...)

$$\begin{aligned} Y(p) &= \sum_{k=0}^M a_k p^k \cdot X(p) + \sum_{k=1}^N b_k p^k \cdot Y(p) \\ &= \frac{\sum_{k=0}^M a_k p^k}{1 - \sum_{k=1}^N b_k p^k} \cdot X(p) =: H(p) \cdot X(p) . \end{aligned}$$

Here, the transfer function $H(p)$ is defined for the whole complex plane using the coefficients from the differential equation. Its general form is

$$H(p) = \frac{\sum_{k=0}^M a_k p^k}{1 - \sum_{k=1}^N b_k p^k} = \frac{a_M \prod_{k=1}^M (p - p_{0k})}{-b_N \prod_{k=1}^N (p - p_{pk})} .$$

Factorizing is always possible. p_{0k} are the **zeros** and p_{pk} the **poles** of the transfer function. The transfer function is fully determined by its poles and zeros (except for a complex factor $\frac{a_M}{b_N}$)!

9.4 The transfer function

Consider you know all poles and zeros of a system, you can immediately estimate without too much calculating what the frequency response of the system will be. Therefore we look at the absolute value of the transfer function (it is also possible and also very easy to calculate the phases³):

$$|H(p)| = \left| \frac{a_M}{b_N} \right| \cdot \frac{\prod_{k=1}^M |p - p_{0k}|}{\prod_{i=1}^N |p - p_{pi}|} .$$

As a matter of interpretation you can think of this as

$$|H(p)| = \frac{\prod \text{distances between } z \text{ and zeros}}{\prod \text{distances between } z \text{ and poles}} .$$

Figure 34 illustrates how you can read the frequency response from a small diagram. You scan from zero along the imaginary axis (which gives you the real frequency ω) and from each point z you measure the distances between z and zeros and the distances between z and poles, multiply and divide them together and plot the result in the diagram in dependency of ω as shown in Fig. 34. This is the way your filter design tools do it (no magic).

9.5 The inverse Laplace transform

We have already defined the *inverse Laplace transformation* as

$$S(p) \xrightarrow{\text{L}^{-1}} s(t) := \frac{1}{2\pi i} \int_{\sigma-i\infty}^{\sigma+i\infty} S(p) e^{pt} dp \quad \text{for } t \geq 0 . \quad (18)$$

³Have a look at Section 11.1. It works the same here.

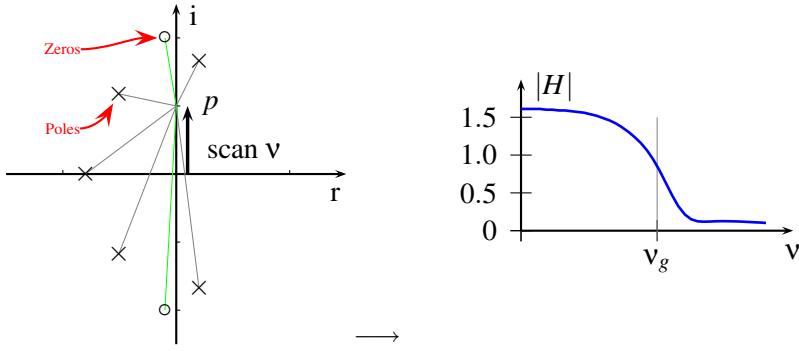


Fig. 34: Calculating the frequency response of a system from the poles and zeros of its transfer function

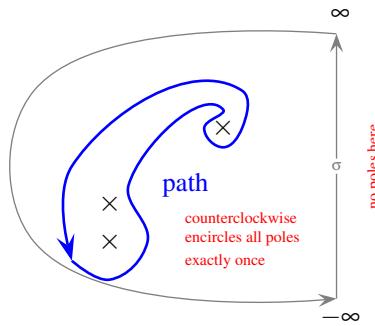


Fig. 35: Different integration paths around poles for the inverse Laplace transform

Now the question might be, why does the integration go from minus infinity to infinity exactly on the boundary of the convergence area? Indeed it is not necessary to do it this way. But, the integration path needs to encircle all poles (exactly once) anticlockwise. From residual theory we know that the contribution of a holomorph function on an area where there are no poles is zero for any closed integration loop. So we define

$$s(t) = \frac{1}{2\pi i} \oint_{\text{path}} S(p) e^{pt} dp$$

$$s(t) = \sum_{p_{pk}} \text{Res}_{p_{pk}} (S(p) e^{pt}) .$$

Recapitulate the definition of the *residuum* of a pole of function f at p_0 of order k :

$$\text{Res}_{p_0}(f) := \frac{1}{(k-1)!} \frac{d^{k-1}}{dp^{k-1}} [f(p) \cdot (p - p_0)^k] . \quad (19)$$

Looking at Fig. 35, we see how the definition (18) fits into that picture. The integration path $\int_{\sigma-i\infty}^{\sigma+i\infty}$ which lies completely inside or at least at the left border of the region of convergence (no poles are on the right side) already contains the whole residue information. As you can see in Fig. 35, the integration path can be extended by an anticlockwise encirculation of all poles in the far region of the p -plane. Now, the *initial value theorem* (also a calculation rule we have not yet mentioned)

$$s(0) = \lim_{p \rightarrow \infty} pS(p) < \text{const}$$

tells us that the behaviour of $S(p)$ for large $|p|$ should be at least a decay of the order of $|S(p)| < \frac{1}{|p|}$ so that for $\lim_{p \rightarrow \infty}$ the contribution to this path of integration is zero.

Examples

Finally, let us do two examples of inverse Laplace transforms to see how it works out:

1. p_0 single pole of

$$S(p) := \frac{1}{p+a}, \quad k=1, \quad p_0 = -a.$$

$$\begin{aligned} s(t) &= \text{Res}_{-a} [S(p) e^{pt}] \\ &= \frac{1}{(1-1)!} \cdot \frac{d^0}{dp^0} \left(\frac{1}{p+a} e^{pt} (p+a)^1 \right) \Big|_{p=-a} = e^{-at}. \end{aligned}$$

2. p_0 pole of third order of

$$S(p) := \frac{1}{p^3}, \quad k=3, \quad p_0 = 0.$$

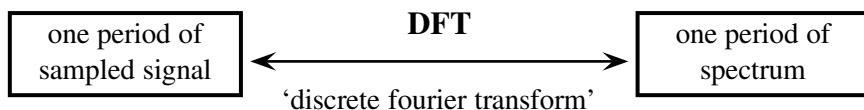
$$\begin{aligned} s(t) &= \text{Res}_0 [S(p) e^{pt}] \\ &= \frac{1}{2!} \cdot \frac{d^2}{dp^2} \left(\frac{1}{p^3} e^{pt} p^3 \right) \Big|_{p=0} = \frac{t^2}{2}. \end{aligned}$$

10 Discrete transforms and digital filters

In the previous section, we have mainly developed the mathematical formalism for analog signal processing and for continuous linear systems. Since the aim of this lecture is to treat digitized data and realize systems with digital signal processing, our job is now to transform the concepts in such a way that we can make use of them for the design of digital systems. Many concepts are the same or at least similar. The quantization of time and value has only a small effect, and in the limit for an infinitely high sampling rate and real numbers, it should be the same. Nevertheless, as mentioned before, we have to deal with these quantization effects, and there are fundamental differences, some of which we have already discussed. Remember: The spectrum of sampled signals is always **periodic**, because of aliasing, and the spectrum of a (continuous) periodic signal is also always periodic — this is also true for sampled signals.

10.1 The discrete Fourier transform

Because of this, we can define a transformation which maps between the periods in time domain and the periods in frequency domain. This can be done with the *discrete Fourier transform*. In contrast to the continuous Fourier transform, the discrete Fourier transform is always defined and maps uniquely between these domains, without ambiguities.



Given a period (of length N) of samples $s[n]$ ($\in \mathbb{R}$ or \mathbb{C}) with $n \in [0, \dots, N] \subset \mathbb{N}_0$, the *discrete Fourier transform* is defined as

$$S[k] = \sum_{n=0}^{N-1} s[n] e^{-2\pi i \frac{nk}{N}},$$

where $S[k] \in \mathbb{C}$, $k \in [0, \dots, N] \subset \mathbb{N}_0$.

The **inverse discrete Fourier transform** is

$$s[n] = \frac{1}{N} \sum_{k=0}^{N-1} S[k] e^{2\pi i \frac{nk}{N}}.$$

Calculation rules for the DFT are exactly the same as for the continuous Fourier transforms (**linearity, symmetry, etc.**), just replace ω with the *discrete frequency*

$$\omega_d : 2\pi \frac{k}{N} \Rightarrow S[\omega_d] = \sum_{n=0}^{N-1} s[n] e^{-i\omega_d n}$$

and then substitute $k = \frac{\omega_d}{2\pi} \cdot N$, $k \in \mathbb{N}_0$.

But there are also two important differences, one is the

Scaling: ($\lambda \in \mathbb{Z}$)

$$\text{DFT}\{x[\lambda n]\} = \frac{1}{|\lambda|} X\left(\frac{\omega_d}{\lambda}\right) = X[???$$

which will not work, because the length of the period itself is modified. A little modification needs to be applied to the

Time-shift:

$$\text{DFT}\{x[n + n_0]\} = e^{i\omega_d n_0} X[k].$$

And finally, with the **convolution**, one needs to pay attention, because if the result has more samples than the period, it needs to be folded back into the period.

10.2 The Fast Fourier Transform (FFT)

If the number of samples of the data snapshot is $N = 2^m$, $m \in \mathbb{Z}_0$, there is a fast and efficient algorithm to compute the discrete Fourier transform (DFT) and its inverse. The details of these (there are many) algorithms are beyond the scope of this lecture and so you may refer to the literature.

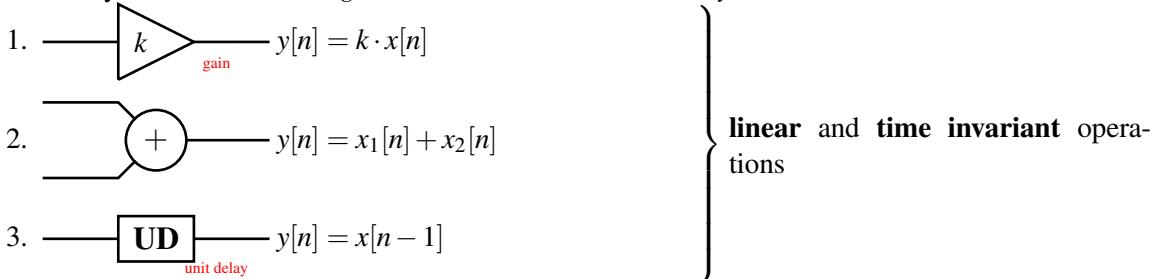
Since this algorithm can only be applied to snapshots with a number of samples that is a power of 2, there are several techniques, which match the number of samples N to 2^m , mainly

1. **zero-padding**
2. **windowing**, also for estimation of FT of an aperiodic signal and time-frequency analysis.

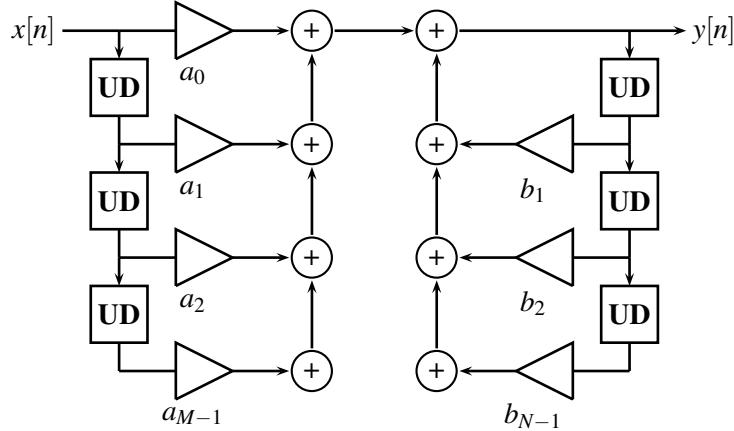
Here, you may also refer to the literature.

10.3 Digital filters

Let us consider the very basic systems and elementary linear operations we can think of. We find that there are only three of them: the *gain*, the *adder* and the *unit delay*.



Any combination of these operations is called a '**digital filter**'.

**Fig. 36:** Alternative presentation of Eq. (20)

In analogy to the differential equation for analog systems (see Eq. (17))

$$y(t) = \sum_{k=0}^N \alpha_k \frac{d^k}{dt^k} x(t) + \sum_{k=1}^M \beta_k \frac{d^k}{dt^k} y(t) ,$$

we can define a similar *equation of differences* for the digital systems which only consists of the above mentioned three operations (compare with the equivalent notation shown in Fig. 36):

$$\Rightarrow y[n] = \underbrace{\sum_{k=0}^{N-1} a_k x[n-k]}_{\text{direct}} + \underbrace{\sum_{k=1}^{M-1} b_k y[n-k]}_{\text{recursive}} . \quad (20)$$

Using the convolution operator we can also write

$$y[n] = a[M] * x[n] + b[N] * y[n] ,$$

where we have two filter kernels, one direct kernel $a[M]$ and one recursive kernel $b[N]$.

10.3.1 Impulse response of a digital filter

Now, what does the impulse response of the general digital filter denoted in Eq. (20) look like? Remember, the digital unit impulse is defined as

$$x[n] = \delta[n] := \begin{cases} 1 & n = 0 \\ 0 & n \neq 0 \end{cases} .$$

Let us write down the response to that input:

$$\Rightarrow h[n] = y[n] = \begin{cases} 0 & n < 0 \\ a_0 & n = 0 \\ a_n + \sum_{k=1}^{\min(n,N)} b_k h[n-k] & n > 0 \end{cases} .$$

Now we can distinguish between two different cases:

1. If $b_k \equiv 0 \rightarrow h[n] = a_n, n \geq 0$ we talk about a **Finite Impulse Response filter (FIR)** and
2. if at least one $b_{k_0} \neq 0$, then we call it **Infinite Impulse Response filter (IIR)**.

It is clear why it is named in this way: for the FIR filter the impulse response has only a finite number of non-zero values, which means that there is a n_f where $h[i] = 0 \quad \forall i > n_f$. In contrast to this, the impulse response will (in general) be of infinite length, although only a finite set of the coefficients (a_k, b_k) generate it.

10.3.2 Order of a filter

Besides the class of the digital filter (FIR or IIR), another important characteristic parameter is the *order* of the filter. It is defined as follows:

$$\exists(N, M) : (a_n = 0 \quad \forall n > N) \wedge (b_n = 0 \quad \forall n > M)$$

$$\text{Order} := \max(N, M).$$

So the order is the minimum number of coefficients needed to implement it. The order is also a measure for the maximum **latency** (or delay) of a filter, because it counts the maximum number of unit delays needed to complete the output (refer to Fig. 36).

For an FIR filter, the order of the filter is equal to the length of the impulse response. For an IIR filter this is not the case.

10.3.3 Transfer function of a digital filter

With the help of the discrete Fourier transform, it is straightforward to find an expression for the general form of the *transfer function* of such a digital filter, starting with Eq. (20):

$$y[n] = \sum_{k=0}^N a_k x[n-k] + \sum_{k=1}^M b_k y[n-k]$$

DFT time shift rule

$$Y(\omega_d) = \sum_{k=0}^N a_k \overbrace{X(\omega_d) e^{-i\omega_d k}}^{\text{DFT}} + \sum_{k=1}^M b_k \overbrace{Y(\omega_d) e^{-i\omega_d k}}^{\text{time shift rule}}$$

$$X(\omega_d) \neq 0 \quad \forall \omega_d$$

$$H(\omega_d) := \frac{Y(\omega_d)}{X(\omega_d)} = \frac{\sum_{k=0}^N a_k (e^{-i\omega_d})^k}{1 - \sum_{k=1}^M b_k (e^{-i\omega_d})^k}. \quad (21)$$

Remember that the digital frequency ω_d is periodic with ω_s ($-\pi < \omega_d < \pi$).

Further, remember that we developed a similar formula in Section 9.3. In that case, we used the Laplace transform (as a more general expression which was extended to complex frequencies) instead of the Fourier transform. It is also possible to do (more or less) the same thing here for the digital systems. We can substitute $z := e^{i\omega_d}$ and extend it to the complex by including a damping term σ

$$z := e^{i\omega_d - \sigma}.$$

The resulting transform (which is a modified DFT) will be called *z transform*.

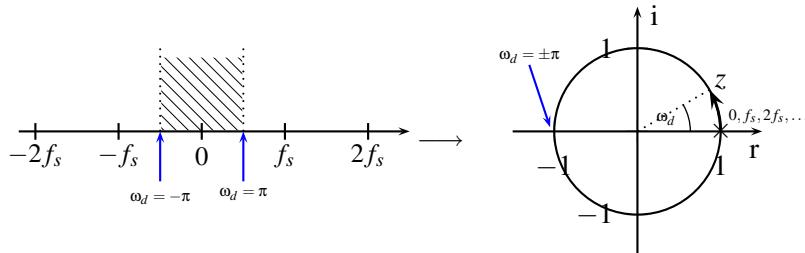
11 The z transform

Introducing the z -transform, we develop a tool which is as powerful as the Laplace transform mentioned in Section 9, but also applicable for digital systems and digital signals. The concept is based on the periodicity of the spectra of digital signals. With a suitable transformation, all tools and methods developed for analog systems and analog signals using the Laplace transform can be adapted for use with digital ones.

Starting with the discrete transfer function, we simply do the substitution $z := e^{i\omega_d} (= e^{2\pi i \frac{k}{N}})$ in Eq. (21):

$$H(\omega_d) \xrightarrow{\text{substitution}} H(z) = \frac{\sum_{k=0}^N a_k z^{-k}}{1 - \sum_{k=1}^M b_k z^{-k}}.$$

This substitution maps the frequency axis to the unit circle in the complex z -plane:



This concept is useful because it automatically accounts for the periodicity of ω_d . The z -plane (or the unit circle) is a representation of one period of the digital frequency. Frequencies above the Nyquist frequency are automatically mapped to the place where its aliasing frequency would be. So there will be no aliasing from now on.

Now, we can extend this concept to the whole complex plane $z \in \mathbb{C}$. We therefore add a damping term to the digital frequency ω_d :

$$\begin{aligned} \omega_d \in \mathbb{R}_{[-\pi, \pi]} &\longrightarrow \omega_{d_c} \in \mathbb{C} \\ \omega_{d_c} &= \omega_d + i\sigma , \\ \Rightarrow z &= e^{i\omega_{d_c}} = e^{i\omega_d - \sigma} . \end{aligned}$$

As shown in Fig. 37, different points and regions for z correspond to different classes of (sampled) functions. As with the p -plane for the Laplace transform, besides discrete sine and cosine functions there are also discrete exponential functions, as well as exponentially damped and growing functions. Together, this set of functions forms a basis for the decomposition of any discrete signal. In particular, for the expression of the transfer functions of discrete systems, we can find a very handy way; it is similar to what we did with the transfer functions of analog systems, factorized in poles and zeros in the p -plane.

11.1 Poles and zeros in the z -plane

Also, in the z -plane, a factorization of Eq. (21) is always possible:

$$H(z) = \frac{\alpha_0 \prod_{k=1}^M (1 - z_{0k} z^{-1})}{\prod_{i=1}^N (1 - z_{pi} z^{-1})},$$

where z_{0k} are the *zeros* and z_{pi} are the *poles* of the function. The absolute value of the frequency response can be calculated in a similar way, by scanning z along the unit circle as shown in Fig. 38

$$|H(z)| = \frac{\prod \text{distances between } z \text{ and zeros}}{\prod \text{distances between } z \text{ and poles}}.$$

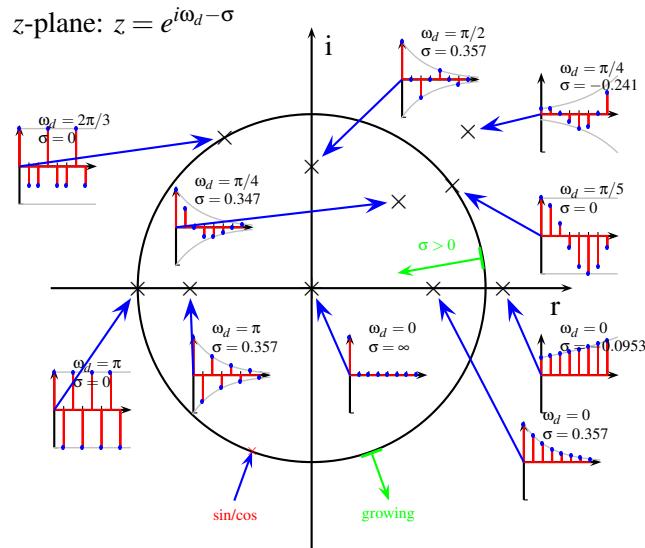


Fig. 37: The z -plane

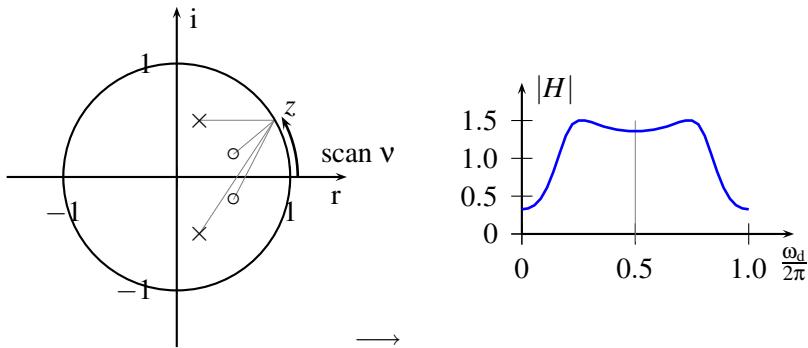


Fig. 38: Calculation of the frequency response of a digital system from its poles and zeros of its transfer function in the z -plane

A handy formula for the phases of $H(z)$ is also available

$$\angle H(z) = \sum \angle(z - \text{zeros}) - \sum \angle(z - \text{poles}).$$

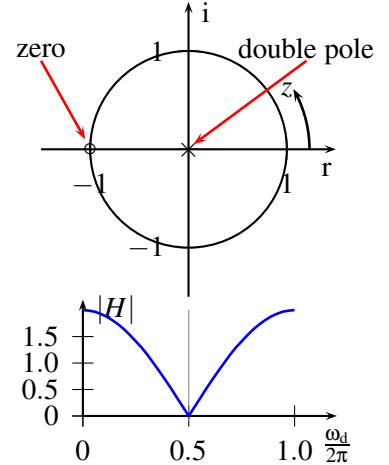
All this can be done by a very easy calculation, or even graphically, if you like.

Examples

1. 2nd order non-recursive filter (FIR)

$$a_0 = \frac{1}{2} \quad ; \quad a_1 = 1 \quad ; \quad a_2 = \frac{1}{2} \quad ; \quad b_1 = b_2 = 0$$

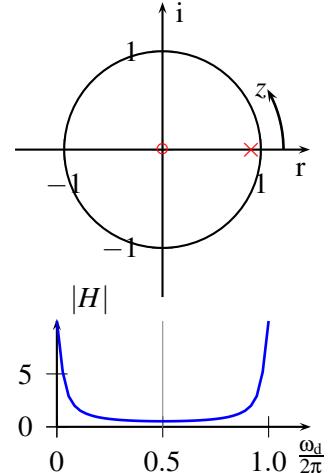
$$\begin{aligned}
 h[n] &= \left\{ \frac{1}{2}, 1, \frac{1}{2} \right\} \\
 \rightarrow y[n] &= \frac{1}{2}x[n] + x[n-1] + \frac{1}{2}x[n-2] \\
 \rightarrow H(e^{j\omega_d}) &= \frac{\frac{1}{2} + e^{-j\omega_d} + \frac{1}{2}e^{-2j\omega_d}}{1} \\
 \rightarrow H(z) &= \frac{1}{2} + z^{-1} + \frac{1}{2}z^{-2} = \frac{z^{-2}}{2}(z+1) \\
 \text{Poles: } z_{p1} &= z_{p2} = 0, \quad \text{Zeros: } z_{01} = -1
 \end{aligned}$$



2. 1st order recursive filter (IIR)

$$a_0 = 1 \quad ; \quad a_1 = a_2 = \dots = a_n = 0 \quad ; \quad b_1 = 0.9 \quad ; \quad b_2 = \dots = b_m = 0$$

$$\begin{aligned}
 h[n] &= (0.9)^n \quad ; \quad n \geq 0 \\
 \rightarrow H(z) &= \frac{1}{1 - 0.9z^{-1}} = \frac{z}{z - 0.9} \\
 \text{Poles: } z_{p1} &= 0.9, \quad \text{Zeros: } z_{01} = 0
 \end{aligned}$$



11.2 The z -transformation

Given $h : \mathbb{Z}_0^+ \rightarrow \mathbb{R}$, the **z -transformation** of h is

$$h[n] \xrightarrow{Z} H(z) := \sum_{n=-\infty}^{\infty} h[n]z^{-n}$$

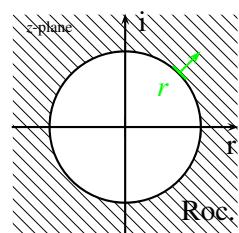
causal: $h[n] = 0 \quad \forall n < 0$

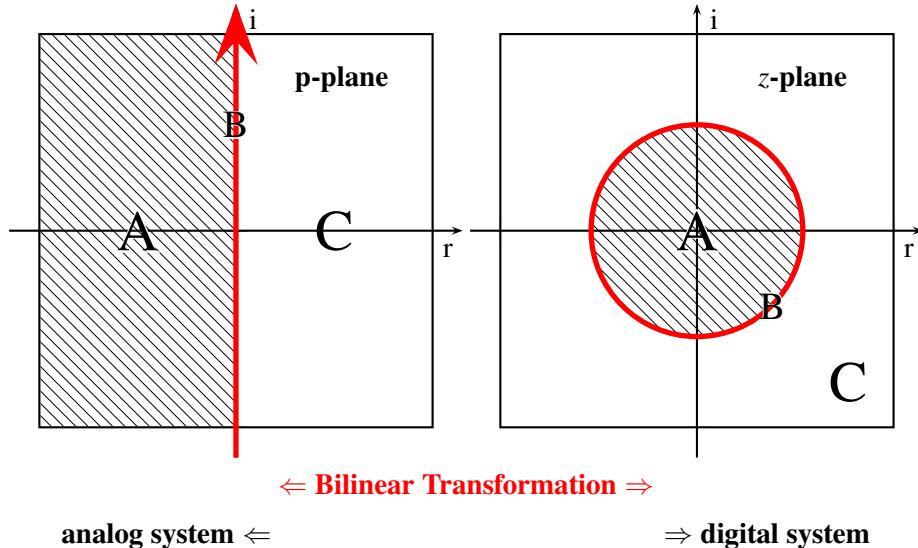
This is the same as DFT plus substitution $z := e^{j\omega_d}$. [2ex]

Region of convergence

The Region of convergence (Roc) can be defined as follows:

$$\begin{aligned}
 \text{Roc} &:= \left\{ z : \left| \sum_{n=-\infty}^{\infty} h[n]z^{-n} \right| < \infty \right\} \\
 \Rightarrow \text{if } |h[n]| &< M r^n \quad \forall n, \quad r \in \mathbb{R}^+ \quad \Rightarrow h(z) \text{ exists } \forall z \in \mathbb{C} : |z| > r \\
 &\text{(all the poles of } h(z) \text{ lie inside a circle of } |z| < r.)
 \end{aligned}$$




Fig. 39: p -plane and z -plane

Signal	z -Transform	Pole	Zero	Roc
$\delta[n]$	$1 \cdot z^0 = 1$	-	-	$z \in \mathbb{C}$
$\delta[n - n_0]; \quad n_0 \in \mathbb{N}$	z^{-n_0}	0	-	$z \neq 0$
step[n]	$\sum_{k=0}^{\infty} z^{-k} = \frac{1}{1-z^{-1}} = \frac{z}{z-1}$	1	0	$ z > 1$
$h[n] = \alpha n$	$\frac{\alpha z}{(z-1)^2}$	1	0	$ z > 1$
$h[n] = b^n; \quad b \in \mathbb{R}$	$\sum_{k=0}^{\infty} \left(\frac{b}{z}\right)^k = \frac{z}{z-b}$	b	0	$ z > b$
$h[n] = \cos(\omega n)$	$\frac{z(z - \cos(\omega))}{z^2 - 2z\cos(\omega) + 1}$?	$0, \cos(\omega)$	$ z > 1 - ?$
$h[n] = \sin(\omega n)$	$\frac{z\sin(\omega)}{z^2 - 2z\cos(\omega) + 1}$?	0	$ z > 1 - ?$

Fig. 40: z -transform of common signals

If the impulse response of the system is decaying faster than or approximately exponentially, then all poles lie inside a circle of finite size, and the z -transform exists outside of that circle.

11.3 z -transforms of common signals

Some z -transforms of common signals are given in Fig. 40 and the most common calculation rules are summarized in Fig. 41. As you can see, the behaviour is quite similar to what you already know from the Fourier and the Laplace transforms. The convolution rules, in particular, are the same. Also, the regions of the planes can be comparable if you want to compare analog systems and corresponding discrete systems by looking at their poles and zeros in the p -plane or z -plane, respectively. If you map both planes with the so-called *bilinear transformation* on each other, as shown in Fig. 39, you can directly compare frequency responses, stability issues, and a lot more. This mapping from analog to digital is also a common technique in digital filter design.

	Signal	z-Transform	$r_2 < z < r_1$
Linearity:	$a_1x_1[n] + a_2x_2[n]$	$a_1X_1(z) + a_2X_2(z)$	$\mathbb{D}_1 \cap \mathbb{D}_2$
Time shifting:	$x[n-k]$	$z^{-k}X(z)$	$\begin{cases} z \neq 0 \text{ if } k > 0 \\ z \neq \infty \text{ if } k < 0 \end{cases}$
Scaling:	$a^n x[n]$	$X(a^{-1}z)$	$ a r_2 < z < a r_1$
Time reversal:	$x[-n]$	$X(z^{-1})$	$\frac{1}{r_1} < z < \frac{1}{r_2}$
Convolution:	$x_1[n] * x_2[n]$	$X_1(z)X_2(z)$	$\mathbb{D}_1 \cap \mathbb{D}_2$
Multiplication:	$x_1[n] \cdot x_2[n]$	$\frac{1}{2\pi i} \oint_C X_1(v)X_2(\frac{z}{v}) v^{-1} dv$	$r_{1l}r_{2l} < z < r_{1u}r_{2u}$
Differentiation:	$x[n] - x[n-1]$	$\frac{z-1}{z} X(z)$	
initial value:	$x[0] =$	$\lim_{z \rightarrow \infty} X(z)$	
final value:	$x[\infty] =$	$\lim_{z \searrow 1} (z-1)X(z)$	

Fig. 41: Calculating with the *z*-transform

11.4 The inverse *z*-transformation

Similar to the inverse Laplace transform, we now define the *inverse z-transform* as follows:

$$X(z) \xrightarrow{\text{Z}^{-1}} x[n] = \frac{1}{2\pi i} \oint_C X(z)z^{n-1} dz , \quad (22)$$

where C is an anticlockwise, closed path encircling the origin and entirely in the **region of convergence**. C must encircle all of the poles of $X(z)$. In this case Eq. (22) can be expressed using the calculus of residuals

$$x[n] = \sum_{z_{pk}} \text{Res}_{z_{pk}} (X(z)z^{n-1}) .$$

Example

z_0 single pole of

$$X(z) := \frac{1}{z} , \quad k = 1 , \quad z_0 = 0 .$$

$$\begin{aligned} x[n] &= \text{Res}_0 [X(z)z^{n-1}] \\ &= \frac{1}{(1-1)!} \cdot \frac{d^0}{dz^0} \left(\frac{1}{z} z^{n-1} \cdot (z-0)^1 \right) \Big|_{z=0} = \begin{cases} 1 & n=1 \\ 0 & n \neq 1 \end{cases} = \delta[n-1] . \end{aligned}$$

Remember the definition of the residuum, Eq. (19).

12 Digital filter design

In this section, I would like to give you some hints about how you can design a filter for your applications. We cannot go into details here, since filter design is a profession in itself and there are many books about it and also advanced toolboxes for computer-aided design. The sessions about modelling tools and control theory cover parts of it.

Having the mathematical concepts in mind, we can now use them. A common problem is to find the filter coefficients α_i and β_i (for analog filters) or a_i and b_i (for digital filters), or, if you want, to have a simple implementation, the filter kernel $h[n]$ of a FIR filter. You should have at least some idea about the frequency response; should it have low-pass, high-pass or band-pass characteristics, what is the centre or

edge frequency and what is the phase response of the system to be created? This is especially necessary if you design feedback loops, and stability is your concern.

Well, this is how you could start:

- Do not specify a_i, b_i but **zeros** z_{0k} and **poles** z_{pi} by the transfer function $H(z)$, $H(\omega_d)$, the impulse response $h[n]$, or the step-response $s[n]$. Usually, you do this by trial and error: you place some poles and zeros on the z -plane and calculate the frequency response (if you are interested in the frequency domain), or the step response (if you are interested in time domain) or both. Then you can move these poles around to see how that changes the responses. You could add more zeros or poles and try to cancel out resonances if they bother you, etc.
- Then calculate a_i and b_i or $h[n]$ for implementation. The implementation is straightforward and not very difficult; if you keep the order of your filter small, there will not be so many surprises later.

To make this trial-and-error job a little more sophisticated, you should know that

1. Because a_i, b_i usually should be real (for implementation), $\rightarrow z_{0k}$ and z_{pi} need to be real or they appear in complex conjugate **pairs**.
2. The filter kernel should be finite or at least

$$\lim_{n \rightarrow \infty} h[n] \stackrel{!}{=} 0$$

otherwise the filter might be unstable. A consequence of this boundary is that $|z_{pk}| < 1$, which means the poles need to be located inside the unit circle.

Filter design check-list

Finally I shall give you a check-list for filter design:

1. Specify the *transfer function* $H(f)$.
2. Specify the type of the filter (*FIR* or *IIR*) *numerical stability, dispersion*.
3. Specify the *order* of the filter (number of poles and zeros) *compromise between implementational effort and approximation of ideal transfer function*.
4. Select the *method* for filter design:
 - numerical (computer-based) optimization of coefficients;
 - convert analog filter to digital filter *impulse response invariant (windowing) design, transfer function invariant design, bilinear transformation*;
 - use filter transformation prototypes.

13 The Kalman filter

In this section, I present the idea of a special, highly useful filter, the *Kalman filter*. Though this filter can be implemented as an ordinary digital IIR filter, the concept behind it may be more difficult to understand. The Kalman filter is useful to filter out the noise of a signal whose signal-to-noise ratio is very poor, but about which you know something of the underlying process producing the input stream of measurements. From this extra knowledge, one can take advantage and improve the signal quality, effectively removing noise from the signal. The Kalman filter does this best, it is the optimal filter with respect to virtually *any criterion* that makes sense.

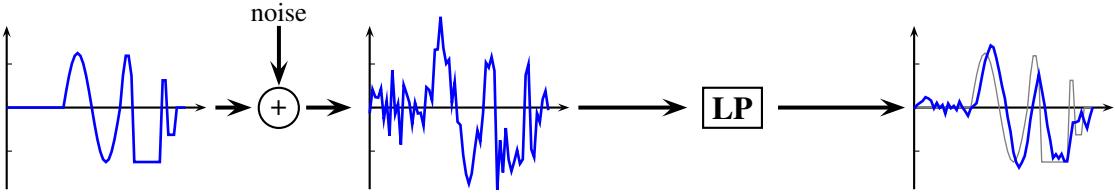


Fig. 42: Filtering a noisy signal with a low-pass filter. The result is time-shifted and the high-frequency components (the sharp edges) of the signal are not well reconstructed.

13.1 Fighting the noise

We discussed common sources of noise earlier in this lecture. The noise may come from the nature of the process that gives you the measurements or it may come from the detector or sensor (including the noise sources which belong to the digitization process). In any event, you very often end up with a stream of measurements $x[n]$ which has a bad signal-to-noise-ratio. For signal processing, you need to improve the signal quality, remove the noise, and since you cannot improve the measurement hardware, you will have to do your best to do it within the digital signal process itself.

The first idea which comes to mind is to use a low-pass filter (do some averaging of the input signal). This idea is not too bad and can work well if your sampling frequency is high enough and two side effects do not bother you: mainly the latency, time shift and phase response introduced with the filter and the fact that you remove only the high-frequency noise components. As a consequence, the higher harmonics of your signal may be smeared out and you keep the low-frequency noise components on your signal. If you can live with this, fine, but in many situations, like the one shown in Fig. 42, you might not be very satisfied with the result.

The Kalman filter can improve the situation. This means that it introduces nearly no latency while doing a good job of noise filtering and conserving the high-frequency components of your signal. Last but not least, the Kalman filter is still a causal system, but it can only work if you have some extra knowledge about the underlying process and if you are able to create a model (at least a very simple one) of it.

If you also have the chance to use non-causal filtering (maybe because the data is produced in chunks and can be processed as a whole), then techniques which are described in the section about wavelets may also be applicable.

13.2 Predictive/adaptive filters

First, the Kalman filter is an *adaptive filter*; this means that its filter coefficients are not constant, but instead change in adaptation to the filtered data itself. Figure 43 explains the principle. A special system identification block within the filter analyses the data and calculates new optimal filter coefficients from it using sophisticated algorithms.

The second attribute of the Kalman filter is that it is *predictive*. This simply means that it has some algorithm which allows it to calculate a prediction (expectation) of the current measurement or input value, based on the latest measurements and a *model* of the underlying process. Both the measured value and the predicted value are then combined to produce the output of the filter.

The trick is how to do this combination in such a way that, depending on the certainty of either the predicted or the measured value, the output represents the best certainty of both together. This trick is based on the rule of ‘propagation of error’ (a well-known concept which will be discussed soon). This way, it is guaranteed that the output variance is always smaller than (or equal to) the variance of the input signal.

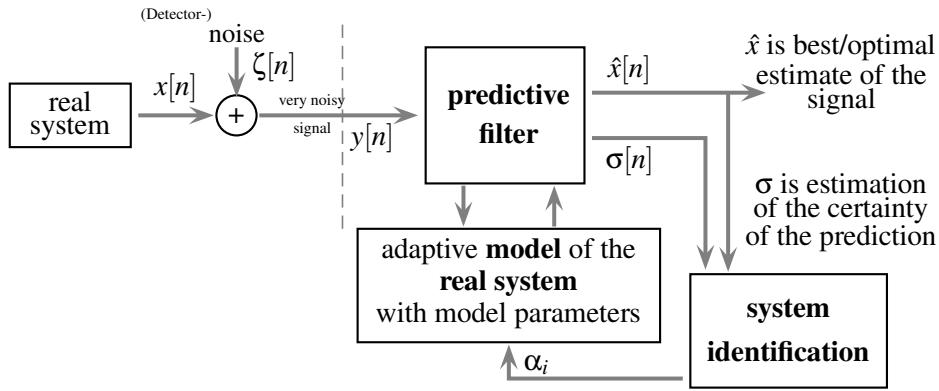


Fig. 43: Principle of an adaptive and predictive filter (like the Kalman filter). The filter consists of a model of the underlying process which can calculate predicted values from the model parameters (and the latest measured values). The model parameters are adapted from a system identification block. The algorithm is essential for the optimality of the Kalman filter; it always follows the variance of the measured data and the predicted data, based on the rule of ‘propagation of error’. In this way, it is guaranteed that the output variance is always smaller than (or equal to) the variance of the input signal.

13.3 Example: navigation

To understand how that works, we are going to develop a simple, one-dimensional example: the estimation of the position of a boat on the (one-dimensional) ocean. Suppose we are only interested in one parameter (e.g., the latitude or longitude). For position measurements, the crew can use different methods, let us say a sextant for navigation with the sun or the stars and a modern GPS receiver. Depending on the person who is doing the position determination, the position values $x[n]$ may have different precision (expressed by the variances $\sigma[n]$), depending on the method used or the person who does it.

First think of the simple situation where the boat is more or less at rest and a first position measurement is done (at time t_1) with an uncertainty known to be σ_1 , which might be very large, because let us say a beginner does this navigation:

$$\text{first measurement: } x_1 \pm \sigma_1 := (x(t_1) \pm \Delta x(t_1)) .$$

Now we analyse how a second measurement (nearly at the same time $t_2 \approx t_1$)⁴

$$\text{second measurement: } x_2 \pm \sigma_2 := (x(t_2) \pm \Delta x(t_2))$$

can improve the knowledge of the position. Assume the uncertainty σ_2 of this second measurement is smaller than the first one (because now the captain himself did the measurement). You could throw away the first measurement and only use the second one. But this would be not the best solution, because the first measurement also contains information we could benefit from. So the clever way is to combine both measurements

$$\Rightarrow \text{best estimate: } \hat{x} = \frac{\sigma_2^2}{\sigma_1^2 + \sigma_2^2} \cdot x_1 + \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2} \cdot x_2$$

$$\text{uncertainty: } \hat{\sigma} = \sqrt{\frac{1}{\sigma_1^2 + \sigma_2^2}} \leq \min(\sigma_1, \sigma_2)$$

⁴For the moment time does not play a role because we assumed the boat to be at rest.

so that the variance $\hat{\sigma}$ of the resulting position measurement \hat{x} is even better than the best of each single measurement.

But what if some noticeable/relevant time has passed between the measurements?

To be more general we can say:

$$\begin{aligned}\hat{x}(t_2) &= \frac{\sigma_2^2}{\sigma_1^2 + \sigma_2^2} \cdot x(t_1) + \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2} \cdot x(t_2) \\ &= x(t_1) + \frac{\sigma_1^2}{\sigma_1^2 + \sigma_2^2} \cdot (x(t_2) - x(t_1)) .\end{aligned}$$

Now we consider a stream of (new) input data $x_{n+1} := x(t_{n+1})$ which should be combined with the latest best value \hat{x}_n to produce a new best value \hat{x}_{n+1} . (Remember: the variances σ_{n+1} of each measurement are assumed to be known.) This is trivial if the measurement device is the same for all measurements, since σ_{n+1} can be assumed to be constant. But even if σ_{n+1} is not known in advance, one can estimate it by calculating the variance (e.g., with the method of running statistics) of the input signal stream.

$$\hat{x}_{n+1} = \hat{x}_n + K_{n+1} (x(t_{n+1}) - \hat{x}_n) \quad (23)$$

$$\hat{\sigma}_{n+1} = \frac{1}{\sqrt{1/\hat{\sigma}_n^2 + 1/\sigma_{n+1}^2}} ; \quad (24) \quad \begin{array}{l} \text{'prediction equation'} \\ \text{'Kalman gain'} \end{array}$$

$$\text{where } K_{n+1} := \frac{\hat{\sigma}_n^2}{\hat{\sigma}_n^2 + \sigma_{n+1}^2} .$$

The new prediction (say, best estimate of the position) is based on the old prediction and the new measurement value. There is one curiosity: $\hat{\sigma}$ is shrinking all the time! Becoming smaller and smaller and approaching zero. This means that with time you can get a really precise value of your position with any bad and ugly measurement device. But remember that this holds only if the boat is really at rest and the position does not change with time.

Finally, we want to discuss the more realistic case that the boat is moving (with a constant velocity v). We can now extend the example to be even more general:

Since we know the physical influence of a velocity of the boat's position

$$\frac{dx}{dt} = v + \Delta v$$

(this is the underlying process) we can introduce a model:

$$x(t) = v(t) \cdot (t - t_0) + x(t_0) ,$$

where $v(t) \pm \Delta v(t)$ is assumed to be known by a different measurement (called system identification). Besides this, we also assume that v is constant or changing only adiabatically (slowly compared to the sampling rate of the position measurements). The model also tells us the expected uncertainty of the calculated position value:

$$\sigma(t) = \sqrt{(\Delta v \cdot (t - t_0))^2 + (\sigma(t_0))^2} .$$

If, at this moment, you do not understand this formula, read the paragraph about 'propagation of error'. Figure 44 shows you what this means: Because the velocity also has a non-zero variance, the variances of the position derived from it become larger with time (σ is growing!), so the uncertainty is increasing!

Now let us see what this means for our example. Since the boat is moving, we cannot simply combine the latest best value with the new measurement, because some time has passed since the last

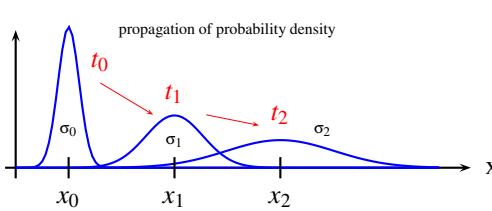


Fig. 44: Evolution of the probability density of the position derived from the model. Because the velocity also has a non-zero variance, the variances of the positions derived from it become larger with time, so the uncertainty is increasing.

measurement and we know (by our model) that the position must have changed in the meantime and we cannot simply combine it (to produce an average). Instead, we have to consider this position change since the last measurement. This can be done by a prediction of the actual position by our model: $x(t_{n+1}) := \bar{x}_{n+1}$ based on the model parameter v , the last ‘known’ position $\hat{x}(t_n)$:

$$\bar{x}_{n+1} := v \cdot (t_{n+1} - t_n) + \hat{x}(t_n), \quad (25)$$

and a measure of the certainty of this prediction:

$$\bar{\sigma}_{n+1} := \sqrt{(\Delta v \cdot (t_{n+1} - t_n))^2 + \hat{\sigma}_n^2}. \quad (26)$$

Propagation of error

Consider a function

$$f = f(\alpha_1, \alpha_2, \dots, \alpha_n)$$

which is a function of one or more (model) parameters α_i , each with corresponding errors $\Delta\alpha_i$. Now you want to know the consequence of these errors on the overall error or uncertainty of f .

$$\Rightarrow \boxed{\Delta f = \sqrt{\sum_i \left(\frac{\partial f}{\partial \alpha_i} \Delta \alpha_i \right)^2}}.$$

Maybe you have seen this before, because this a very common formula in physics and applies everywhere where measurements are done. In our example this means:

$$x(t) = v \cdot t + x_0$$

$$\begin{aligned} \Rightarrow \Delta x &= \sqrt{\left(\frac{\partial x}{\partial v} \Delta v \right)^2 + \left(\frac{\partial x}{\partial x_0} \Delta x_0 \right)^2 + \left(\frac{\partial x}{\partial t} \Delta t \right)^2} \\ &= \sqrt{(\Delta v \cdot t)^2 + (\Delta x_0)^2}, \end{aligned}$$

(assuming $\Delta t = 0$).

This assumes that the individual errors are not correlated and are Gaussian distributed. This is likely because of the central limit theorem, but not guaranteed!

The Kalman gain

Now we use the Kalman prediction Eqs. (23) and (24) to combine the new measurement x_{n+1} with the value from the model, which is a propagation of the latest predicted value \hat{x}_n for the time $(t_{n+1} - t_n)$ using the model parameters (v) [see Eqs. (25) and (26)]. For all values, the uncertainty or measurement error is taken into account.

The output (\hat{x}_n from input $x_n := x(t_n)$) of the Kalman filter becomes:

$$\hat{x}_n = \bar{x}_n + \bar{K}_n (x_n - \bar{x}_n) ; \quad \hat{\sigma}_n = \frac{1}{\sqrt{1/\bar{\sigma}_n^2 + 1/\sigma_n^2}}$$

where

$$\bar{K}_n := \frac{\bar{\sigma}_n^2}{\bar{\sigma}_n^2 + \sigma_n^2}$$

is the redefined *Kalman gain*.

With some additional substitutions, $T := t_{n+1} - t_n$,

$\hat{x}_n \longrightarrow$	$y[n]$	Kalman filter output
$\hat{x}_{n-1} \longrightarrow$	$y[n-1]$	last output value
$x_n \longrightarrow$	$x[n]$	input value ,

one can see the general structure (difference equation) of the digital filter:

$$\begin{aligned}
 y[n] &= v_n \cdot T + y[n-1] + \bar{K}_n(x[n] - v_n \cdot T - y[n-1]) \\
 &= \bar{K}_n \cdot \underbrace{x[n]}_{\text{measurement}} + \underbrace{(1 - \bar{K}_n)(v_n \cdot T + y[n-1])}_{\text{model}}
 \end{aligned}$$

weights which represent the accuracy
of the data and the model

And this is also the way the Kalman filter could be implemented. Notice that the second term is a recursive part of the filter. The Kalman gain is the weight which decides how much model and how much input data goes to the output. If the prediction from the model is bad (the corresponding estimated variance $\bar{\sigma}$ is large), the Kalman gain tends to $\bar{K} = 1$ and so the input will be directly passed to the output without using the model at all, but also without making the output data more noisy than the input. On the contrary, if the input data occasionally has a lot of noise and the model and its model parameters are still fine, \bar{K} will be closer to zero and the output data of the Kalman filter will be dominated by the model predictions and its statistics.

13.4 The internal structure

Now let us summarize, what we learned about the Kalman filter:

- The Kalman filter makes use of an internal model and model parameters.
- The ‘internal’ system/model parameters ($\sigma, v, \Delta v$) are calculated from the input data itself.
- Also the variances of the input data stream and the variances of the derived predicted values belong to the internal parameters.
- The Kalman filter makes use of the ‘propagation of error’ principle.
- The Kalman filter has three fundamental functional blocks:
 1. The combination of model predicted with input data stream.
 2. The prediction block for the next model value.
 3. The system identification block for update of the model parameters.

The internal structure of the Kalman Filter is shown in Fig. 45.

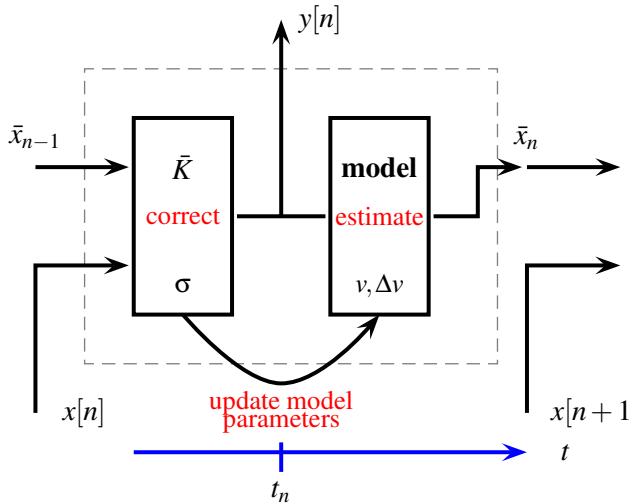


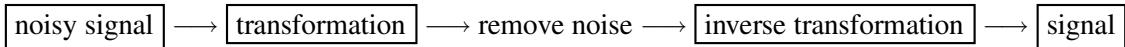
Fig. 45: Internal structure of the Kalman filter. In one block, the model prediction \bar{x}_{n-1} from the last step is combined with the actual input value $x[n]$ and passed to the output $y[n]$. The second block calculates the prediction for the next time-step based on the model parameters and their variances. In parallel, the model parameters need to be updated by a system identification algorithm.

We discussed a fairly simple example. In more realistic applications, the model can be very complex. But with more and more model parameters, more error contributions are added; this means that the optimal model complexity needs to be evaluated. The model should be as complex as necessary to reduce the noise, but also as simple as possible.

The trick is that the σ of the output will always be smaller than (or in worst case equal to) the σ of the input⁵. So the output will be best noise-filtered (depending on the model). A bad model generates a \bar{K} near to one, so the input is not corrected much (no effective filtering).

14 Wavelets

As mentioned in the previous section, wavelets can be helpful (among other things) at removing noise with a special spectral characteristics from a (non-causal) signal.



A similar method can also be used to select only especially desired spectral components with special filters; this is also often used for (lossy) data compression as for audio signals or images. Last but not least, the wavelet transformation also has applications in solving special classes of differential equations. We shall not go into these very popular fields, but instead restrict ourselves to the question of how we can make use of the wavelets for noise removal.

14.1 Fighting noise with the DFT

A quick solution will be to use the digital Fourier transformation, defined in Section 10.1, to do this job. Let us see how that works out (Fig. 46).

As you can see in this example, the reconstruction of the original signal is not too bad (in contrast with the method of the low-pass filter discussed in the previous section, there is no problem with time shift or amplitude degradation of the higher frequency components of the signal), but one major problem is immediately visible: High frequencies are there, but the phase information of the higher harmonics is distorted by cutting away some of the (noisy) components. Modifying only single spectral components has effects everywhere in time! This is the nature of the Fourier decomposition. To avoid this, one wants to use a different transformation like the wavelet transformation, because wavelets are not only localized in frequency (like $\sin()$ and $\cos()$), but also localized in time. This means that if you remove some of the wavelet components from the wavelet spectrum, the time domain signal is affected only locally. In this

⁵The principle only works if the errors of the signal measurements are independent of each other and the distribution of the noise and of all internal parameters is Gaussian. If this is not the case, the ‘propagation of error’ formula underestimates the resulting variance.

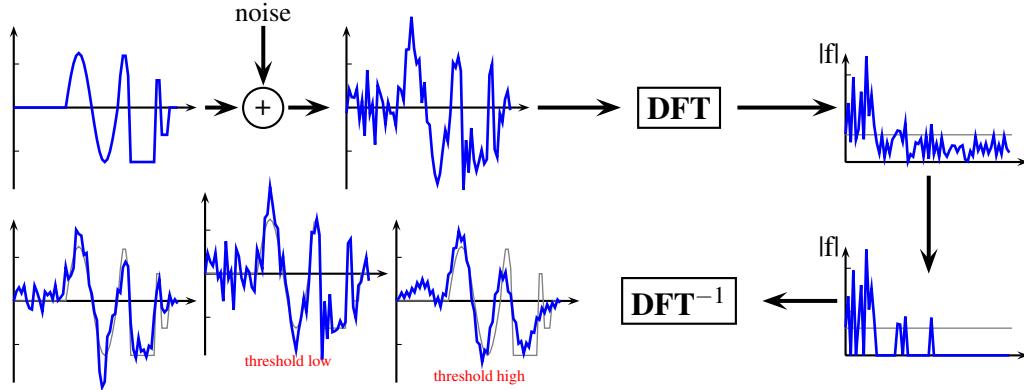


Fig. 46: Example for using the digital Fourier transformation for noise filtering. The noisy signal is transformed to frequency domain. Then all spectral components which are below a certain threshold (for amplitude) are removed (which means they are set to zero) and finally the data is transformed back into time domain. Depending on the threshold used, the result is fairly good, or still too noisy if the threshold was too low, or the reconstruction of the signal is bad, if the threshold was set too high.

way, it is possible to remove high-frequency noise where the original signal is smooth and still conserve the sharp edges of the waveform.

14.2 Localized functions

Localized functions have a body that is located around some time t_0 . This means that, for all times far away from t_0 , the function tends to zero. It especially goes asymptotically to zero for $t \rightarrow \pm\infty$. Localized in frequency simply means that for $\omega \rightarrow \infty$, the spectrum goes to zero. Sometimes it is also required that the spectrum goes to zero for $\omega \rightarrow 0$ (which is the case for wavelets). There is also a rigorous definition of localized functions, which are required to be exactly zero outside a region around the body.

There is nothing special about localized functions (see Fig. 47). In contrast to the $\sin()$ function which is well localized in frequency (only one frequency component is present) but not at all localized in time⁶, the product of a Gauss function with a $\sin()$ function is localized both in time and in frequency⁷.

Wavelets are special functions $\Psi(t) : \mathbb{R} \xrightarrow{\Psi} \mathbb{R}$ with special requirements. One which is already mentioned is that Ψ should be well localized in time and frequency. Second, it is required that

$$\int \Psi dt = 0 .$$

And finally, more technical requirements are needed, in particular, applications to make the calculation easy.

14.3 Wavelet families

There are many kinds of wavelets and **wavelet families** coming from practical applications:

- smooth wavelets,
- compactly supported wavelets (**Daubechies, 1988**),
- wavelets with simple mathematical expressions (**Haar, 1900, Meyer, Morlet**),

⁶On the contrary, any function consisting of a δ -function will probably be localized in time but definitely not in frequency.

⁷The Gauss function itself would also be localized in frequency, but it does not fulfil the more restrict requirement that the spectrum go to zero for $\omega \rightarrow 0$.

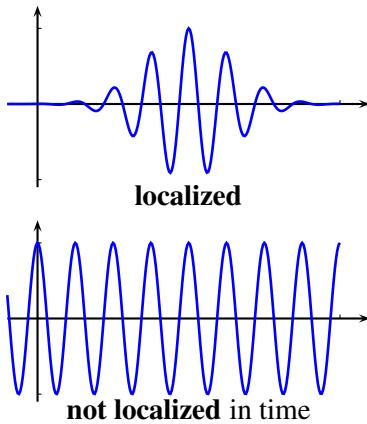


Fig. 47: Example of localized and not localized functions

- wavelets with simple associated filters,
- discrete wavelets,
- etc.

Each wavelet family is generated from a ‘**mother wavelet**’ $\Psi_{1,0}$ (which fulfils the requirements mentioned above) by a **transformation** which is a combination of translation and dilatation

$$\text{family: } \Psi_{1,0}(x) \longmapsto \Psi_{a,b} := \frac{1}{\sqrt{a}} \Psi_{1,0}\left(\frac{x-b}{a}\right) \quad ; a \in \mathbb{R}^+, b \in \mathbb{R} .$$

If you do a proper selection of a 's and b 's, you can get a wavelet family which forms a **basis** (like $\{\sin(n\omega t), \cos(n\omega t)\}$ do).

With the following set of parameters a and b :

$$a := 2^{-j} \quad ; \quad b := k \cdot 2^{-j} \quad ; \quad j, k \in \mathbb{Z} ,$$

which is called ‘critical sampling’, one gets

$$\Rightarrow \psi_{j,k}(x) := 2^{j/2} \psi(2^j x - k) , \quad (27)$$

an orthonormal Hilbert basis.

14.4 Discrete wavelet transformation

As with the discrete Fourier transformation, one can decompose an arbitrary function f to this basis formed by the wavelets (27). The ‘spectral’ components are expressed by the coefficients $c_{j,k}$.

$$f = \sum_{j,k \in \mathbb{Z}} c_{j,k} \cdot \psi_{j,k} .$$

The difference, compared with the ordinary Fourier transformation, is that the coefficients form a two-dimensional array; and you may ask what the benefit of having even more coefficients than with a Fourier transformation will be. The answer is that the wavelet decomposition can be done in a tricky way: that only (the first) very few coefficients hold most of the information of your function f and almost all other components can be neglected. Of course, this depends on the class of functions and on the selected (mother) wavelets.

The second big task—after having selected the wavelet family you want to use—is how to get the coefficients, or, more generally, how to perform the *Discrete Wavelet Transformation*

$$f(x) \xrightarrow{\text{DWT}} \{c_{j,k}\} \quad ; \quad c_{j,k} \in \mathbb{R}; j, k \in \mathbb{Z} .$$

The algorithm for the DWT is a bit tricky, but this problem has been solved and a very efficient algorithm exists. Unfortunately it is out of the scope of this lecture to explain how it works. So please consult the literature. But still one word: it makes use of iterative (digital) filter banks and the problem can best be understood in frequency domain. Also, the concept of *the scaling function* plays an important role here, limiting the number of coefficients to a *finite* number.

14.5 Wavelet applications

What are wavelets good for? Functions can be better approximated with wavelets if they have discontinuities, sharp spikes, or a non-periodic structure (they are localized). Why not use the Fourier basis?

Fourier basis	Wavelets
<ul style="list-style-type: none"> – Basis functions are localized in frequency, but not localized in time domain. – Small changes in the spectrum will produce changes of the signal everywhere in time. – Functions with discontinuities and/or sharp spikes need a big number of spectral coefficients, sometimes even an infinite number, to be properly approximated. 	<ul style="list-style-type: none"> – Basis functions are localized in frequency (scale/dilatation) and time (translation). – This can be an advantage for signal processing: Many signals can better be represented in wavelet-basis than in ‘spectral lines’, fewer coefficients → data compression.

14.6 Example: the Haar wavelet

The simplest and oldest of all (mother) wavelets is the **Haar wavelet**:

$$\Psi_{00}(x) := \begin{cases} 1 & 0 \leq x < 0.5 \\ -1 & 0.5 \leq x < 1 \\ 0 & \text{else} \end{cases}$$

With critical sampling (27), the wavelet family can be expressed as

$$\Psi_{j,k}(x) := 2^{j/2} \cdot \Psi_{00}(2^j x - k) \quad ; j, k \in \mathbb{Z} .$$

Let us see if they fulfil the wavelet criteria: Obviously, they are localized in x , $\int \psi dx = 0$, and

$$\int \Psi_{j,k} \cdot \Psi_{j',k'} = \begin{cases} 1 & j = j' \text{ and } k = k' \\ 0 & \text{else} \end{cases} ,$$

so they really form an orthonormal basis of the Hilbert space. Of course, there is also a disadvantage: The Haar wavelets are not smooth, so they may not fit best for smooth functions, but they will do their job fairly well for discrete (sampled) data.

The trick now is that—let us say you have $n = 2^m$ samples of a digitized function—you first map the data to the interval $[0; 1]$. Then write down all wavelets which have a body inside this interval, stop if the wavelets become small enough to fit a single sample, and then do the decomposition. You can do this straightforwardly with a big set of equations (one for each sample point) and solve it. As already mentioned, this is not the most efficient way to do this, but you will get the idea.

In case you really do this homework, I expect the following problem: How can a function with $\int_0^1 f(x) dx \neq 0$ be transformed? The answer is: either you try to extend the number of coefficients to

infinity (especially all the coefficients with $j < 0$) or—and this is, of course, recommended—you add (at least) one additional function to the set of wavelets which replaces the infinite number of smaller and smaller scale wavelets; namely, $\Phi_{j_0,k}; k \in \mathbb{Z}$ (j_0 is fixed here, so these functions form only a one dimensional array), the *scaling function*. The scaling function is not a wavelet, since $\int \Phi(x) dx = 1$ is required, but you can prove that the set

$$\{\Phi_{j_0,k}, \Psi_{j,k} ; j \geq j_0, k \in \mathbb{Z}\}$$

spans the same space as the full basis $\{\Psi_{j,k} ; j, k \in \mathbb{Z}\}$.

Now you might still be worried about the k within the definition, but consider our example: let us choose $j_0 = 0$. The restriction of the domain to $[0; 1[$ means that we need only consider wavelets with $0 \leq k < j$ and there is a maximal j because of our sampling resolution ($j < m$). All in all, the number of non-zero wavelet components is limited to a finite number. Finally, the missing scaling function is simply

$$\Phi_{0,k} := \begin{cases} 1 & \text{for } 0 \leq x < 1, \\ 0 & \text{else} \end{cases}$$

independent of k , so we need only one. Now all functions (with a limited number of samples) can be transformed to a finite set of wavelet coefficients. If the number of non-zero wavelet coefficients is smaller than the number of samples, you might be happy.

Unfortunately, the application of the wavelets is limited: Although the discrete wavelet transformation is well defined, and efficient algorithms have been worked out, the success of using the wavelets depends on the *choice* of the wavelet family. If you cannot find a clever wavelet family which fits well with your particular problem, you will be lost, and there is no generic way to help you out there.

Acknowledgements

I would like to thank Kirsten Hacker for proofreading the manuscript. Thanks to all those who sent me their comments and also pointed out some bugs and confusions after the presentation at the CERN school. If this started fruitful discussions, I am happy.

Bibliography

Many ideas for instructive pictures are taken from Smith's book, which is pretty much a beginner's guide to digital signal processing. Figures 12, 14, 23, 25, 28, and 29 have their origin there. There are many other books on digital signal processing, wavelets, and the Kalman filter. Here, I just list a short collection of textbooks and similar papers which inspired and taught me the latter. You have to find out by yourself if they will also be useful to you.

- S.W. Smith, *The Scientist and Engineer's Guide to Digital Signal Processing* (California Technical Pub., San Diego, CA, 1997).
- W. Kester, *Mixed-Signal and DSP Design Techniques* (Newnes, Amsterdam, 2003).
- W.H. Press, B.P. Flannery, S.A. Teukolsky and W.T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing*, 2nd ed. (Cambridge University Press, 1992).
- B.D.O. Anderson and J.B. Moore, *Optimal Filtering* (Prentice-Hall, Englewood Cliffs, NJ, 1979).
- G.F. Franklin, J.D. Powell and M.L. Workman, *Digital Control of Dynamic Systems*, 3rd ed. (Addison-Wesley, Menlo Park, CA, 1998).
- E. Kreyszig, *Advanced Engineering Mathematics*, 8th ed. (Wiley, New York, 1999).
- D. Lancaster, *Don Lancaster's Active Filter Cookbook*, 2nd ed. (Newnes, Oxford, 1996).
- P.M. Clarkson, *Optimal and Adaptive Signal Processing* (CRC Press, Boca Raton, 1993).

- G. Strang and T. Nguyen, *Wavelets and Filter Banks* (Cambridge Univ. Press, Wellesley, MA, 1997).
- B. Widrow and S.D. Stearns, *Adaptive Signal Processing* (Prentice-Hall, Englewood Cliffs, NJ, 1985).
- G. Welch and G. Bishop *An Introduction to the Kalman Filter*, University of North Carolina at Chapel Hill, Department of Computer Science, <http://www.cs.unc.edu/~{welch,gb}>.
- Wikipedia, The Free Encyclopedia. May 1, 2007, <http://en.wikipedia.org/>.

Control theory

S. Simrock

DESY, Hamburg, Germany

Abstract

In engineering and mathematics, control theory deals with the behaviour of dynamical systems. The desired output of a system is called the reference. When one or more output variables of a system need to follow a certain reference over time, a controller manipulates the inputs to a system to obtain the desired effect on the output of the system. Rapid advances in digital system technology have radically altered the control design options. It has become routinely practicable to design very complicated digital controllers and to carry out the extensive calculations required for their design. These advances in implementation and design capability can be obtained at low cost because of the widespread availability of inexpensive and powerful digital processing platforms and high-speed analog IO devices.

1 Introduction

The emphasis of this tutorial on control theory is on the design of digital controls to achieve good dynamic response and small errors while using signals that are sampled in time and quantized in amplitude. Both transform (classical control) and state-space (modern control) methods are described and applied to illustrative examples. The transform methods emphasized are the root-locus method of Evans and frequency response. The state-space methods developed are the technique of pole assignment augmented by an estimator (observer) and optimal quadratic-loss control. The optimal control problems use the steady-state constant gain solution.

Other topics covered are system identification and non-linear control. System identification is a general term to describe mathematical tools and algorithms that build dynamical models from measured data. A dynamical model in this context is a mathematical description of the dynamic behaviour of a system or process.

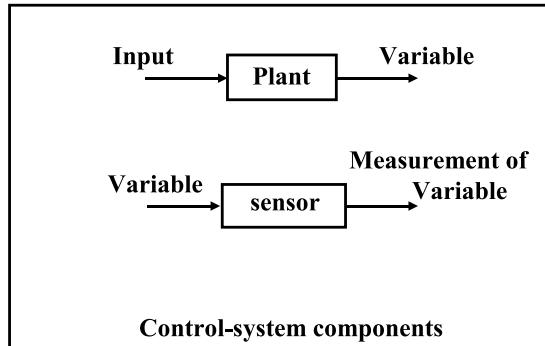
Non-linear control is a sub-division of control engineering which deals with the control of non-linear systems. The behaviour of a non-linear system cannot be described as a linear function of the state of that system or the input variables to that system. There exist several well-developed techniques for analysing non-linear feedback systems. Control design techniques for non-linear systems also exist. These can be subdivided into techniques which attempt to treat the system as a linear system in a limited range of operation and use (well-known) linear design techniques for each region.

The main components of a control system (Fig. 1) are the plant and a sensor which is used to measure the variable to be controlled. The closed-loop system (Fig. 2) requires a connection from the system outputs to the inputs. For feedback control a compensator amplifies and filters the error signal (Fig. 3) to manipulate the input signals to the plant to minimize the errors of the variables.

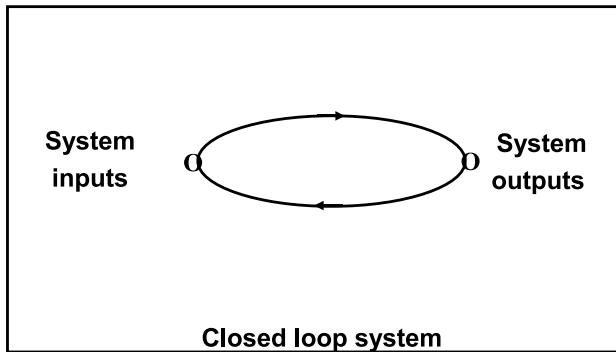
Objective: Control theory is concerned with the analysis and design of a closed-loop control system.

Analysis: Closed-loop system is given to determine characteristic or behaviour.

Design: Desired system characteristics or behaviour are specified → configure or synthesize a closed-loop system.

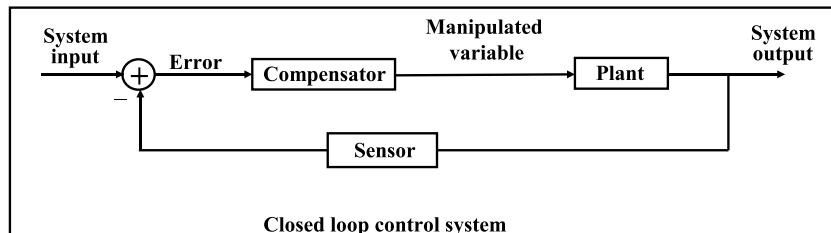
**Fig. 1:** Components of a control system

Definition: A closed-loop system is a system in which certain forces (we call these inputs) are determined, at least in part, by certain responses of the system (we call these outputs).

**Fig. 2:** Schematic of a closed-loop system

Definitions:

- The system for measurement of a variable (or signal) is called a sensor
- A **plant** of a control system is the part of the system to be controlled
- The **compensator** (or controller or simply filter) provides satisfactory characteristics for the total system.

**Fig. 3:** Arrangement of controller and plant in a closed-loop system

Two types of control system:

- A **regulator** maintains a physical variable at some constant value in the presence of perturbances.
- A **servo mechanism** describes a control system in which a physical variable is required to follow or track some desired time function (originally applied in order to control a mechanical position or motion).

2 Examples of control systems

In the following we study two examples where feedback control is applied. The first example shows how to model an RF control system. The second and third examples show electronic circuits which can be used to design the compensator (or controller) filter.

2.1 RF control system

RF control systems are used to stabilize amplitude and phase of the accelerating fields. Amplitude and phase are measured, compared with the desired setpoints, and the amplified error signals drive actuators for amplitude and phase thereby forming a negative feedback loop.

Goal:

Maintain stable gradient and phase in presence of field perturbations.

Solution: Feedback for gradient amplitude and phase.

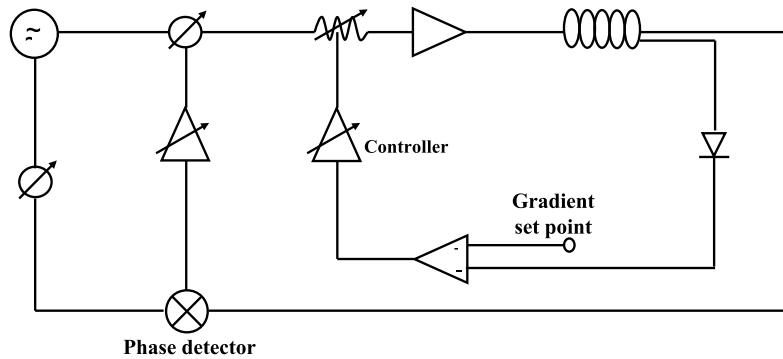


Fig. 4: RF control system with feedback control for amplitude and phase

Model: Mathematical description of input–output relation of components combined with block diagram.

Amplitude loop (general form):

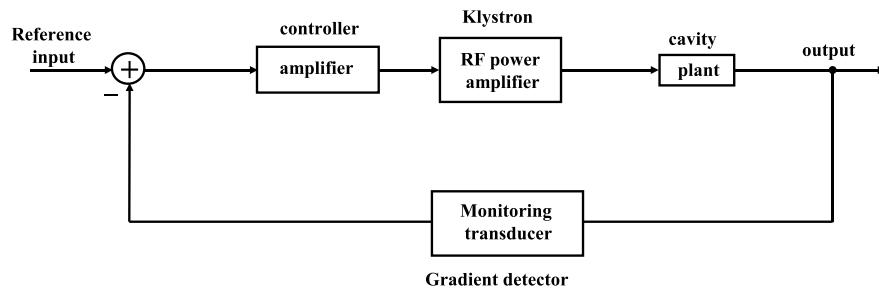


Fig. 5: General form feedback loop for amplitude control with blocks for the system components

RF control model using ‘*transfer functions*’

A transfer function of a *linear* system is defined as the ratio of the Laplace transform of the output and the Laplace transform of the input with ICs = zero.

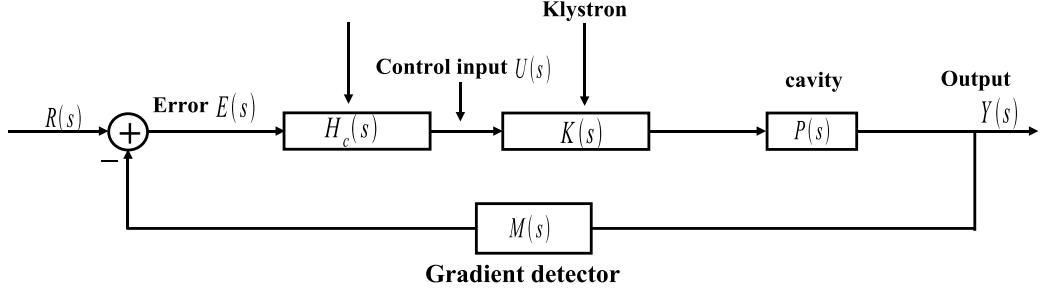


Fig. 6: Description of system components in the amplitude control loop by transfer functions

Input–output relations

Input	Output	Transfer function
$U(s)$	$Y(s)$	$G(s) = P(s)K(s)$
$E(s)$	$Y(s)$	$L(s) = G(s)H_c(s)$
$R(s)$	$Y(s)$	$T(s) = (1 + L(S)M(s))^{-1}L(s)$

2.2 Passive network circuit

For high frequencies the network shown in (Fig. 7) forms a voltage divider determined by the ratio of the resistors, while at low frequencies the input voltage is unchanged at the output.

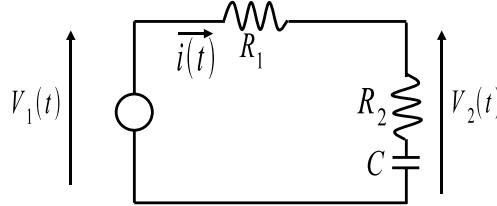


Fig. 7: Passive network for a frequency-dependent voltage divider

Differential equations:

$$R_1 i(t) + R_2 i(t) + \frac{1}{C} \int_0^t i(\tau) d\tau = v_1(t)$$

$$R_2 i(t) + \frac{1}{C} \int_0^t i(\tau) d\tau = v_2(t)$$

Laplace transform:

$$R_1 I(s) + R_2 I(s) + \frac{1}{sC} I(s) = V_1(s)$$

$$R_2 I(s) + \frac{1}{sC} I(s) = V_2(s)$$

Laplace transform:

$$G(s) = \frac{V_2(s)}{V_1(s)} = \frac{R_2 C s + 1}{(R_1 + R_2) C s + 1}$$

Input V_1 , Output V_2 .

2.3 Operational amplifier circuit

The transfer function of the circuit shown in (Fig. 8) represents an amplifier with very high gain at low frequencies and a gain determined by the ratio of the resistors at high frequencies. The more general case is shown in (Fig. 9) and allows the realization of a wide range of transfer functions.

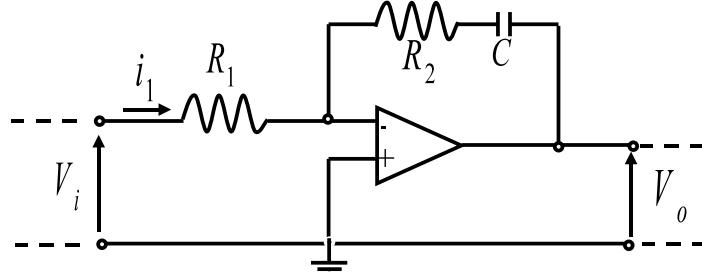


Fig. 8: Active circuit with operational amplifier

$$V_i(s) = R_1 I_1(s) \text{ and } V_o(s) = -(R_2 + \frac{1}{sC}) I_1(s)$$

$$G(s) = \frac{V_o(s)}{V_i(s)} = -\frac{R_2 Cs + 1}{R_1 Cs}$$

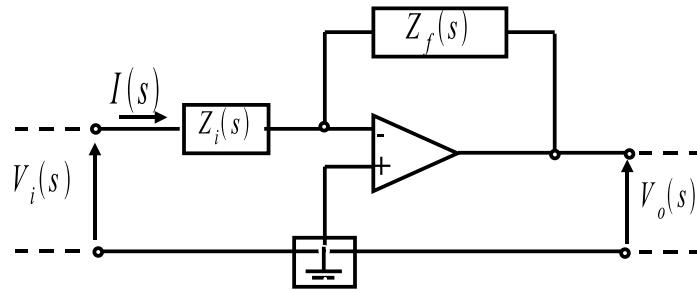


Fig. 9: Generalized version of active circuit with operational amplifier using impedances in the input and feedback section

It is convenient to derive a transfer function for a circuit with a single operational amplifier that contains input and feedback impedance:

$$V_i(s) = Z_i(s) I(s) \text{ and } V_o(s) = -Z_f(s) I(s) \rightarrow G(s) = \frac{V_o(s)}{V_i(s)} = -\frac{Z_f(s)}{Z_i(s)}$$

3 Model of dynamic systems

Models of dynamic systems can be described by differential equations. A system of order n can be reduced to a set of n first-order equations. These can be written in a matrix formalism called state-space representation.

We shall study the following dynamic system:

Parameters: k : spring constant, γ : damping constant, $u(t)$: force

Quantity of interest: $y(t)$: displacement from equilibrium

Differential equation: Newton's third law ($m = 1$)

$$\ddot{y}(t) = \sum F_{ext} = -ky(t) - \gamma\dot{y}(t) + u(t)$$

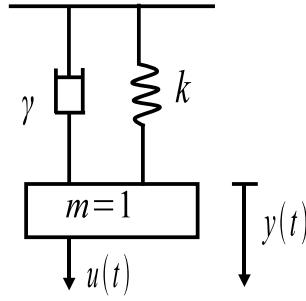


Fig. 10: Harmonic oscillator with mass m , spring constant k , and damping term γ

$$\ddot{y} + \gamma\dot{y}(t) + ky(t) = u(t)$$

$$y(0) = y_0, \dot{y}(0) = \dot{y}_0$$

1. Equation is linear (i.e., no y^2 like terms).
2. Ordinary (as opposed to partial e.g., $\frac{\partial}{\partial x} \frac{\partial}{\partial t} f(x, t) = 0$).
3. All coefficients constant: $k(t) = k, \gamma(t) = \gamma$ for all t .

Stop calculating, let us paint. Picture to visualize differential equation.

1. Express highest-order term (put it to one side)

$$\ddot{y}(t) = -ky(t) - \gamma\dot{y}(t) + u(t)$$

2. Put adder in front.

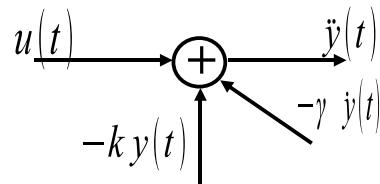


Fig. 11: Visualization of the differential equation for the harmonics oscillator

3. Synthesize all other terms using integrators.

3.1 Linear Ordinary Differential Equation (LODE)

General form of LODE:

$$y^{(n)}(t) + a_{n-1}y^{(n-1)}(t) + \dots + a_1\dot{y}(t) + a_0y(t) = b_m u^{(m)}(t) + \dots + b_1\dot{u}(t) + b_0u(t)$$

m, n positive integers, $m \leq n$; coefficients $a_0, a_1, \dots, a_{n-1}, b_0, \dots, b_m$ real numbers.

Mathematical solution: I hope you know it

Solution of LODE: $y(t) = y_h(t) + y_p(t)$.

The solution is the sum of the homogeneous solution $y_h(t)$ (natural response) solving

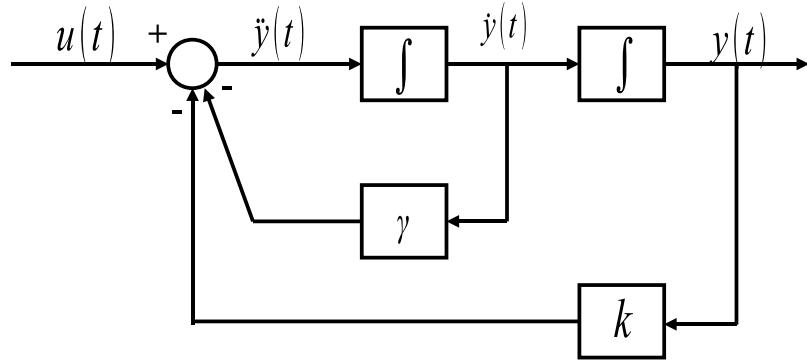


Fig. 12: Visualization of the second-order equation for the simple harmonic oscillator. The diagram can be used to program an analog computer.

$$y^{(n)}(t) + a_{n-1}y^{(n-1)}(t) + \dots + a_1\dot{y}(t) + a_0y(t) = 0$$

and the particular solution $y_p(t)$.

How to get natural response $y_h(t)$? Characteristic polynomial

$$x(\lambda) = \lambda_n + a_{n-1}\lambda_{n-1} + a_1\lambda + a_0 = 0$$

$$(\lambda - \lambda_1)^r \cdot (\lambda - \lambda_{r+1}) \dots (\lambda - \lambda_n) = 0$$

$$y_h(t) = (c_1 + c_2 t + \dots + c_r t^{r-1}) e^{\lambda_1 t} + c_{r+1} e^{\lambda_{r+1} t} \dots + c_n e^{\lambda_n t}$$

The determination of $y_p(t)$ is relatively simple, if the input $u(t)$ yields only a finite number of independent derivates, e.g., $u(t) \cong e^{xit}, \beta_r t^r$.

Most important for control system/feedback design:

$$y_{(n-1)}(t) + a_{n-1}y^{(n-1)}(t) + \dots + a_1\dot{y}(t) + a_0y(t) = b_m u^{(m)}(t) + \dots + b_1 u(t) + b_0 u(t).$$

In general: Any linear time-invariant system described by a LODE can be realized, simulated, and easily visualized in a block diagram ($n = 2, m = 2$)

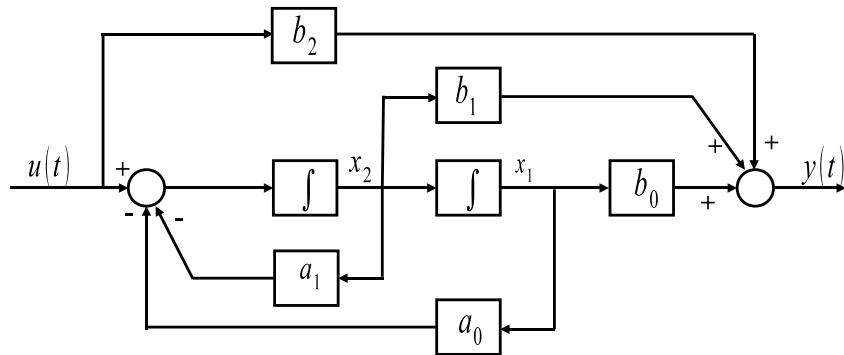


Fig. 13: Control-canonical form of a second-order differential equation

It is very useful to visualize *interaction* between the variables.

What are x_1 and x_2 ?

More explanations later, for now: Please simply accept it.

3.2 State-space equation

Any system which can be represented by a LODE can be represented in *state space form* (matrix differential equation).

Let us go back to our first example (Newton's law):

$$\ddot{y}(t) + \gamma\dot{y}(t) + ky(t) = u(t)$$

Step 1: Deduce set of first-order differential equation in variables

$x_j(t)$ (so-called states of system)

$x_1(t) \cong \text{Position} : y(t) :$

$x_2(t) \cong \text{Velocity} : \dot{y}(t) :$

$$\dot{x}_1(t) = \dot{y}(t) = x_2(t)$$

$$\dot{x}_2(t) = \ddot{y}(t) = -ky(t) - \gamma\dot{y}(t) + u(t) = -kx_1(t) - \gamma x_2(t) + u(t)$$

One LODE of order n transformed into n LODEs of order 1

Step 2: Put everything together in a matrix differential equation:

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ -k & -\gamma \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix} u(t)$$

$\dot{x}(t) = Ax(t) + Bu(t)$ State equation

$$y(t) = [1 \ 0] \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}$$

$y(t) = Cx(t) + Du(t)$ Measurement equation

Definition: The *system state* x of a system at any time t_0 is the ‘amount of information’ that, together with all inputs for $t \geq t_0$, uniquely determines the behaviour of the system for all $t \geq t_0$.

The linear time-invariant (LTI) analog system is described by the standard form of the state-space equation $\dot{x}(t) = Ax(t) + Bu(t)$ state equation

$y(t) = Cx(t) + Du(t)$ state equation,

where $\dot{x}(t)$ is the time derivative of the vector $x(t) = \begin{bmatrix} x_1(t) \\ \dots \\ x_n(t) \end{bmatrix}$.

The system is completely described by the state-space matrices A, B, C, D (in most cases $D = 0$).

Declaration of variables

Variable	Dimension	Name
$X(t)$	$nx1$	State vector
A	nXn	System matrix
B	nXr	Input matrix
$u(t)$	$rX1$	Input vector
$y(t)$	$pX1$	Output vector
C	pXn	Output matrix
D	pXr	Matrix representing direct coupling between input and output

Why all this work with state-space equation? Why bother with it?

Because, given any system of the LODE form

$$y^{(n)}(t) + a_{n-1}y^{(n-1)}(t) + \dots + a_1\dot{y}(T) + a_0y(t) = b_m u^{(m)}(t) + \dots + b_1\dot{u}(t) + b_0u(t)$$

the system can be represented as

$$\dot{x}(t) = Ax(t) + Bu(t)$$

$$y(t) = Cx(t) + Du(t)$$

With, for example, **control-canonical form** (case $n = 3, m = 3$):

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -a_0 & -a_1 & -a_2 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad C = \begin{bmatrix} b_0 & b_1 & b_2 \end{bmatrix}, \quad D = b_3$$

or **observer-canonical form**:

$$A = \begin{bmatrix} 0 & 0 & -a_0 \\ 1 & 0 & -a_1 \\ 0 & 1 & -a_2 \end{bmatrix}, \quad B = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}, \quad C = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}, \quad D = b_3$$

1. Notation is very compact, but not unique.
2. Computers love the state-space equation. (Trust us!)
3. Modern control (1960–now) uses the state-space equation
4. General (vector) block diagram for easy visualization

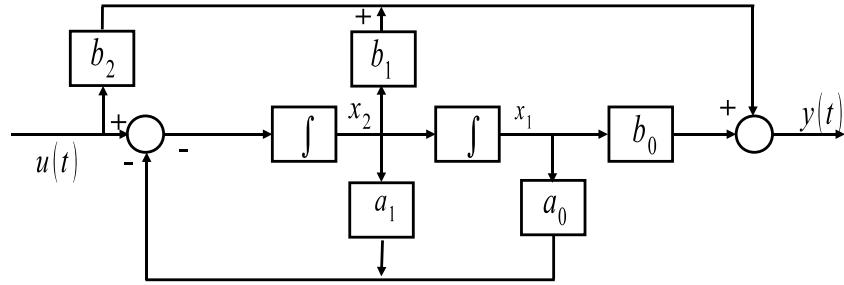
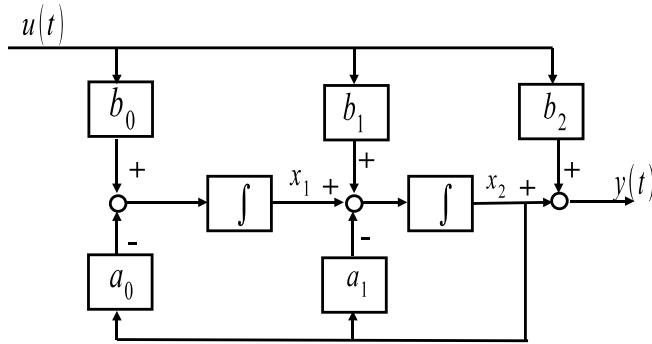
3.2.1 Block diagrams:

Now: solution of the state-space equation in the time domain:

$$x(t) = \phi(t)x(0) + \int_0^t \phi(\tau)Bu(t-\tau)d\tau.$$

Natural response + particular solution

$$y(t) = Cx(t) + Du(t)$$

**Fig. 14:** Control-canonical form of a differential equation of second order**Fig. 15:** Observer-canonical form of a differential equation of second order

$$= C\phi(t)x(0) + C \int_0^t \phi(\tau)Bu(t-\tau)d\tau + Du(t).$$

With the **state transition matrix**

$$\phi(t) = I + At + \frac{A^2}{2!}t^2 + \frac{A^3}{3!}t^3 + \dots = e^{At},$$

which is an exponential series in the matrix A (time evaluation operator).

Properties of $\phi(t)$ (state transition matrix):

1. $\frac{d\phi(t)}{dt} = A\phi(t)$
2. $\phi(0) = I$
3. $\phi(t_1 + t_2) = \phi(t_1)\phi(t_2)$
4. $\phi^{-1} = \phi(-t)$

Example:

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \implies A^2 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \phi(t) = I + At = \begin{bmatrix} 1 & t \\ 0 & 1 \end{bmatrix} = e^{At}$$

Matrix A is a nilpotent matrix.

3.2.2 Example

We have the following differential equation:

$$\frac{d^2}{dt^2}y(t) + 4\frac{d}{dt}y(t) + 3u(t) = 2u(t).$$

1. State equations of the differential equation:

Let $x_1(t) = y(t)$ and $x_2(t) = \dot{y}(t)$ then

$$\dot{x}_1(t) = \dot{y}(t) = x_2(t)$$

$$\dot{x}_2(t) + 4x_2(t) + 3x_1(t) = 2u(t)$$

$$\dot{x}_2 = -3x_1(t) - 4x_2(t) + 2u(t)$$

2. Write the state equations in matrix form:

Define system state $x(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}$ $\dot{x}(t) = \begin{bmatrix} 0 & 1 \\ -3 & -4 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 2 \end{bmatrix} u(t)$
 $y(t) = \begin{bmatrix} 1 & 0 \end{bmatrix} x(t)$.

3.2.3 Cavity model

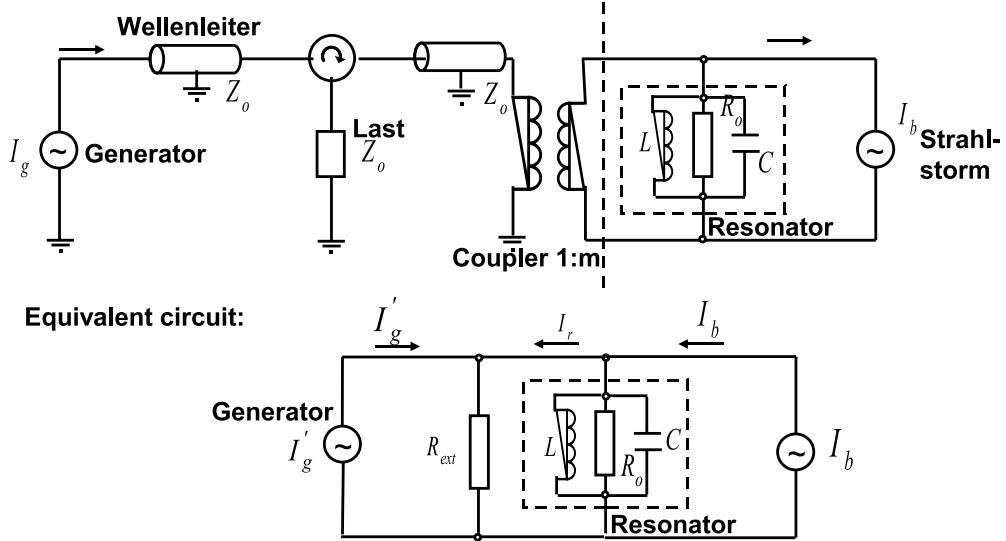


Fig. 16:

$$C\ddot{U} + \frac{1}{R_L}\dot{U} + \frac{1}{L}U = \dot{I}'_g + \dot{I}_b$$

$$\omega_{\frac{1}{2}} := \frac{1}{2R_L C} = \frac{\omega_0}{2Q_L}$$

$$\ddot{U} + 2\omega_{\frac{1}{2}}\dot{U} + \omega_0^2 U = 2R_L \omega_{\frac{1}{2}} \cdot (\frac{2}{m} \dot{I}'_g + \dot{I}_b)$$

Only the envelope of RF (real and imaginary part) is of interest:

$$U(t) = (U_r(t) + iU_i(t)) \exp(i\omega_{HF}(t))$$

$$I_g(t) = (I_g r(t) + iI_{gi}(t)) \exp(i\omega_{HF}(t))$$

$$I_b(t) = (I_{b\omega r}(t) + iI_{b\omega i}(t)) \exp(i\omega_{HF} t) = 2(I_{bor}(t) + iI_{boi}(t)) \exp(i\omega_{HF} t)$$

Neglect small terms in derivatives for U and I .

$$\ddot{U}_r + i\ddot{U}_i(t) \ll \omega_{HF}^2(U_r(t) + iU_i(t))$$

$$2\omega_{\frac{1}{2}}(\dot{U}_r + i\dot{U}_i(t)) \ll \omega_{HF}^2(U_r(t) + iU_i(t))$$

$$\int_{t_1}^{t_2} (\dot{I}_r(t) + i\dot{I}_i(t))dt \ll \int_{t_1}^{t_2} \omega_{HF}(I_r(t) + iI_i(t))dt$$

Envelope equations for real and imaginary component.

$$\dot{U}_r(t) + \omega_{\frac{1}{2}}U_r + \Delta\omega U_i = \omega_{HF}\frac{r}{Q} \cdot \left(\frac{1}{m}I_{gr} + I_{bor} \right)$$

$$\dot{U}_i(t) + \omega_{\frac{1}{2}}U_i - \Delta\omega U_r = \omega_{HF}\left(\frac{r}{Q}\right) \cdot \left(\frac{1}{m}I_{gi} + I_{boi} \right)$$

Matrix equations:

$$\begin{bmatrix} \dot{U}_r(t) \\ \dot{U}_i(t) \end{bmatrix} = \begin{bmatrix} -\omega_{\frac{1}{2}} & -\Delta\omega \\ \Delta\omega & -\omega_{\frac{1}{2}} \end{bmatrix} \cdot \begin{bmatrix} U_r(t) \\ U_i(t) \end{bmatrix} + \omega_{HF}\left(\frac{r}{Q}\right) \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \frac{1}{m}I_{gr}(t) + I_{bor}(t) \\ \frac{1}{m}I_{gi}(t) + I_{boi}(t) \end{bmatrix}$$

With system Matrices: $A = \begin{bmatrix} -\omega_{\frac{1}{2}} & -\Delta\omega \\ \Delta\omega & -\omega_{\frac{1}{2}} \end{bmatrix}$ $B = \omega_{HF}\left(\frac{r}{Q}\right) \cdot \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

$$\vec{x}(t) = \begin{bmatrix} U_r(t) \\ U_i(t) \end{bmatrix} \quad \vec{u}(t) = \begin{bmatrix} \frac{1}{m}I_{gr}(t) + I_{bor}(t) \\ \frac{1}{m}I_{gi}(t) + I_{boi}(t) \end{bmatrix}$$

General form: $\dot{\vec{x}} = A.\vec{x}(t) + B.\vec{u}(t)$.

Solution:

$$\vec{x}(t) = \phi(t).\vec{x}(0) + \int_0^t \phi(t-t').B.\vec{u}(t')dt'$$

$$\phi(t) = e^{-\omega_{\frac{1}{2}}t} \begin{bmatrix} \cos(\Delta\omega t) & -\sin(\Delta\omega t) \\ \sin(\Delta\omega t) & \cos(\Delta\omega t) \end{bmatrix}$$

Special case:

$$\vec{u}(t) = \begin{bmatrix} \frac{1}{m}I_{gr}(t) + I_{bor}(t) \\ \frac{1}{m}I_{gi}(t) + I_{boi}(t) \end{bmatrix} =: \begin{bmatrix} I_r \\ I_i \end{bmatrix}$$

$$\begin{bmatrix} U_r(t) \\ U_i(t) \end{bmatrix} = \frac{\omega_{HF}\left(\frac{r}{Q}\right)}{\omega_{\frac{1}{2}}^2 + \Delta\omega^2} \cdot \begin{bmatrix} \omega_{\frac{1}{2}} & -\Delta\omega \\ \Delta\omega & \omega_{\frac{1}{2}} \end{bmatrix} \cdot \left\{ 1 - \begin{bmatrix} \cos(\Delta\omega t) & -\sin(\Delta\omega t) \\ \sin(\Delta\omega t) & \cos(\Delta\omega t) \end{bmatrix} e^{-\omega_{\frac{1}{2}}t} \right\} \cdot \begin{bmatrix} I_r \\ I_i \end{bmatrix}$$

3.3 Mason's rule

Mason's rule is a simple formula for reducing block diagrams. It works for continuous and discrete systems. In its most general form it is messy, except for the special case when all paths touch:

$$H(s) = \frac{\Sigma(\text{forward path gains})}{1 - \Sigma(\text{loop path gains})}$$

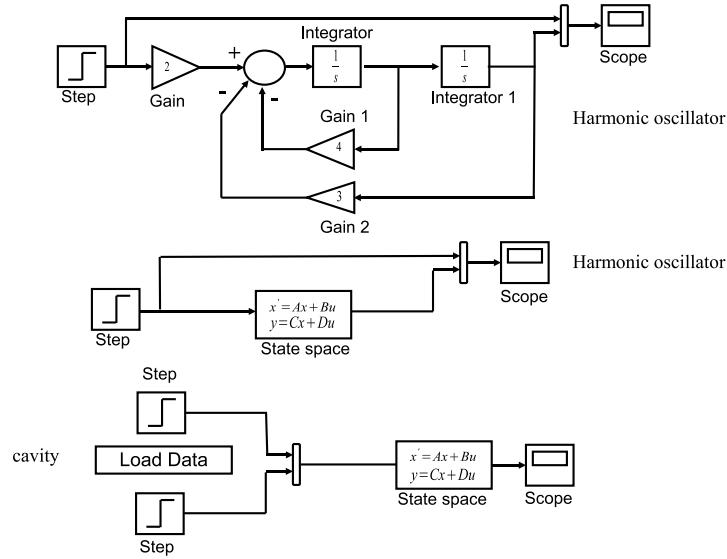


Fig. 17: Simulink diagram of the harmonic oscillator as analog computer and in state-space form. The state-space form of the cavity model follows the same structure.

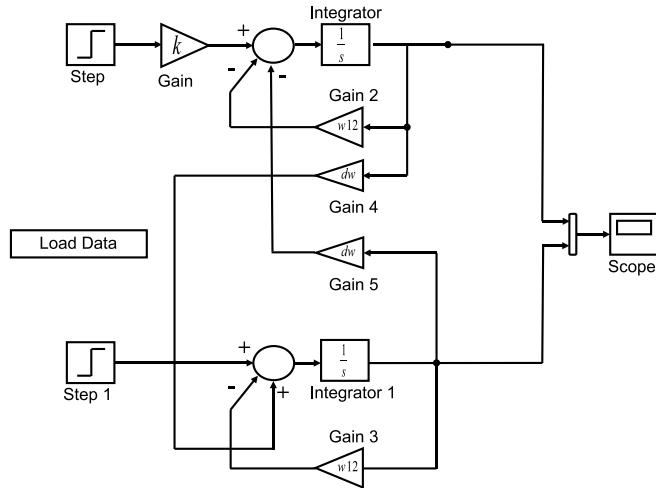


Fig. 18: Simulink diagram of the cavity model with two inputs and output for real and imaginary parts of the drive and field vectors

Two paths are said to *touch* if they have a component in common, e.g., an adder.

Forward path: $F_1:1 - 10 - 11 - 5 - 6$ and $F_2:1 - 2 - 3 - 4 - 5 - 6$

Loop path: $I_1:3 - 4 - 5 - 8 - 9$ and $I_2:5 - 6 - 7$

$$G(f_1) = H_5 H_3, G(f_2) = H_1 H_2 H_3, G(I_1) = H_2 H_4, G(I_2) = H_3$$

Check: All paths touch (contain adder between 4 and 5)

$$\implies \text{By Mason's rule: } H = \frac{G(f_1)+G(f_2)}{1-G(l_1)-G(l_2)} = \frac{H_5 H_3 + H_1 H_2 H_3}{1-H_2 H_4 - H_3} = \frac{H_3(H_5 + H_1 H_2)}{1-H_2 H_4 - H_3}.$$

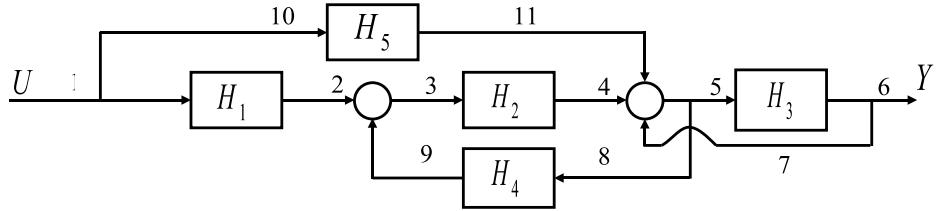


Fig. 19: Example for Mason's rule. All connections in the forward and loop paths are shown.

4 Transfer function $G(s)$

The transfer function is commonly used in the analysis of single-input single-output electronic filters, for instance. It is mainly used in signal processing, communication theory, and control theory. The term is often used exclusively to refer to linear, time-invariant systems (LTI), as covered in this article. Most real systems have non-linear input/output characteristics, but many systems, when operated within nominal parameters (not ‘over-driven’) have behaviour that is close enough to linear that LTI system theory is an acceptable representation of the input/output behaviour.

Continuous-time space model:

$$\dot{x}(t) = Ax(t) + Bu(t) \text{ State equation}$$

$$y(t) = Cx(t) + Du(t) \text{ Measurement equation}$$

The transfer function describes the input–output relation of the system.



Fig. 20: System with the Laplace transforms of input and output signals. The ratio is the transfer function.

$$sX(s) - x(0) = AX(s) + BU(s)$$

$$X(s) = (sI - A)^{-1}x(0) + (sI - A)^{-1}BU(s)$$

$$= \phi(s)x(0) + \phi(s)BU(s)$$

$$Y(s) = CX(s) + DU(s)$$

$$= C[(sI - A)^{-1}]x(0) + [c(sI - A)^{-1}B + D]U(s)$$

$$= C\phi(s)x(0) + C\phi(s)BU(s) + DU(s)$$

Transfer function $G(s)$ (pxr) (case: $X(0) = 0$):

$$G(s) = C(sI - A)^{-1}B + D = C\phi(s)B + D$$

Transfer function of TESLA cavity including $\frac{8}{9} - \pi$ mode

$$H_{cont}(s) \approx H_{cav}(s) = H_{\Pi}(s) + H_{\frac{8\pi}{9}}(s)$$

$$\begin{aligned} \text{π- mode } H_{\Pi}(s) &= \frac{(\omega_{\frac{1}{2}})\pi}{\Delta\omega_{\pi}^2 + (s + (\omega_{\frac{1}{2}})\pi)^2} \begin{bmatrix} s + (\omega_{\frac{1}{2}})\pi & -\Delta\omega_{\pi} \\ -\Delta\omega_{\pi} & s + (\omega_{\frac{1}{2}})\pi \end{bmatrix} \\ \text{$\frac{8}{9}\pi$- mode } H_{\frac{8\pi}{9}}(s) &= -\frac{(\omega_{\frac{1}{2}})\frac{8\pi}{9}}{\Delta\omega_{\frac{8\pi}{9}}^2 + (s + (\omega_{\frac{1}{2}})\frac{8\pi}{9})^2} \begin{bmatrix} s + (\omega_{\frac{1}{2}})\frac{8\pi}{9} & -\Delta\omega_{\frac{8\pi}{9}} \\ \Delta\omega_{\frac{8\pi}{9}} & s + (\omega_{\frac{1}{2}})\frac{8\pi}{9} \end{bmatrix} \end{aligned}$$

4.1 Transfer function of a closed-loop system

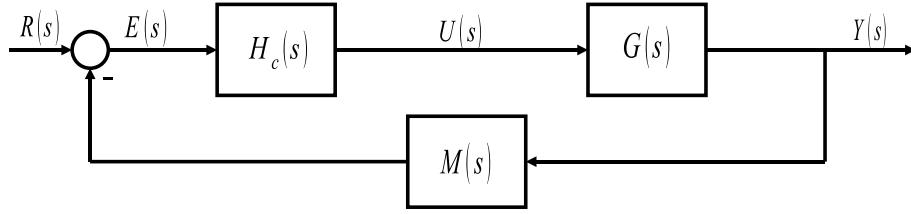


Fig. 21: Diagram of a closed-loop system. Mason's rule can be applied to determine the closed-loop transfer function.

We can deduce for the output of the system:

$$Y(s) = G(s)U(s) = G(s)H_c(s)E(s)$$

$$= G(s)H_c(s)[R(s) - M(s)Y(s)]$$

$$= L(s)R(s) - L(s)M(s)Y(s)$$

With $L(s)$ the transfer function of the open-loop system (controller and plant)

$$(1 + L(s)M(s))Y(s) = L(s)R(s)$$

$$Y(s) = (I + L(s)M(s))^{-1}L(s)R(s) = T(S)R(s),$$

where $T(s)$ is called the reference transfer function.

4.2 Sensitivity

The ratio of change of the transfer function $T(s)$ by the parameter b can be defined as

$$\text{System characteristics change with system parameter variations } S = \frac{\Delta T(s)}{T(s)} \cdot \frac{b}{\Delta b}.$$

$$\text{The sensitivity function is defined as: } S_b^T = \lim_{\Delta b \rightarrow 0} \frac{\Delta T(s)}{\Delta b} \cdot \frac{b}{T(s)} = \frac{T(s)}{b} \cdot \frac{b}{T(s)}.$$

Or, in general, sensitivity function of a characteristic W with respect to the parameter b :

$$S_b^W = \frac{W}{b} \cdot \frac{b}{W}.$$

Example: Plant with proportional feedback given by $G_c(s) = K_p$ and $G_p(s) = \frac{K}{s+0.1}$

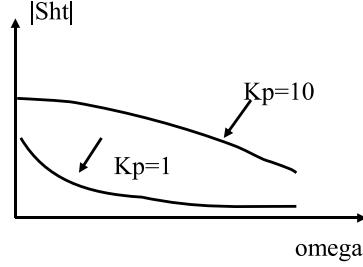


Fig. 22: Sensitivity as function of frequency for different feedback gains

Plant transfer function $T(s)$: $T(s) \frac{K_p G_p(s)}{1 + K_p G_p(s) H_k}$

$$T_H^T(j\omega) = \frac{-K_p G_p(j\omega) H_k}{1 + K_p G_p(j\omega) H_k} = \frac{-0.25 K_p}{0.1 + 0.25 K_p + j\omega}$$

Increase of H results in decrease of T . System cannot be insensitive to both H, T .

4.3 Disturbance rejection

Disturbances are system influences we do not control and whose impact on the system we want to minimize.

$$\begin{aligned} C(s) &= \frac{G_c(s) \cdot G_p(s)}{1 + G_c(s) \cdot G_p(s) \cdot H(s)} \cdot R(s) + \frac{G_d(s)}{1 + G_c(s) \cdot G_p(s) \cdot H(s)} \\ &= T(s) \cdot R(s) + T_d(s) \cdot D(s). \end{aligned}$$

To reject disturbances, make $T_d(s) \cdot D(s)$ small!

1. Use frequency response approach to investigate disturbance rejection.
2. In general $T_d(j\omega)$ cannot be small for all $-\omega$. Design $T_d(j\omega)$ small for significant portion of system bandwidth.
3. Reduce the gain $G_d(j\omega)$ between disturbance input and output.
4. Increase the loop gain $G_c G_p(j\omega)$ without increasing the gain $G_d(j\omega)$. Usually accomplished by the choice of $G_c(j\omega)$ compensator.

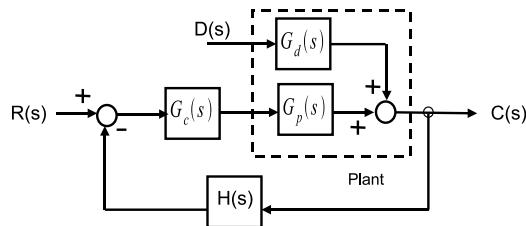


Fig. 23: Diagram with all transfer functions and signals used to define the disturbance rejection

5. Reduce the disturbance magnitude $d(t)$: should always be attempted if reasonable.
6. Use feed forward compensation, if disturbance can be measured.

5 Stability

In electrical engineering, specifically signal processing and control theory, BIBO stability is a form of stability for signals and systems. BIBO stands for Bounded-Input Bounded-Output. If a system is BIBO stable then the output will be bounded for every input to the system that is bounded.

Now we have learnt so far that

the impulse response tells us everything about the system response to arbitrary input signal $u(t)$.

What we have not learnt:

If we know the transfer function $G(s)$, how can we deduce the system behaviour?

What can we say about the system stability?

Definition: A linear time invariant system is **BIBO** stable (Bounded-Input-Bounded-Output) for all bounded inputs $|u(t)| \leq M_1$ (for all t) exists a boundary for the output signal M_2 , so that $|y(t)| \leq M_2$ (for all t) with M_1 and M_2 are positive real numbers.

If input never exceeds M_1 and output never exceeds M_2 then we have BIBO stability!

Note: It has to be valid for ALL bounded input signals!

Example: $Y(s) = G(s)U(s)$, integrator $G(s) = \frac{1}{s}$

1. Case $u(t) = \delta(t), U(s) = 1$

$$|y(t)| = |L^{-1}[Y(s)]| = |L^{-1}[\frac{1}{s}]| = 1$$

2. Case $u(t) = 1, U(s) = \frac{1}{s}$

$$|y(t)| = |L^{-1}[Y(s)]| = |L^{-1}[\frac{1}{s^2}]| = t$$

BIBO stability has to be proven for any input. It is not sufficient to show its validity for a single input signal!

Condition for BIBO stability:

We start from the input–output relation

$$Y(s) = G(s)U(s).$$

By means of the convolution theorem we get

$$|y(t)| = \left| \int_0^t g(\tau)u(t-\tau)d\tau \right| \leq \int_0^t |g(\tau)||u(t-\tau)|d\tau \leq M_1 \int_0^\infty |g(\tau)|d\tau \leq M_2.$$

Therefore it follows immediately that if the impulse response is absolutely integrable

$$\int_0^\infty |g(t)|dt < \infty,$$

then the system is BIBO stable.

5.1 Poles and zeros

Can stability be determined if we know the TF of a system?

$$G(s) = C\phi(s)B + D = C \frac{[sI - A]_{adj}}{\chi(s)} B + D$$

Coefficients of transfer function $G(s)$ are rational functions in the complex variable s

$$g_{ij}(s) = a \cdot \frac{\prod_{k=1}^m (s - z_k)}{\prod_{l=1}^n (s - p_l)} = \frac{N_{ij}(s)}{D_{ij}(s)}$$

z_k zeros, p_l poles, α real constant, and it is $m \leq n$ (we assume common factors have already been cancelled!).

What do we know about the zeros and the poles?

Since numerator $N(s)$ and denominator $D(s)$ are polynomials with real coefficients, poles and zeros must be real numbers or must arise as complex conjugated pairs!

5.2 Stability directly from state-space

Recall that: $H(s) = C(sI - A)^{-1}B + D$.

Assuming $D = 0$ (D could change zeros but not poles),

$$H(s) = \frac{C_{adj}(sI - A)B}{\det(sI - A)} = \frac{b(s)}{a(s)}.$$

Assuming there are no common factors between the poly $C_{adj}(sI - A)B$ and $\det(sI - A)$ i.e., no pole-zero cancellations (usually true, system called ‘minimal’) then we can identify and

$$b(s) = C_{adj}(sI - A)B$$

$$a(s) = \det(sI - A)$$

i.e., poles are root of $\det(sI - A)$.

Let λ_i be the i -th eigen value of A if $\mathbf{Re}(\lambda_i) \leq 0$ for all $i \implies$ system stable.

So with a computer, with eigenvalue solver, one can determine system stability directly from coupling Matrix A .

A system is BIBO stable if, for every bounded input, the output remains bounded with increasing time.

For a LTI system, this definition requires that all poles of the closed-loop transfer-function (all roots of the system characteristic equation) lie in the left half of the complex plane.

5.3 Several methods are available for stability analysis:

1. Routh Hurwitz criterion
2. Calculation of exact locations of roots

- root locus technique
- Nyquist criterion
- Bode plot

3. Simulation (only general procedures for nonlinear systems)

While the first criterion proofs whether a feedback system is stable or unstable, the second method also provides information about the setting time (damping term). Pole locations tell us about impulse response i.e., also stability:

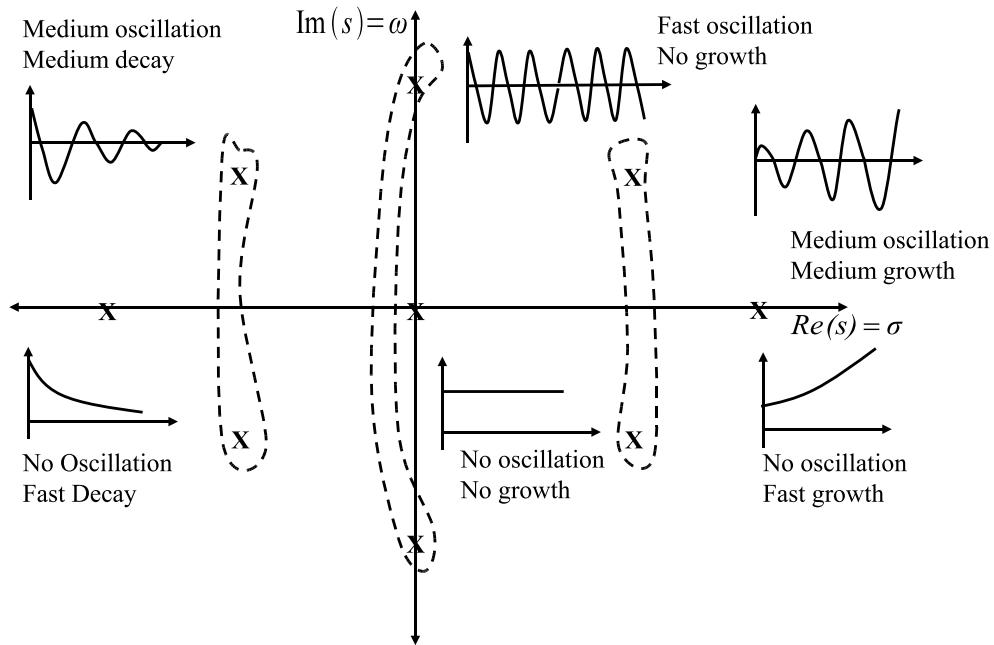


Fig. 24: Stability diagram in the complex plane. The system is stable if the poles are located in the left half of the plane. Oscillations occur for conjugate complex poles.

Furthermore: Keep in mind the following picture and facts!

1. Complex pole pair: Oscillation with growth to decay
2. Real pole: exponential growth or decay
3. Poles are the eigenvalues of the matrix A .
4. Position of zeros goes into the size of c_j

In general a complex root must have a corresponding conjugate root ($N(s), D(s)$) polynomials with real coefficients.

Bode Diagram

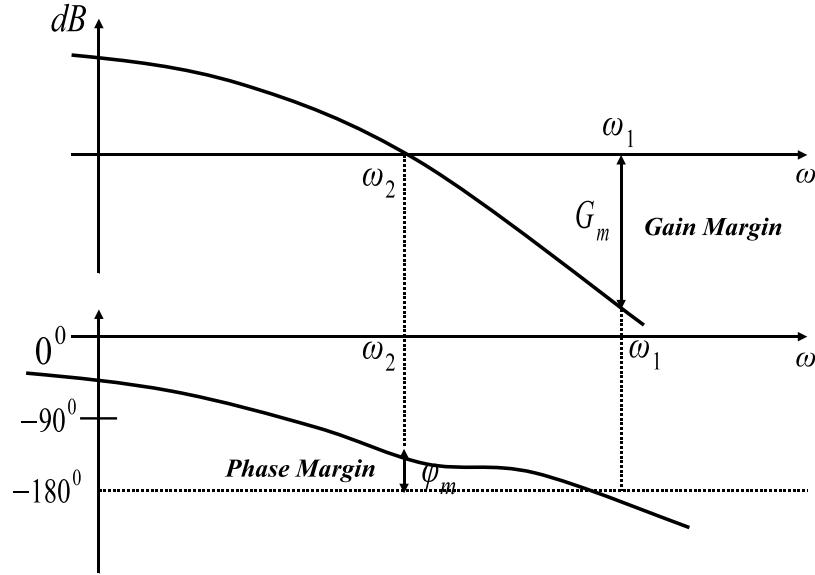


Fig. 25: Bode diagram showing phase and gain margins for stable systems

The closed loop is stable if the phase of the unity crossover frequency of the *open loop* is larger than -180 degrees.

5.4 Root locus analysis

Definition: A root locus of a system is a plot of the roots of the system characteristic equation (the poles of the closed-loop transfer function) while some parameter of the system (usually the feedback gain) is varied. $KH(s) = \frac{K}{(s-p_1)(s-p_2)(s-p_3)}$.

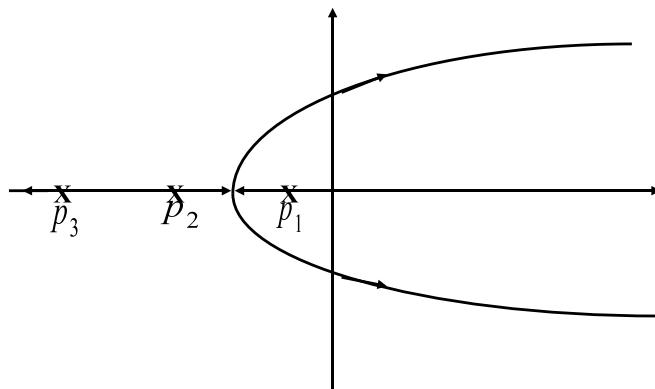


Fig. 26: Concept of root locus method



Fig. 27: Transfer function model of the closed-loop system used for the root locus method

$$G_{cl}(s) = \frac{KH(s)}{1+KH(s)} \text{ roots at } 1 + KH(s) = 0.$$

How do we move the poles by varying the constant gain K ?

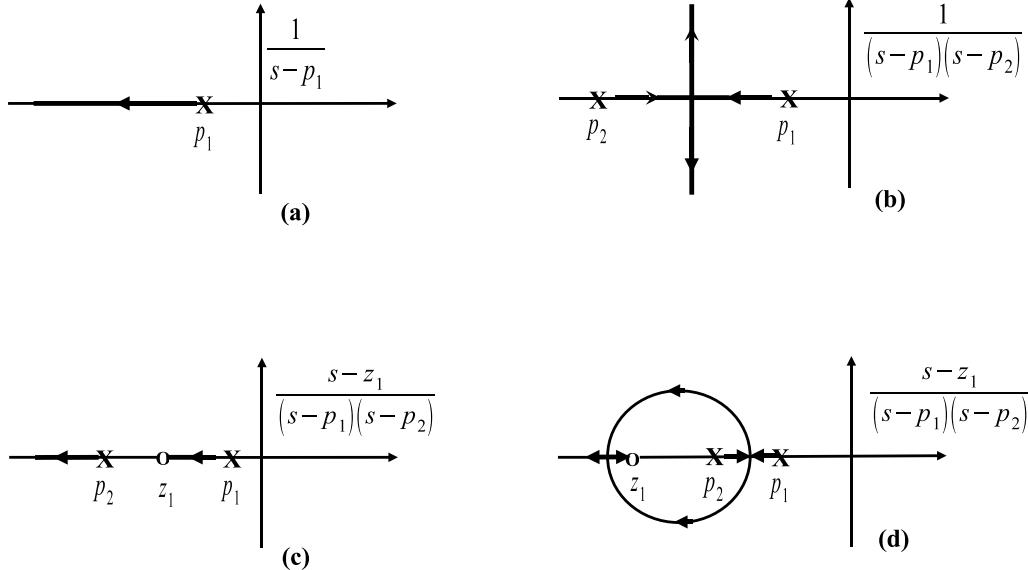


Fig. 28: Root locus curves: some typical transfer functions

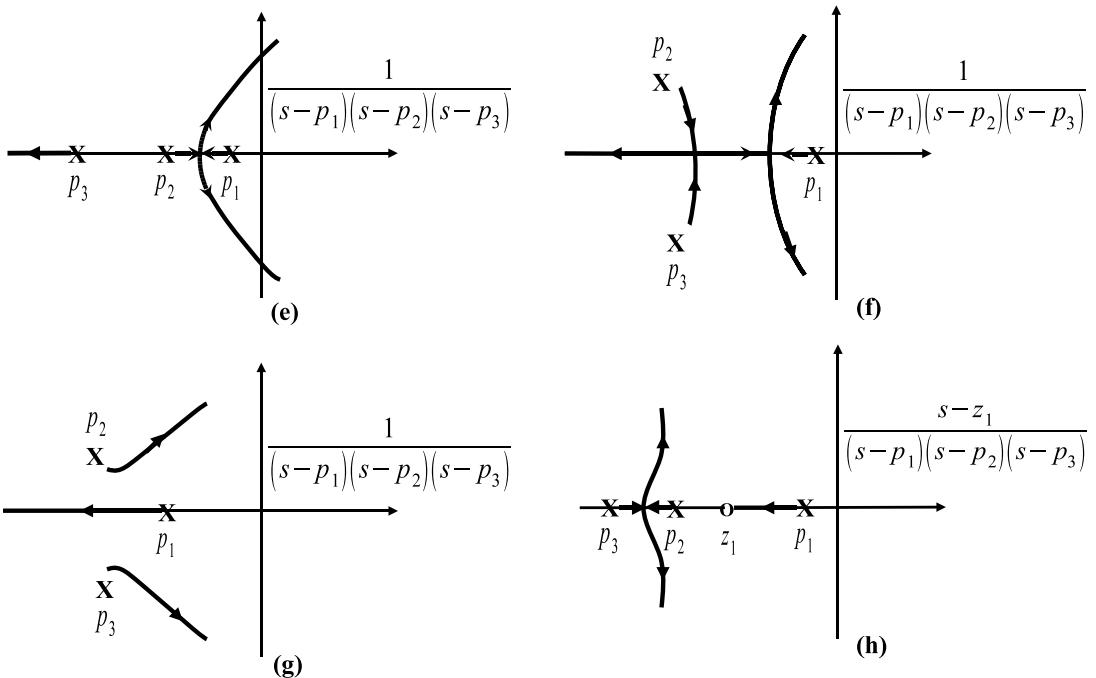


Fig. 29: More root locus curves: some typical transfer functions

6 Feedback

Feedback is both a mechanism, process, and signal that is looped back to control a system within itself. This loop is called the feedback loop. A control system usually has input and output to the system; when

the output of the system is fed back into the system as part of its input, it is called the ‘feedback’. Feedback and regulation are self-related. The negative feedback helps to maintain stability in a system in spite of external changes.

The idea:

Suppose we have a system or ‘plant’

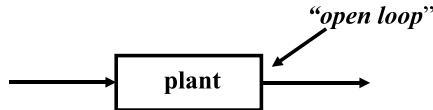


Fig. 30: Plant with input and output signals. Open-loop diagram.

We want to improve some aspect of the plant’s performance by observing the output and applying an appropriate ‘correction’ signal. **This is done by the feedback.**

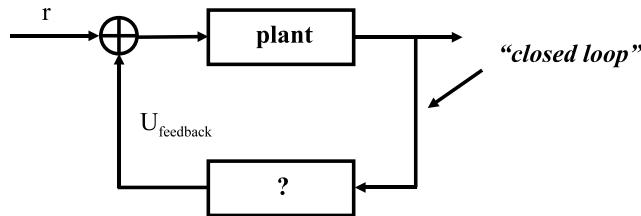


Fig. 31: Closed-loop system with reference input and plant output

Question: What should this be?

Open-loop gain: $G^{O.L}(s) = G(s) = (\frac{u}{y})^{-1}$

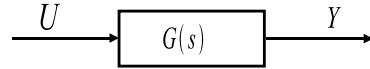


Fig. 32: Transfer functions and signals in the open-loop system

Closed-loop gain: $G^{C.L}(s) = \frac{G(s)}{1+G(s)H(s)}$

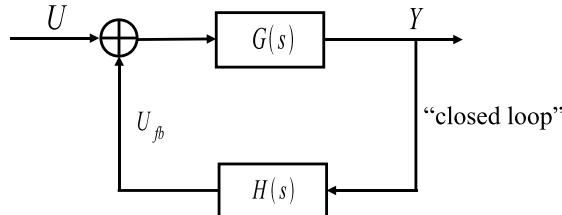


Fig. 33: Transfer functions and signals in the closed-loop system

Proof: $y = G(u - u_{fb}) = Gu - Gu_{fb} \implies y + GH_y = Gu$

$$= Gu - GHy \implies \frac{y}{u} = \frac{G}{(1+GH)}.$$

6.1 Simple harmonic oscillator

Consider a simple harmonic oscillator with feedback proportional to x i.e.:

where

$$\ddot{x} + \gamma\dot{x} + \omega_n^2 x = u + u_{fb}$$

$$u_{fb}(t) = -\alpha x(t).$$

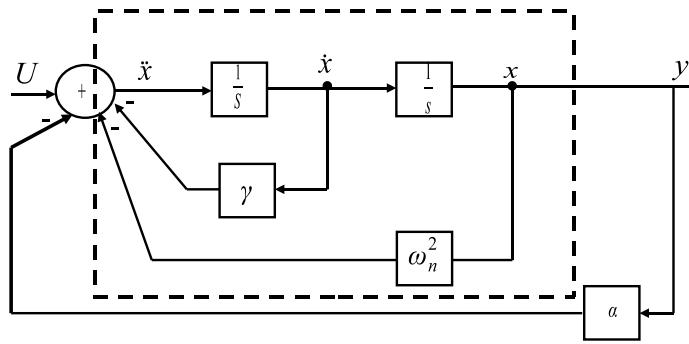


Fig. 34: Simple harmonic oscillator with proportional feedback

Then

$$\ddot{x} + \gamma\dot{x} + \omega_n^2 x = u - \alpha x$$

$$\implies \ddot{x} + \gamma\dot{x} + (\omega_n^2 + \alpha)x = u.$$

Same as before, except that ‘natural’ frequency $\omega_n^2 + \alpha$

now the closed-loop transfer function is $G^{C.L.}(s) = \frac{1}{s^2 + \gamma s + (\omega_n^2 + \alpha)}$.

So the effect of the proportional feedback in this case is *to increase the bandwidth of the system* (and reduce gain slightly, but this can easily be compensated for by adding a constant gain in front).

6.1.1 Simple harmonic oscillator with integral feedback

Suppose we use *integral feedback* in a simple harmonic oscillator

$$u_{fb}(t) = -\alpha \int_0^t x(\tau) d\tau$$

$$\text{i.e. } \ddot{x} + \gamma\dot{x} + \omega_n^2 x = u - \alpha \int_0^t x(\tau) d\tau$$

Differentiating once more yields $\ddot{x} + \gamma\ddot{x} + \omega_n^2 \dot{x} + \alpha x = \dot{u}$.

No longer just simple SHO add another state $G^{C.L.}(s) = \frac{\frac{1}{s^2 + \gamma s + \omega_n^2}}{1 + \left(\frac{\alpha}{s}\right)\left(\frac{1}{s^2 + \gamma s + (\omega_n^2 + \alpha)}\right)} = \frac{s}{s(s^2 + \gamma s + \omega_n^2) + \alpha}$

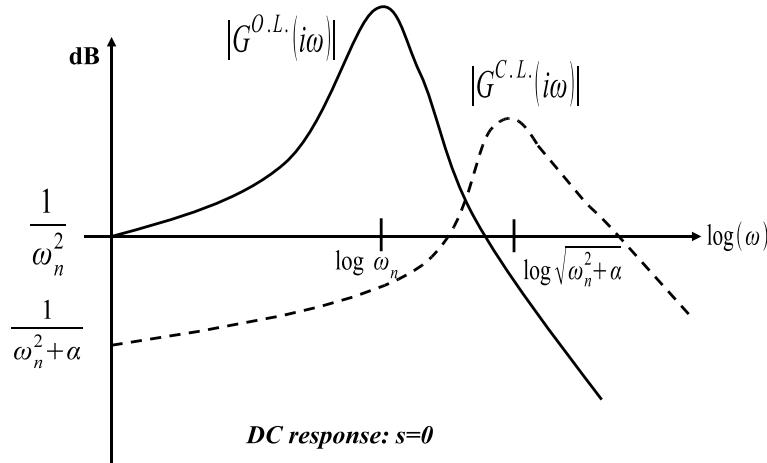


Fig. 35: Bode plot of SHO with proportional feedback. Open and closed-loop transfer function are shown.

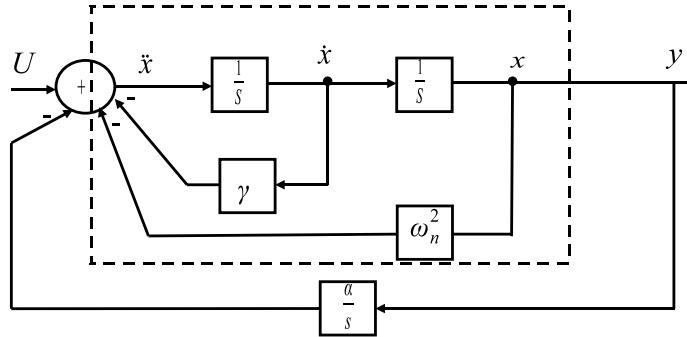


Fig. 36: Simple harmonic oscillator with integral feedback

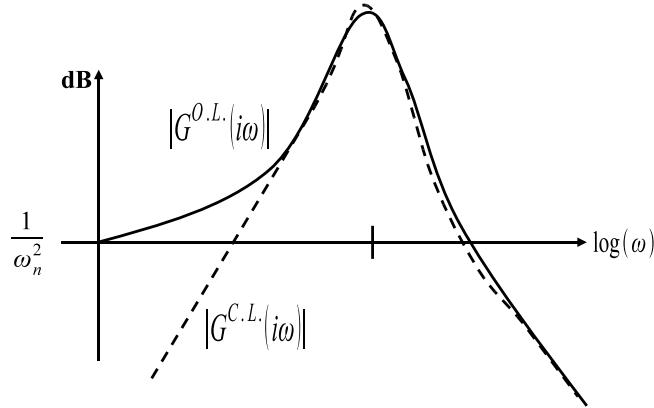


Fig. 37: Bode plot of SHO with integral feedback. Open and closed-loop transfer function are shown.

Observe that

1. $G^{C.L.}(0 = 0)$.
2. For large s (and hence for large ω)

$$G^{C.L.}(s) \approx \frac{1}{(s^2 + \gamma s + \omega_n^2)} \approx G^{O.L.}(s).$$

So integral feedback has killed the DC gain i.e., the system *rejects constant disturbances*.

6.1.2 Simple harmonic oscillator with differential feedback

Suppose we now apply *differential feedback* in a simple harmonic oscillator i.e.

$$u_{fb}(t) = -\alpha \dot{x}(t)$$

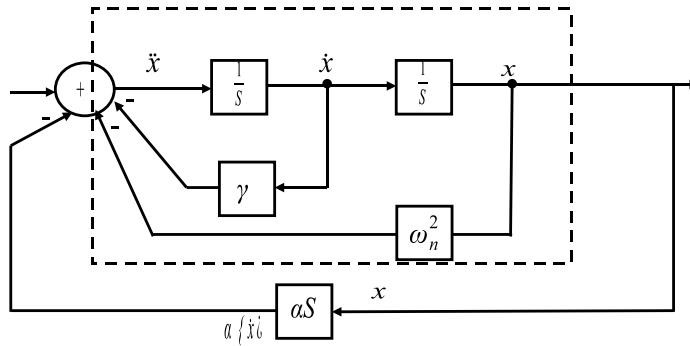


Fig. 38: Simple harmonic oscillator with differential feedback

Now we have $\ddot{x} + (\gamma + \alpha)\dot{x} + \omega_n^2 x = u$.

So the effect of differential feedback is to increase damping.

$$\text{Now } G^{C.L.}(s) = \frac{1}{s^2 + (\gamma + \alpha)s + \omega_n^2}.$$

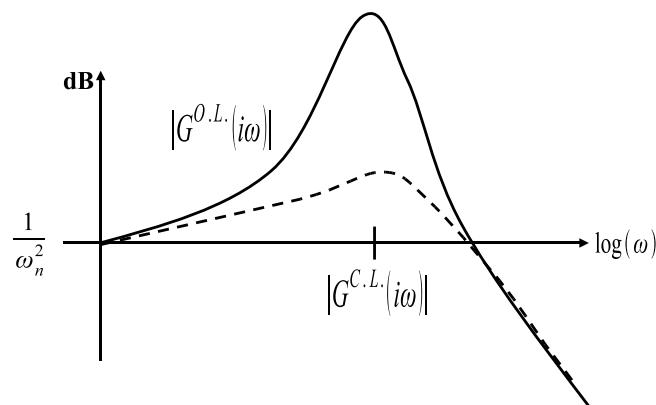


Fig. 39: Bode plot of SHO with differential feedback. Open and closed-loop transfer function are shown.

So the effect of differential feedback here is to ‘flatten the resonance’ i.e. **damping is increased**.

Note: Differentiators can never be built exactly, only approximately.

6.2 PID controller

1. The last three examples of feedback can all be combined to form a PID controller (proportional-integral-differential). $u_{fb} = u_p + u_d + u_l$

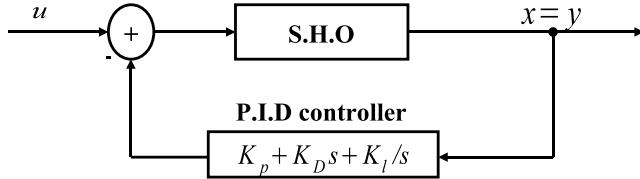


Fig. 40: SHO with PID controller

2. In the example above SHO was a very simple system and it was clear what the physical interpretation of P. or I. or D. did. But for *large complex systems* it is not obvious.

⇒ x **Require arbitrary ‘tweaking’**

That is what we are trying to avoid.

For example, if you are so clever let us see you do this with your PID controller:

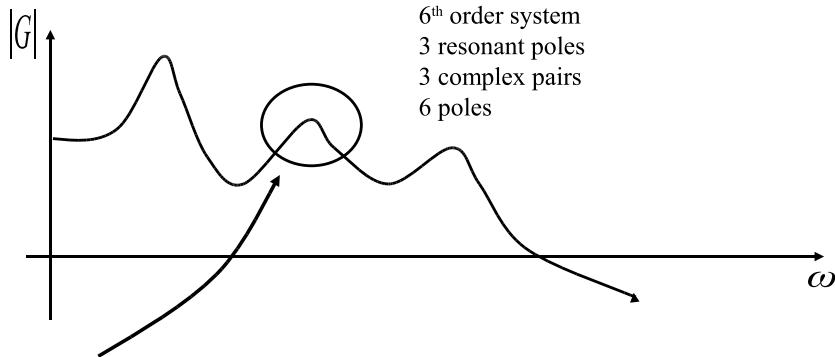


Fig. 41: PID controller adjustment problem

Damp this mode, but leave the other two modes undamped, just as they are.

This could turn out to be a tweaking nightmare. It will get you nowhere fast!

We shall see how this problem can be solved easily.

6.3 Full state feedback

Suppose we have a system $\dot{x}(t) = Ax(t) + Bu(t)$, $y(t) = Cx(t)$.

Since the state vector $x(t)$ contains all current information about the system the most general feedback makes use of **all** the state information.

$$u = -K_1 x_1 - \dots - K_n x_n = -Kx$$

where $k = [k_1 \dots k_n]$ (Row matrix)

Where example: In SHO examples

Proportional feedback: $u_p = -k_p x = -[K_p \ 0]$

Differential feedback: $u_D = -k_D \dot{x} = -[0 \ k_D]$

Theorem:

If there are no pole cancellations in $G_{O.L.}(s) = \frac{b(s)}{a(s)} = C(sI - A)^{-1}B$,

then we can move the eigen values of $A - BK$ anywhere we want using full state feedback.

Proof: Given **any** system as L.O.D.E or state space it can be written as

$$\begin{bmatrix} x_1 \\ \dots \\ x_n \end{bmatrix} = \begin{bmatrix} 0 & 1 & \dots & 0 \\ 0 & \dots & \dots & \dots \\ 0 & \dots & \dots & 1 \\ -a_0 & \dots & \dots & -a_{n-1} \end{bmatrix} \begin{bmatrix} x_1 \\ \dots \\ x_n \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \dots \\ 1 \end{bmatrix} u$$

$$y = [b_0 \ \dots \ \dots \ b_{n-1}] \begin{bmatrix} x_1 \\ \dots \\ x_n \end{bmatrix}$$

where $G^{O.L.} = C(sI - A)^{-1}B = \frac{b_{n-1}s^{n-1} + \dots + b_0}{s^n + a_{n-1}s^{n-1} + \dots + a_0}$ i.e. the first row of $A^{O.L.}$ gives the coefficients of the denominator

$$a^{O.L.}(s) = \det(sI - A^{O.L.}) = s^n + a_{n-1}s^{n-1} + \dots + a_0.$$

$$\text{Now } A^{C.L.} = A^{O.L.} - BK = \begin{bmatrix} 0 & 1 & \dots & 0 \\ 0 & \dots & \dots & 1 \\ 0 & \dots & \dots & 1 \\ -a_0 & \dots & \dots & -a_{n-1} \end{bmatrix} -$$

$$\begin{bmatrix} 0 \\ 0 \\ \dots \\ 1 \end{bmatrix} [k_0 \ \dots \ \dots \ K_{n-1}] \begin{bmatrix} 0 & 1 & \dots & 1 \\ 0 & \dots & \dots & \dots \\ 0 & \dots & \dots & 1 \\ -(a_0 + k_0) & \dots & \dots & -(a_{n-1} + k_{n-1}) \end{bmatrix}$$

So the closed-loop denominator

$$a^{C.L.}(s) = \det(sI - A^{C.L.}) = s^n + (a_0 + k_0)s^{n-1} + \dots + (a_{n-1} + k_{n-1}).$$

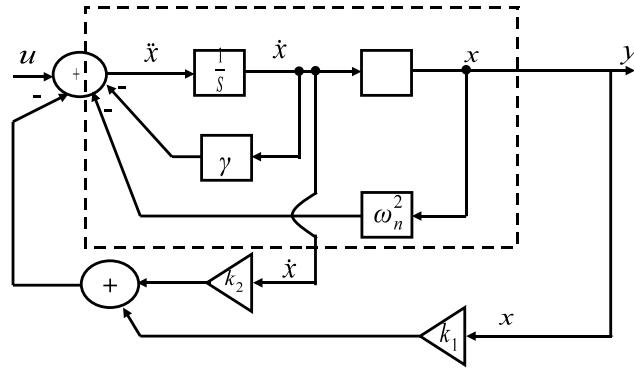
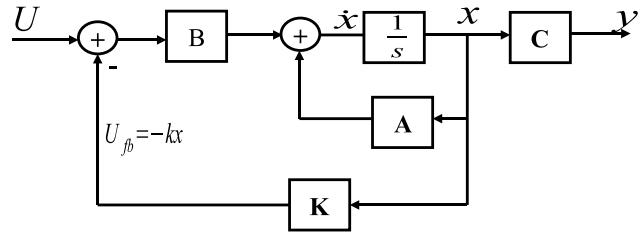
Using $u = -Kx$ we have direct control over every closed-loop denominator coefficient. We can place the root anywhere we want in the s plane.

Example: Detailed block diagram of SHO with full state feedback.

Of course this **assumes** we have access to the \dot{x} state, which we actually do not in practice.

However, let us ignore that ‘minor’ practical detail for now. (The Kalman filter will show us how to get \dot{x} from x .)

With full state feedback we have (assume $D = 0$).

**Fig. 42:** SHO with full state feedback**Fig. 43:** Diagram of state space system with full state feedback

So

$$\dot{x} = Ax + B[u + u_{fb}] = Ax + Bu + BKu_{fb}$$

$$\dot{x} = (A - BK)x + Bu$$

$$u_{fb} = -Kx$$

$$y = Cx$$

With full state feedback, we get a new closed loop matrix.

$$A^{C.L} = (A^{O.L} - BK)$$

Now all stability information is given by the eigen values of the new A matrix.

The linear time-invariant system $\dot{x} = Ax + Bu$ $y = Cx$ is said to be controllable if it is possible to find some input $u(t)$ that will transfer the initial state $x(0)$ to the origin of state-space, $x(t_0) = 0$, with t_0 finite.

The solution of the state equation is

$$x(t) = \phi(t)x(0) + \int_0^t \phi(\tau)Bu(t-\tau)d\tau.$$

For the system to be controllable, a function $u(t)$ must exist that satisfies the equation

$$0 = \phi(t_0)x(0) + \int_0^{t_0} \phi(\tau)Bu(t_0-\tau)d\tau.$$

With t_0 finite, it can be shown that this condition is satisfied if the controllability matrix

$C_M = [B \ AB \ A^2B \ \dots \ A^{n-1}B]$ was inverse. This is equivalent to the matrix C_M having full rank (rank n for an n^{th} order differential equation).

Observable:

- The linear time-invariant system is said to be observable if the initial conditions $x(0)$ can be determined from the output function $y(t), 0 \leq t \leq t_1$ where t_1 is finite with $y(t) = Cx = C\phi(t)x_0 + C \int_0^t \phi(\tau)Bu(\tau)d\tau$.
- The system is observable if this equation can be solved for $x(0)$. It can be shown that the system is observable if the matrix: $O_M = \begin{bmatrix} C \\ CA \\ \dots \\ CA^{n-1} \end{bmatrix}$.
- Was inverse. This is equivalent to the matrix C_M having full rank (rank n for an n -th order differential equation).

7 Discrete systems

Digital control is a branch of control theory that uses digital computers to act as a system. Depending on the requirements, a digital control system can take the form of a microcontroller to an ASIC to a standard desktop computer. Since a digital computer is a discrete system the Laplace transform is replaced with the Z-transform. Also since a digital computer has finite precision (see Quantization) extra care is needed to ensure the error in coefficients, A/D conversion, D/A conversion, etc. is not producing undesired or unplanned effects.

A discrete system or discrete-time system, as opposed to a continuous-time system, is one in which the signals are sampled periodically. It is usually used to connote an analog sampled system, rather than a digital sampled system, which uses quantized values.

Where do discrete systems arise?

Typical control engineering example: Assume the DAC+ADC are clocked at sampling period T . Then $u(t)$ is given by: $u(k) \equiv u_c(t); kT \leq t < (k+1)T$

$$y(k) \equiv y_c(kT); k = 0, 1, 2, \dots$$

Suppose: time continuous system is given by state-space $\dot{x}_c(t) = Ax_c(t) + Bu_c(t); x_c(0) = x_0$

$$y_c(t) = Cx_c(t) + Du_c(t).$$

7.1 State space description

Can we obtain a direct relationship between $u(k)$ and $y(k)$? We want an equivalent discrete system:
Yes! we can obtain an equivalent discrete system.

$$\text{Recall } x_x(t) = e^{At}x_c(0) + \int_0^t e^{A\tau} \cdot Bu_c(t-\tau)d\tau.$$

$$\text{From this } x_c(kT + T) = e^{AT}x_c(kT) + \int_0^T e^{A\tau} \cdot Bu_c(kT - \tau)d\tau.$$

Observe that $u(kT + T - \tau) = u(kT)$ for $\tau \in [0, T]$ i.e. $u(kT + T - \tau)$ is constant $u(kT)$ over

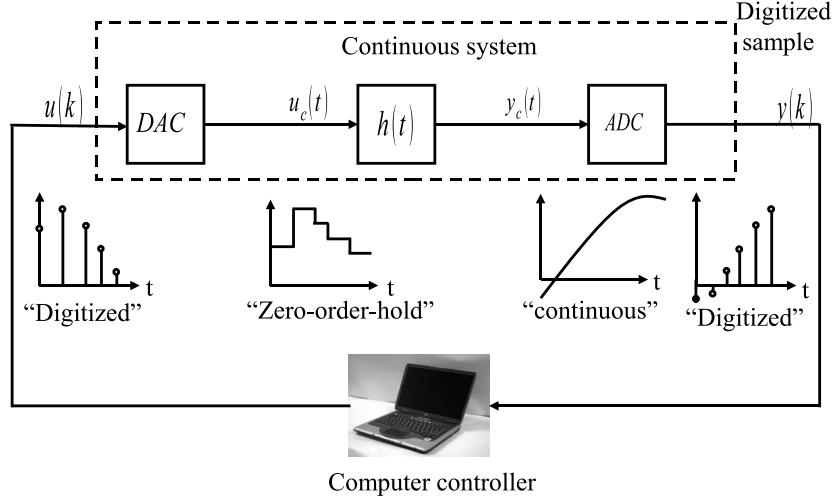


Fig. 44: Schematic of digital controller with analog-to-digital and digital-to-analog converters. The continuous and discretized signals are shown in the feedback loop.

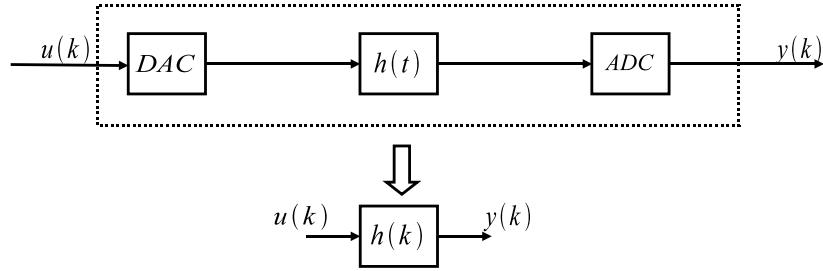


Fig. 45: Conversion from continuous system to discrete transfer function

$$\tau \epsilon [0, T]$$

can pull out of integral

$$\Rightarrow x_c(kT + T) = e^{At}x_c(kT) + (\int_0^t e^{A\tau} \cdot B d\tau)u_c(kT)$$

$$x(k+1) = A_d x(k) + B_d u(k)$$

$$y(k) = C_d x(k) + D_d u(k)$$

$$x(0) = x_c(0).$$

$$\text{So } A_d = e^{AT}, B_d = \int_0^T e^{A\tau} \cdot B d\tau, C_d = C, D_d = D.$$

So we have an exact (note: $x(k+1) = x(k) + \dot{x}(k)T + O(.)$) discrete time equivalent to the time continuous system at sample times $t = kT$ – no numerical approximation! A linear ordinary difference equation looks similar to a LODE

$$y(k+n) + a_{n-1}y(k+n-1) + \dots + a_1y(k+1) + a_0y(k) = b_m u(k+m) + \dots + b_1 u(k+1) + b_0 u(k) \quad n \geq m; \\ \text{assumes initial values } y(n-1), \dots, y(1), y(0) = 0.$$

7.2 Z transform

Z transform of the LODE yields (linearity of Z transform):

$$z_n Y(z) + z^{n-1} a_{n-1} Y(z) + \dots + z a_1 Y(z) + a_0 Y(z) = z_m b_m U(z) + \dots + z b_1 U(z) + b_0 U(z)$$

it follows the input–output relation:

$$(z^n + z^{n-1} a_{n-1} + \dots + z a_1 + a_0) Y(z) = (z^m b_m + \dots + z b_1 + b_0) U(z)$$

$$Y(z) = \frac{z^m b_m + \dots + z b_1 + b_0}{z^n + \dots + z a_1 + a_0} U(z)$$

$$Y(z) = G(z)U(z).$$

Once again: if $U(z) = 1, (u(k) = \delta(k))$, then $Y(z) = G(z)$.

Transfer function of system is the Z transform of its pulse response!

$$x(k+1) = A_d x(k) + B_d u(k)$$

$$y(k) = C x(k) + D u(k).$$

Applying the Z transform on first equation:

$$z.X(z) - zx(0) = A_d X(z) + B_d U(z)$$

$$(zI - A_d)X(z) = zx(0) + Bu(z)$$

$$X(z) = (zI - A)^{-1}zx(0) + (zI - A_d)^{-1}BU(z). \quad \text{Homogeneous solution.}$$

NOW: Particular solution:

$$Y(z) = CX(z) + DU(z) = C(zI - A_d)^{-1}zx(0) + (C(zI - A_d)^{-1}B + D)U(z)$$

if $x(0) = 0$ then we get the input–output relation:

$$Y(z) = G(z)U(z) \text{ with } G(z) = C(zI - A_d)^{-1}B + D.$$

Exactly as for the continuous systems.

To analyse discrete-time systems: the Z transform (analog to Laplace transform for time-continuous system).

It converts linear ordinary difference equation into algebraic equations: easier to find a solution of the system!

It gives the frequency response for free!

Z transform == generalized discrete-time Fourier transform

Given any sequence $f(k)$ the discrete-time Fourier transform is $F(\tilde{\omega}) = \sum_{k=-\infty}^{\infty} f(k)e^{-i\omega k}$

$\omega = 2\pi f, f = \frac{1}{T}$. The sampling frequency in Hz, T : difference/time between two samples.

In the same spirit: $F(z) = Z[f(k)] = \sum_{k=0}^{\infty} f(k)z^{-k}$

with z a complex variable. Note: if $f(k) = 0$ for $k = -1, -2, \dots$ then $\tilde{F}(\omega) = F(z = e^{j\omega})$.

7.3 Stability

A discrete LTI system is BIBO stable if $|u(k)| < M, \forall k = |y(k)|; \forall k$

$$|y(k)| = |\sum_0^k u(k-i)h(i)| \leq \sum_0^k |u(k-i)||h(i)| \leq M \sum_0^k |h(i)| \leq M \sum_0^\infty |h(i)|.$$

$$\text{For LODE state space system: } H(z) = \alpha \frac{\pi_{i=1}(z-z_i)}{\pi_{i=1}^n(z-p_i)} = \sum_{i=1}^k \beta_i T_i(z).$$

With partial fraction of the rational function, once again pole locations tell us a lot about the shape of pulse response.

Zeros determine the size of β_i . In general a complex pair \rightarrow oscillatory growth/damping.

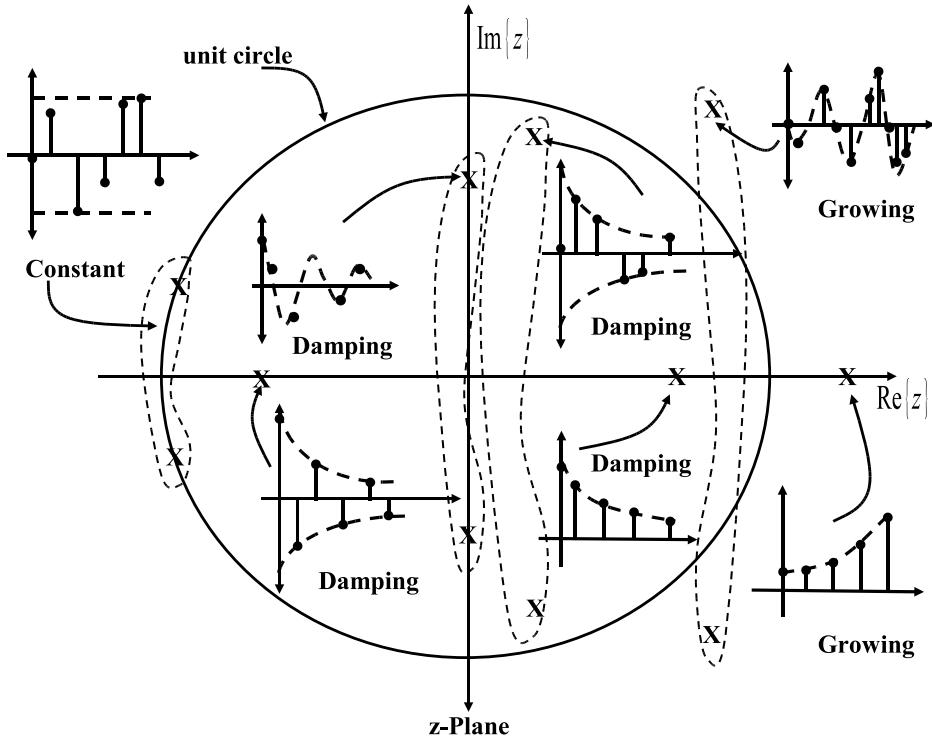


Fig. 46: Stability diagram for discrete system. The system is stable if the poles are located inside the unity circle.

A real pole \rightarrow exponential growth/decay but may be oscillatory too (e.g. $r^n 1(n)$ where $r < 0$).

The farther inside the unit circle \rightarrow the faster the damping \rightarrow the higher the stability, i.e. $|p_i| \leq 1 \rightarrow$ system stable. Stability directly from state space: exactly as for cts systems, assuming no pole-zero cancellations and $D = 0$

$$H(z) = \frac{b(z)}{a(z)} = C(zI - A_d)^{-1} B_d = \frac{C_{adj}(zI - A_d)B_d}{\det(zI - A_d)}$$

$$b(z) = C_{adj}(zI - A_d)B_d \quad a(z) = \det(zI - A_d).$$

The poles are eigenvalues of A_d . So check stability, use an eigenvalue solver to get eigenvalues of the matrix A_d , then if $|\lambda_i| < 1$ for all i then the system is stable. Where $|\lambda_i|$ is the i^{th} e-value of A_d .

7.4 Cavity model

Converting the transfer function from the continuous cavity model to the discrete model:

$$H(s) = \frac{\omega_{12}}{\Delta\omega^2 + (s + \omega_{12})^2} \begin{bmatrix} s + \omega_{12} & -\Delta\omega \\ \Delta\omega & s + \omega_{12} \end{bmatrix}.$$

The discretization of the model is represented by the z transform:

$$H(z) = (1 - \frac{1}{z})z(\frac{H(s)}{s}) = \frac{z-1}{z}.zL^{-1}\frac{H(s)}{s}|_{t=kT_s}$$

$$H(z) = \frac{\omega_{12}}{\Delta\omega^2 + \omega_{12}^2} \cdot \begin{bmatrix} \omega_{12} & -\Delta\omega \\ \Delta\omega & \omega_{12} \end{bmatrix} - \left(\frac{\omega_{12}}{\Delta\omega^2 + \omega_{12}^2} \cdot \frac{z-1}{z^2 - 2ze^{\omega_{12}T_s} \cdot \cos(\Delta\omega T_s) + e^{2\omega_{12}T_s}} \right) \cdot ((z - e^{\omega_{12}T_s} \cdot \cos(\Delta\omega T_s)) \cdot \begin{bmatrix} \omega_{12} & -\Delta\omega \\ \Delta\omega & \omega_{12} \end{bmatrix}) - e^{\omega_{12}T_s} \cdot \sin(\Delta\omega T_s) \cdot \begin{bmatrix} \Delta\omega & \omega_{12} \\ -\omega_{12} & \Delta\omega \end{bmatrix}.$$

Given: $x(k+1) = Ax(k) + Bu(k)$

$z(k) = Cx(k)$. Assume $D = 0$ for simplicity.

8 Optimal control:

Optimal control deals with the problem of finding a control law for a given system such that a certain optimality criterion is achieved. A control problem includes a cost functional that is a function of state and control variables. An optimal control is a set of differential equations describing the paths of the control variables that minimize the cost functional.

Suppose the system is unstable or almost unstable. We want to find $u_{fb}(k)$ which will bring $x(k)$ to zero, quickly, from any initial condition.

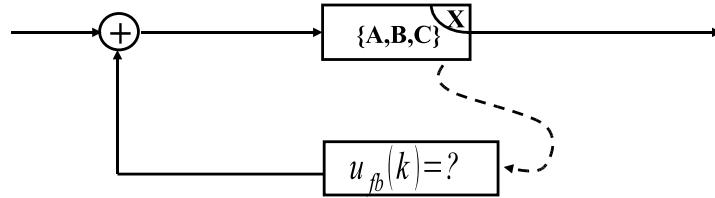


Fig. 47: Formulation of the problem of the optimal controller

A quadratic form is a quadratic function of the components of a vector:

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \quad f(x) = f(x_1, x_2) = ax_1^2 + bx_1x_2 + cx_2^2$$

$$[x_1, x_2] \begin{bmatrix} a & \frac{1}{2}b \\ \frac{1}{2}b & d \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + [c \ 0] \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$f(x) = x^T Q x$, quadratic part, $+P^T x$, linear part, $+e$ constant.

What do we mean by ‘bad’ damping and ‘cheap’ control? We now define precisely what we mean. Consider:

$$J \equiv \sum_{i=0}^{\infty} x_i^T Q x_i + u_i^T R u_i.$$

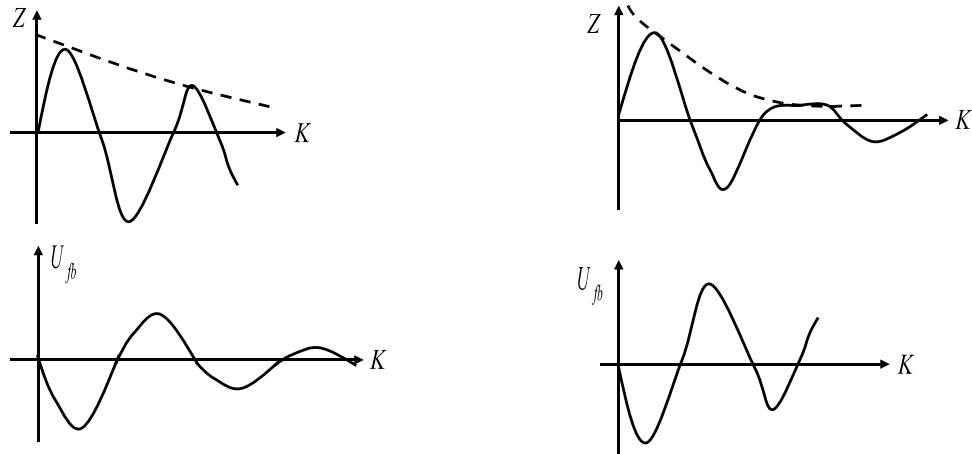


Fig. 48: Tradeoff between bad damping and cheap control (left side) and good damping and expensive control (right)

The first term penalizes large state executions, the second penalizes large control. $Q \geq 0, R > 0$

Can there be a tradeoff between state excursions and control by varying Q and R ?

Large Q — ‘good’ damping important

Large R — actuator effort ‘expensive’

(Linear quadratic regulator)

$$x_{i+1} = Ax_i + Bu_i; \quad x_0$$

Find control sequence u_0, u_1, u_2, \dots such that $J = \sum_{i=0}^{\infty} x_i^T Q x_i + u_i^T R u_i$ = minimum.

Answer: The optimal control sequence is a state feedback sequence $u_i^{\infty}_0$

$$u_i = -K_{opt} x_i$$

$$K_{opt} = (R + B^T S B)^{-1} B^T S A$$

$$S = A^T S A + Q - A^T A B (R + B^T S B)^{-1} B^T S A$$

Algebraic Riccati Equation (ARE) for discrete-time systems.

Note: Since u_i = state feedback, it works for any initial state x_0 .

Remarks:

1. So optimal control, $u_i = K_{opt} x_i$ is state feedback! This is why we are interested in state feedback.
2. Equation ARE is a matrix quadratic equation. Looks pretty intimidating but computer can solve in a second.
3. No tweaking! Just specify A, B, C, D and Q and R , press return button, LQR routine spits out K_{opt} — done (of course guaranteed optimal in the sense that it minimizes).
4. Design is guaranteed optimal in the sense that it minimizes.

$$J_{lqr}(x_0, u_i^\infty x_i^T Q x_i + u_i^T R u_i)$$

(Of course that does not mean it is the ‘best’ in the absolute sense.)

As we vary the Q/R ratio we get whole family of k_{lqr} ’s, i.e., can trade off between state excursion (damping) vs actuator effort (control)

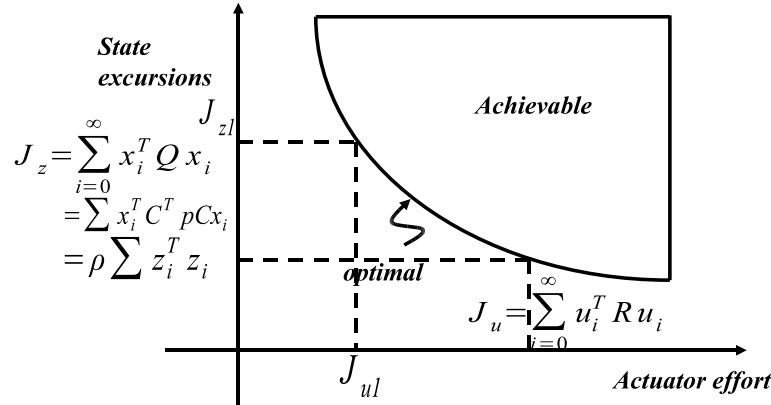


Fig. 49: Tradeoff between control effort and rms state excursions

Our optimal control has the form $u_{opt}(k) = -K(k)x_{opt}(k)$.

This assumes that we have complete state information $x_{opt}(k)$. This is not actually the case, e.g., in SHO, we might have only a position sensor but not a velocity sensor.

8.1 The Kalman filter

How can we obtain ‘good’ estimates of the velocity state by just observing the position state?

The sensors may be noisy and the plant itself may be subject to outside disturbances (process noise) i.e., we are looking for this:

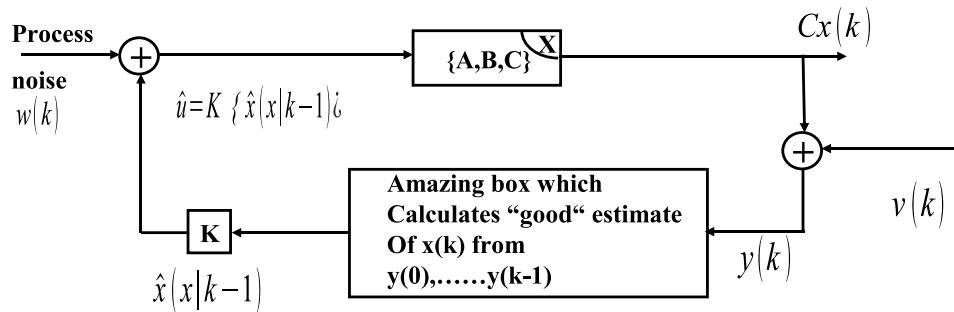


Fig. 50: Formulation of the problem to be solved by the Kalman filter

$$x(k+1) = Ax(k) + Bw(k)$$

$$z(k) = Cx(k)$$

$y(k) = Cx(k) + v(k)$. Assumes also $x(0)$ is random and Gaussian and that $x(k), w(k) + v(k)$ are

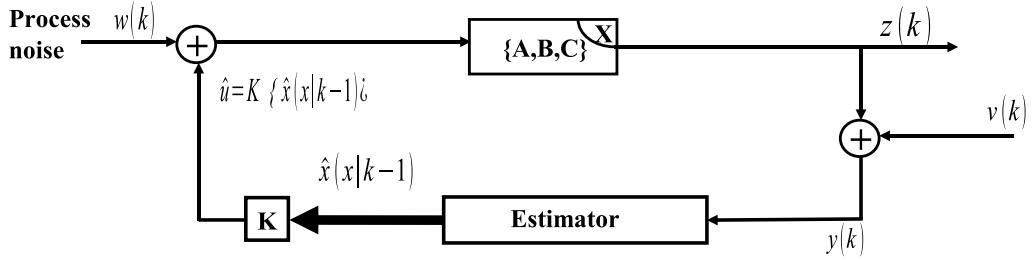


Fig. 51: Concept of the estimator which provides a best state estimate from model and noisy sensor signal

all mutually independent for all k .

Find: $\hat{x}(k|k-1)$ optimal estimate of $x(k)$ given y_0, \dots, y_{k-1} such that ‘mean squared error’.

$$E[\|x(k) - \hat{x}(k|k-1)\|_2^2] = \text{minimal.}$$

Fact from statistics: $\hat{x}(k|k-1) = E[x(k)|(y_0, \dots, y_{k-1})]$ The Kalman filter is an efficient algorithm that computes the new $\hat{x}_{i+1|i}$ (the linear-least-mean-square estimate) of the system state vector x_{i+1} , given y_0, \dots, y_i , by updating the old estimate $\hat{x}_{i|i-1}$ and old $x_{i|i-1}$ (error). The Kalman filter produces

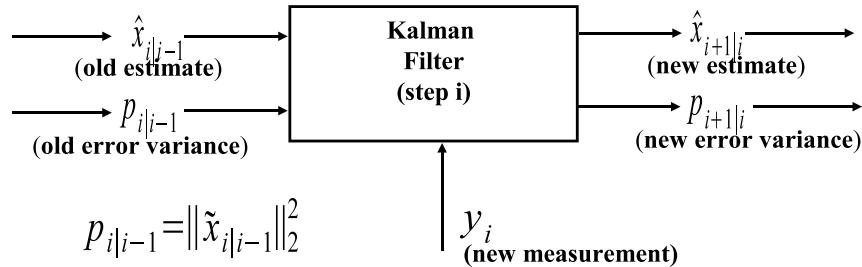


Fig. 52: Update algorithm for the Kalman filter

$\hat{x}_{i+1|i}$ from $\hat{x}_{i|i-1}$ (rather than $\hat{x}_{i|i}$, because it ‘tracks’ the system ‘dynamics’). By the time we compute $\hat{x}_{i|i}$ from $\hat{x}_{i|i-1}$, the system state has changed from x_i .

The Kalman filter algorithm can be divided into a measurement update and a time update:

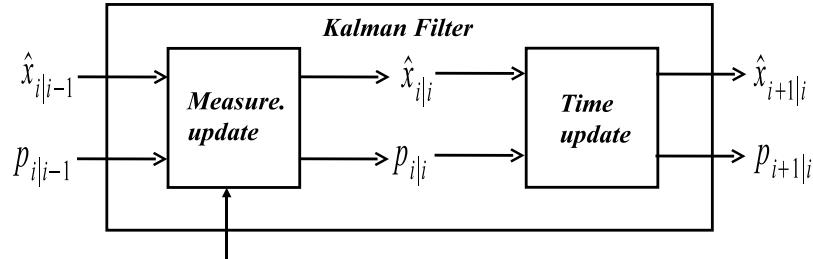


Fig. 53: Separation between measurement and time updates in the Kalman filter

Measurement update(M.U): $\hat{x}_{i|i} = \hat{x}_{i|i-1} + P_{i|i-1} C^T (C P_{i|i-1} C^T + V)^{-1} (y_i - C \hat{x}_{i|i-1})$

$$P_{i|i} = P_{i|i-1} - P_{i|i-1} C^T (C P_{i|i-1} C^T + V)^{-1} C P_{i|i-1}.$$

Time update (T.U.): $\hat{x}_{i+1|i} = A\hat{x}_{i|i}$ and $P_{i+1|i} = Ap_{i|i}A^T + BWB^T$.

With initial conditions: $\hat{x}_{0|-1} = 0$ and $p_{0|-1} = X_0$.

By plugging M.U equations in to T.U equations one can do both steps at once:

$$\hat{x}_{i+1|i} = A\hat{x}_{i|i} = A\hat{x}_{i|i-1} + Ap_{i|i-1}C^T(Cp_{i|i-1}C^T + V)^{-1}(y_i - C\hat{x}_{i|i-1}).$$

$$\hat{x}_{i+1|i} = A\hat{x}_{i|i-1} + L_i(y_i - C\hat{x}_{i|i-1}) \text{ where } L_i \equiv A(p_{i|i-1}C^T(Cp_{i|i-1}C^T + V)^{-1})$$

$$p_{i+1|i} = Ap_{i|i}A^T + BWB^T = A[p_{i|i-1} - p_{i|i-1}C^T(Cp_{i|i-1}C^T + V)^{-1}CP_{i|i-1}]A^T + BWB^T$$

$$p_{i+1|i} = Ap_{i|i}A^T + BWB^T - Ap_{i|i-1}C^T(Cp_{i|i-1}C^T + V)^{-1}(Cp_{i|i-1} - 1)A^T$$

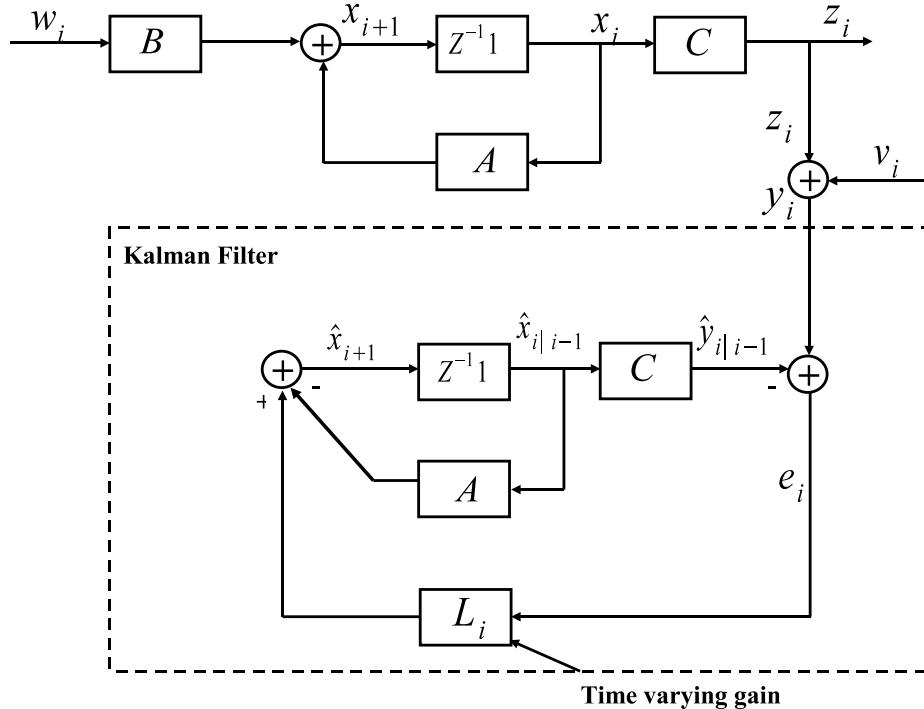


Fig. 54: Model of Kalman filter with time varying gain

Plant equations: $x_{i+1} = Ax_i + Bu_i$ and $y_i = Cx_i + v_i$.

Kalman filter: $\hat{x}_{i+1|i} = A\hat{x}_{i|i-1} + L_i(y_i - \hat{y}_{i|i-1})$ and $y_{i|i-1} = C\hat{x}_{i|i-1}$. If $v = w = 0$ Kalman filter can estimate the state precisely in a finite number of steps.

Remarks:

1. Since $y_i = Cx_i + v_i$ and $\hat{y}_{i|i-1} = C\hat{x}_i$ can write estimator equation as $\hat{x}_{i+1|i} = A\hat{x}_{i|i-1} + L_i(Cx_i + v_i - C\hat{x}_{i|i-1}) = (A - L_iC)x_i + L_iC\hat{x}_{i|i-1} + v_i$. We can combine this with equation for x_{i+1}

$$\begin{bmatrix} x_{i+1} \\ \hat{x}_{i+1|i} \end{bmatrix} = \begin{bmatrix} A & 0 \\ L_iC & A - L_iC \end{bmatrix} \begin{bmatrix} x_i \\ \hat{x}_{i|i-1} \end{bmatrix} + \begin{bmatrix} B & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} w_i \\ v_i \end{bmatrix}$$

$$\begin{bmatrix} z_i \\ \hat{y}_{i|i-1} \end{bmatrix} = \begin{bmatrix} C & 0 \\ 0 & C \end{bmatrix} \begin{bmatrix} x_i \\ \hat{x}_{i|i-1} \end{bmatrix}$$

2. In practice, the Riccati equation reaches a steady state in a few steps. People often run with steady-state Kalman filters, i.e.

$$L_s s = A p_{ss} C^T (C P_{ss} C^T + V)^{-1},$$

where $p_{ss} = A p_{ss} A^T + B W B^T - A p_{ss} C^T (C P_{ss} C^T + V)^{-1} C P_{ss} A$.

8.2 LQG controller

Now we are finally ready to solve the full control problem.

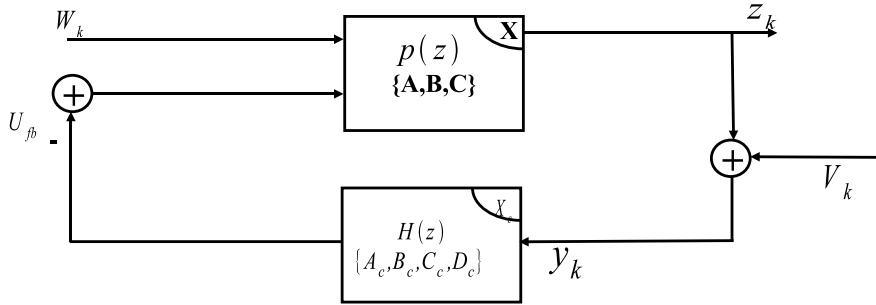


Fig. 55: LQG controller problem definiton

Given: $x_{k+1} = Ax_k + Bu_k + B_w w_k$, $z_k = Cx_k$, $y_k = Cx_k + v_k$, $\langle w_i, w_j \rangle = w\delta_{ij}$, $\langle v_i, v_j \rangle = V\delta_{ij}$, $\langle w_i, v_j \rangle = 0$, where w_k and v_k are both Gaussian. For Gaussian, K.L. gives the absolute best estimate.

Separation principle: (We will not prove this.) The separation principle states that the LQG optimal controller is obtained by:

1. using a Kalman filter to obtain least-squares optimal estimate of the plant state,

i.e., Let $x_c(k) = \hat{x}_{k|k-1}$.

2. Feedback estimated LQR-optimal state feedback

$u(k) = -K_{LQR}x_c(k) = -K_{LQR}\hat{x}_{k|k-1}$. One can treat problems of optimal feedback and a state estimate separately.

Plant: $\|x_{k+1} = Ax_k + (-Bu_k) + B_w w_k, z_k = Cx_k, y_k = Cx_k + v_k\|$.

LQG Controller $\|\hat{x}_{k+1|k} = A\hat{x}_{k|k-1} + Bu_k + L(y_k - C\hat{x}_{k|k-1}), u_k = -K\hat{x}_{k|k-1}\|$

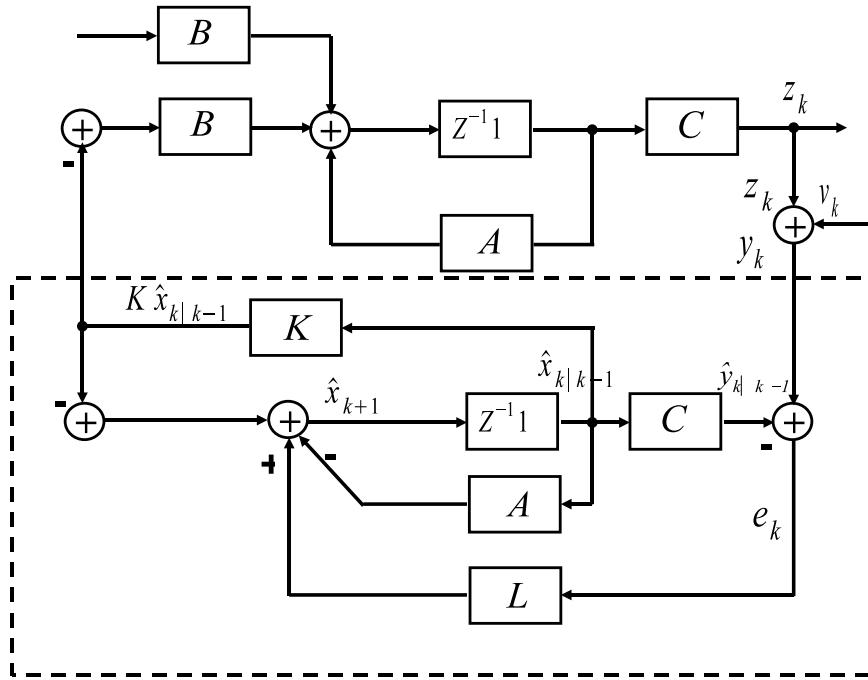
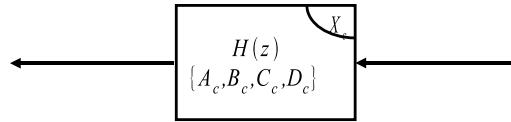
where $k = -[R + B^T S B]^{-1} + s = A^T S A + Q - A^T S B [R + B^T S B]^{-1} B^T S A$, and

$$L = A P C^T [V + C P C^T]^{-1} + P = A P A^T + B W B^T [V + C P C^T]^{-1} C P C^T.$$

We want a controller which takes as input noisy measurements y and produces as output a feedback signal u which will minimize excursions of the regulated plant outputs (if no pole-zero cancellation, then this is equivalent to minimizing state excursions).

We also want to achieve ‘regulation’ with as little actuator effort u as possible.

Problem statement (mathematically) Find: controller $H(z) = C_c(zI - A_c)^{-1}B_c + D_c$.

**Fig. 56:** Visual representation of LQG controller**Fig. 57:** LQG controller

Controller: $x_c(k+1) = A_c x(k+1) = A_c x(k+1) + B_c y(k)$, $y_c(k) = C_c x_c(k)$ which will minimize the cost, where

$J_{LQG} = \lim_{k \rightarrow \infty} E[x_k^T Q x_k]$ are r.m.s. ‘state’ excursions and $+u_k^T R u_k$ is the r.m.s. ‘actuator’ effort.

$$\|x_{k+1} = Ax_k + (-Bu_k) + B_w w_k, z_k = Cx_k, y_k = Cx_k + v_k\|$$

Remarks:

1. Q and R are weighting matrices that allow trading off r.m.s. u and r.m.s. x .
2. If $Q = C^T \rho C$; $\rho > 0$ then trade off r.m.s. z vs r.m.s. u .
3. In the stochastic LQR case, the only difference is that now we do not have complete state information $y_i = Cx_i + v_i$ we have only noisy observations i.e. cannot use full state feedback.

Idea: Could we use estimated state feedback? (i.e. $-K \hat{x}_{k|k-1}$)

4. We can let Q/R ratio vary and we will obtain the family of LQG controllers. We can plot r.m.s. Z r.m.s. u for each one.

So by specifying (1) system model, (2) noise variance, (3) optimally criterion J_{LQG} and plotting trade off curve completely specifies the limit of performance of the system i.e. which combinations of (Z_{rms}, U_{rms}) are achievable by any controller’s good ‘benchmark curve’.

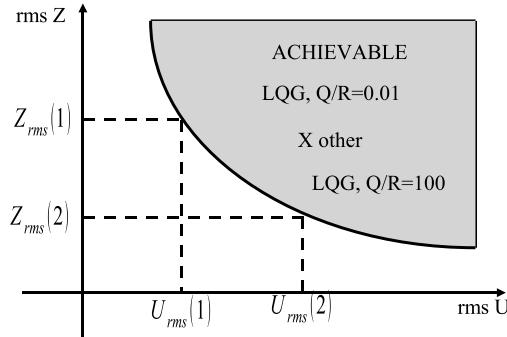


Fig. 58: Tradeoff between control effort and r.m.s. error

9 System identification

System identification is a general term to describe mathematical tools and algorithms that build dynamical models from measured data. A dynamical mathematical model in this context is a mathematical description of the dynamic behaviour of a system or process.

What is system identification? Using experimental data obtained from input/output relations to model dynamic systems.

There are different approaches to system identification depending on the model class.

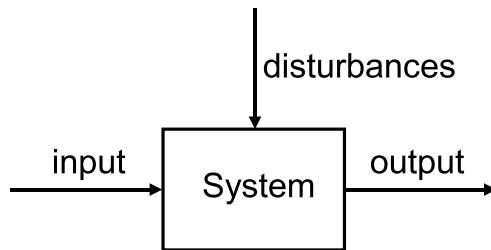


Fig. 59: System with input, output and disturbances. Goal is the identification of the system from measurements of the input and output signals.

1. Linear/non-linear
2. Parametric/non-parametric
 - Non-parametric methods try to estimate a generic model (step responses, impulse responses, frequency responses)
 - Parametric methods estimate parameters in a user-specified model (transfer functions, state-space matrices)

Identification steps: System identification is the experimental approach to process modelling.

System identification includes the following steps:

1. **Experimental design:** Its purpose is to obtain good experimental data, and it includes the choice of the measured variables and of the character of the input signals.
2. **Selection of model structure:** A suitable model structure is chosen using prior knowledge and trial and error.

3. **Choice of the criterion to fit:** A suitable cost function is chosen, which reflects how well the model fits the experimental data.
4. **Parameter estimation:** An optimization problem is solved to obtain the numerical values of the model parameters.
5. **Model validation:** The model is tested in order to reveal any inadequacies.

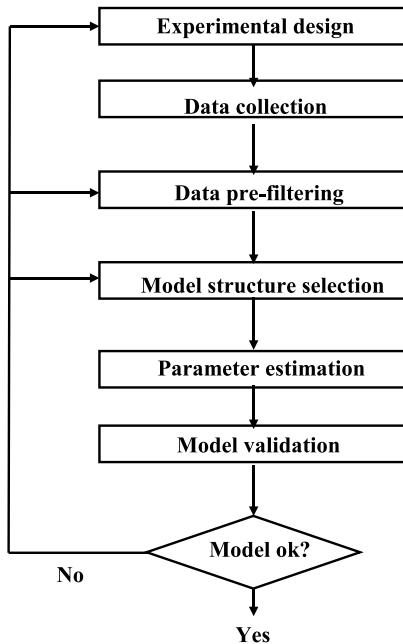


Fig. 60: Procedure for system identification

Different mathematical models: Model description:

1. Transfer functions
2. State-space models
3. Block diagrams (e.g. Simulink)

Notation for continuous time models: Complex Laplace variable s and differential operator p

$$\dot{x}(t) = \frac{\partial x(t)}{\partial t} = px(t) .$$

Notation for discrete time models: Complex z transform variable and shift operator q

$$x(k+1) = qx(k) .$$

Experiments and data collection: It is often good to use a two-stage approach.

1. Preliminary experiments
 - Step/impulse response tests to get basic understanding of system dynamics
 - Linearity, static gains, time delays constants, sampling interval
2. Data collection for model estimation
 - Carefully designed experiment to enable good model fit

- Operating point, input signal type, number of data points to collect

Preliminary experiments-step response: Useful for obtaining qualitative information about the system.

- Dead-time (delay)
- Static gain
- Time constants
- Resonance frequency

Sample time can be determined from time constants.

Rule-of-thumb: 4–10 sample points over the rise time.

Design experiment for model estimation: Input signal should excite all relevant frequencies

- Estimated model more accurate in frequency ranges where input has high energy.
- Pseudo-Random Binary Sequence (PRBS) is usually a good choice.

Trade-off in selection of signal amplitude

- Large amplitude gives high signal-to-noise ratio (SNR), low parameter variance.
- Most systems are non-linear for large input amplitudes.

Big difference between estimation of system under closed-loop control or not.

Collecting data: Sampling time selection and anti-alias are central.

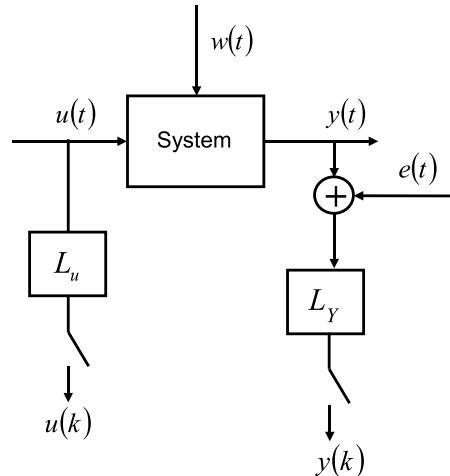


Fig. 61: Concept of the measurement of discretized input and output signals

Data pre-filtering:

Remove:

- Transients needed to reach desired operating point
- Mean values of input and output values, i.e. work with

$$\Delta u[t] = u[t] - \frac{1}{N} \sum_{t=1}^N u[t]$$

$$\Delta y[t] = y[t] - \frac{1}{N} \sum_{t=1}^N y[t]$$

- Trends (use ‘detrend’ in MATLAB)
- Outliers (obvious erroneous data points)

General model structure

1. Systems described in terms of differential equations and transfer functions.
2. Provides insight into the system physics and a compact model structure general-linear polynomial model or the general-linear model.

$$y(k) = q^{-k} G(q^{-1}, \Theta) u(k) + H(q^{-1}, \Theta) e(k)$$

- $u(k)$ and $y(k)$ are the input and output of the system, respectively.
- $e(k)$ is zero-mean white noise, or the disturbance of the system.
- $G(q^{-1}, \Theta)$ is the transfer function of the stochastic part of the system.
- $H(q^{-1}, \Theta)$ is the transfer function of the stochastic part of the system.
- General-linear model structure:

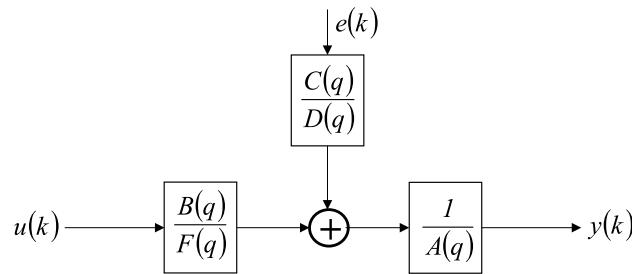


Fig. 62: Transfer functions in universal model structure

3. Provides flexibility for system and stochastic dynamics. Simpler models that are a subset of the linear model structure.

Model structures based on input–output:

Model	$\tilde{p}(q)$	$\tilde{p}_e(q)$
ARX	$\frac{B(q)}{A(q)}$	$\frac{1}{A(q)}$
ARMAX	$\frac{B(q)}{A(q)}$	$\frac{C(q)}{A(q)}$
Box–Jenkins	$\frac{B(q)}{F(q)}$	$\frac{C(q)}{D(q)}$
Output error	$\frac{B(q)}{F(q)}$	1
FIR	$B(q)$	1

$$A(q)y[k] = \frac{B(q)}{F(q)}u[k] + \frac{C(q)}{D(q)}y[k] \text{ory}[k] = \tilde{p}(q)u[k] + \tilde{p}_e(q)e[k]$$

Provides flexibility for system and stochastic dynamics. Simpler models that are a subset of the general linear model structure.

Parametric models:

1. Each of these methods has their own advantages and disadvantages and is commonly used in real-world applications
2. Choice of the model structure to use depends on the dynamics and the noise characteristics of the system.
3. Using a model with more freedom or parameters is not always better as it can result in the modelling of non-existent dynamics and noise characteristics.
4. This is where physical insight into a system is helpful.

AR model

ARX model

ARMAX model

Box–Jenkins model

Output error model

State space model

9.1 AR model:

1. Process model where outputs are only dependent on previous outputs.
2. No system inputs or disturbances are used in the modelling.
3. Class of solvable problems is limited. For signals not for systems.
4. Time series analysis, such as linear prediction coding commonly use the AR model.

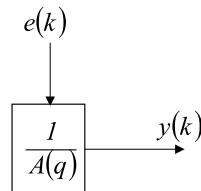


Fig. 63: AR model structure

9.2 ARX model:

Advantages:

- Is the most efficient of the polynomial estimation methods → solving linear regression equations in analytic form.
- Unique solution that satisfies the global minimum of the loss function.
- Preferable, especially when the model order is high.

Disadvantages:

- Disturbances are part of the system dynamics.
- The transfer function of the deterministic part $G(q^{-1}, \Theta)$ of the system and the transfer function of the stochastic part $H(q^{-1}, \Theta)$ of the system have the same set of poles.
- This coupling can be unrealistic.

- The system dynamics and stochastic dynamics of the system do not share the same set of poles all the time.
- However, you can reduce this disadvantages if you have a good signal-to-noise ratio.
- Set the model order higher than the actual model order to minimize the equation error, especially when the signal-to-noise ratio is low.
- However, increasing the model order can change some dynamic characteristics of the model, such as the stability of the model.

$$y[k] = \frac{B(q)}{A(q)}u[k] + \frac{1}{A(q)}e[k]$$

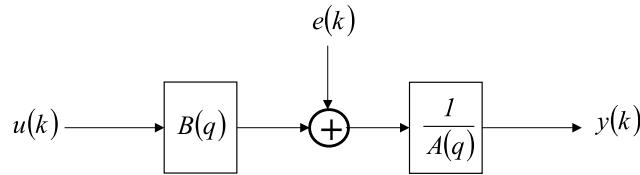


Fig. 64: ARX model structure

9.3 ARMAX model:

1. Includes disturbances dynamics. ARMAX models are useful when you have dominating disturbances that enter early in the process, such as at the input.
For example, a wind gust affecting an aircraft is a dominating disturbance early in the process.
2. More flexibility in the handling of disturbance modelling.

$$y[k] = \frac{B(q)}{A(q)}u[k] + \frac{C(q)}{A(q)}e[k]$$

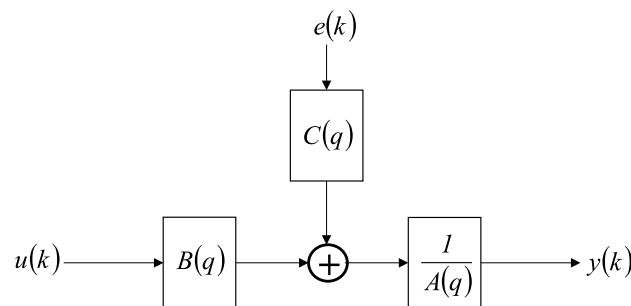


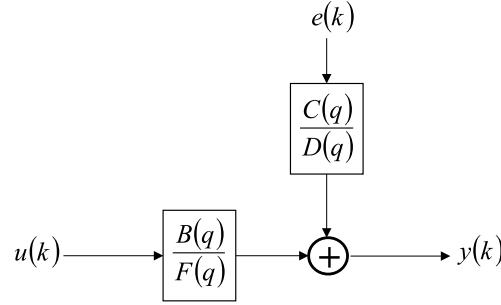
Fig. 65: ARMAX model structure

9.4 Box-Jenkins model:

- Provides a complete model with disturbance properties modelled separately from system dynamics.
- The Box-Jenkins model is useful when you have disturbances that enter late in the process.

For example, measurement noise on the output is a disturbance late in the process (not output error model).

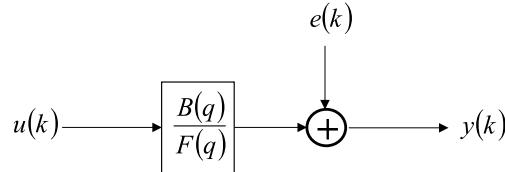
$$y[k] = \frac{B(q)}{F(q)}u[k] + \frac{C(Q)}{D(q)}e[k]$$

**Fig. 66:** Box–Jenkins model structure

9.5 Output error model:

- Describes the system dynamics separately.
- No parameters are used for modelling the disturbance characteristics.

$$y[k] = \frac{B(q)}{F(q)}u[k] + e[k]$$

**Fig. 67:** Output error model structure

9.6 State space model:

1. All previous methods are based on minimizing a performance function.
2. For complex high-order systems the classical methods can suffer from several problems.
3. Find many local minima in the performance function and do not converge to global minima.
4. The user will need to specify complicated parametrization. They also may suffer potential problems with numerical instability and excessive computation time to execute the iterative numerical minimization methods needed.
5. In addition, modern control methods require a state-space model of the system.
6. For cases such as these the State-Space (SS) identification method is the appropriate model structure.

Hint: When the model order is high, use an ARX model because the algorithm involved in ARX model estimation is fast and efficient when the number of data points is very large. The state-space model estimation with a large number of data points is slow and requires a large amount of

memory. If you must use a state-space model, for example in modern control methods, reduce the sampling rate of the signal in case the sampling rate is unnecessarily high.

$$x(k+1) = Ax(k) + Bu(k) + Ke(k)$$

$$y(k) = Cx(k) + Du(k) + e(k)$$

$x(k)$: state vector (Dim. setting you need to provide for the state-space model)

$y(k)$: system output

$u(k)$: System input

$e(k)$: Stochastic error. A, B, C, D, K are system matrices.

7. In general, they provides a more complete representation of the system, especially for complex MIMO systems, because similar to first-principle models.
8. The identification procedure does not involve non-linear optimization so the estimation reaches a solution regardless of the initial guess.
9. Parameter settings for the state-space model are simpler.
10. You need to select only the order/states, of the model.
11. Prior knowledge of the system or by analysing the singular values of A .

Determine parameters: Determining the delay and model order for the prediction error methods, ARMAX, BJ, and OE, is typically a trial-and-error process.

The following is a useful set of steps that can lead to a suitable model (this is not the only methodology you can use, nor is this a comprehensive procedure).

1. Obtain useful information about the model order by observing the number of resonance peaks in the non-parametric frequency response function. Normally, the number of peaks in the magnitude response equals half the order of $A(q), F(q)$.
2. Obtain a reasonable estimate of delay using correlation analysis and/or by testing reasonable values in a medium-sized ARX model. Choose the delay that provides the best model fit based on prediction errors or other fit criteria.
3. Test various ARX model orders with this delay choosing those that provide the best fit.
4. Since the ARX model describes both the system dynamics and noise properties using the same set of poles, the resulting model may be unnecessarily high in order.
5. By plotting the zeros and poles (with the uncertainty intervals) and looking for cancellations you can reduce the model order. The resulting order of the poles and zeros are a good starting point for ARMAX, OE and/or BJ models with these orders used as the B and F model parameters and first- or second-order models for the noise characteristics.
6. Are there additional signals? Measurements can be incorporated as extra input signals!
7. If you cannot obtain a suitable model following these steps, additional physical insight into the problem might be necessary (compensating for non-linear sensors or actuators and handling of important physical non-linearities are often necessary in addition to using a ready-made model).

From the prediction error standpoint, the higher the order of the model, the better the model fits the data because the model has more degrees of freedom. However, you need more computation time and memory for higher orders. The parsimony principle advocates choosing the model with the smallest degree of freedom, or number of parameters, if all the models fit the data well and pass the verification test.

Conclusion:

- Variety of model structures available
- Choice is based upon an understanding of the system identification method and algorithm
- System and disturbance are important
- Handling a wide range of system dynamics without knowing system physics
- Reduction of engineering effort
- Identification tools (Matlab, Matrix or LabVIEW) are available for developing, prototyping, and deploying control algorithms.

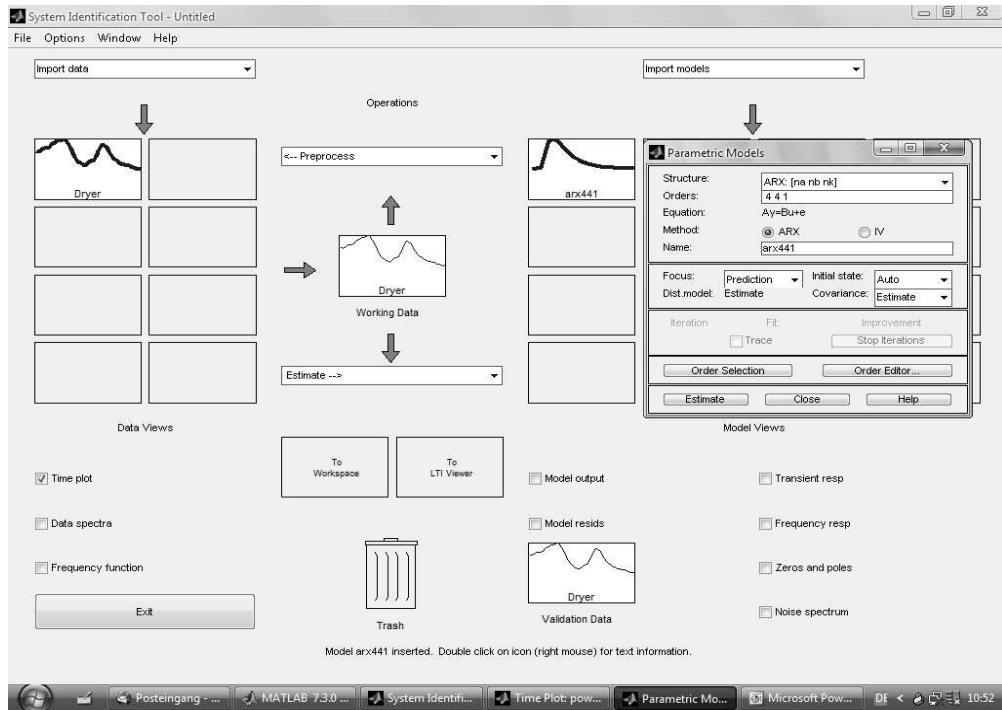


Fig. 68: Matlab system identification tool

10 Non-linear systems

In mathematics, a non-linear system is a system which is not linear i.e. a system which does not satisfy the superposition principle. Less technically, a non-linear system is any problem where the variable(s) to be solved for cannot be written as a linear sum of independent components. A non-homogeneous system, which is linear apart from the presence of a function of the independent variables, is non-linear according to a strict definition, but such systems are usually studied alongside linear systems because they can be transformed to a linear system as long as a particular solution is known. Generally, non-linear problems are difficult (if possible) to solve and are much less understandable than linear problems. Even if not exactly solvable, the outcome of a linear problem is rather predictable, while the outcome of a non-linear is inherently not.

Properties of non-linear systems: Some properties of non-linear dynamic systems are:

- They do not follow the principle of **superposition** (linearity and homogeneity).
- They may have multiple isolated equilibrium points (**linear systems** can have only one).
- They may exhibit properties such as **limit-cycle**, **bifurcation**, **chaos**

- Finite escape time: The state of an **unstable** non-linear system can go to infinity in finite time.
- For a **sinusoidal** input, the output signal may contain many harmonics and sub-harmonics with various amplitudes and phase differences (a linear system's output will only contain the sinusoid at the output).

Analysis and control of non-linear systems: There are several well-developed techniques for analysing non-linear feedback systems:

- Describing function method
- Phase plane method
- Lyapunov stability analysis
- Singular perturbation method
- Popov criterion (described in The Luré Problem below)
- Centre manifold theorem
- Small-gain theorem
- Passivity analysis

Control design of non-linear systems: Control design techniques for non-linear systems also exist. These can be subdivided into techniques which attempt to treat the system as a linear system in a limited range of operation and use (well-known) linear design techniques for each region:

- Gain scheduling
- Adaptive control

Those that attempt to introduce auxiliary non-linear feedback in such a way that the system can be treated as linear for purposes of control design:

- Feedback linearization
- Lyapunov based methods
- Lyapunov redesign
- Back-stepping
- Sliding mode control

10.1 Describing function method:

- Procedure for analysing non-linear control problems based on quasi-linearization.
- Replacement of the non-linear system by a system that is linear except for a dependence on the amplitude of the input waveform. Must be carried out for a specific family of input waveforms.
- An example might be the family of sine-wave inputs; in this case the system would be characterized by an SIDF or sine input describing function $H(A, \omega)$ giving the system response to an input consisting of a sine wave of amplitude A and frequency ω (this SIDF is a generalization of the transfer function $H(\omega)$ used to characterize linear systems).
- Other types of describing functions that have been used are DFs for level inputs and for Gaussian noise inputs. The DFs often suffice to answer specific questions about control and stability.

10.2 Phase plane method

- Refers to graphically determining the existence of limit cycles.

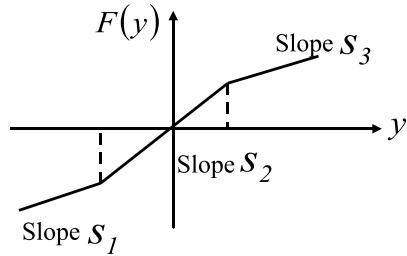


Fig. 69: Example of a piecewise linear function

- The phase plane, applicable for second-order systems only, is a plot with axes being the values of the two state variables, x_2 vs x_1 .
- Vectors representing the derivatives at representative points are drawn. With enough of these arrows in place the system behaviour over the entire plane can be visualized and limit cycles can be easily identified.

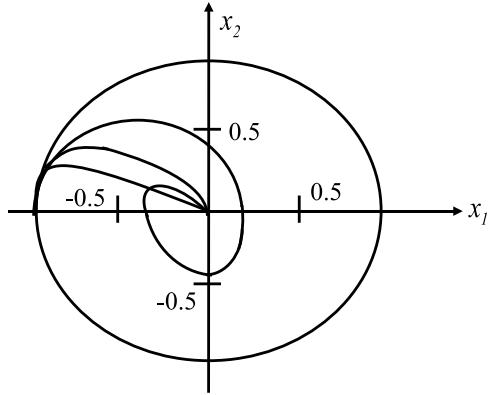


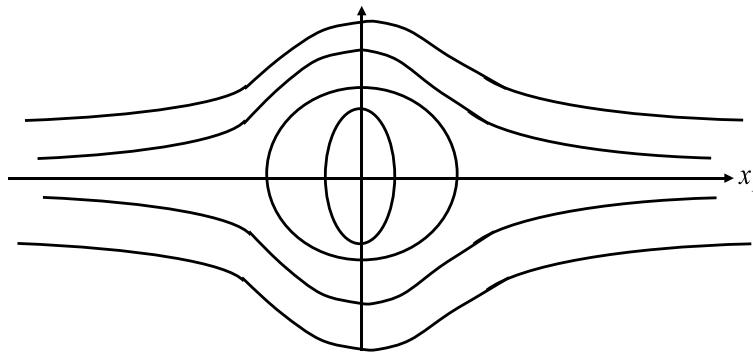
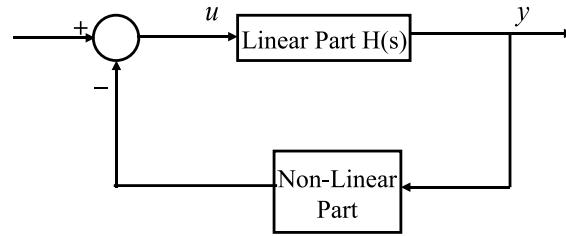
Fig. 70: Identification of limit cycles in the phase plane

10.3 Lyapunov stability

- Lyapunov stability occurs in the study of dynamical systems. In simple terms, if all solutions of the dynamical system that start out near an equilibrium point x_e stay near x_e forever, then x_e is Lyapunov stable.
- More strongly, if all solutions that start out near x_e converge to x_e , then x_e is asymptotically stable.
- The notion of exponential stability guarantees a minimal rate of decay, i.e., an estimate of how quickly the solutions converge.
- The idea of Lyapunov stability can be extended to infinite-dimensional manifolds, where it is known as **structural stability**, which concerns the behaviour of different but ‘nearby’ solutions to differential equations.

10.4 Luré’s problem

1. Control systems have a forward path that is linear and time-invariant, and a feedback path that contains a memoryless, possibly time-varying, static non-linearity.
2. The linear part can be characterized by four matrices (A, B, C, D) , while the non-linear part is $\Phi(y)$ with a sector non-linearity.

**Fig. 71:** Lyapunov stability analysis in the phase plane**Fig. 72:** The Luré problem decomposes the control problem in a linear forward path and a non-linear feedback path

3. by A.I. Luré.

10.5 Popov criterion

1. The sub-class of Luré's systems studied by **Popov** is described by:

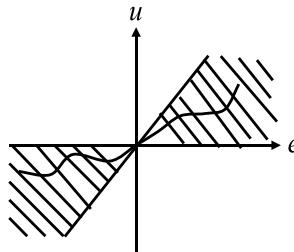
$$\dot{x} = Ax + bu, \dot{\xi} = u$$

$$y = cx + d\xi, u = -\Phi(y), \text{ where}$$

2. $x \in R^n, \xi, u, y$ are scalars and A, b, c, d have commensurate dimensions. The non-linear element $\Phi : R \rightarrow R$ is a time-invariant non-linearity belonging to open sector $(0, \infty)$. This means that

$$\Phi(0) = 0, y\Phi(y) > 0 \forall y \neq 0;$$

Popov stability criteria: →Absolute stability. The marked region in the figure is called a sector

**Fig. 73:** Stability in a sector described by the Popov inequality

$$0 \leq F(e) \leq ke \text{ for } e > 0$$

$$ke \leq F(e) \leq 0 \text{ for } e < 0$$

or shorter

$$0 \leq \frac{F(e)}{e} \leq k.$$

The standard control loop is absolute stable in sector $[0, k]$, if the closed loop only has global asymptotic stable equilibrium for any characteristic $F(e)$ lying in this sector.

Prerequisites:

$$G(s) = \frac{1}{s^p} \frac{1 + b_1 s + \dots + b_m s^m}{1 + a_1 s + \dots + a_n s^n}, \quad m < n + p.$$

No common zeros of numerators and denominators.

$F(e)$ is unique and at least steady in intervals $F(0) = 0$.

One distinguishes the main case $p = 0$, i.e. without integrator eigen value and the singular case $p \neq 0$.

Cases	p	Sector	Inequality
Ia	0	$\frac{F(e)}{e} \geq 0$	$\Re((1 + qj\omega)G(j\omega)) > 0$
Ib	0	$k \geq \frac{F(e)}{e} \geq 0$	$\Re((1 + qj\omega)G(j\omega)) > -\frac{1}{k}$
IIa	$1, \dots, n-m$	$k \geq \frac{F(e)}{e} \geq 0$	$\Re((1 + qj\omega)G(j\omega)) > -\frac{1}{k}$
IIb	$1, \dots, n-m$	$\frac{F(e)}{e} > 0$	$\Re((1 + qj\omega)G(j\omega)) \geq 0$

- The control loop is absolute stable in the given sector, when a real number q is found for all $\omega \leq 0$ apply on the right side of inequality.
- The Popov criteria is a sufficient condition for absolute stability of the control loop.

Geometric interpretation of the Popov inequality:

- For the case Ib and IIa , the inequality can be rewritten as follows

$$\begin{aligned} \Re((1 + qj\omega)G(j\omega)) &> -\frac{1}{k} \\ \Re(G(j\omega)) + q\Re(j\omega G(j\omega)) &> -\frac{1}{k} \\ \Re(G(j\omega)) + q\Re(j\omega \Re(G(j\omega))) &> -\omega \Im(G(j\omega)) > -\frac{1}{k} \\ \Re(G(j\omega)) - q\omega \Im(G(j\omega)) &> -\frac{1}{k} \longrightarrow 1 \end{aligned}$$

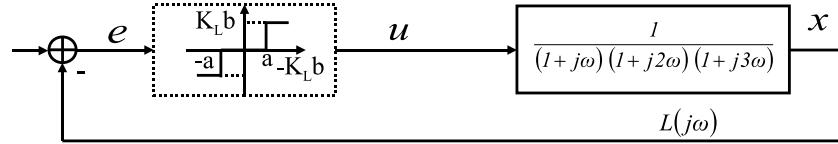
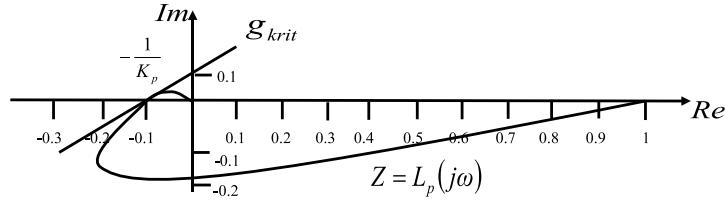
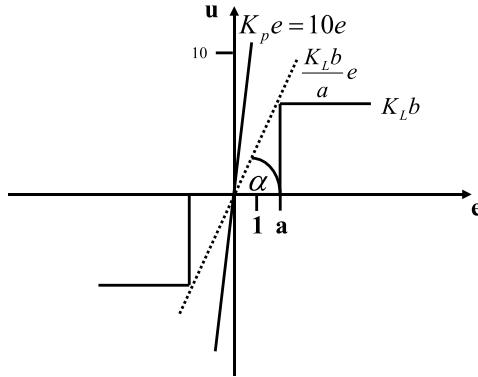
- One defines the modified frequency plot

$$G_p(j\omega) = \Re(G(j\omega)) + j\omega \Im(G(j\omega))$$

- This frequency plot is called Popov curve, so equation 1 can be represented as

$$\Im(G_p(j\omega)) < \frac{1}{q}(\Re(G_p(j\omega)) + \frac{1}{k}).$$

Example of Popov criteria: The non-linearity is shown as a three-point characteristic which already hosts the amplification K of the linear subsystem. Popov frequency response locus $Z = Lp(j\omega)$. The line g_{krit} is tangential to the Popov frequency response locus curve and intersects the real axis at 0.1.

**Fig. 74:** Example for controller non-linearity described by a three-point characteristic**Fig. 75:** Graphical determination of g_{krit} **Fig. 76:** Stability sector for the example

Popov sector is defined by the line $K_p e$. To verify Popov criteria the gradient of three-point characteristic must be smaller the calculated gradient

$$\tan \alpha = \frac{K_L b}{a}$$

$$\frac{K_L b}{a} < 10 .$$

If this is true for the given inequality, then it is idle state global asymptotic stable.

11 Non-linear control

Non-linear control is a sub-division of control engineering which deals with the control of non-linear systems. The behaviour of a non-linear system cannot be described as a linear function of the state of that system or the input variables to that system. For linear systems, there are many well-established control techniques, for example root-locus, Bode plot, Nyquist criterion, state-feedback, pole-placement etc.

11.1 Gain scheduling

Gain scheduling is an approach to control of **non-linear systems** that uses a family of **linear controllers**, each of which provides satisfactory control for a different operating point of the system.

One or more observable variables, called the scheduling variables, are used to determine what operating region the system is currently in and to enable the appropriate linear controller.

Example: In an aircraft flight control system, the altitude and Mach number might be the scheduling variables, with different linear controller parameters available (and automatically plugged into the controller) for various combinations of these two variables.

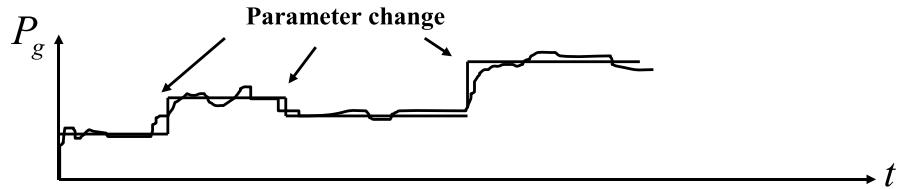


Fig. 77: Concept of gain scheduling of controller parameters based on the scheduling variable

11.2 Adaptive control

1. Modifying the controller parameters to compensate slow time \tilde{U} varying behaviour or uncertainties.
2. For example, as an aircraft flies, its mass will slowly decrease as a result of fuel consumption; we need a control law that adapts itself to such changing conditions.
3. Adaptive control does not need a priori information about the bounds on these uncertain or time-varying parameters.
4. Robust control guarantees that if the changes are within given bounds the control law need not be changed.
5. Adaptive control is precisely concerned with control law changes.

There are several broad categories of feedback adaptive control (classification can vary):

Dual adaptive controllers optimal controllers. – Suboptimal dual controllers

Non-dual adaptive controllers – Model Reference Adaptive Controllers (MRACs) [incorporate a reference model defining desired closed loop performance]

MRAC

- | | |
|------------|---|
| MIA | <ul style="list-style-type: none"> – Gradient optimization MRACs [use local rule for adjusting parameters when performance differs from reference] – Stability optimized MRACs – Model Identification Adaptive Controllers (MIACs) [perform system identification while the system is running] – Cautious adaptive controllers [use current SI to modify control law, allowing for SI uncertainty]. Non-parametric adaptive controllers – Parametric adaptive controllers – Explicit parameter adaptive controllers – Implicit parameter adaptive controllers |
|------------|---|

11.3 Feedback linearization

1. Common approach used in controlling nonlinear systems.
2. Transformation of the non-linear system into an equivalent linear system through a change of variables and a suitable control input.
3. Feedback linearization may be applied to non-linear systems of the following form:

$$\dot{x} = f(x) + g(x)u$$

$$y = h(x)$$

where x is the state vector, u is the vector of inputs, and y is the vector of outputs.

4. The goal, then, is to develop a control input u that renders either the input-output map linear, or results in a linearization of the full state of the system.

Existence of a state feedback control?

$$u = \alpha(x) + \beta(x)y$$

And variable change?

$$z = T(x)$$

Pendulum example: Task: Stabilization of origin of pendulum eq.

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = a[\sin(x_1 + \delta) - \sin(\delta)] - bx_2 + cu$$

1. Choose u as follows to cancel non-linear term:

$$u = \frac{a}{c}[\sin(x_1 + \delta) - \sin(\delta)] + \frac{v}{c}$$

From the above two equations: linear system

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = -bx_2 + v$$

2. Stabilizing feedback controller

$$v = -k_1x_1 - k_2x_2$$

Closed loop

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = k_1x_1 - (k_2 + b)x_2$$

3. Re-substituting to or feedback control law

$$u = \frac{a}{c}[\sin(x_1 + \delta) - \sin(\delta)] - \frac{1}{c}(k_1x_1 + k_2x_2)$$

11.4 Sliding mode control

In control theory sliding mode control is a type of variable structure control where the dynamics of a non-linear system is altered via application of a high-frequency switching control. This is a state feedback control scheme where the feedback is not a continuous function of time.

This control scheme involves the following two steps:

- Selection of a hyper surface or a manifold such that the system trajectory exhibits desirable behaviour when confined to this manifold.
- Finding feedback gains so that the system trajectory intersects and stays on the manifold.

Advantages:

- Controllers with switching elements are easy to realize.
- The controller outputs can be restricted to a certain region of operation.
- Can also be used inside the region of operation to enhance the closed-loop dynamics.

Two types of switching controllers

- Switching between constant values
- Switching between state or output dependent values

Design of a switching controller therefore needs:

- The definition of a switching condition
- The choice of the control law inside the areas separated by the switching law

Phase plot of a closed-loop system with sliding mode controller

$$(y = 0, \dot{y} = 1 \text{ and } m = 1) .$$

Consider a linear system

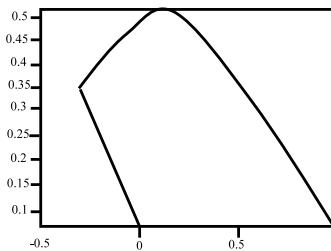


Fig. 78: Phase plot of a closed-loop system with sliding mode controller

$$\ddot{y} = u$$

The control law

$$u = -ky \text{ with } k > 0 .$$

It is clear that the controller is not able to stabilize the semi stable system that has two open-loop eigen values of zero. The root locus plot with two branches along the imaginary axis.

Switching function:

$$s(y, \dot{y}) = my + \dot{y} .$$

With a scalar $m > 0$ then a variable structure control law

$$u = -1 \ s(y, \dot{y}) > 0, 1 \ s(y, \dot{y}) < 0 .$$

Phase portrait with example: Consider a second-order system

$$\dot{x}_1 = x_2$$

$$\dot{x}_2 = h(x) + g(x)u$$

$g(x)$ and $h(x)$ are unknown non-linear functions.

On the manifolds $s = 0$ motion

$$s = a_1 x_1 + x_2 = 0 \rightarrow \dot{x} = -ax .$$

Independent of $g(x)$ and $h(x)$

Goal: state feedback \rightarrow stabilize the origin.

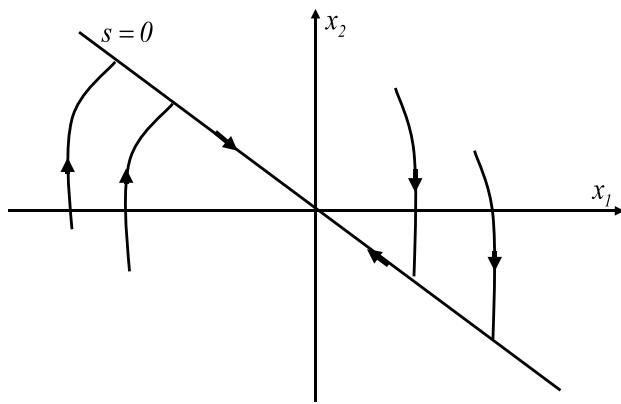


Fig. 79: Manifolds for the example

Chattering on sliding manifold

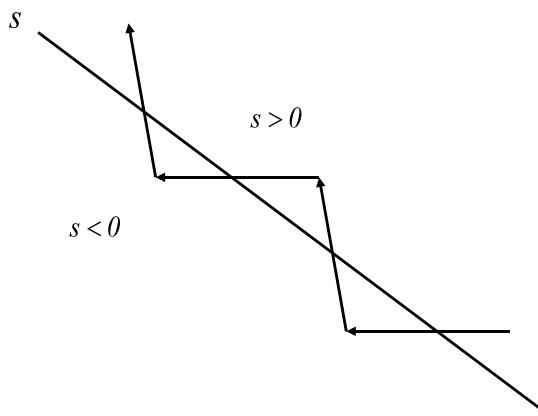


Fig. 80: Chattering on sliding manifold

- Caused by delays (no ideal would run on manifold).
- Region $s < 0 \rightarrow$ trajectory reverses
- Delay leads to crossing again

Disadvantages

- Low control accuracy
- Power losses
- Excitation of un-modelled dynamics
- Can result in instabilities.

Bibliography

- [1] C.T. Chen, *Linear System Theory and Design*, 3rd. ed. (Oxford University Press, Oxford, 1999).
- [2] H.K. Khalil, *Nonlinear Systems*, 3rd. ed. (Prentice-Hall, Upper Saddle River, NJ, 2002), (ISBN 0 – 13 – 067389 – 7).
- [3] N.S. Nise, *Control Systems Engineering*, 4th ed. (John Wiley and Sons, Inc., Chichester, 2004).
- [4] D. Hinrichsen and A.J. Pritchard, *Mathematical Systems Theory I, Modelling, State Space Analysis, Stability and Robustness* (Springer, Berlin, 2003).
- [5] E.D. Sontag, *Mathematical Control Theory: Deterministic Finite Dimensional Systems*, 2nd ed. (Springer, New York, 1998).
- [6] C. Kilian, *Modern Control Technology: Component and Systems*, 3rd ed. (Delmar Thompson Learning, 2005).
- [7] V. Bush, *Operational Circuit Analysis* (John Wiley and Sons, Inc., New York, 1929).
- [8] R.F. Stengel, *Optimal Control and Estimation* (Dover Publications, New York, 1994) (ISBN 0 – 486 – 68200 – 5, ISBN-13 : 978 – 0 – 486 – 68200 – 6).
- [9] G.F. Franklin, A. Emami-Naeini and J.D. Powell, *Feedback Control of Dynamic Systems*, 4th ed. (Prentice-Hall, Upper Saddle River, NJ, 2002) (ISBN 0 – 13 – 032393 – 4).
- [10] J.L. Hellerstein, D.M. Tilbury and S. Parekh , *Feedback Control of Computing Systems* (John Wiley and Sons, Newark, NJ, 2004).
- [11] G.E. Carlson, *Signal and Linear System Analysis with Matlab*, 2nd ed. (Wiley, Chichester, 1998).
- [12] J.G. Proakis and D.G. Mandalokis, *Digital Signal Processing Principles, Algorithms and Applications*, 3rd ed. (Prentice-Hall, Upper Saddle River, NJ, 1996).
- [13] R.E. Ziemer, W.H. Tranter and D.R. Fannin, *Signals and Systems: Continuous and Discrete*, 4th ed. (Prentice-Hall, Upper Saddle River, NJ, 1998).
- [14] B. Porat, *A Course in Digital Signal Processing* (Wiley, Chichester, 1997).
- [15] L. Ljung, *Systems Identification: Theory for the User*, 2nd ed. (Prentice-Hall, Englewood Cliffs, NJ, 1999).
- [16] Jer-Nan Juang, *Applied System Identification* (Prentice-Hall, Englewood Cliffs, NJ, 1994).
- [17] A.I. Luré and V.N. Postnikov, *On the theory of stability of control systems*, *Applied Mathematics and Mechanics*, **8** (3), 1944, in Russian.
- [18] M. Vidyasagar, *Nonlinear Systems Analysis*, 2nd ed. (Prentice-Hall, Englewood Cliffs, NJ, 1993).
- [19] A. Isidori, *Nonlinear Control Systems*, 3rd ed. (Springer, London, 1995).

From analog to digital

J. Belleman

CERN, Geneva, Switzerland

Abstract

Analog-to-digital conversion and its reverse, digital-to-analog conversion, are ubiquitous in all modern electronics, from instrumentation and telecommunication equipment to computers and entertainment. We shall explore the consequences of converting signals between the analog and digital domains and give an overview of the internal architecture and operation of a number of converter types. The importance of analog input and clock signal integrity will be explained and methods to prevent or mitigate the effects of interference will be shown. Examples will be drawn from several manufacturers' datasheets.

1 Introduction

Since the 1970s, the supremacy of numerical processing over analog signal treatment has become ever more evident. More and more functions that traditionally were in the analog realm are being replaced by digital electronic hardware. Yet before any numerical signal treatment is possible, the analog nature of most electrical signals must first be converted into a numerical representation. That task falls to an analog circuit that is not likely to disappear for some time to come: the Analog-to-Digital Converter (ADC).

The basic function of an ADC is to convert a voltage applied to its input into a number with a limited range of possible values. On top of that, it will only do so at a finite rate. Thus, it replaces the continuous input signal by a sequence of numbers, with discrete steps in both amplitude and time.

A huge variety of ADCs exist, using many different architectures and covering a wide range of resolution and speed.

1.1 Amplitude quantization

In order to make a signal suitable for treatment by numerical circuitry, it must first be represented in a numerical format, or quantized. That is, a continuous range of values is replaced by a limited set of values separated by discrete steps (Fig. 1).

Usually the number of steps is chosen to be a power of two, because that yields the most economical representation in binary digital electronics. Naturally, the quality of the approximation depends on the number of steps used to approximate the original signal.

It is customary to specify the accuracy of an ADC by comparing the power of an ideal full-scale sinusoidal input signal with its numerical representation, because it is relatively straightforward to produce a very clean sinewave signal.

Consider the example (Fig. 2), where a sinusoidal signal is compared with its numerical representation expressed in 2^n equidistant discrete steps, such that the difference between the sine and its quantized approximation is never greater than one half of the size of a step. This difference is referred to as the quantization error (ε).

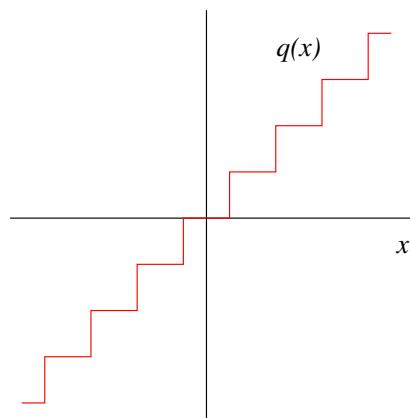


Fig. 1: Quantization function

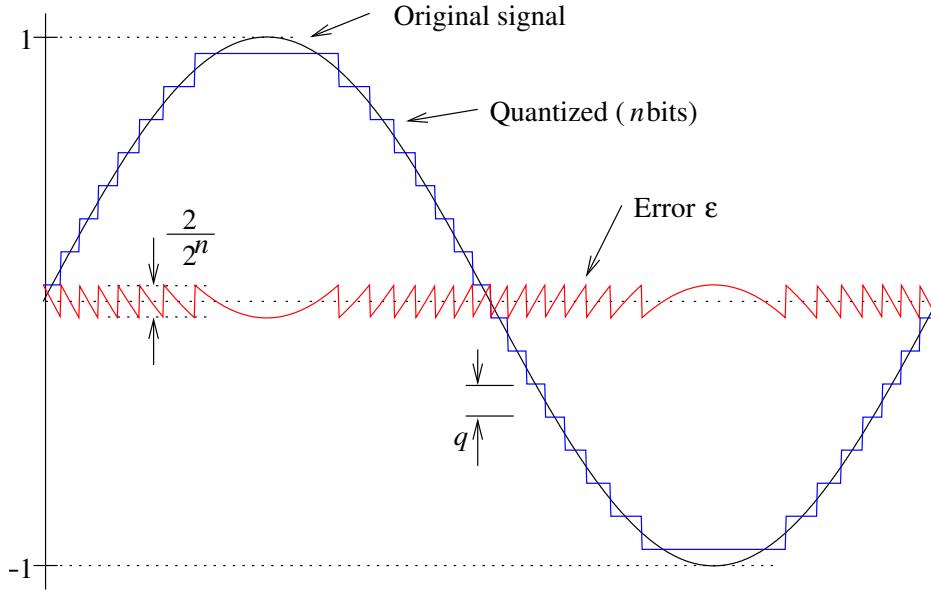


Fig. 2: Original signal, quantized signal, and quantization error

The mean power in a sinusoidal input signal with unit amplitude is simply

$$P_s = \frac{1}{T} \int_0^T \sin^2 \omega t dt = \frac{1}{2}. \quad (1)$$

Quantized to 2^n discrete levels, equivalent to n bits, one quantization step q has a size of

$$q = \frac{2}{2^n} = 2^{-(n-1)}. \quad (2)$$

We have arranged things so that the quantization error has a maximum value of plus or minus half the step size, so

$$|\varepsilon| \leq \frac{q}{2} \text{ and therefore, } |\varepsilon| \leq 2^{-n}. \quad (3)$$

For a large enough number of quantization steps, the probability density function of the quantization error tends toward being flat [1]. Its value within the quantization error bounds is

$$p(\varepsilon) \approx \frac{1}{q}. \quad (4)$$

So we can calculate its mean power or variance as the 2nd moment of its distribution:

$$P_\varepsilon = \int_{-q/2}^{q/2} p(\varepsilon) \varepsilon^2 d\varepsilon \approx \frac{2^{-2n}}{3}. \quad (5)$$

Thus, an ideal ADC would have a signal-to-noise ratio

$$\text{SNR} = \frac{P_s}{P_\varepsilon} = 1.5 \cdot 2^{2n}, \quad (6)$$

or, expressed in decibels, $1.76 + 6.02n$ dB¹. This is an expression that appears in many texts on ADCs, and it is the *best* an n -bit ADC can do. It serves to set a standard against which to compare the performance of a real ADC. Usually, a real ADC performs substantially worse than that.

¹Decibels: See Appendix A.

By measuring the actual signal-to-noise ratio of an ADC, and solving for n in Eq. (6), one can determine the Effective Number Of Bits (ENOB) of an ADC. This number is also often specified in ADC datasheets.

The quantization error is a distortion of the original signal. It is an irreversible loss of detail. Contrary to analog circuitry, in which distortion usually decreases for lower signal levels, the quantization error is comparatively worse for small signals. It is therefore important to scale the signal amplitude so as to fill the ADC range as completely as possible without running into saturation.

1.2 Quantization in the time domain

An ADC delivers values that correspond to discrete instants in time. Provided the sampling rate is at least twice the bandwidth of the signal and neglecting, for the moment, the effect of the finite resolution of the amplitude quantization, the information contained in the sample stream is sufficient to restore the original signal without loss of information. This was worked out in the first half of the twentieth century by E. Whittaker, V. Kotelnikov, H. Nyquist, R. Hartley and C. Shannon [2]–[6].

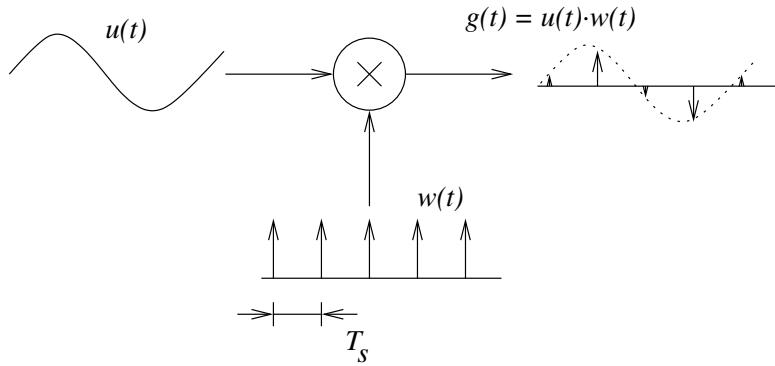


Fig. 3: The sampling process

Conceptually, sampling is a modulation process (Fig. 3). The input waveform $u(t)$ is multiplied by the modulating or sampling waveform $w(t)$. The modulating waveform is a sequence of equally spaced Dirac δ impulses:

$$w(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT_s) \quad (7)$$

where T_s is the sampling period. Its reciprocal $1/T_s$ is the sampling frequency F_s . The output of the modulation process, $g(t)$, consists of a sequence of impulses with varying ‘amplitude’, according to the value of the input signal at each sampling instant. The value $u(t)$ taken in between the sampling instants does not affect the result. Only the values at the sampling instants matter.

$$g(t) = \sum_{n=-\infty}^{\infty} u(t) \cdot \delta(t - nT_s) = \sum_{n=-\infty}^{\infty} u(nT_s) \cdot \delta(t - nT_s). \quad (8)$$

Let us examine the properties of $g(t)$ in the frequency domain. To do so, we shall first derive the frequency-domain representation $W(f)$ of the modulating waveform $w(t)$:

$$W(f) = \mathcal{F}\{w(t)\} = \int_{-\infty}^{\infty} w(t) e^{-j2\pi ft} dt = \int_{-\infty}^{\infty} \sum_{n=-\infty}^{\infty} \delta(t - nT_s) e^{-j2\pi ft} dt. \quad (9)$$

Integrating each term of the series separately, we obtain

$$W(f) = \sum_{n=-\infty}^{\infty} e^{-j2\pi n f T_s} = 1 + 2 \sum_{n=1}^{\infty} \cos 2\pi n f T_s = \sum_{n=-\infty}^{\infty} \delta(f - n F_s). \quad (10)$$

This shows three different perspectives of the same expression. The spectrum of the sampling waveform turns out to be a repetition of spectral lines at multiples of the sampling frequency F_s .

In order to get the spectrum of the sampled signal $G(f)$, we can apply convolution:

$$G(f) = U(f) * W(f) \quad (11)$$

$$= \int_{-\infty}^{\infty} U(\phi) W(f - \phi) d\phi \quad (12)$$

$$= \int_{-\infty}^{\infty} U(\phi) \sum_{n=-\infty}^{\infty} \delta(f - n F_s - \phi) d\phi \quad (13)$$

$$= \sum_{n=-\infty}^{\infty} \int_{-\infty}^{\infty} U(\phi) \cdot \delta(f - n F_s - \phi) d\phi. \quad (14)$$

$$G(f) = \sum_{n=-\infty}^{\infty} U(f - n F_s). \quad (15)$$

Equation (15) shows that the spectrum of the original signal is repeated at all harmonics of the sampling waveform (Fig. 4). If the spectrum of the original signal $U(f)$ extends beyond $F_s/2$, adjacent sidebands would overlap and it would no longer be possible to tell to which image a given frequency in the spectrum belongs. Adjacent images get mixed up inseparably. This limit is known as the Nyquist criterion [4], [6]. We shall subsequently refer to $F_s/2$ as the *Nyquist frequency* F_N .

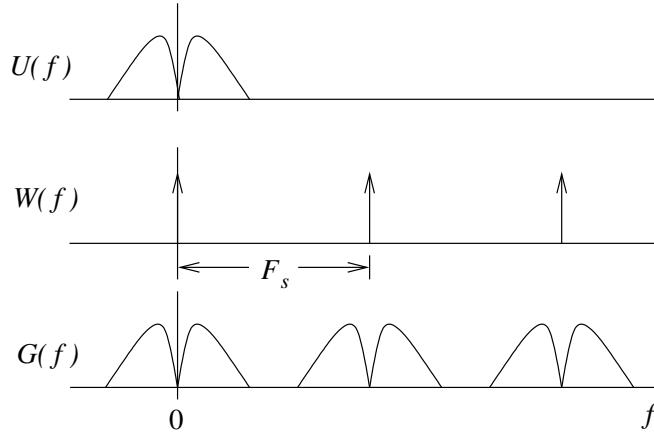
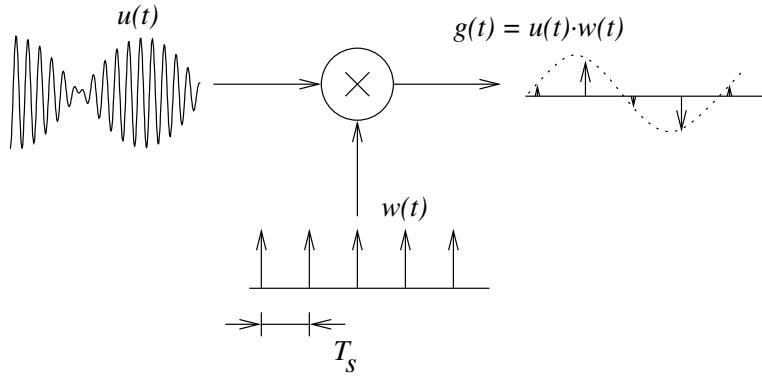


Fig. 4: Spectrum of a sampled signal

Note that Nyquist's criterion sets a limit on the *bandwidth* of a signal to be sampled; It says nothing of its frequency. Indeed, it is quite possible to sample a band-limited signal at a rate much slower than its actual frequency, while still keeping the spectral images separate. Consider for example a signal as depicted by $u(t)$ in Fig. 5. This signal has frequencies greater than the sampling rate. Its spectrum might look like $U(f)$ in Fig. 6. Let us describe it as frequency shifted by $m \cdot F_s$, with integer m . To get the spectrum of the sampled signal, we again apply convolution with the spectrum of the sampling waveform, as in Eq. (12), replacing $U(f)$ with $U(f + m F_s)$:

$$G(f) = U(f + m F_s) * W(f) = \int_{-\infty}^{\infty} U(\phi + m F_s) W(f - \phi) d\phi. \quad (16)$$

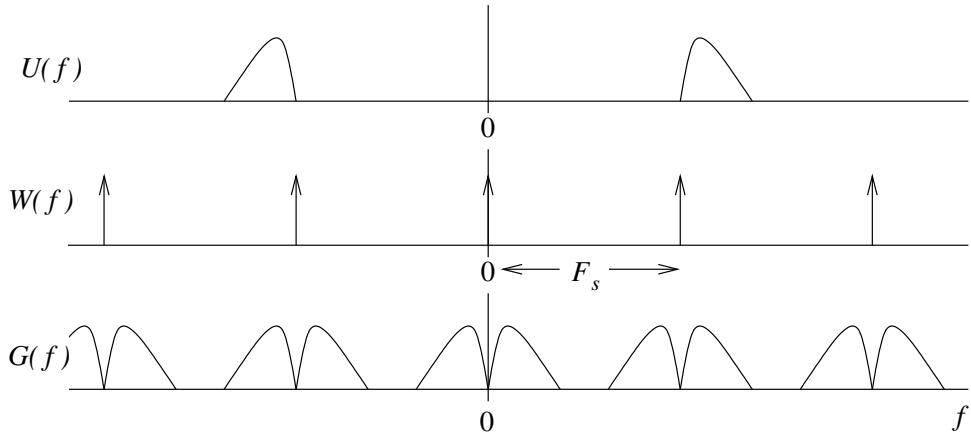
**Fig. 5:** Sub-sampling

$$G(f) = \sum_{n=-\infty}^{\infty} U(f + (m - n)F_s). \quad (17)$$

Since the sum over n runs from $-\infty$ to ∞ , we may add an arbitrary integer constant to n without affecting the value of the result, so let us choose $-m$. The resulting expression for the spectrum of the sampled signal is

$$G(f) = \sum_{n=-\infty}^{\infty} U(f - nF_s), \quad (18)$$

which is exactly the same as in Eq. (15), above.

**Fig. 6:** Spectra with sub-sampling

The spectrum of the sampled signal does not change if the sampled signal is shifted by some integer times F_s . The practice of sampling a signal at a sampling frequency below the frequencies contained in the signal is called *sub-sampling*. Provided the signal bandwidth is less than half the sampling frequency, it is possible to reconstruct the original signal without loss of information. All the information of the signal is present in every spectral image. The multiple copying of frequencies in the spectrum of a sampled signal is known as *aliasing*.

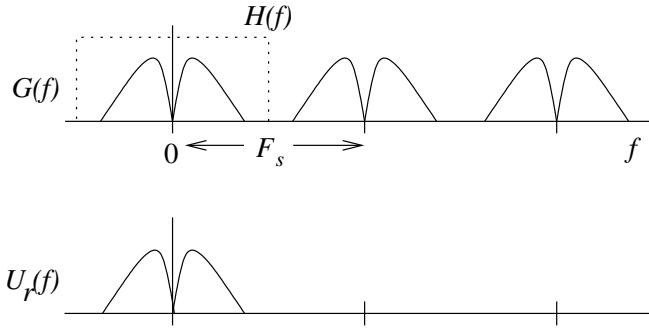


Fig. 7: Signal reconstruction by filtering

1.3 Signal reconstruction

To recover the original continuous signal, conceptually, we can filter the sampled signal using a filter $H(f)$ with a rectangular passband. In Fig. 7, we have chosen the baseband, which is the image immediately around $f = 0$, so $H(f) = 1$ for $-F_s/2 < f < F_s/2$ and zero elsewhere. Filtering is a multiplication in the frequency domain, and a convolution in the time domain. So, to obtain the expression for the reconstructed signal $u_r(t)$, we first get the inverse Fourier transform of the filter function $H(f)$ and then convolve that with the expression for the sample sequence:

$$h(t) = \mathcal{F}^{-1}\{H(f)\} = \int_{-\infty}^{\infty} H(f) e^{j2\pi ft} df = \int_{-F_s/2}^{F_s/2} e^{j2\pi ft} df = F_s \frac{\sin \pi F_s t}{\pi F_s t}, \quad (19)$$

$$u_r(t) = g(t) * h(t) \quad (20)$$

$$= \int_{-\infty}^{\infty} g(\tau) \cdot h(t - \tau) d\tau \quad (21)$$

$$= \int_{-\infty}^{\infty} \sum_{n=-\infty}^{\infty} u(\tau) \cdot \delta(\tau - nT_s) \cdot h(t - \tau) d\tau \quad (22)$$

$$= \sum_{n=-\infty}^{\infty} u(nT_s) \cdot h(t - nT_s) \quad (23)$$

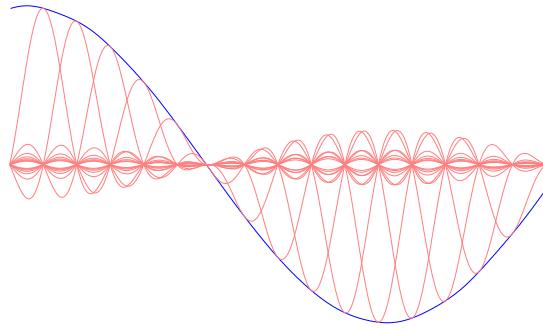
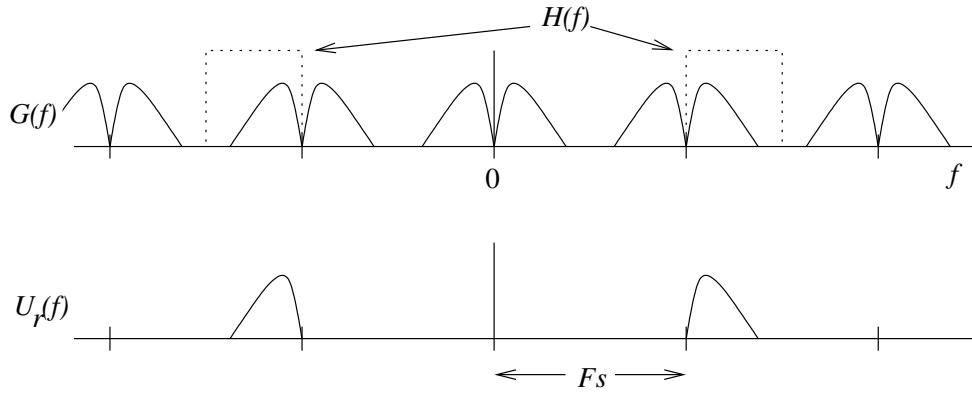
$$u_r(t) = \sum_{n=-\infty}^{\infty} u(nT_s) \cdot F_s \frac{\sin \pi F_s (t - nT_s)}{\pi F_s (t - nT_s)}. \quad (24)$$

The form $\sin(x)/x$ is the *cardinal sine* of x . It appears very often in signal processing mathematics and is also written as $\text{sinc}(x)$. We can thus reconstruct the signal exactly by summing an infinite series of sinc functions, weighted and displaced according to each sample. (Fig. 8). This method is attributed to Kotelnikov [3] and Shannon [6].

It is also possible to reconstruct a signal from one of the other spectral images, thus obtaining a frequency translation of the signal by some integer multiple of F_s (Fig. 9). This is termed frequency conversion and is useful in, for example, digital receivers.

Let us look at one case in some more detail. The reconstruction filter is shifted by just F_s . The filter is then described by

$$H(f) = 1 \text{ for } F_s < |f| < \frac{3}{2}F_s \text{ and zero elsewhere.} \quad (25)$$

**Fig. 8:** Reconstruction of a continuous signal by summing sinc functions**Fig. 9:** Signal reconstruction from a different spectral image

The inverse Fourier transform of the filter function gives its time-domain representation:

$$h(t) = \mathcal{F}^{-1}\{H(f)\} \quad (26)$$

$$= \int_{-\infty}^{\infty} H(f) e^{j2\pi ft} df \quad (27)$$

$$= \int_{-\frac{3}{2}F_s}^{\frac{3}{2}F_s} e^{j2\pi ft} df - \int_{F_s}^{F_s} e^{j2\pi ft} df \quad (28)$$

$$h(t) = 3F_s \text{sinc}(3\pi t F_s) - 2F_s \text{sinc}(2\pi t F_s) . \quad (29)$$

Convolution of Eq. (29) with the time-domain representation of the sampled signal will then yield the reconstructed function $u_r(t)$, which is now centred on F_s rather than on $f = 0$.

$$u_r(t) = g(t) * h(t) . \quad (30)$$

$$u_r(t) = \sum_{n=-\infty}^{\infty} u(nT_s) \{3F_s \text{sinc}(3\pi(t - nT_s)F_s) - 2F_s \text{sinc}(2\pi(t - nT_s)F_s)\} . \quad (31)$$

The reconstructed signal consists of the sum of two infinite series of sinc functions, weighted and displaced according to the successive samples. The resultant waveform is illustrated in Fig. 10.

In practice, signal reconstruction is never done that way. Dirac impulses and filters with rectangular transfer functions are mathematical abstractions, and even if they could be realized to a sufficient

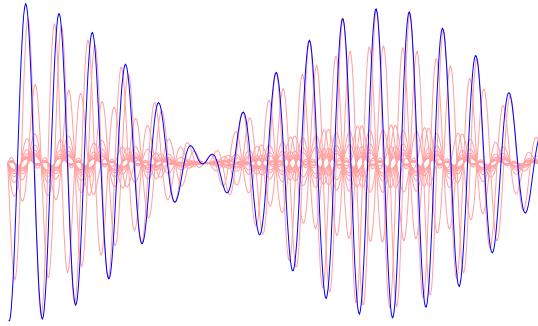


Fig. 10: Reconstruction with frequency conversion

degree of accuracy, each term of the series would extend far to both the past and the future, which is clearly impractical. Instead, each sample is held for a full sampling period. At that stage, the reconstructed signal looks like a staircase approximation of the reconstructed signal (Fig. 11). This sequence of rectangular steps is then smoothed by filtering it with a polynomial filter.

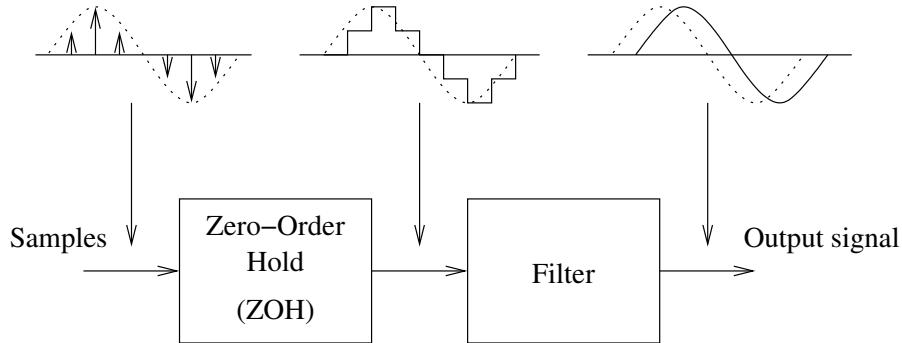


Fig. 11: Signal reconstruction with staircase approximation

The expression for the staircase approximation is

$$u_r(t) = \sum_{n=-\infty}^{\infty} g(nT_s) \cdot \square(t - nT_s), \quad (32)$$

where $\square(t)$ is the rectangle function, of unit value for $0 < t < T_s$ and zero elsewhere (Fig. 12).

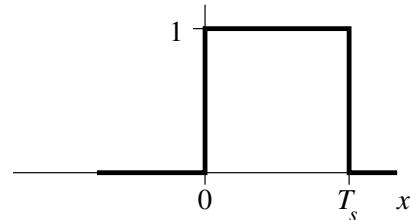


Fig. 12: The rectangle function $\square(t)$

As a consequence of the staircase approximation, the spectrum of the reconstructed signal is distorted, convolved by the spectrum of the rectangular pulses of length T_s :

$$\square(f) = \mathcal{F}\{\square(t)\} = \int_{-\infty}^{\infty} \square(t) e^{-2j\pi f t} dt \quad (33)$$

$$= \int_0^{T_s} e^{-2j\pi f t} dt \quad (34)$$

$$= T_s e^{-j\pi f T_s} \cdot \text{sinc}(\pi f T_s) \quad (35)$$

$$\square(f) = \frac{1 - e^{-j2\pi f T_s}}{-j2\pi f}. \quad (36)$$

Since this function drops off slowly in the frequency domain (Fig. 13), frequencies from adjacent spectral images will leak into the reconstructed signal, so the polynomial filter following it is necessary to suppress them. Incidentally, this function is also known as the Zero-Order Hold (ZOH), because it holds its value constant during one sampling period. (The Laplace domain expression of the ZOH is readily found by substituting $s = j2\pi f$ in Eq. (36) above.) Higher order hold functions, implementing linear, parabolic, or even higher order interpolation between adjacent samples exist and offer improved reconstruction accuracy, albeit with increased delay and a considerable increase in complexity. In practice, higher order hold functions are never used.

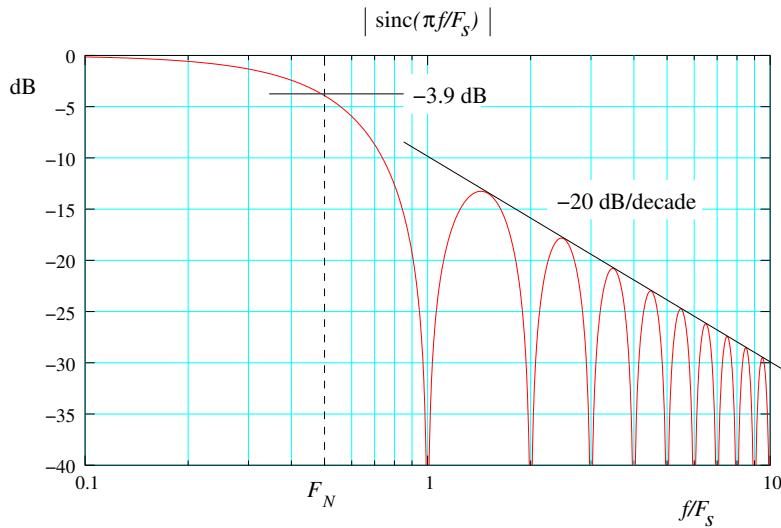


Fig. 13: Frequency response of a ZOH

The polynomial filter can be designed to compensate for the $\text{sinc}(f)$ response of the staircase approximation. As an alternative, you may apply the compensation in the digital domain, before the DAC, and use a simpler flat filter. As a rule, the filter is some compromise between performance, cost, delay, and out-of-band signal rejection. While it is possible to use one of the harmonic bands to reconstruct a frequency-shifted signal, the frequency response of the ZOH drops off too fast to make this an interesting option.

1.4 Spectrum of the quantization error

It is clear from Fig. 2 that even though the quantization error is deterministic, it has a spectrum that bears little relation to $u(t)$, the signal being digitized. It does not yield so easily to analysis though [7] [8] [9]. A numerical Fourier transform shows that if the signal is busy enough, and the number of bits great enough, the quantization noise power (Eq. 5) is basically evenly distributed over the full spectrum from 0 to F_N , thus appearing as white noise. (Frequencies beyond the Nyquist rate are aliased back into the range 0 to F_N and the spectrum repeats every integer multiple of F_s .) Compared to the level of a full-scale sinusoidal test signal, the noise floor ends up at

$$-\left(1.76 + 6.02n + 10 \log_{10} \frac{F_s}{2}\right) \text{ dBFS/Hz}, \quad (37)$$

with n the number of bits of the ADC. Usually the noise floor will be determined from a discrete Fourier transform of a block of samples. It is then more convenient to express the noise floor in dBFS/bin, because that can be read straight off the plot. Equation (37) is modified accordingly:

$$-\left(1.76 + 6.02n + 10 \log_{10} \frac{N}{2}\right) \text{ dBFS/bin ,} \quad (38)$$

with N the number of samples over which the Fourier transform is taken. Figure 14 shows such a plot, the result of an 8 k sample long recording of a full-scale sinewave.

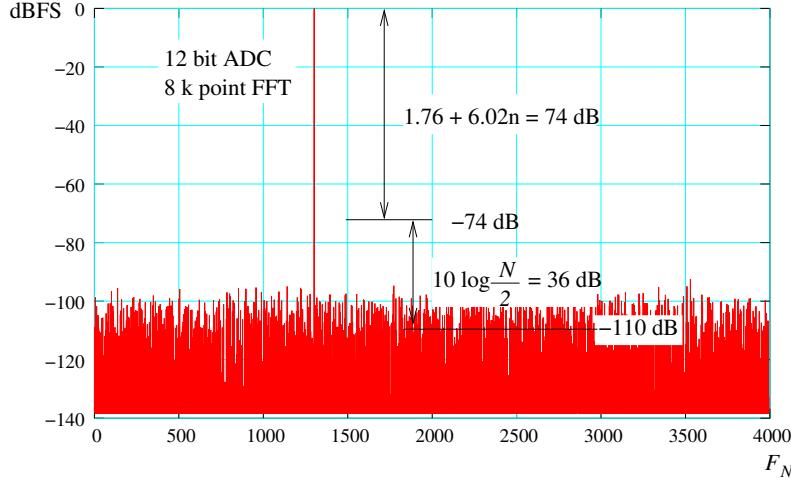


Fig. 14: Discrete Fourier transform of 8 k samples of ADC data

Unfortunately, the quantization noise does not always appear as white noise. Even for a hypothetically perfect ADC, for simple ratios between F_s and the frequency components of $u(t)$, some of the quantization noise power concentrates at discrete frequencies to form spurious signals or spurs. Non-linearities in actual ADCs contribute harmonics of the frequencies in $u(t)$, which look like still more of these spurs (Fig. 15).

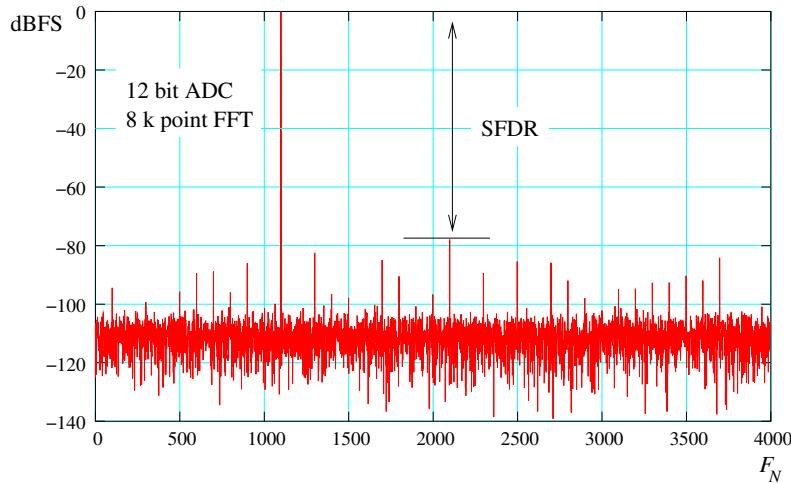


Fig. 15: Spurious components in the spectrum of an ADC

Spurs can be made less objectionable by applying *dither*, an intentional injection of low-level additive noise. This will spread out the energy of a spur over a greater frequency range, without seriously affecting the desired signal. If the spectrum of the dither is chosen so that it does not overlap with the spectrum of the desired signal, the loss of SNR can be made all but negligible.

Datasheets for ADCs used in signal processing will specify a number for the *Spurious-Free Dynamic Range* (SFDR) in dBc, which is the ratio in dB between the greatest spur and the applied test signal, or in dBFS, which is the ratio with respect to a full-scale signal. This is an important specification for digital radio receivers, for example, because a spur may hide or interfere with a weak neighbouring signal. Manufacturers are careful to avoid the problematic frequency ratios mentioned above when specifying the performance of their converters. In practice, the SFDR is measured using a near full-scale sinusoid and applying a Fourier transform to a recording of a few thousand samples. The sine wave must be spectrally clean, with harmonics and phase noise below the quantization noise floor. Even though clean sine waves are the simplest possible test signals, ADCs are getting so good nowadays that this is by no means easy (Fig. 16). (You may end up measuring the quality of your signal generator instead.)

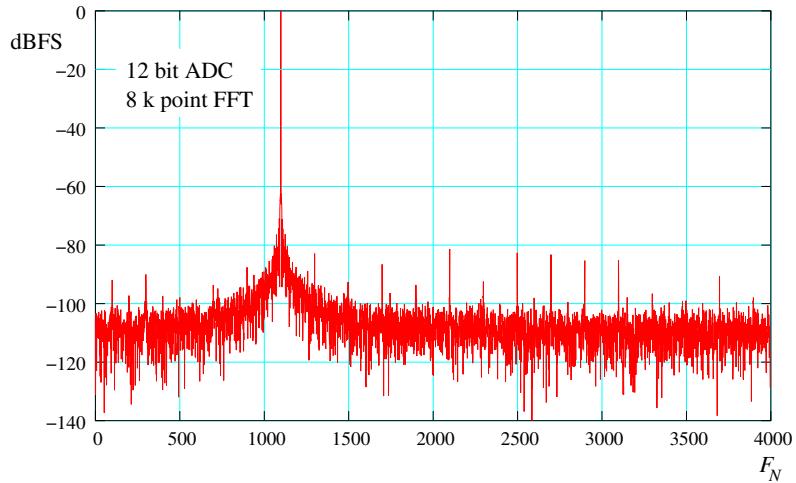


Fig. 16: The signature of excessive $1/f$ phase noise

The total power in the quantization noise depends on the number of bits of the converter, as determined previously (Eq. 6). By taking samples faster than required by the Nyquist criterion, the quantization noise is spread over a larger bandwidth. If we then pass the sample stream through a numerical low-pass filter, the noise beyond the signal bandwidth is removed and the SNR improved (Fig. 17). This

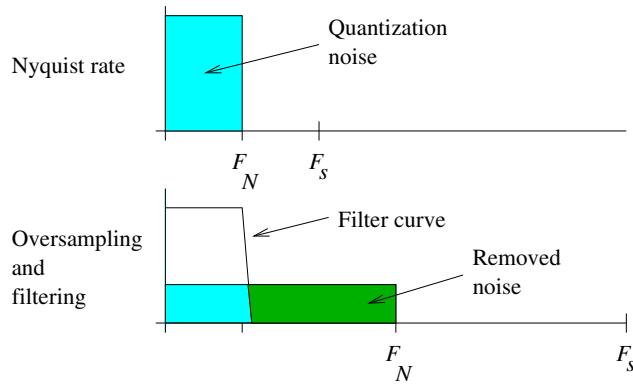


Fig. 17: Oversampling improves SNR

is termed *oversampling*. The improvement is quite modest however: Only 3 dB for every doubling of the sampling rate, or equivalently, one bit for every factor of four increase. For DC inputs and very quiet ADCs, dither may be necessary to make this work at all. It is possible to do much better, as demonstrated by $\Sigma\Delta$ ADCs.

1.5 Gain and offset errors

The gain error is the ratio between the nominal and the actual full-scale value of the ADC. The offset error is the difference between the nominal and the actual value at zero input (Fig. 18). Exact ways of defining these values differ from one manufacturer to another, or from one ADC type to another.

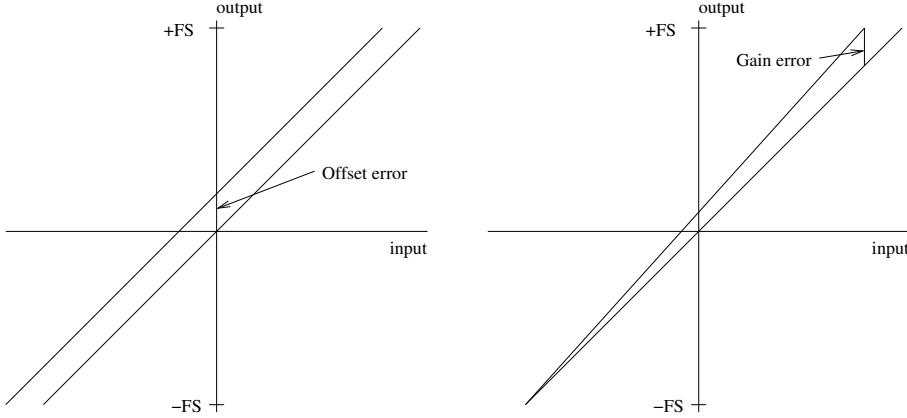


Fig. 18: Offset and gain error definitions

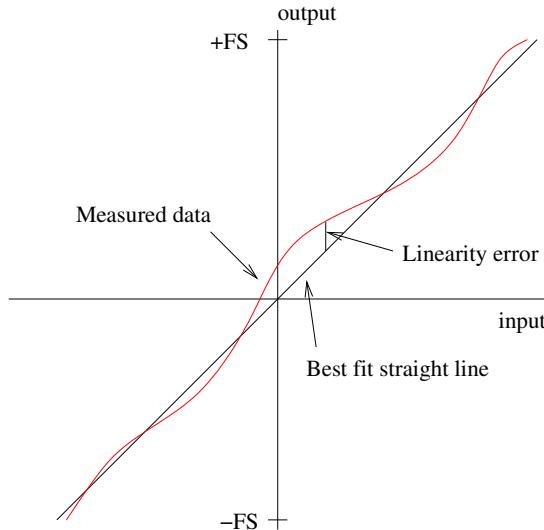
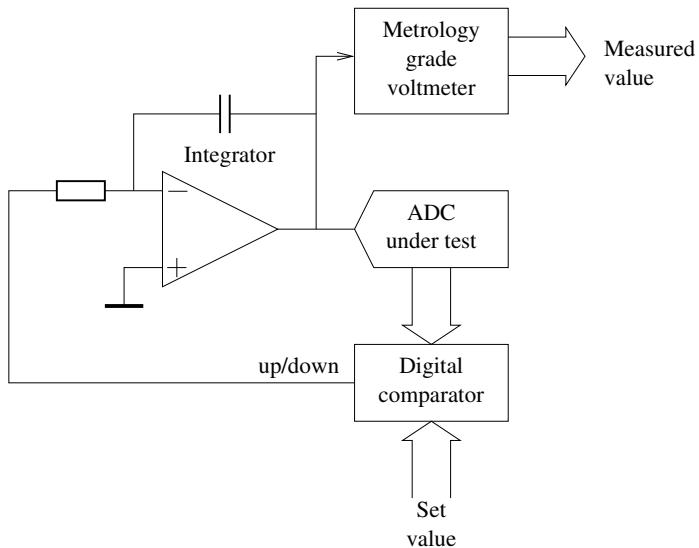
1.6 Integral and differential non-linearity, SINAD

The *Integral Non-Linearity* (INL) is measured by comparing a set of measurements distributed over the whole range of the ADC with the linear regression through all points of the set. (Some manufacturers specify the deviation with respect to a straight line through the end points, Fig. 19. Read the datasheets.) Note that this measurement ignores gain and offset errors. The measurement setup used is something along the lines of Fig. 20. The digital comparator will drive a slow analog integrator to make the output from the ADC equal to the set value. A precision voltmeter then gives the corresponding ADC input value. For high-resolution ADCs, the principal difficulty lies in avoiding errors due to thermocouple effects. The shape of the linearity error curve yields information about the expected harmonic distortion products. For example, a parabolic component will yield a 2nd harmonic. It is customary for manufacturers to quote the SNR of their products excluding the contribution of the first five harmonics. With all noise, harmonics and other spurs included, the performance measure is called SINAD. The abbreviation stands for SIgnal to Noise And Distortion.

The *Differential Non-Linearity* (DNL) measures the span of input values over which each possible digitized value occurs. In other words, it measures the width of each step of the amplitude quantization function. It is usually measured by histogramming converted values over the full range of the ADC, using an input signal with a flat (or at least with a *known*) distribution. This characteristic is important for ADCs used in closed-loop feedback systems and in spectrography applications. A poor DNL translates into greater than ideal quantization noise. Linearity errors can get bad enough to result in missing codes, or non-monotonicity, meaning that the converter output may actually go *down* for an increase in input signal at some places.

1.7 Clock jitter

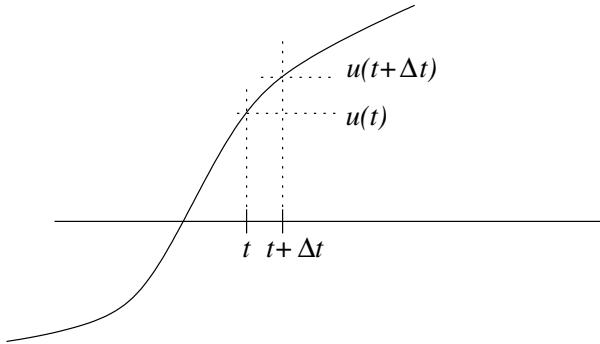
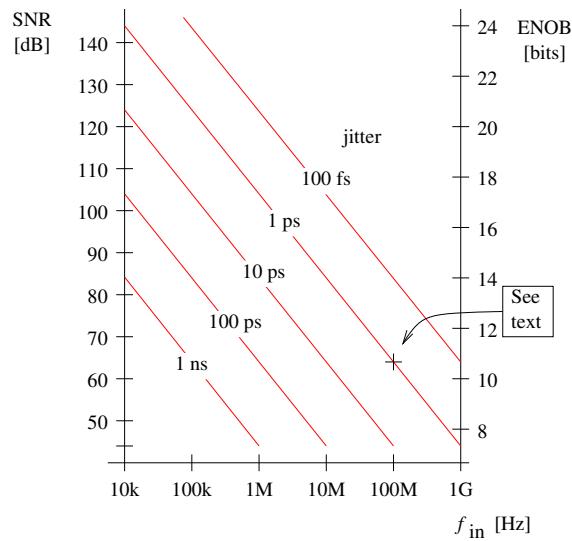
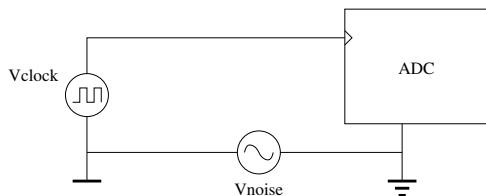
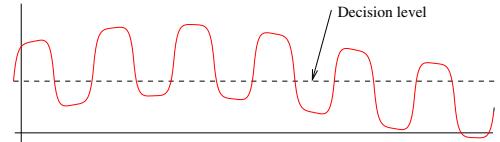
ADCs used in signal processing applications need a stable, clean, clock signal. The effects of clock jitter do not depend on the sampling rate of the ADC, but rather on the rate of change of its input signal. With reference to Fig. 21, suppose an ADC digitizes an input signal $u(t)$. Let us assume that the sampling clock is in error by an amount Δt : Then the input sample is off by an amount

**Fig. 19:** Integral non-linearity error**Fig. 20:** Set-up to measure integral non-linearity

$$\Delta U = \frac{du(t)}{dt} \cdot \Delta t . \quad (39)$$

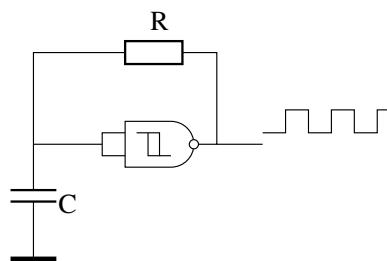
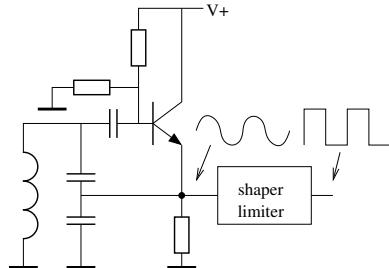
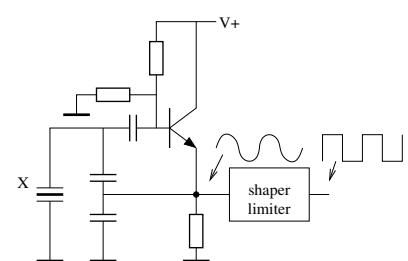
This quickly leads to surprisingly stringent demands on the jitter performance of the ADC conversion clock. As a point of example, consider an ADC digitizing a 100 MHz sine, driven by a clock source with the quite respectable jitter specification of 1 ps_{rms}: Its effective resolution would be limited to about 10.5 bits at best (Fig. 22).

In the light of these considerations, it is evident that the clock of high-performance ADCs should be generated using a high-quality oscillator, mounted close to the ADC and connected to the same ground plane as the ADC (Fig. 23). Owing to the finite rise time of the clock waveform, superimposed ground or power supply noise would deteriorate the jitter specifications of an otherwise good clock signal (Fig. 24). By the same relation as Eq. (39), we can see that only a few millivolts of ground noise on a clock signal with 1 ns rise and fall times are enough to introduce more than 1 ps of jitter. One way to alleviate this might be to use a differential clock signal.

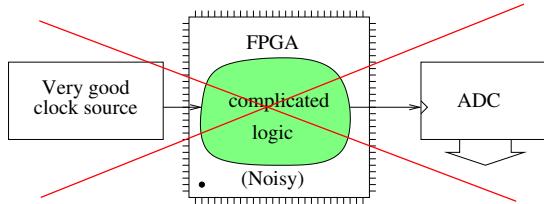
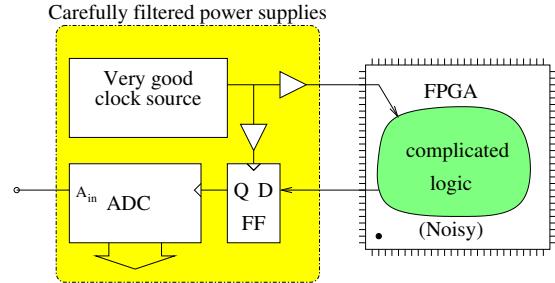
**Fig. 21:** The effect of clock jitter**Fig. 22:** ADC resolution vs. input frequency and clock jitter**Fig. 23:** Ground does not have the same potential throughout**Fig. 24:** Ground noise affects clock jitter

Some simple rules can be given to get a good clock signal. First, use a good oscillator. RC oscillators or logic gate oscillators, like the one in Fig. 25, have poor performance, with jitter usually worse than $100 \text{ ps}_{\text{rms}}$. Colpitts type LC oscillators, as in Fig. 26, are much better, with around $10 \text{ ps}_{\text{rms}}$ of jitter. A Pierce oscillator, see Fig. 27, with quartz or ceramic resonator can be quite good, with jitter at $1 \text{ ps}_{\text{rms}}$ or below. Getting jitter much below 1 ps is an exercise in low-noise electronic design and is an art all by itself.

It is all too easy to deteriorate a good clock by simple lack of care. In particular, do not feed the ADC from a clock taken from or via a nearby DSP or FPGA (Fig. 28), unless re-synchronized using a flipflop driven by the original, clean clock (Fig. 29). Do not use any remaining logic gates in the same package that buffers the clock signal for other purposes. Carefully filter the power supply of the

**Fig. 25:** An RC oscillator**Fig. 26:** A Colpitts oscillator**Fig. 27:** A Pierce oscillator

clock signal generation and distribution circuitry. Digital logic usually does not have very good isolation between gates in the same package, nor does it reject any power line noise: It has not been designed with that purpose in mind, because it usually does not need it. Crosstalk between different sections of a logic circuit can seriously deteriorate a clock signal. The clock signal should be kept clear of both the analog input and the digital outputs of the ADC. Even though the clock is a digital square wave, it should be treated with all the care that should be given to an analog signal.

**Fig. 28:** Do not do this!**Fig. 29:** But if you really must, do it like this

The clock jitter is only a single figure to capture the behaviour of the sampler clock, exactly like the standard deviation is only a single parameter of a distribution. It tells you nothing about the shape of the distribution, nor does it give any information about its spectrum.

More generally, timing jitter is the consequence of oscillator phase noise, the collective effects of various noise sources inherent in their circuitry. Resistors contribute thermal noise and semiconductor devices add shot noise and $1/f$ noise. This noise modulates the generated output tone. Thus, its spectrum acquires sidebands that, for simple oscillators, have a $1/f$ slope for small offsets from the carrier, and that are flat farther out (Fig. 30). Synthesized frequency sources can deviate quite strongly from that simple model. In addition, they may have discrete spurious sidebands in their spectrum. Many manufacturers of oscillators publish phase noise measurements for their products. These may take the form of actual phase noise plots, or of specifications along the lines of: “ -150 dBc at 100 Hz offset from the carrier”.

ADCs are clocked devices. They do not directly respond to phase variations, but only to variations in timing of the effective logic threshold. For the sake of argument, let us assume that the clocking waveform is a sinusoid and that the significant instants are the positive-going zero crossings. That approximation is quite accurate for high-speed ADCs with differential clock inputs:

$$V(t) = \sin(2\pi F_s t + \varphi(t)), \quad (40)$$

where $\varphi(t)$ models the random phase variations. Two different zero crossings at, say, times t_1 and t_2 , about N periods apart, satisfy the conditions

$$2\pi F_s t_1 + \varphi(t_1) = 0 \text{ and } 2\pi F_s t_2 + \varphi(t_2) = 2\pi N \quad (41)$$

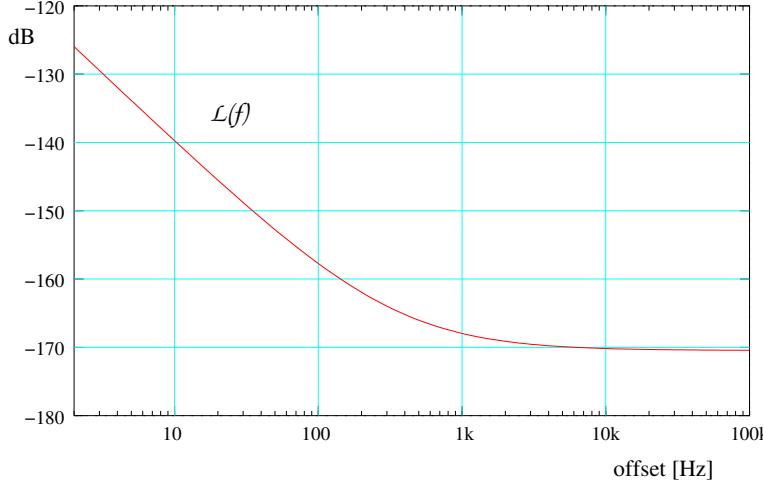


Fig. 30: Typical single-sideband phase noise spectrum

and thus

$$2\pi F_s(t_2 - t_1) + \varphi(t_2) - \varphi(t_1) = 2\pi N. \quad (42)$$

The time between these instants is an integer number of periods plus a bit of random jitter:

$$t_2 - t_1 = \frac{N}{F_s} + \Delta t. \quad (43)$$

Substitution of Eq. (43) into Eq. (42) yields

$$2\pi F_s \left(\frac{N}{F_s} + \Delta t \right) + \varphi(t_2) - \varphi(t_1) = 2\pi N \quad (44)$$

and after solving for the random jitter Δt ,

$$\Delta t = \frac{1}{2\pi F_s} (\varphi(t_1) - \varphi(t_2)). \quad (45)$$

The expected value of the variance is obtained by squaring Eq. (45):

$$\langle \Delta t^2 \rangle = \frac{1}{4\pi^2 F_s^2} (\langle \varphi(t_1)^2 \rangle - 2\langle \varphi(t_1)\varphi(t_2) \rangle + \langle \varphi(t_2)^2 \rangle). \quad (46)$$

Both t_1 and t_2 are affected by the same statistical jitter, so

$$\langle \varphi(t_1)^2 \rangle = \langle \varphi(t_2)^2 \rangle = \langle \varphi(t)^2 \rangle. \quad (47)$$

The variance of this term is the same, whether regarded in the time or in the frequency domain (Parseval's theorem), so, with $S_\varphi(f)$ the spectral density of the phase error,

$$\langle \varphi(t)^2 \rangle = \int_0^\infty S_\varphi(f) df. \quad (48)$$

The variance of the middle term of Eq. (46) is obtained by taking the cosine transform

$$\langle \varphi(t_1)\varphi(t_2) \rangle = \int_0^\infty S_\varphi(f) \cos(2\pi f \tau) df, \quad (49)$$

with $\tau = t_1 - t_2$. Substituting Eq. (48) and (49) back into Eq. (46) gives

$$\langle \Delta t^2 \rangle = \frac{1}{2\pi^2 F_s^2} \int_0^\infty S_\varphi(f) (1 - \cos(2\pi f \tau)) df = \frac{1}{\pi^2 F_s^2} \int_0^\infty S_\varphi(f) \sin^2(\pi f \tau) df . \quad (50)$$

And finally, the standard deviation of the timing error as a function of the phase noise spectrum becomes

$$\Delta t = \frac{1}{\pi F_s} \sqrt{\int_0^\infty S_\varphi(f) \sin^2(\pi f \tau) df} , \quad (51)$$

with τ the time between the significant instants of the clock waveform, usually equal to $1/F_s$. It is seen that the timing jitter depends on the total energy in the sidebands of the clock signal, weighted so that noise at small offsets and at offset frequencies near NF_s contributes little to the total jitter [10]. In practice, the integration interval is constrained by the reciprocal of the measurement time for the lower bound, and by the bandwidth of the ADC's sampler for the upper bound. Finally, the phase noise is often specified as single-sideband phase noise, denoted $\mathcal{L}_\varphi(f)$. Normally, the phase noise spectrum above and below the carrier is uncorrelated, but has equal power density, so that $S_\varphi(f) = 2\mathcal{L}_\varphi(f)$.

2 Converter architectures

There are many different ways of constructing analog-to-digital converters, and for some architectures, there are hundreds of different ADC implementations on the market. Technology constantly evolves, yielding ever higher performance and deeper integration. Each has its own peculiarities and strengths and depending on the application, some type of architecture may be preferable over another.

The main architectures, with their typical application domains and ballpark specifications, are outlined in Table 1. In the paragraphs following, we shall give an overview of the most prevalent architectures in existence, with some selected specific models of each. For some architectures, the choice available is so overwhelmingly large that there is no hope of giving a comprehensive overview.

Table 1: Some converter architectures

Architecture	Speed	Resolution	Linearity	Applications
Flash	Very fast (GS/s)	Poor (8 bits)	Poor	Oscilloscopes, transient recorders
Successive approximation	Fast (MS/s)	Fair (14 bits)	Fair	DSP, digital receivers, instrumentation
$\Sigma-\Delta$	Slow (kS/s)	Excellent (24 bits)	Excellent	Process control, audio, weight, pressure, temperature measurement
Dual slope integration	very slow (S/s)	Very good (18 bits)	Very good	Bench-top and hand-held measuring instruments, battery powered devices

Converters exist that combine some aspects of different architectures. For example, flash and successive approximation (SA) can be combined to yield better resolution than pure flash, at a higher speed

than pure SA. Besides the main architectures mentioned in Table 1, several less commonly used ADC types exist, for example, voltage-to-frequency converters, Wilkinson converters, tracking converters.

Many converters nowadays use switched-capacitor technology internally. Since charge stored on capacitors leaks away over time, this implies that such ADCs not only have a maximum conversion rate, but a minimum conversion rate as well. Moreover, changing the conversion rate of such ADCs on the fly may result in transient distortion effects. These phenomena do not usually appear in the data sheets.

Increasingly, one finds converters that are complete data acquisition subsystems, with many configuration options programmed by setting internal registers. Such ADCs cannot be used without some sort of processor or microcontroller to set all configuration registers.

2.1 Flash ADCs

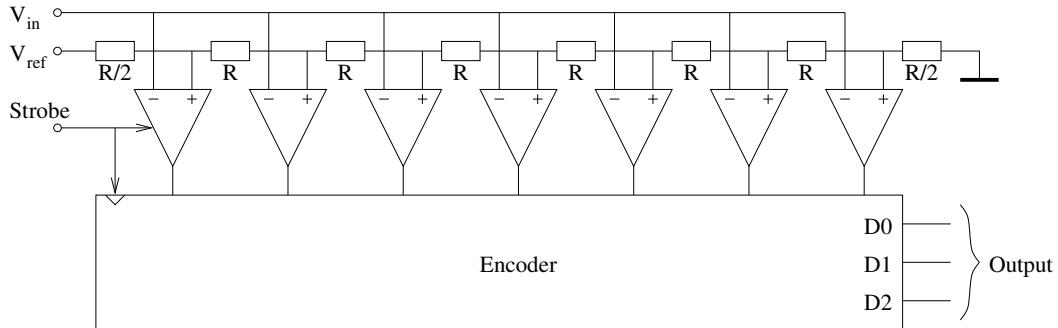


Fig. 31: Principle of a flash ADC

This is the fastest ADC architecture in existence. An n -bit flash ADC has $2^n - 1$ comparators, simultaneously comparing the input signal with the potential on as many taps of a resistor chain divider (Fig. 31). A logic encoder circuit takes the output of the comparators and turns it into a binary code. The converter is very fast: Its conversion time is basically the delay of the comparators plus that of the logic circuits, which added together can be under a nanosecond. The disadvantage is that for every bit extra, the number of comparators roughly doubles, which quickly gets out of hand. The input capacitance goes up linearly with the number of comparators, which quickly leads to unreasonable demands on the driving amplifier. Flash converters commonly have 8 bits, but up to 10 bits is possible.

Owing to differing offsets in individual comparators, the DNL of this kind of ADC is often rather poor. The converter may even be non-monotonous or have missing codes. Differences in switching speed between the comparators may cause transient *sparkle* values to appear at the outputs. Moreover, a comparator's switching speed depends on the magnitude of the difference signal it sees. At least one comparator sees a very small difference and therefore, it will be comparatively slow to settle. (There is even an infinitesimal chance that it will not settle at all, so called *metastability*.) Flash converters are often clocked or strobed to hide transient codes. For high input frequencies, the dynamic performance of the ADC may be much improved by preceding it with a Sample-and-Hold Amplifier (SHA), timed in such a way that the input signal is kept steady while conversion takes place. Flash converters are used in oscilloscopes and transient digitizers.

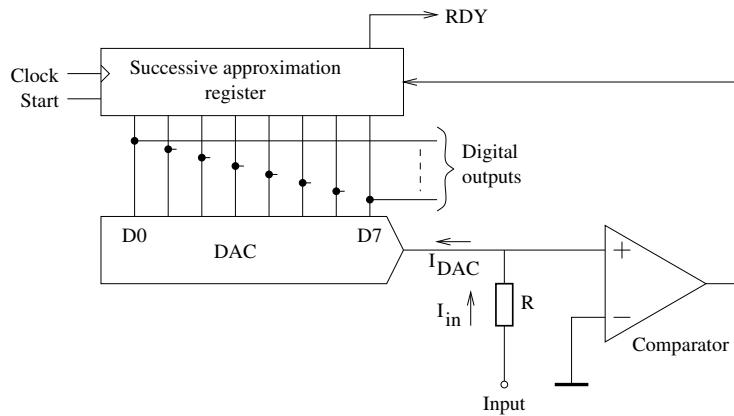
The digital interface of flash ADCs is sometimes de-multiplexed to reduce the necessary transfer rate on the digital buses. Differential signalling is used extensively, both for analog and digital signals. The chip always has multiple ground and power supply connections. Thus, even though a flash ADC may digitize to only eight bits, it is usually housed in packages with over a hundred pins. The power consumption of high-performance flash converters is quite high, often exceeding 1 W.

Table 2: Some flash converters

Type	Bits	Rate	INL	DNL	Power	Interface
ADC081500	8	1.5 GS/s	0.3 LSB	0.15 LSB	1.2 W	LVDS
MAX104	8	1 GS/s	0.25 LSB	0.25 LSB	5.25 W	diff PECL

2.2 Successive approximation ADCs

A very common and popular architecture is the successive approximation ADC (Fig. 32). It starts a conversion by first comparing the half scale value with the input. If the input is greater, the corresponding bit is set; If not, it is cleared. The approximation is then refined with successively smaller increments until the required precision is reached, at which time the conversion is complete.

**Fig. 32:** Principle of a successive approximation ADC

The analog input bandwidth of SAR ADCs often well exceeds the Nyquist frequency. This can be useful in undersampling applications, like in digital receivers, but it makes the use of an anti-alias filter mandatory. It is important to keep the input value steady while conversion is going on. Most converters nowadays have a built-in hold circuit. A SHA should be used if this is not the case and if the input signal can vary during conversion. Even if the converter has its own hold circuit, if undersampling is used, performance may still benefit from a wide-band sample-and-hold preceding the ADC.

The digital interface of SAR ADCs is usually parallel, but lately a lot of serial interface devices have appeared on the market. These are handy for use with small microcontrollers and are available in tiny packages with surprisingly few pins.

The DNL of this type of ADC can be notably poor, sometimes to the point that certain output values never occur: So-called missing codes. SAR converters can never be non-monotonous. If you care about DNL, such as when performing spectrography, avoid using successive approximation ADCs.

Table 3: Some successive approximation converters

Type	Bits	Conv. rate	BW	SFDR	INL	DNL	Notes
AD7476	12	1 MS/s	6.5 MHz	80 dB	1 LSB	0.75 LSB	6-lead SOT23
LTC2356	14	3.5 MS/s	50 MHz	86 dB	0.5 LSB	0.4 LSB	10-lead MSOP
LTC1279	12	600 kS/s	5 MHz	82 dB	1 LSB	1 LSB	SOL-24

2.3 Pipeline converters

These may be regarded as a combination of flash and successive approximation. They are sometimes also referred to as segmented or sub-ranging ADCs. At each clock pulse, the input value is digitized by a coarse, first-stage, flash ADC. This first stage may have just three or four bits of resolution. The coarse

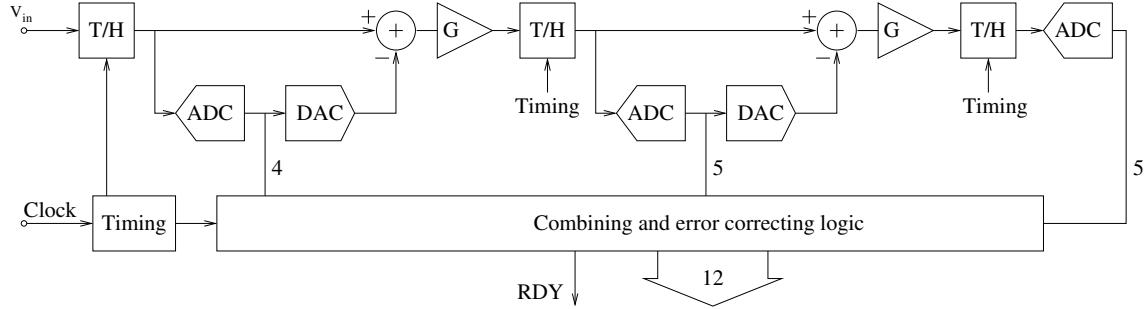


Fig. 33: Architecture of a pipelined ADC

approximation is subtracted from the original input and the residue is converted by the next flash stage on the next clock tick. This may be repeated one or two more times to reach the desired resolution. The partial conversion results are combined using logic circuits. Each converter stage usually spans a little more than the ideal residue of the preceding stage, so that a converter such as the one in Fig. 33, with one 4-bit stage and two 5-bit stages, may finally deliver an overall result of only 12 bits. This is done to accommodate linearity errors in the sub-sections (Fig. 34).

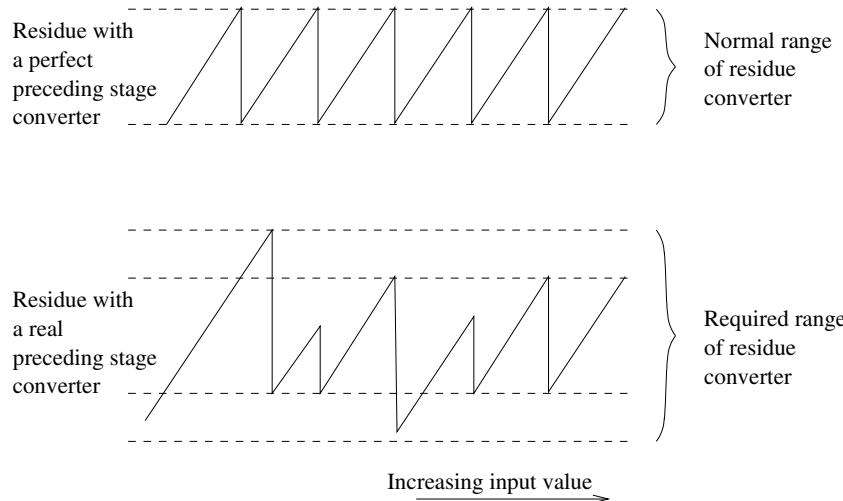
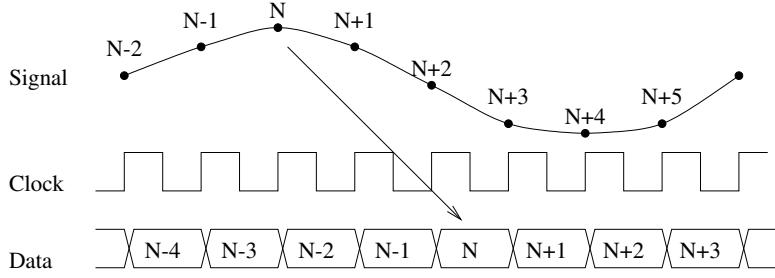


Fig. 34: The effect of imperfect sub-converter linearity

The ADC delivers one output value every clock cycle, but each such value refers to the input level of several clock cycles previous (Fig. 35). This may be important if the ADC is part of a closed-loop feedback system, or if it is multiplexed across several input signals. These ADCs are used in fast signal processing applications like digital radio, radar, and medical ultrasound. The pipeline architecture is currently the subject of intense competition between several manufacturers and new types are introduced almost every month.

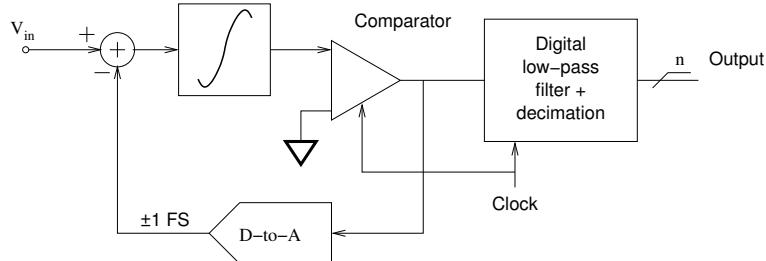
2.4 $\Sigma\Delta$ ADCs

This architecture is very popular for audio converters and high-resolution, low-speed measurement applications such as weighing scales and pressure gauges. It is simple and undemanding with respect to

**Fig. 35:** Pipelined ADCs have several clock periods of latency**Table 4:** Some pipelined converters

Type	Bits	Conv. rate	BW	SFDR	INL	DNL
AD872	12	10 MS/s	35 MHz	75 dB	1.75 LSB	0.5 LSB
AD9432	12	105 MS/s	500 MHz	80 dB	0.5 LSB	0.25 LSB
LTC2255	14	125 MS/s	640 MHz	88 dB	1 LSB	0.5 LSB
LTC2242	12	250 MS/s	1.2 GHz	78 dB	1 LSB	0.4 LSB
ADS5232	12	65 MS/s	300 MHz	85 dB	0.4 LSB	0.3 LSB
TDA9910	12	80 MS/s	370 MHz	70 dB	2 LSB	0.6 LSB

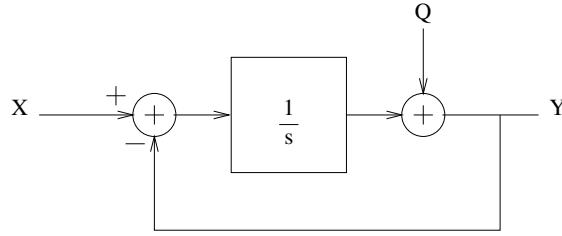
precision of components, but it requires a lot of digital post-processing, lending itself well to low-cost CMOS IC implementation. It is characterized by excellent linearity and resolution, but has low conversion rate and high latency.

**Fig. 36:** Principle of a first-order Σ-Δ ADC

Its principle is shown in Fig. 36. The output of an integrator is sampled by a strobed comparator which effectively operates as a single-bit ADC. The integrator continuously integrates the difference between the input and the comparator output. The output is a sequence of pulses with the average width representing the value of the input. The clock rate is normally many times higher than the Nyquist rate. The feedback loop, comprising the integrator and comparator is called a Σ-Δ modulator. A digital low-pass filter calculates the average over a number of samples, trading sampling speed for resolution, producing a multiple-bit sample stream at a fraction of the clock rate. This reduction of sampling rate is called *decimation*.

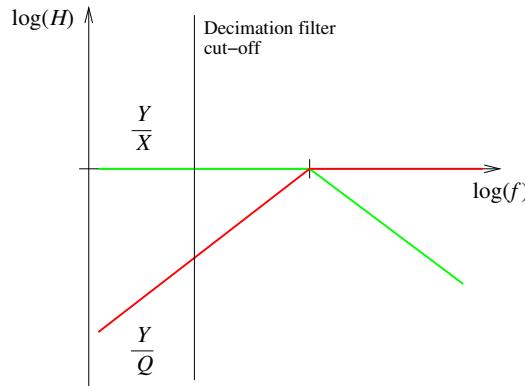
With some stretch of the imagination [11], this converter can be modelled as a linear feedback system, with the quantization due to the comparator considered as an independent source of random uncorrelated noise (Fig. 37). Examination of the loop transfer function teaches that the input signal X is subjected to a low-pass response (Fig. 38).

$$\frac{Y}{X} = \frac{1}{s+1}. \quad (52)$$

**Fig. 37:** Block diagram of a first-order $\Sigma\Delta$ ADC

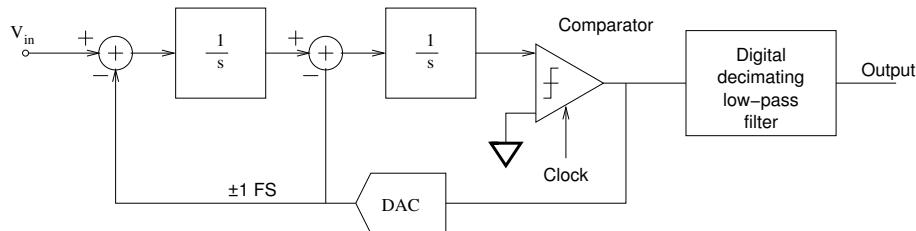
The transfer function from the point of view of the sampling noise Q is a high-pass

$$\frac{Y}{Q} = \frac{s}{s+1}. \quad (53)$$

**Fig. 38:** Noise shaping for a first-order $\Sigma\Delta$ ADC

With a suitable choice of sampling frequency and integrator time constant, most of the quantization noise is beyond the bandwidth of interest and removed by the decimation filter. The combined response of the $\Sigma\Delta$ modulator and the decimation filter affords 9 dB of resolution improvement per octave of oversampling ratio: 6 dB for the modulator and 3 dB for the filter. The low-pass response with respect to the input signal and the high oversampling ratio ease the requirements put on the anti-aliasing filter, sometimes to the point of making it superfluous altogether.

The performance of $\Sigma\Delta$ ADCs can be significantly improved by using higher order modulator loops and/or multi-bit quantizers (Fig. 39). However, ensuring stability of higher order modulators after an input overload event becomes problematic.

**Fig. 39:** Block diagram of a second-order $\Sigma\Delta$ ADC

The decimation filter is usually implemented as a FIR filter with several tens, or even up to a few thousand, taps. Because of the long latency of such filters, this architecture is less suitable for use in

feedback systems or in applications where many different signals are multiplexed into a single converter. Some converters have programmable filters, sometimes combining IIR and FIR filters, to trade settling time against resolution according to the needs of the application.

$\Sigma\Delta$ converters designed for instrumentation are usually DC-accurate. Some have integrated signal conditioning amplifiers, for example, the AD7799 has a built-in instrumentation amplifier (as well as programmable conversion rate and filter bandwidth, and several other fancy features). Those used for audio signal processing have good dynamic properties, but usually very poor gain and zero error specifications.

Table 5: Some $\Sigma\Delta$ converters

Type	Bits	Sampling rate	Delay	Decimation	Notes
ADS1610	16	60 MS/s	$3 \mu s$	6	Instrumentation
AD1871	24	6.1 MS/s	$460 \mu s$	64	Audio
AD7400	16	10 MS/s	NA	NA	$\Sigma\Delta$ mod. (isolated)
AD7799	24	64 kS/s	240 ms	15 k	Instrumentation
LTC2400	24	154 kS/s	160 ms	256	Instrumentation

2.5 Dual-slope integration

This architecture offers good resolution and linearity, combined with low power consumption, but is usually limited to low conversion rates (tens of conversions per second), Fig. 40. It is very tolerant of component value drift and lends itself well to integration on low-cost semiconductor processes. It is often used in measurement instruments like volt meters and weighing scales. It is available in monolithic ICs containing not only the converter, but also auto-calibration and auto-ranging circuitry and even display driver logic. This technology has lost a lot of ground in favour of $\Sigma\Delta$ converters.

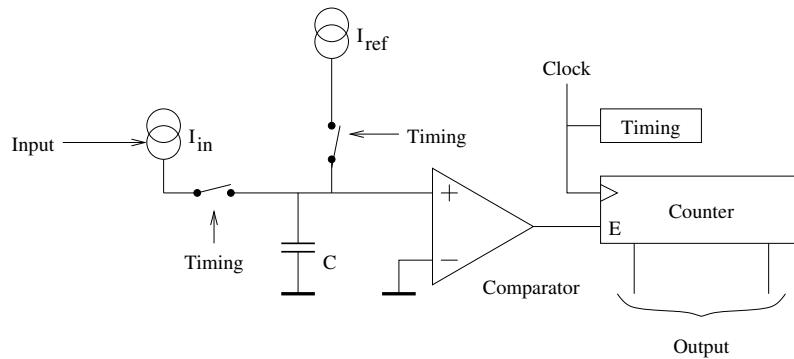
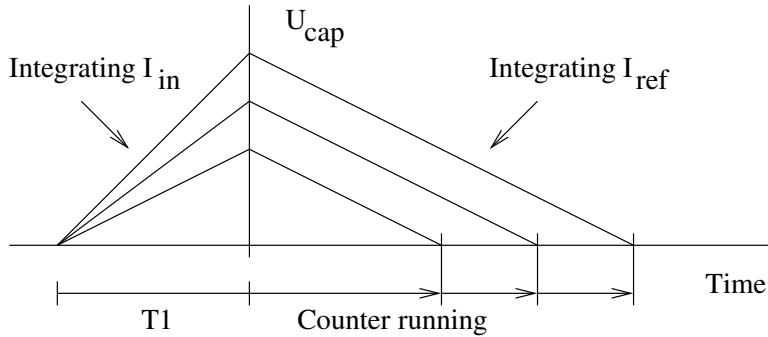


Fig. 40: Principle of a dual-slope ADC

Its operation has two phases: In the first, a current proportional to the input signal is integrated onto a capacitor for a fixed number of clock periods, Fig. 41. Then, a constant current I_{ref} is used to linearly discharge the capacitor, while at the same time incrementing a digital counter at the same clock rate. The counter is stopped when the capacitor voltage reaches zero. The counter then holds the digital representation of the input value. It is clear that the exact value of the capacitor is of no importance (but beware of dielectric absorption effects), and neither is the exact clock frequency. Often, the duration of the charging phase is chosen to be an integer number of mains power periods, in order to reject power line interference. The integral over one period of mains interference is usually zero.

Because of their excellent DNL, variants of this architecture are still widely used in high-energy physics to measure charge from photomultipliers or wire chamber detectors. In this application, they are

**Fig. 41:** Operation of a dual-slope ADC

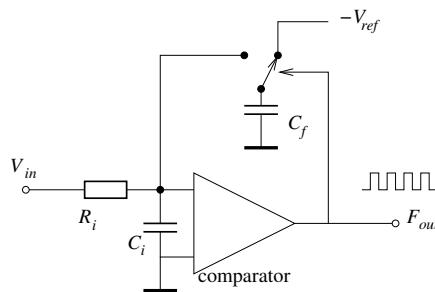
referred to as ‘Wilkinson run-down ADCs’. Another variant implements a time-to-digital converter, by gating a constant current into the capacitor between a START and a STOP pulse input during the first phase. This method easily yields timing resolution in the picosecond domain, well beyond the reach of direct counting logic.

Table 6: Some dual-slope converters

Type	Resolution	Conversion rate	Linearity	Notes
ICL7106	$3\frac{1}{2}$ digits	3 S/s	10^{-4}	LCD display driver
TC7109	12 bits	30 S/s	$5 \cdot 10^{-5}$	μ C compatible
ALD500	$5\frac{1}{2}$ digits	2 S/s	$4 \cdot 10^{-5}$	No counters
MQT300	18 bits	$10 \mu s$	10^{-5}	Wilkinson

2.6 Voltage-to-frequency converters

This is a very cheap and simple converter, yet it can deliver excellent linearity (10^{-4} ballpark). Its output signal, square pulses at a rate proportional to the input voltage, can very easily be sent over long distances with little risk of corruption. A simple counter can be used to acquire the signal. It is very easy to make integrating measurements: Just do not reset the counter. Converting the signal back into an analog voltage is also very simple: A low-pass filter will often do. These properties make it very interesting for remote measurement in industrial process control applications, e.g., in oil refineries and other large chemical plants. It is used in energy meters, flow, pressure and temperature meters, fuel gauges, and speed measurement.

**Fig. 42:** Principle of a voltage-to-frequency converter

Its operation can be easily understood from Fig. 42. The input voltage is integrated on the input capacitor C_i . Every time the voltage on C_i goes positive, the comparator discharges the feedback capacitor C_f into the input, thus removing a fixed amount of charge from C_i and taking its potential negative

again. Immediately after, C_f is once more connected to the (negative) reference and charged up with a new fixed amount of charge. The rate at which this repeats depends linearly on the value of the input voltage. Note that as long that $C_i \gg C_f$, the exact value of C_i does not matter. To first order, the circuit's conversion constant is

$$\frac{F_{out}}{V_{in}} = \frac{1}{R_i V_{ref} C_f}. \quad (54)$$

Table 7: Some voltage-to-frequency converters

Type	Range	Linearity	F_{max}
LM331	1 kHz/V	0.003%	10 kHz
AD537	1 mA	0.25%	150 kHz
AD7740	2.5 V	1%/kHz	500 kHz
XR4151	1 kHz/V	0.05%	10 kHz
VFC110	400 μ A	0.02%	4 MHz

2.7 Other architectures

Some physical quantities lend themselves well to particular conversion techniques. Linear or rotational movement or displacement can be converted into digital format using coding rulers or disks (Fig. 43). Gray-code patterns are used to avoid glitches due to multiple simultaneous bit transitions. Proximity detectors, optical or Hall effect detectors can detect the passage of gear teeth to measure speed or position in vehicles and machinery.

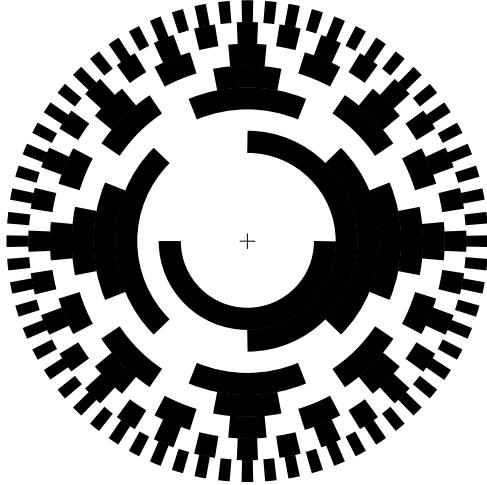


Fig. 43: A Gray-coded resolver disc as used in angular sensors

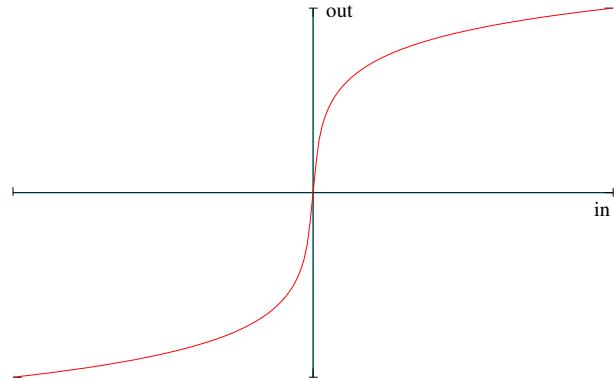


Fig. 44: A-law compression curve used for voice communication converters

There have been attempts to create floating point format ADCs, such as MicroNetworks' MN5420. While the idea seems attractive, none of these were particularly successful commercially.

For voice communication, ADCs have been designed with non-uniform quantization functions, with decreasing step sizes near zero (Fig. 44). This was done to improve the SNR for small signals, common in speech. Nowadays, these are advantageously replaced by ordinary linear Σ - Δ converters, followed by dynamic range compression in the digital domain.

3 Digital-to-analog converters

There exist many different DACs of varying architectures. Some are optimized for instrumentation applications, requiring good DC accuracy. Others may be optimized for the generation of RF signals, and the emphasis there lies on speed and the lowest possible spurious and harmonic components. Audio DACs are optimized for cost and distortion.

Table 8: Some DAC architectures

Architecture	Properties
Kelvin–Varley divider	Very accurate, monotonous.
Thermometer DAC	Monotonous. Limited number of bits.
Binary weighted ladder	Very common, but subject to glitches.
R-2R ladder	Widely used. Not very power efficient.
$\Sigma - \Delta$	Linear, accurate, but complex.

A Digital-to-Analog Converter (DAC) produces a staircase approximation of the desired signal. As demonstrated in Section 1.3, this signal must be filtered to remove the out-of-band energy.

In addition to the aliases of the desired signal, a DAC output also contains harmonics, due to the INL of the DAC, and spurious signals that may have multiple origins. Note that harmonics above the Nyquist rate also have aliases, so the spectrum of a DAC output can look positively busy. If an alias of a harmonic falls within the desired signal bandwidth, it will be a source of trouble. It is impossible to filter this out! Careful frequency planning is used to avoid these pitfalls.

The quality of the DAC clock is just as important as it is for ADCs. The DAC output signal is convolved with the spectrum of the clock source.

Many DACs have an internal fixed full-scale reference voltage. Some can be used with an externally applied reference and a few will even accept reference voltages of either polarity and with a reasonable bandwidth. These are called *multiplying* DACs and can be very useful in certain applications.

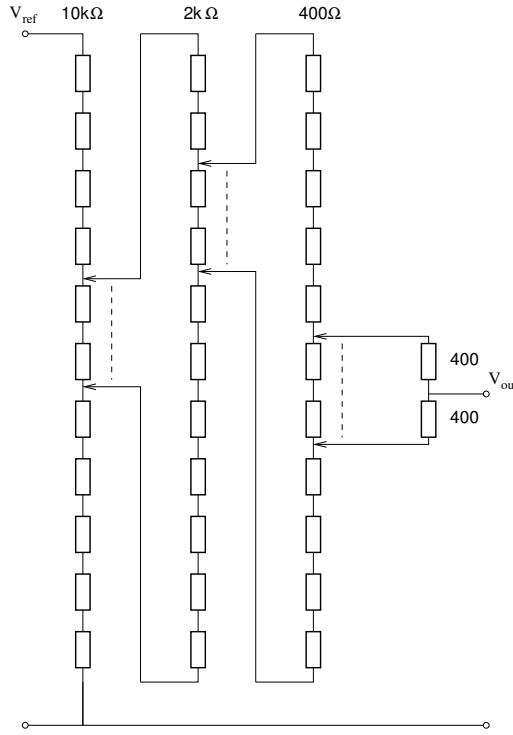
3.1 The Kelvin–Varley divider

The grandfather of all DACs is perhaps the Kelvin–Varley Divider (KVD), which, despite its age, is still one of the most accurate D-to-A converters in existence. It is still used in metrology applications. Its basic building block is a divider made of a string of carefully matched resistors (Fig. 45). It is guaranteed to be monotonic. The two switches selecting the taps always move together. This keeps the total resistance between the end points constant. KVDs are available with up to 7 decades of resolution. The output impedance of a KVD depends on the selected output value, and thus is accurate only at zero current. It can therefore only be used in either nulling bridge measurements, or buffered with a high input impedance precision amplifier. KVD-like structures are used as subsections of segmented DACs.

Considering that commercially available, manually switched KVDs are basically nothing more than resistors and switches, they may seem surprisingly expensive. However, they are precision instruments, targeted at calibration and standards laboratories.

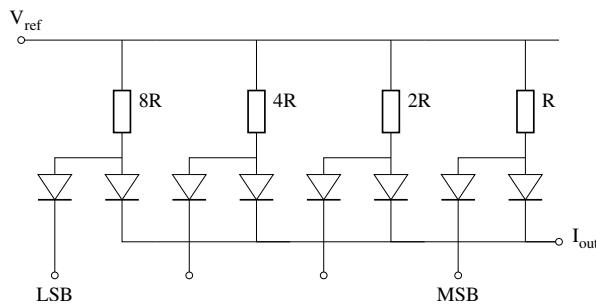
Table 9: Kelvin–Varley dividers

Type	Input resistance	Resolution	Accuracy	Notes
KVD720A	100 k Ω	10^{-7}	10^{-7}	Precision, metrology
GR1455A	10 k Ω	10^{-4}	$1.5 \cdot 10^{-4}$	Routine lab use
DAC081S101	NA	8 bits	$7 \cdot 10^{-4}$	String DAC
DAC5571	NA	8 bits	0.5 LSB	String DAC

**Fig. 45:** The Kelvin–Varley divider

3.2 The binary-weighted ladder DAC

This very simple DAC architecture adds together the currents flowing through a set of resistors dimensioned such that each differs by a factor of two from its immediate neighbours. The device as depicted in Fig. 46 (albeit with 8 bits and an additional single-transistor output buffer amplifier) was used by Hybrid Systems as a half-serious, give-away, promotional item at an exposition in the 1970s, but it proved so popular that it is still available today as the DAC371-8. In the schematic as drawn, the currents are set with resistors and the switches are diodes. The output should feed a zero impedance point, like the virtual signal ground at the negative input of an operational amplifier buffer. In more serious implementations of this principle, current sources and transistor differential switches may be used, and the output voltage may be allowed to vary.

**Fig. 46:** Binary-weighted DAC

Both current output and voltage output variants are possible. The figure is an example of the former. Current output DACs have only a limited *compliance*, i.e., they can deliver their nominal current over only a limited range of output voltage, often not even 500 mV. Amplifiers are then needed to bring up the signal to a useful level. The total capacitance of the internal current switches is often enough to push the amplifier into instability unless extra frequency compensation is used.

The wide range of resistor values required makes this architecture less suitable for integration. The different currents in the switches affect the switching speeds, leading to glitches in the output. This architecture is not inherently monotonic.

Table 10: Some binary-weighted DACs

Type	Bits	Update rate	Settling time	glitch	Application
THS5641A	8	100 MHz	35 ns	5 pVs	Video, communication
TLC5602	8	30 MHz	30 ns	–	Video

3.3 The R–2R ladder

This very common architecture uses only two different resistor values, and all the switches conduct the same current, thus correcting two of the deficiencies of the binary-weighted ladder, Fig. 47. Its disadvantage is that only a fraction of the current of each stage contributes to the output, making it less attractive for low-power applications. Otherwise this architecture is very similar to the binary-weighted ladder. It is not inherently monotonic.

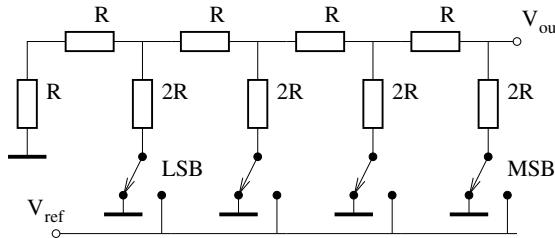


Fig. 47: R–2R ladder DAC

Table 11: Some R–2R DACs

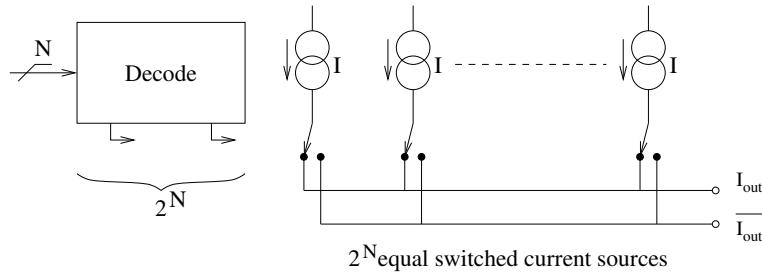
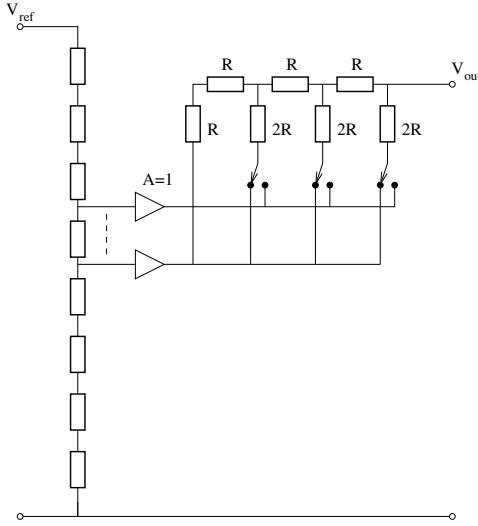
Type	Bits	Update rate	Settling time	Glitch	Notes
AD5445	12	20.4 MHz	80 ns	2 nVs	Multiplying DAC
LTC7545	12	9 MHz	1 μ s	2 nVs	4Q multiplying DAC
MAX7524	8	6 MHz	400 ns	–	CMOS multiplying DAC

3.4 Thermometer DAC

This architecture is most often used as a sub-circuit in segmented DACs (see below). For N bits, it uses 2^N identical current sources, with digital decoder logic to switch on the appropriate number of sources. To remain practical, the number of bits is limited, Fig. 48. It is inherently monotonic and glitch-free.

3.5 Segmented DACs

Segmented DACs combine several architectures in an attempt to strike a balance between speed, power consumption, distortion, glitch energy and maybe other considerations. Most high-performance DACs, such as those used in communication equipment and signal generators use some variant of this architecture. Figure 49 shows an example that combines a Kelvin–Varley first stage with an R–2R ladder secondary stage. A few bits of the applied digital input are used to select the KVD tap, and the remaining bits control the switches of the R–2R section. Other combinations implying binary-weighted and thermometer DAC sections are also encountered.

**Fig. 48:** Thermometer DAC**Fig. 49:** A segmented DAC**Table 12:** Some segmented DACs

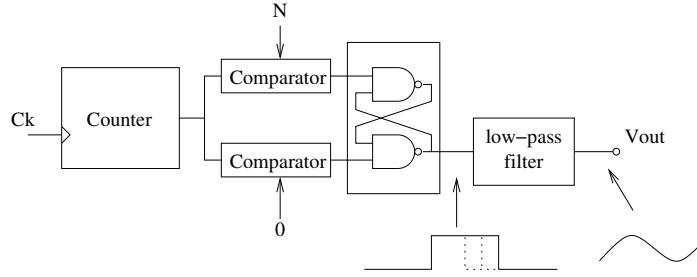
Type	Bits	Update rate	Settling time	SFDR	Notes
AD9753	12	300 MS/s	11 ns	69 dB	Communication
AD9735	12	1.2 GS/s	1 ns	75 dB	Communication
LTC1591	14	9 MS/s	1 μ s	94 dB	4Q multiplying DAC
TDA9935	14	80 MS/s	–	73 dB	Dual, communication

3.6 Pulse-width modulation

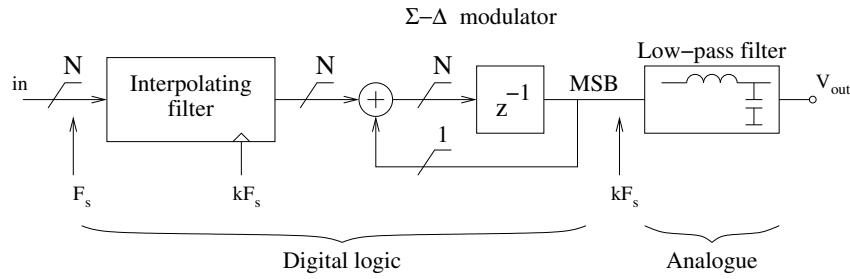
This is a very simple and cheap method that finds application in motor controllers, audio amplifiers (class-D) and single-chip micro controllers. The duty cycle of a fixed-amplitude square pulse is varied according to the desired output level, Fig. 50. A low-pass filter averages the sequence of pulses into a smooth output signal. The architecture is inherently linear. In power applications, such as portable audio amplifiers and motor controllers, its excellent power efficiency is a decisive advantage.

3.7 $\Sigma\Delta$ DACs

This architecture is similar in its general idea to the pulse-width modulation (PWM) DAC, but much more sophisticated in its execution, Fig. 51. At the same sample rate, its resolution comfortably surpasses that of the PWM DAC. A fully digital implementation of a $\Sigma\Delta$ modulator produces a train of output pulses with an average on-off ratio proportional to the desired output value. The pulse rate is many times higher than the maximum signal frequency. A low-pass filter smooths the pulses into a continuous output

**Fig. 50:** Pulse-width modulation DAC

signal. Like for $\Sigma-\Delta$ ADCs, the modulator may be of higher order, and the single-bit internal DAC may be replaced by a multi-level device.

**Fig. 51:** Architecture of a first-order $\Sigma-\Delta$ DAC

This architecture is relatively recent, because of the complexity of the digital $\Sigma-\Delta$ modulator. It is inherently linear and monotonic and can be built with excellent resolution. It is used in instrumentation and audio equipment. It is available as a logic design file (IP, Intellectual Property) that can be programmed into FPGAs, leaving the designer only the burden of designing the output stage and filter. The principle can also be applied to extend the resolution of ordinary DACs.

Table 13: Some $\Sigma-\Delta$ DACs

Type	Bits	Rate	SNR	
AD1955	24	192 kS/s	120 dB	Audio
MAX5556	16	50 kS/s	86 dB	Audio
CWda30	≤ 24	NA	Variable	IP core

3.8 DAC ailments

DACs suffer from basically the same imperfections as ADCs regarding linearity and harmonic distortion, leading to spurs, harmonics of the signal, and clock feedthrough artefacts in the spectrum of the output signal. However, where an ADC would have missing codes, a DAC would be non-monotonous. DACs used in closed-loop feedback systems should be monotonous, or the loop might hang on such imperfections. At some points in its transfer function, a DAC may generate glitches, where the output value briefly departs from the vicinity of the final output value, even for small changes in the digital input. These typically occur when many input bits change value simultaneously. Manufacturers will usually specify the importance of these glitches in units of volt·seconds, improperly referred to as *glitch energy*.

4 Signal integrity

This is a whole subject by itself and cannot possibly be properly dealt with in just a few pages. We shall merely give an overview. For more detailed treatment, refer to the literature [12], [13]. First some generalities: We distinguish between noise, which is an inherent property of the circuit components, and interference, which comes from elsewhere. Noise may be present on the input signal as an inherent property of the input transducer, or generated by components in the circuit itself. The term noise is in practice often used for what is actually interference. Examples of noise are the shot noise from a photodiode or the thermal noise of resistors. Examples of interference may be mains hum, power supply noise, or radio frequency leakage.

Interference is hard to deal with because it is usually determined by undesirable, parasitic circuit ‘components’ that do not appear explicitly in any schematic diagram, and that are usually neglected. Most often, it involves inductance and resistance of solid conductors, which are usually thought of as having the same potential throughout, or capacitive and inductive coupling between parts of the circuit that have no explicit components connecting them. This makes identifying the causes of interference difficult and requires a fair idea of the location and importance of these parasitic components.

On top of that, there is usually no catch-all solution that will solve all problems completely. Reducing one particular source of interference may aggravate another. Compromises must sometimes be made. However, in all but the most obstinate cases, an acceptable solution can usually be found.

There are three main coupling mechanisms by which interfering signals can get into a circuit:

Common impedance coupling: This is probably the most frequent cause of interference in any electronic circuitry. Conductors, wires or printed circuit board (PCB) traces have a finite non-zero impedance, both inductive and resistive. If two distinct circuits share a piece of wire or PCB trace, here modelled as Z_p (Fig. 52), the current flowing in one circuit will cause some fraction of its signal to be superimposed on the other. To reduce this effect, common current return paths should be generously dimensioned, to minimize Z_p . Bear in mind that the inductive part of Z_p is often dominant. On PCBs, this leads to using full-surface ground planes. Alternatively, you may consider giving some or all circuits their own individual return path to some common point (Fig. 53). Taken to the extreme, this results in a star topology. This latter strategy is most appropriate to low-frequency circuits.

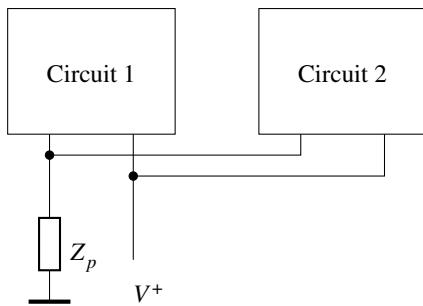


Fig. 52: Common impedance coupling

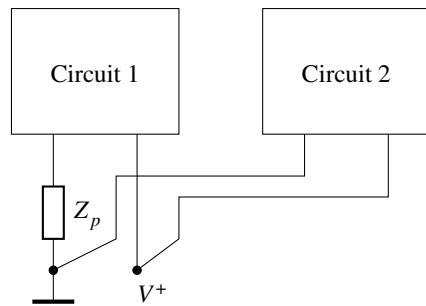


Fig. 53: Using a star layout to reduce common impedance coupling

Inductive coupling: If any closed loop is traversed by a changing magnetic flux, a voltage and/or current will be induced in that loop. Conversely, any loop carrying a current produces a flux (Fig. 54). Keep loop areas small, use twisted-pair wires or coax cable, and put continuous, uninterrupted ground planes under PCB tracks. Keep direct and return paths close together. Keep loops with high dI/dt well away from loops that carry sensitive low-level signals. Consider using differential signalling. Decoupling or bypass capacitors also serve to confine fast changing currents to small loops. Magnetic shielding is rarely practical, but at high frequencies, a simple conductive shield may be effective.

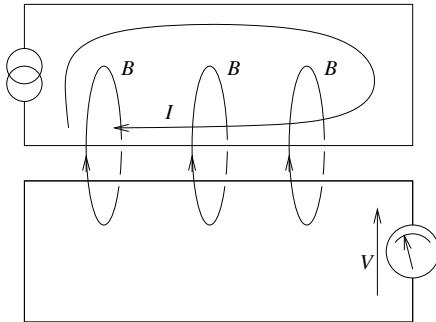


Fig. 54: Inductive coupling

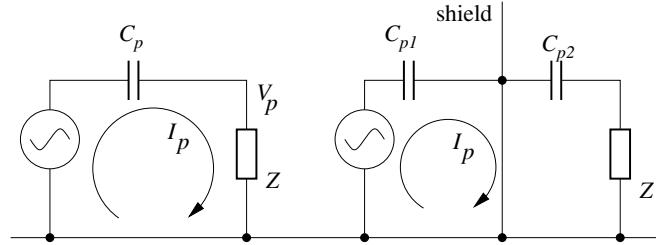


Fig. 55: Capacitive coupling and shielding

Capacitive coupling: Changing electric fields will also induce currents in nearby conductors (Fig. 55), effectively acting as parasitic capacitance C_p between circuit elements. This current produces a voltage V_p across victim impedance Z . Keep nodes with rapidly changing voltages compact. Keep high-impedance nodes far away from the first kind. If possible, reduce the dE/dt of aggressor nodes and lower the impedance of victim nodes. Put grounded shields (coax, again) or guard tracks between them (so-called Faraday shields), so that the parasitic current I_p flows on the shield rather than through the victim impedance.

4.1 Input-signal conditioning

Many considerations guide the design of input-signal conditioning circuitry. This circuitry should adapt the input signal to the ADC input; filter, scale and offset it if needed, protect against out-of-bound inputs, reject common mode interference, etc. Many circuit topologies exist to deal with each of these requirements. Manufacturers will usually recommend some conditioning circuits with suitable device type numbers and component values.

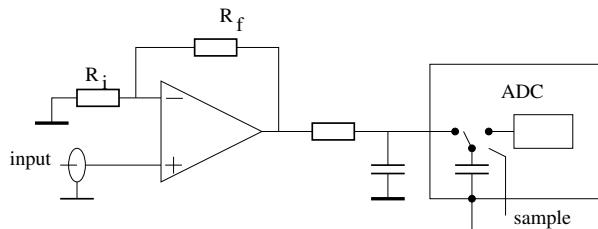


Fig. 56: Simple signal buffer

A simple amplifier can buffer and possibly scale the signal to the ADC, presenting a high impedance to the signal source and a stiff source to the ADC (Fig. 56). A low-pass RC circuit isolates the amplifier from the capacitive ADC input and absorbs sampler kick-back. The amplifier may actually be an instrumentation amplifier (Fig. 57), in cases where considerable gain is required to reveal a small useful signal riding on a large common-mode voltage. This arrangement is good for low-frequency signals, as produced by Wheatstone-bridge-like temperature or force transducers, for example.

Recent ADCs often have differential inputs in order to retain a reasonable dynamic range in spite of ever lower supply voltages. A pair of operational amplifiers can be used to convert a single-ended signal into a differential one (Fig. 58). A number of manufacturers produce monolithic differential amplifiers, which have both differential inputs and outputs for this sort of application (Fig. 59). (An example is the Analog Devices ADA4937.)

High input impedance is not always desirable, as it is more susceptible to interference and requires shielding. For higher frequencies, inputs are usually terminated, in order to avoid reflecting signal back

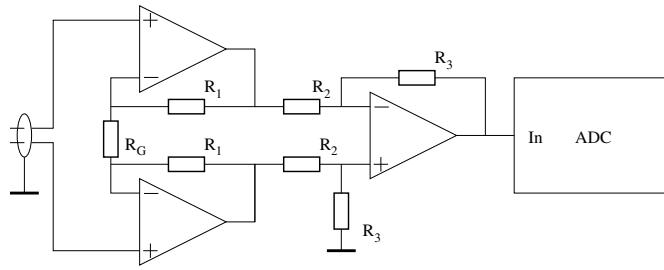


Fig. 57: Instrumentation amplifier

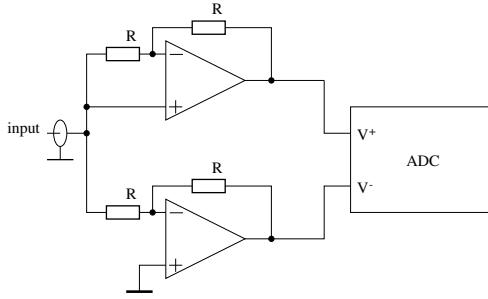


Fig. 58: Converting a single-ended signal to differential

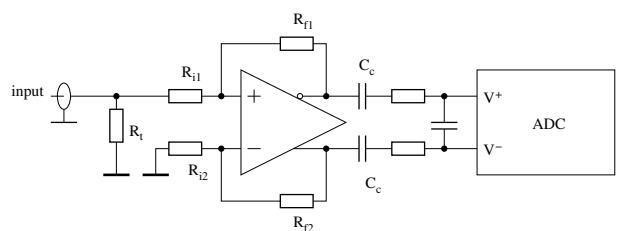


Fig. 59: Using a fully differential buffer amplifier

to the source. Termination consists of a resistor, of value equal to the characteristic impedance of the cable, across the input terminals. The frequency at which termination becomes necessary depends on the length of the cable between the signal source and the ADC.

For higher frequency signals, the single-ended to differential conversion can be obtained using baluns (Fig. 60) or transformers (Fig. 61). Even if the input signal is already differential, it may be beneficial to use a balun or transformer in order to reject common-mode interference. A transformer will not pass DC and low-frequency signals, whereas a balun will. Concerning common-mode rejection, a balun is poor at low frequencies, but it works very well at high frequencies. For the transformer, it is the reverse (due to unavoidable capacitive coupling between the windings). Both may be designed to match the input signal amplitude to best fit it into the ADC input range. Neither will prevent sampler kick-back from travelling back to the source.

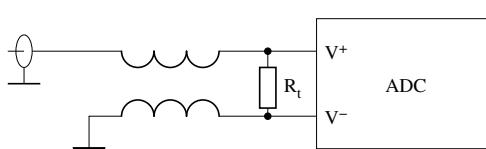


Fig. 60: Using a balun for single-ended to differential conversion

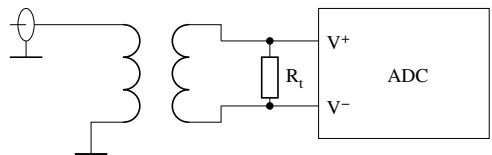


Fig. 61: Using a transformer for single-ended to differential conversion

4.2 Ground pins

Precision or high-speed ADCs have separate pins labelled AGND and DGND. Despite what is often believed, this does not mean that these pins should be connected to separate ground planes. In fact, AGND is the reference zero level for the analog part of the chip and DGND is the pin that carries the return current from the digital in- and outputs. These pins are brought out separately in order to reduce common impedance coupling between the analog and digital sections of the chip. Both should normally be connected to the same solid ground plane.

The often seen advice of splitting the ground plane into a digital and an analog section, with a single link connecting them under the ADC, is not very practical. If both ground planes were also to be tied together at the power supply, this creates a loop with unknown geometry, inviting interference. Another loop may be formed by the screens of the cables carrying the input signal from the—probably grounded—source to the ADC. And where should the link go when your gadget has several ADCs and DACs?

A much better way is to use a single uninterrupted ground plane, and to route digital signals so that their return currents do not find their way into the analog sections. The unavoidable differences in ground potential between the acquisition system and the signal source should be dealt with in the signal conditioning at the ADC input, for example, using differential amplifiers, baluns or transformers.

The digital signals of the ADC itself are an important source of noise in any case. The layout should carefully keep digital, clock, and analog signals apart. For differential signals, pay attention to symmetry. Digital outputs should drive as light a load as possible. This will reduce the intensity of the return currents. Never connect an ADC to long, shared bus lines. Always connect it to a buffer first, using the shortest possible connections, and then connect the buffer outputs to the bus instead. It may be useful to insert small-valued series resistors ($\approx 50 \Omega$) in the data lines between the ADC and the buffer to limit transient digital signal currents.

5 Conclusion

The digital revolution has afforded signal treatment with high fidelity. Analog signals are increasingly to be found only at the very ends of the signal processing chain, the intent seemingly being to eliminate any remaining analog hardware altogether. There are a huge number of different A-to-D and D-to-A converters, with properties tailored to a variety of applications. Many manufacturers compete for a share of the market. The rate of development is furious, with many new, faster and better converters appearing on the market every year.

Appendix

A Decibels

Signal levels in electronics, and also in control system theory, are often specified in decibels (dB). In fact, the dB is a value that relates a given level to some specified or implied reference. The definition in terms of power levels is

$$\text{dB} = 10 \cdot \log_{10} \frac{P}{P_r} . \quad (\text{A.1})$$

Since the amplitude of a signal is proportional to the square root of its power, the definition in terms of amplitude is

$$\text{dB} = 20 \cdot \log_{10} \frac{A}{A_r} . \quad (\text{A.2})$$

Implied is the assumption that both signals are working into the same resistance. This is usually, but not systematically, adhered to by RF specialists, and totally ignored by control system engineers.

Often, some reference level is specified by one or more trailing symbols. For example, the ratio of a signal with respect to 1 mW is given in dBm. Other forms that appear frequently are dBV or dB μ V, meaning dB with respect to a signal with r.m.s. value 1 V, respectively 1 μ V. Also often encountered are dBc, meaning decibels with respect to the level of a carrier signal, or dBFS, meaning decibels with respect to a full-scale signal.

Even though the dB is one tenth of something called a *bel*, it is never used with other SI multiplier prefixes.

References

- [1] B. Widrow, I. Kollár, Ming-Chang Liu, Statistical theory of quantization, *IEEE Trans. Instrum. Meas.*, Vol. 45, April 1996, pp. 353–361.
- [2] E.T. Whittaker, On the functions which are represented by the expansion of the interpolation theory, *Proc. Royal Soc. Edinburgh*, Sec. A, Vol. 35, 1915, pp. 181–194.
- [3] V.A. Kotelnikov, On the carrying capacity of the ether and wire in telecommunications, *Izd. Red. Upr. Svyazi RKKA*, Moscow, 1933 (Russian), <http://ict.open.ac.uk/classics/1.pdf>
- [4] H. Nyquist, Certain topics in telegraph transmission theory, *Trans. AIEE*, Vol. 47, April 1928, pp. 617–644.
- [5] R.V.L. Hartley, Transmission of information, *Bell Syst. Tech. J.*, Vol. 7, July 1928, pp. 535–563.
- [6] C. Shannon, Communication in the presence of noise, *Proc. IRE*, Vol. 37, January 1949, pp. 10–21.
- [7] W.R. Bennett, Spectra of quantized signals, *Bell Syst. Tech. J.*, Vol. 27, July 1948, pp. 446–472.
- [8] B. Widrow, A study of the rough amplitude quantization by means of Nyquist sampling theory, *IRE Trans. Circ. Theory*, Vol. CT-3, 1956, pp. 226–276.
- [9] R.M. Gray, Quantization noise spectra, *IEEE Trans. Inf. Theory*, Vol. 37, No. 6, November 1990, pp. 1220–1244.
- [10] B. Drakhlis, Calculate oscillator jitter by using phase noise analysis – Part 1, *Microwaves & RF*, Vol. 50, No. 1, January 2001, pp. 82–90, 157.
- [11] N. Thao, Asymptotic MSE law of n^{th} -order $\Sigma\text{-}\Delta$ modulators, *IEEE Trans. Circ. Syst. II: Analog and Digital Signal Processing*, Vol. 50, No. 5, May 2003, pp. 234–238.
- [12] H. Ott, *Noise-Reduction Techniques in Electronic Systems*, 2nd ed., (Wiley, New York 1988).
- [13] R. Morrison, *Grounding and Shielding Techniques*, 4th ed., (Wiley, New York, 1998).

Digital signal processor fundamentals and system design

M.E. Angoletta
CERN, Geneva, Switzerland

Abstract

Digital Signal Processors (DSPs) have been used in accelerator systems for more than fifteen years and have largely contributed to the evolution towards digital technology of many accelerator systems, such as machine protection, diagnostics and control of beams, power supply and motors. This paper aims at familiarising the reader with DSP fundamentals, namely DSP characteristics and processing development. Several DSP examples are given, in particular on Texas Instruments DSPs, as they are used in the DSP laboratory companion of the lectures this paper is based upon. The typical system design flow is described; common difficulties, problems and choices faced by DSP developers are outlined; and hints are given on the best solution.

1 Introduction

1.1 Overview

Digital Signal Processors (DSPs) are microprocessors with the following characteristics:

- a) Real-time digital signal processing capabilities. DSPs typically have to process data in real time, i.e., the correctness of the operation depends heavily on the time when the data processing is completed.
- b) High throughput. DSPs can sustain processing of high-speed streaming data, such as audio and multimedia data processing.
- c) Deterministic operation. The execution time of DSP programs can be foreseen accurately, thus guaranteeing a repeatable, desired performance.
- d) Re-programmability by software. Different system behaviour might be obtained by re-coding the algorithm executed by the DSP instead of by hardware modifications.

DSPs appeared on the market in the early 1980s. Over the last 15 years they have been the key enabling technology for many electronics products in fields such as communication systems, multimedia, automotive, instrumentation and military. Table 1 gives an overview of some of these fields and of the corresponding typical DSP applications.

Figure 1 shows a real-life DSP application, namely the use of a Texas Instruments (TI) DSP in a MP3 voice recorder–player. The DSP implements the audio and encode functions. Additional tasks carried out are file management, user interface control, and post-processing algorithms such as equalization and bass management.

Table 1: A short selection of DSP fields of use and specific applications

Field		Application	
Communication	Broadband	Video conferencing / phone	
		Voice / multimedia over IP	
		Digital media gateways (VOD)	
	Wireless	Satellite phone	
		Base station	
	Security	Biometrics	
Consumer		Video surveillance	
Entertainment	Digital still /video camera		
	Digital radio		
	Portable media player / entertainment console		
Toys	Interactive toys		
	Video game console		
Industrial and entertainment	Medical	MRI	
		Ultrasound	
		X-ray	
	Point of sale	Scanner	
		Vending machine	
	Industrial	Factory automation	
		Industrial / machine / motor control	
		Vision system	
Military and aerospace		Guidance (radar, sonar)	
		Avionics	
		Digital radio	
		Smart munitions, target detection	

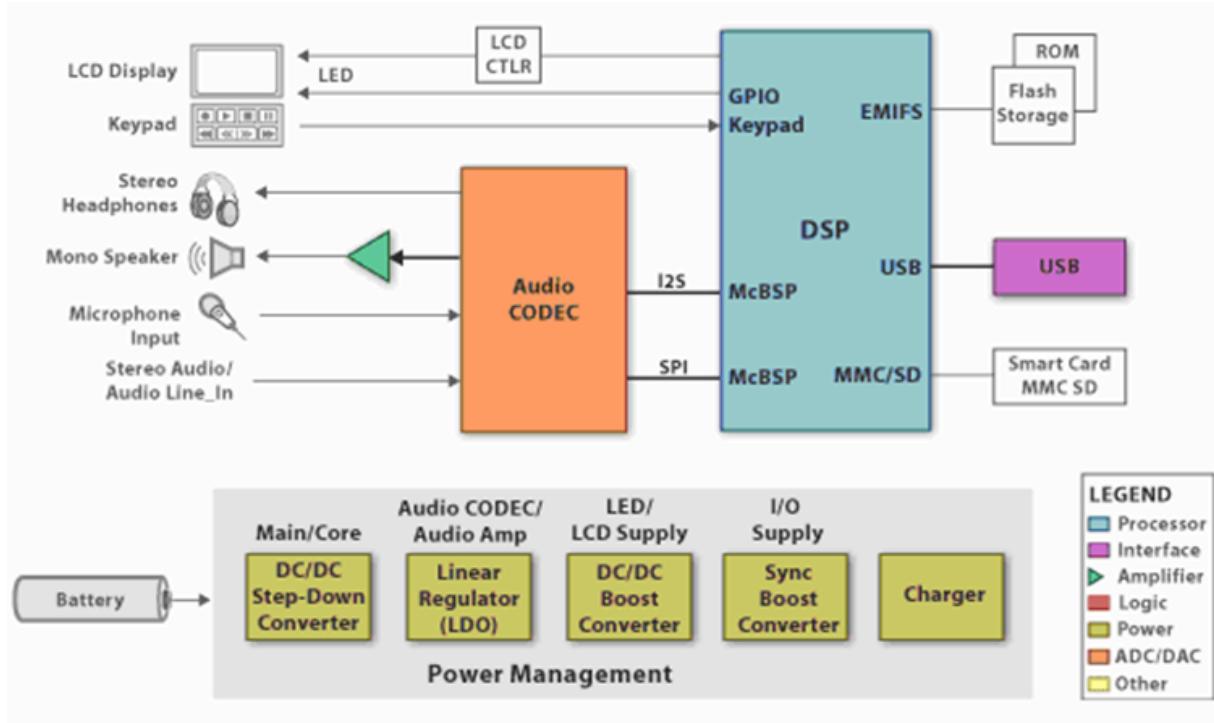


Fig. 1: Use of Texas Instruments DSP in a MP3 player/recorder system. Picture courtesy of Texas Instruments from www.ti.com.

1.2 Use in accelerators

DSPs have been used in accelerators since the mid-1980s. Typical uses include diagnostics, machine protection and feedforward/feedback control. In diagnostics, DSPs implement beam tune, intensity, emittance and position measurement systems. For machine protection, DSPs are used in beam current and beam loss monitors. For control, DSPs often implement beam controls, a complex task where beam dynamics plays an important factor for the control requirements and implementations. Other types of control include motor control, such as collimation or power converter control and regulation. The reader can find more information on DSP applications to accelerators in Refs. [1–3].

DSPs are located in the system front-end. Figure 2 shows CERN's hierarchical controls infrastructure, a three-tier distributed model providing a clear separation between Graphical User Interface (GUI), server, and device (front-end) tiers.

DSPs are typically hosted on VME boards which can include one or more programmable devices such as Complex Programmable Logic Devices (CPLDs) or Field Programmable Gate Arrays (FPGAs). Daughtercards, indicated in Fig. 2 as dashed boxes, are often used; their aim is to construct a system from building blocks and to customize it by different FPGA/DSP codes and by the daughtercards type. DSPs and FPGAs are often connected to other parts of the system via low-latency data links. Digital input/output, timing, and reference signals are also typically available. Data are exchanged between the front-end computer and the DSP over the VME bus via a driver.

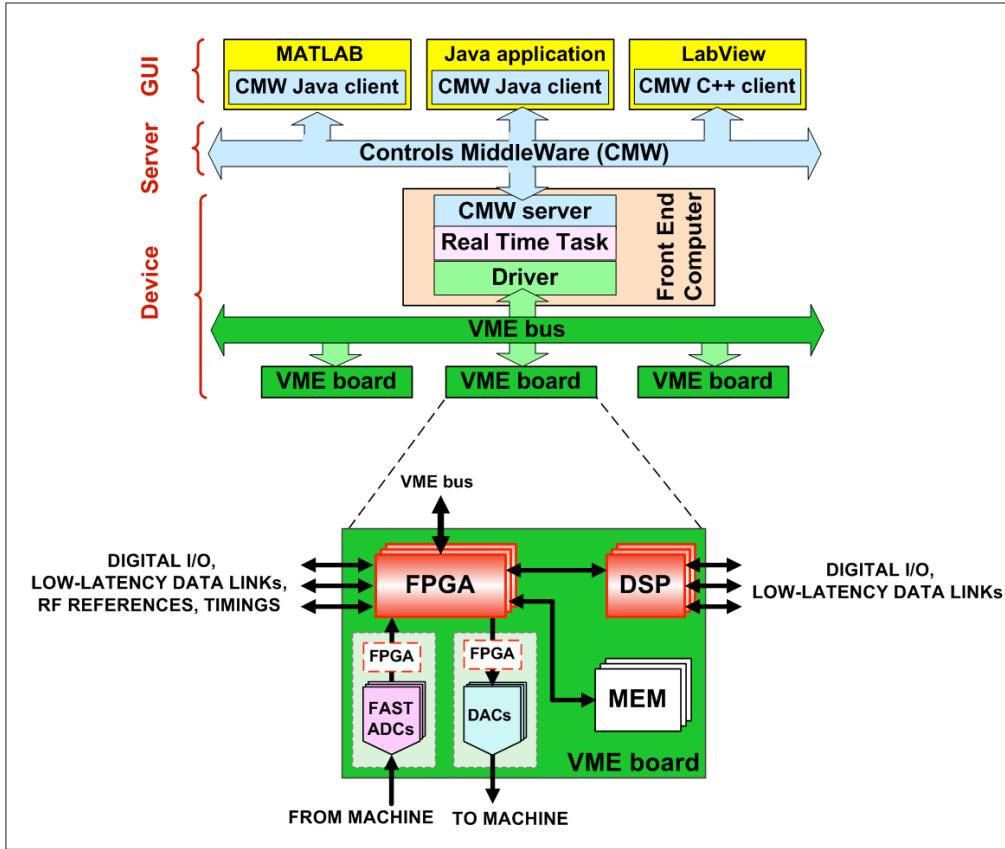


Fig. 2: Typical controls infrastructure used at CERN and DSP characteristics location

2 DSP evolution and current scenery

DSPs appeared on the market in the early 1980s. Since then, they have undergone an intense evolution in terms of hardware features, integration, and software development tools. DSPs are now a mature technology. This section gives an overview of the evolution of the DSP over their 25-year life span; specialized terms such as ‘Harvard architecture’, ‘pipelining’, ‘instruction set’ or ‘JTAG’ are used. The reader is referred to the following paragraphs for explanations of their meaning. More detailed information on DSP evolution can be found in Refs. [4], [5].

2.1 DSP evolution: hardware features

In the late 1970s there were many chips aimed at digital signal processing; however, they are not considered to be digital signal processing owing to either their limited programmability or their lack of hardware features such as hardware multipliers. The first marketed chip to qualify as a programmable DSP was NEC’s MPD7720, in 1981: it had a hardware multiplier and adopted the Harvard architecture (more information on this architecture is given in Section 3.1). Another early DSP was the TMS320C10, marketed by TI in 1982. Figure 3 shows a selective chronological list of DSPs that have been marketed from the early 1980s until now.

From a market evolution viewpoint, we can divide the two and a half decades of DSP life span into two phases: a development phase, which lasted until the early 1990s, and a consolidation phase, lasting until now. Figure 3 gives an overview of the evolution of DSP features together with the first year of marketing for some DSP families.

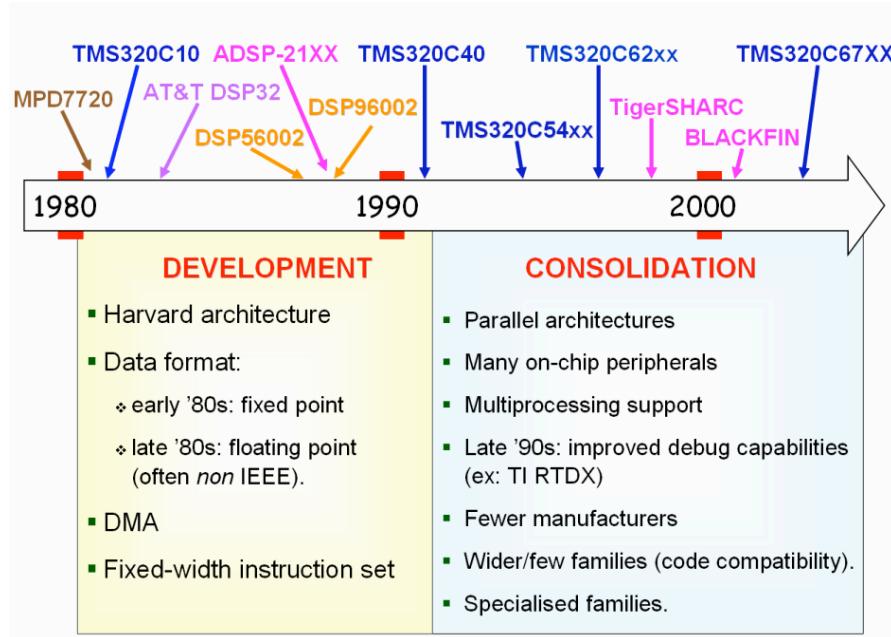


Fig. 3: Evolution of DSP features from their early days until now. The first year of marketing is indicated at the top for some DSP families.

During the market development phase, DSPs were typically based upon the Harvard architecture. The first generation of DSPs included multiply, add, and accumulator units. Examples are TI's TMS320C10 and Analog Devices' (ADI) ADSP-2101. The second generation of DSPs retained the architectural structure of the first generation but added features such as pipelining, multiple arithmetic units, special address generator units, and Direct Memory Access (DMA). Examples include TI's TMS320C20 and Motorola's DSP56002. While the first DSPs were capable of fixed-point operations only, towards the end of the 1980s DSPs with floating point capabilities started to appear. Examples are Motorola's DSP96001 and TI's TMS320C30. It should be noted that the floating-point format was not always IEEE-compatible. For instance, the TMS320C30 internal calculations were carried out in a proprietary format; a hardware chip converter [6] was available to convert to the standard IEEE format. DSPs belonging to the development phase were characterized by fixed-width instruction sets, where one of each instruction was executed per clock cycle. These instructions could be complex, and encompassing several operations. The width of the instruction was typically quite short and did not overcome the DSP native word width. As for DSP producers, the market was nearly equally shared between many manufacturers such as AT&T, Fujitsu, Hitachi, IBM, NEC, Toshiba, Texas Instruments and, towards the end of the 1980s, Motorola, Analog Devices and Zoran.

During the market consolidation phase, enhanced DSP architectures such as Very Long Instruction Word (VLIW) and Single Instruction Multiple Data (SIMD) emerged. These architectures increase the DSP performance through parallelism. Examples of DSPs with enhanced architectures are TI's TMS320C6xxx DSPs, which was the first DSP to implement the VLIW architecture, and ADI's TigerSHARC, that includes both VLIW and SIMD features. The number of on-chip peripherals increased greatly during this phase, as well as the hardware features that allow many processors to work together. Technologies that allow real-time data exchange between host processor and DSP started to appear towards the end of the 1990s. This constituted a real sea change in DSP system debugging and helped the developers enormously. Another phenomenon observed during this phase was the reduction of the number of DSP manufacturers. The number of DSP families was also greatly reduced, in favour of wider families that granted increased code compatibility between DSPs of

different generations belonging to the same family. Additionally, many DSP families are not ‘general-purpose’ but are focused on specific digital signal processing applications, such as audio equipment or control loops.

2.2 DSP evolution: device integration

Table 2 shows the evolution over the last 25 years of some key device characteristics and their expected values after the year 2010.

Table 2: Overview of DSP device characteristics as a function of time. The last column refers to expected values.

Characteristic	Year	1980	1990	2000	> 2010
Wafer size	[inches]	3	6	12	18
Die size	[mm]	50	50	50	5
Feature	[μm]	3	0.8	0.1	0.02
RAM	[Bytes]	256	2000	32000	1 million
Clock frequency	[MHz]	20	80	1000	10000
Power	[mW/MIPS]	250	12.5	0.1	0.001
Price	[USD]	150	15	5	0.15

Wafer, die, and feature sizes are the basic key factors that define a chip technology. The wafer size is the diameter of the wafer used in the semiconductor manufacturing process. The die size is the size of the actual chips carved up in a wafer. The feature size is the size of the smallest circuit component (typically a transistor) that can be etched on a wafer; this is used as an overall indicator of the density of an Integrated Circuit (IC) fabrication process. The trend in industry is to go towards larger wafers and chip dies, so as to increase the number of working chips that can be obtained from the same wafer; also called yield. For instance, the current typical wafer size is 12 inches (300 mm), and some leading chip maker companies plan to move to 18 inches (450 mm) within the first half of the next decade. (It should be added that the issue is somewhat controversial, as many equipment manufacturers fear that the 18 inches wafer size will lead to scale problems even worse than for the 12 inches.) Feature size is decreasing, allowing one to either have more functionality on a die or to reduce the die size while keeping the same functionality. Transistors with smaller sizes require less voltage to drive them; this results in a decrease of the core voltage from 5 V to 1.5 V. The I/O voltage has been lowered as well, with the caveat that it remains compatible with the external devices used and their standard. A lower core voltage has been one of the key factors enabling higher clock frequencies: in fact, the gap between high and low state thresholds is tightened thus allowing a faster logic level transition. Additionally, the reduced die size and lowered core voltage allow lower power consumption, an important factor for portable or mobile system. Finally, the global cost of a chip has decreased by at least a factor 30 over the last 25 years.

The trend towards a faster switching hardware (including chip over-clocking) and smaller feature size carries the benefit of increased processing power and throughput. There is a downside to it, however, represented by the electromigration phenomenon. Electromigration occurs when some of the momentum of a moving electron is transferred to a nearby activated ion, hence causing the ion to move from its original position. Gaps or, on the contrary, unintended electrical connections can develop with time in the conducting material if a significant number of atoms are moved far from their

original position. The consequence is the electrical failure of the electronic interconnects and the consequent shortened chip lifetime.

2.3 DSP evolution: software tools

The improvement of DSP software tools from the early days until now has been spectacular.

Code compilers have evolved greatly to be able to deal with the underlying hardware complexity and the enhanced DSP architectures. At the same time, they allow the developer to program more and more efficiently in high-level languages as opposed to assembly coding. This speeds up considerably the code development time and makes the code itself more portable across different platforms.

Advanced tools now allow the programming of DSPs graphically, i.e., by interconnecting pre-defined blocks that are then converted to DSP code. Examples of these tools are MATLAB Code Generation and embedded target products and National Instruments' LabVIEW DSP Module.

High-performance simulators, emulator and debugging facilities allow the developer to have a high visibility into the DSP with little or no interference on the program execution. Additionally, multiple DSPs can be accessed in the same JTAG chain for both code development and debugging.

2.4 DSP current scenery

The number of DSP vendors is currently somewhat limited: Analog Devices (ADI), Freescale (formerly Motorola), Texas Instruments (TI), Renesas, Microchip and VeriSilicon are the basic players. Amongst them, the biggest share of the market is taken by only three vendors, namely ADI, TI and Freescale. In the accelerator sector one can find mostly ADI and TI DSPs, hence most of the examples in this document will be focused on them. Table 3 lists the main DSP families for ADI and TI DSPs, together with their typical use and performance.

Table 3: Main ADI and TI DSP families, together with their typical use and performance

Manufacturer	Family	Typical use and performance
TI	TMS320C2x	Digital signal controllers
	TMS320C5x	Power efficient
	TMS320C6x	High performance
ADI	SHARC	Medium performance. First ADI family (now three generations)
	TigerSHARC	High performance for multi-processor systems
	Blackfin	High performance and low power

3 DSP core architecture

3.1 Introduction

DSP architecture has been shaped by the requirements of predictable and accurate real-time digital signal processing. An example is the Finite Impulse Response (FIR) filter, with the corresponding mathematical equation (1), where y is the filter output, x is the input data and a is a vector of filter coefficients. Depending on the application, there might be just a few filter coefficients or many hundreds or more.

$$y(n) = \sum_{k=0}^M a_k \cdot x(n - k) . \quad (1)$$

As shown in Eq. (1), the main component of a filter algorithm is the ‘multiply and accumulate’ operation, typically referred to as MAC. Coefficients data have to be retrieved from the memory and the whole operation must be executed in a predictable and fast way, so as to sustain a high throughput rate. Finally, high accuracy should typically be guaranteed. These requirements are common to many other algorithms performed in digital signal processing, such as Infinite Impulse Response (IIR) filters and Fourier Transforms. Table 4 shows a selection of processing requirements together with the main DSP hardware features satisfying them. These hardware features are discussed in more detail in Sections 3.2 to 3.5 and a full overview of a typical DSP core will be built step by step (see Figs. 4, 7, 10, 13). More detailed information on DSP architectural features can be found in Refs. [7] –[14].

Table 4: Main requirements and corresponding DSP hardware implementations for predictable and accurate real-time digital signal processing. The numbers in the first column refer to the section treating the topic.

Processing requirements	Hardware implementations satisfying the requirement
3.2 Fast data access	<ul style="list-style-type: none"> • High-bandwidth memory architectures • Specialized addressing modes • Direct Memory Access (DMA)
3.3 Fast computation	<ul style="list-style-type: none"> • MAC-centred • Pipelining • Parallel architectures (VLIW, SIMD)
3.4 Numerical fidelity	<ul style="list-style-type: none"> • Wide accumulator registers, guard bits, etc.
3.5 Fast execution control	<ul style="list-style-type: none"> • Hardware-assisted, zero-overhead loops, shadow registers, etc.

3.2 Fast data access

Fast data access refers to the need of transferring data to / from memory or DSP peripherals, as well as retrieving instructions from memory. The hardware implementations considered for this are three, namely a) high-bandwidth memory architectures, discussed in Sub-section 3.2.1; b) specialized addressing modes, discussed in Sub-section 3.2.2; c) direct memory access discussed in Sub-section 3.2.3.

3.2.1 High-bandwidth memory architectures

Traditional general-purpose microprocessors are based upon the Von Neumann architecture, shown in Fig. 4(a). This consists of a single block of memory, containing both data and program instructions, and of a single bus (called data bus) to transfer data and instructions from/to the CPU. The disadvantage of this architecture is that only one memory access per instruction cycle is possible, thus constituting a bottleneck in the algorithm execution.

DSPs are typically based upon the Harvard architecture, shown in Fig. 4(b), or upon modified versions of it, such as the Super-Harvard architecture shown in Fig. 4(c). In the Harvard architecture there are separate memories for data and program instructions, and two separate buses connect them to the DSP core. This allows fetching program instructions and data at the same time, thus providing better performance at the price of an increased hardware complexity and cost. The Harvard

architecture can be improved by adding to the DSP core a small bank of fast memory, called ‘instruction cache’, and allowing data to be stored in the program memory. The last-executed program instructions are relocated at run time in the instruction cache. This is advantageous for instance if the DSP is executing a loop small enough so that all its instructions can fit inside the instruction cache: in this case, the instructions are copied to the instruction cache the first time the DSP executes the loop. Further loop iterations are executed directly from the instruction cache, thus allowing data retrieval from program and data memories at the same time.

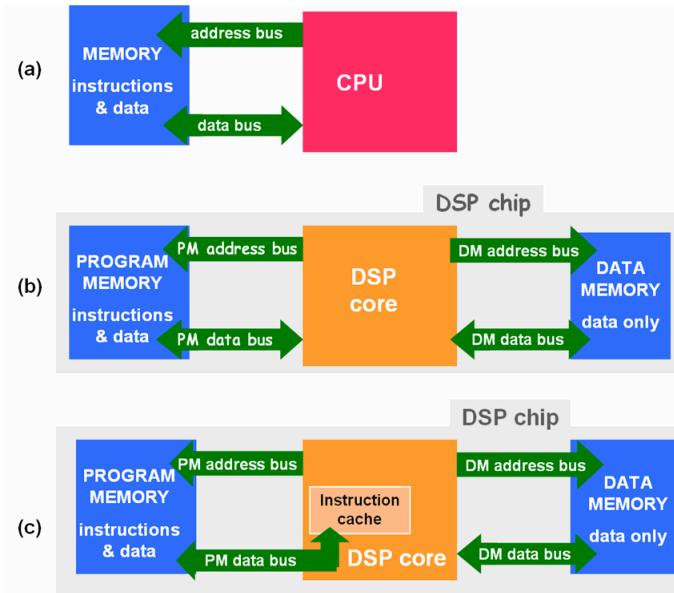


Fig. 4: (a) Von Neumann architecture, typical of traditional general-purpose microprocessors.
(b) Harvard and (c) Super-Harvard architectures, typical of DSPs.

Another more recent improvement of the Harvard architecture is the presence of a ‘data cache’, namely a fast memory located close to the DSP core which is dynamically loaded with data. Of course, the fact of having the cache memory very close to the DSP allows clocking it at high speed, as routing wire delays are short. Figure 5 shows the cache architecture for TI TMS320C67xx DSP, including both program and data cache. There are two levels of cache, called Level 1 (L1) and Level 2 (L2). The L1 cache comprises 8 kbyte of memory divided into 4 kbyte of program cache and 4 kbyte of data cache. The L2 cache comprises 256 kbyte of memory divided into 192 kbyte mapped-SRAM memory and 64 kbyte dual cache memory. The latter can be configured as mapped memory, cache or a combination of the two. The reader can find more information on TI TMS320C67xx DSP two-level memory architecture and configuration possibilities in Ref. [12].

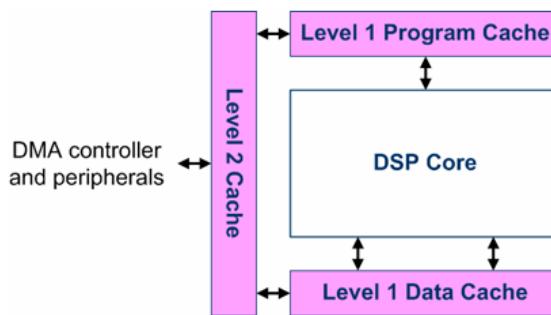


Fig. 5: TI DSP TMS320C67xx family two-level cache architecture

Figure 6 shows the hierarchical memory architecture to be found in a modern DSP [13]. Typical levels of memory and corresponding access time, hardware implementation, and size are also shown. As remarked above, a hierarchical memory allows one to take advantage of both the speed and the capacity of different memory types. Registers are banks of very fast internal memory, typically with single-cycle access time. They are a precious DSP resource used for temporary storage of coefficients and intermediate processing values. The L1 cache is typically high-speed static RAM made of five or six transistors. The amount of L1 cache available thus depends directly on the available chip space. A L2 cache needs typically a smaller number of transistors hence can be present in higher quantities inside the DSPs. Recent years have also seen the integration of DRAM memory blocks into the DSP chip [14], thus guaranteeing larger internal memories with relatively short access times. The Level 3 (L3) memory shown in Fig. 6 is rarely present in DSPs while the external memory is typically available. This is often a large memory with long access times.

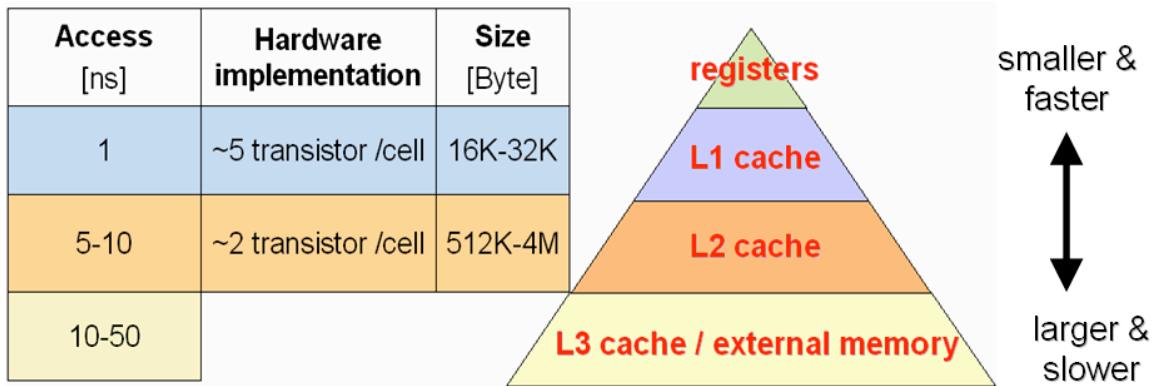


Fig. 6: DSP hierarchical memory architecture and typical number of access clock cycles, hardware implementation, and size for different memory types

As shown above, cache memories improve the average system performance. However, there are drawbacks to the presence of a cache in DSP-based systems, owing to the lack of full predictability for cache hits. A missing cache hit happens when the data or the instructions needed by the DSP are not stored in cache memory, hence they have to be fetched from a slower memory with an execution speed penalty. A situation causing a missing cache hit is, for instance, the flow change due to branch instructions. The consequence is a difficult worst-case-scenario prediction, which is particularly negative for DSP-based systems where it is important to be able to calculate and predict the system time response. There may, however, be methods used to limit these effects, such as the possibility for the user to lock the cache so as to execute time-critical sections in a deterministic way. Advanced cache organizations characterized by a uniform memory addressing are also under study [15].

3.2.2 Specialized addressing modes

DSPs include specialized addressing modes and corresponding hardware support to allow a rapid access to instruction operands through rapid generation of their location in memory. DSPs typically support a wide range of specialized addressing modes, tailored for an efficient implementation of digital signal processing algorithms.

Figure 7 adds the address generator units to the basic DSP architecture shown in Fig. 4(c). As in general-purpose processors, DSPs include a Program Sequencer block, which manages program structure and program flow by supplying addresses to memory for instruction fetches. Unlike general-purpose processors, DSPs include address generator blocks, which control the address generation for specialized addressing modes such as indexing addressing, circular buffers, and bit-reversal addressing. The two last addressing modes are discussed below.

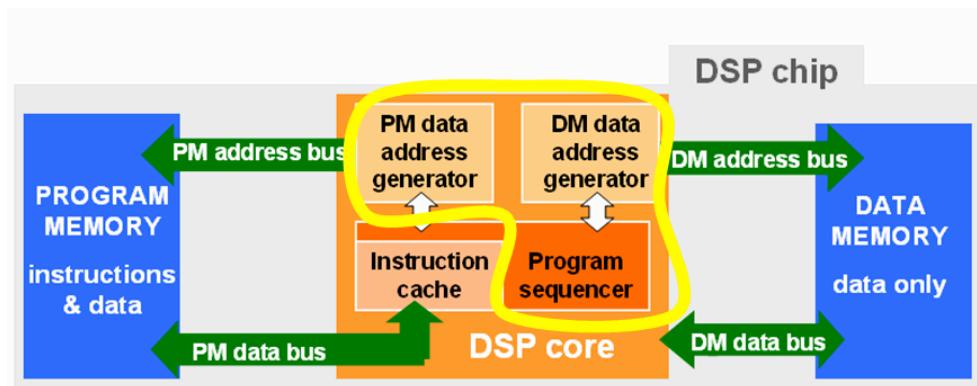


Fig. 7: Program sequencer and address generator units location within a generic DSP core architecture

Circular buffers are limited memory regions where data are stored in a First-In First-Out (FIFO) way; these memory regions are managed in a ‘wrap-around’ way, i.e., the last memory location is followed by the first memory location. Two sets of pointers are used, one for reading and one for writing; the length of the step at which successive memory locations are accessed is called ‘stride’. Address generator units allow striding through the circular buffers without requiring dedicated instructions to determine where to access the following memory location, error detection and so on. Circular buffers allow storing bursts or continuous streams of data and processing them in the order in which they have arrived. Circular buffers are used for instance in the implementation of digital filters; strides higher than one are useful in case of multi-rate signal processing. Figure 8 shows the order in which data are accessed for a read operation in case of an eleven-element circular buffer and with a stride equal to four.

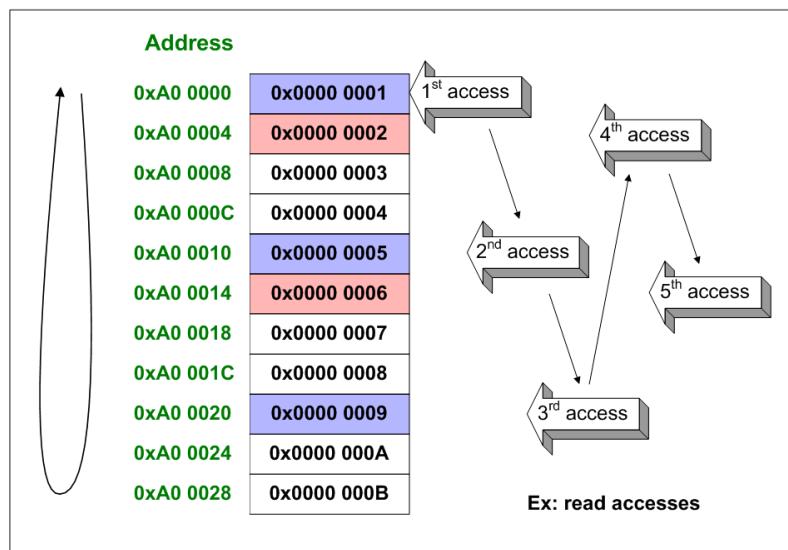


Fig. 8: Example of read data access order in a circular buffer composed of 11 elements and with stride equal to 4 elements

Bit-reversal addressing, shown in Fig. 9, is an essential step in the discrete Fourier transforms calculation. In fact, many implementations of the Fourier transforms require a re-ordering of either the input or the output data that corresponds to reversing the order of the bits in the array index. Figure 9 gives an example of the bit-reversal mechanism. Carrying it out by software is very demanding and

would result in using many CPU cycles, which are saved thanks to the hardware bit-reversal functionality.

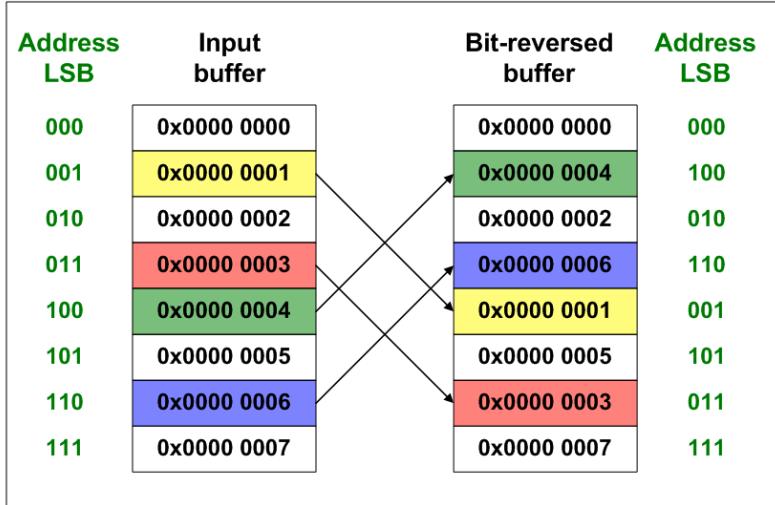


Fig. 9: Bit-reversal mechanism

3.2.3 Direct Memory Access (DMA) controller

The DMA controller is a second processor working in parallel with the DSP core and dedicated to transferring information between two memory areas or between peripherals and memory. In doing so the DMA controller frees the DSP core for other processing tasks. Figure 10 shows an example of the DMA location within a general DSP core architecture.

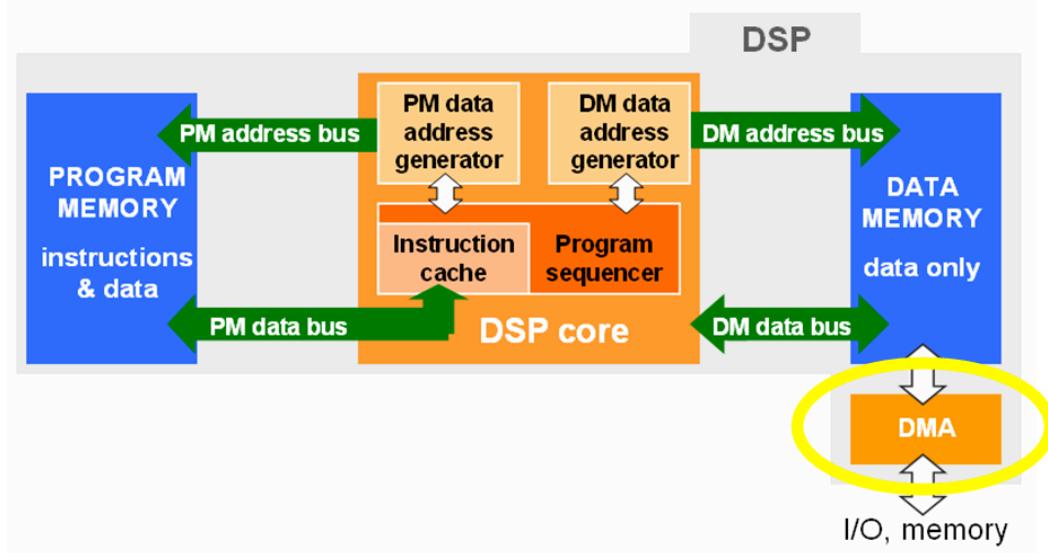


Fig. 10: An example of DMA controller location within a generic DSP core architecture

A DMA coprocessor can transfer data as well as program instructions, the latter transfer corresponding typically to the case of code overlay, i.e., of code stored in an external memory and moved to an internal memory (for instance L1) when needed. Multiple and independent DMA channels are also available for greater flexibility. Bus arbitration between the DMA and the DSP core is needed to avoid colliding memory accesses when the DMA and the DSP core share the same bus to access peripherals and/or memories. To prevent bottlenecks, recent DSPs typically fit DMA controllers with dedicated buses.

Figure 11 shows the advantages of DMA for the DSP core efficient use: the DSP core must set up the DMA but still there is a net gain in the DSP core availability for other processing activities. Nowadays there are two classes of DMA transfer configurations: register-based and RAM-based, the latter one also called descriptor-based. In register-based DMA controllers the transfer set-up is done by the DSP core via the registers set-up. This method is very efficient but allows mainly simple DMA operations. In RAM-based DMA controllers the set-up parameters are stored in memory. This method is preferred by powerful and recent DSPs as it allows great DMA transfer flexibility.

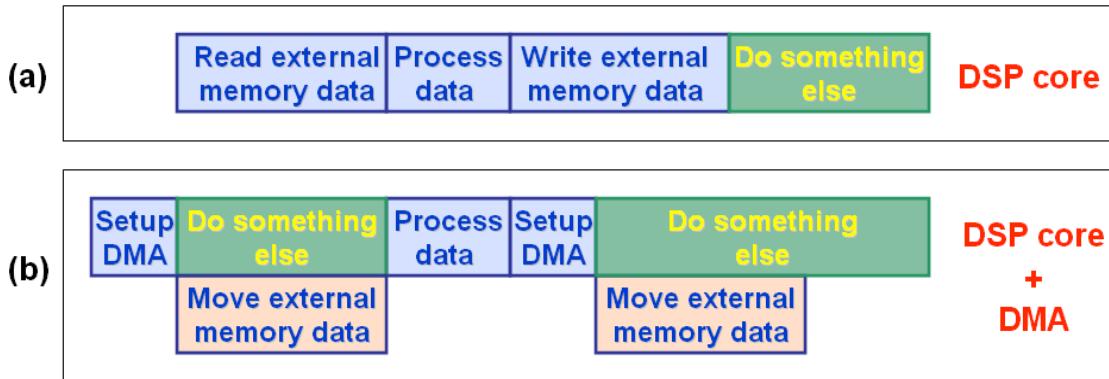


Fig. 11: (a) Read–process–write data when the DSP core only is present; (b) same activity when the DMA takes care of data transfers

Figure 12 provides two examples of transfer configurations. Plot (a) shows a chained DMA transfer, where the completion of a data transfer triggers a new transfer. This type of data transfer is particularly suited to applications that require a continuous data stream in input. Plot (b) shows a multi-dimensional data transfer, obtained by changing the stride of the DMA transfer. This type of data transfer is particularly useful for video applications.

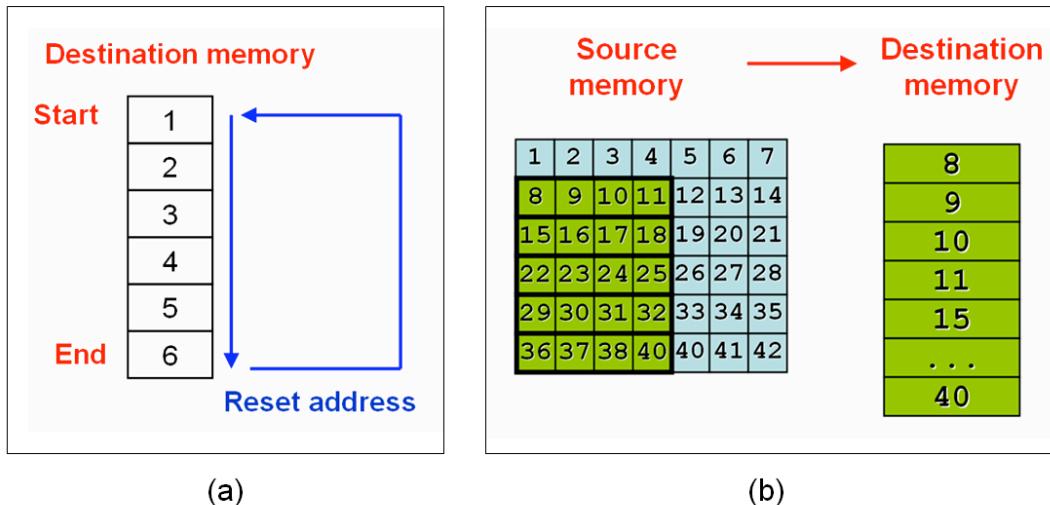


Fig. 12: Examples of DMA transfer configurations. (a): chained DMA transfer; (b): Multi-dimensional data transfer.

DSP external events and interrupts can be used to trigger a DMA data transfer. DMA controllers can also generate interrupts to communicate with the DSP core, for instance to inform it that a data transfer has been completed. An example of a powerful and highly flexible DMA controller is that implemented for TI's TMS320C6000 family [16].

3.3 Fast computation

Here we discuss techniques and architectures used in DSPs for a fast computation. The MAC-centred architecture described in Sub-section 3.3.1 has been common to all DSPs since their early days. The techniques and architectures described in Sub-sections 3.3.2 and 3.3.3 were introduced from the 1990s onwards.

3.3.1 MAC-centred

The MAC operation is used by many digital processing algorithms, as discussed at the beginning of Section 3; consequently its execution must be optimized so as to improve the DSP overall performance. The basic DSP arithmetic processing blocks are a) many registers; b) one or more multipliers; c) one or more Arithmetic Logic Units (ALUs); d) one or more shifters. These blocks work in parallel during the same clock cycle thus optimizing MAC as well as other arithmetic operations. The blocks are shown in Fig. 13 and are briefly described below.

- a) Registers: these are banks of very fast memory used to store intermediate data processing. Very often they are wider than the DSP normal word width, so as to provide a higher resolution during the processing.
- b) Multiplier: it can carry out single-cycle multiplications and very often it includes very wide accumulator registers to reduce round-off or truncation errors. As a consequence, truncation and round-off errors will happen only at the end of the data processing, when the data is stored onto memory. Sometimes an adder is integrated in the multiplier unit.
- c) ALU: it carries out arithmetic and logical operations.
- d) Shifters: it shifts the input value by one or more bits, left or right. In the latter case, the shifter is called a barrel shifter and is especially useful in the implementation of floating point add and subtract operations.

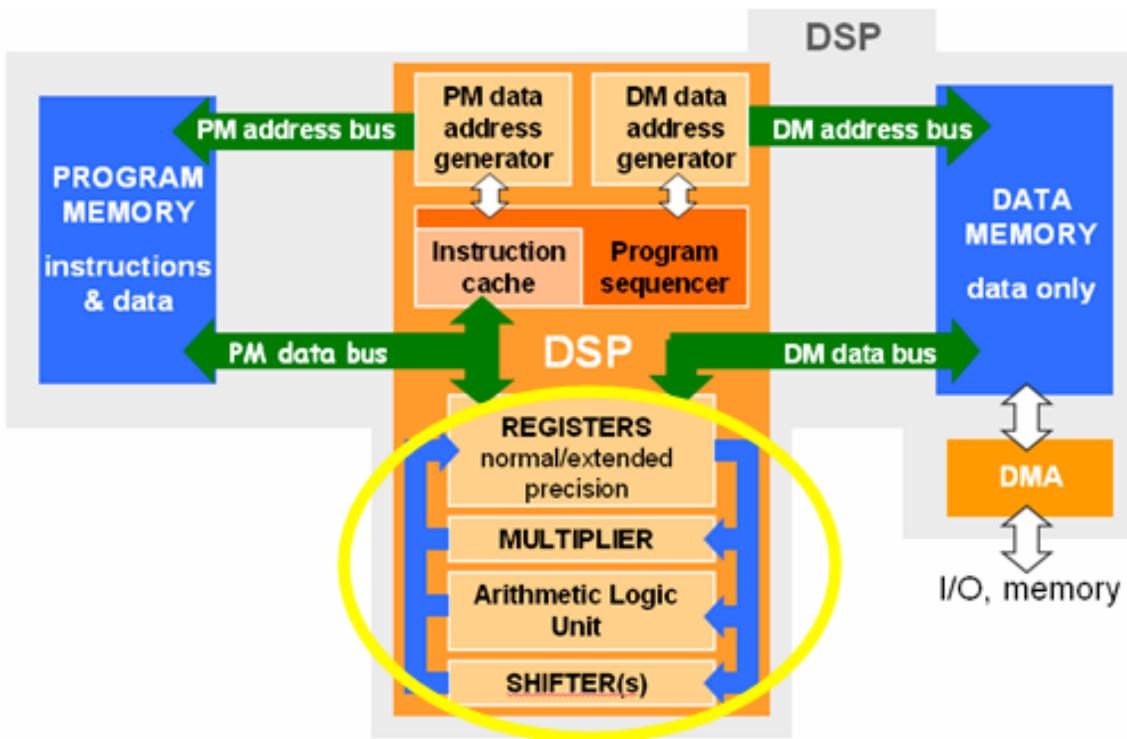


Fig. 13: Basic DSP arithmetic processing blocks. The structure shown is that of ADI SHARC.

3.3.2 Instruction pipelining

Instruction pipelining has become an important element to achieve high DSP performance. It consists of dividing the execution of instructions into different stages and executing the different instructions in parallel stages. The net result is an increased throughput of the instruction execution. The whole process can be compared to a factory assembly line, which produces cars for instance: more than one car is in the assembly line at the same moment, at different stages of assembly. This provides a production higher than the case where only one car at a time is produced, where many specialized crews are idle waiting for the next car to require their work.

Table 5 shows the basic pipelining stage into which each instruction is divided:

1. Fetch. The DSP calculates the address of the next instruction to execute and retrieve the op-code, i.e., the binary word containing the operands and the operation to be carried out on them.
2. Decode. The op-code is interpreted and sent to the corresponding functional unit. The instruction is interpreted and the operands are retrieved.
3. Execute. The instruction is executed and the results are written onto the registers.

Table 5: The three basic pipelining stages and corresponding actions

Basic pipelining stages	Action
Fetch	<ul style="list-style-type: none"> • Generate program fetch address • Read op-code
Decode	<ul style="list-style-type: none"> • Route op-code to functional unit • Decode instruction • Read operands
Execute	<ul style="list-style-type: none"> • Execute instruction • Write results back to registers

Figure 14 shows the advantage of a pipelined CPU with respect to a non-pipelined CPU, in terms of processing time gain. In a non-pipelined CPU the different instructions are executed serially, while in a pipelined CPU only the same type of stages (e.g. Fetch, Decode and Execute) are serialized and different instructions are executed in parallel. A pipeline is called fully-loaded if all stages are executed at the same time; this corresponds to the maximum possible instruction throughput. The depth of the pipeline, i.e., the number of stages into which an instruction is divided, can vary from one processor to another. Generally speaking a deeper pipeline allows the processor to execute faster, hence many processors sub-divide pipeline stages into smaller steps, each one executed at each clock cycle. The smaller the step, the faster the processor clock speed can be. An example of deep pipeline is the TI TMS320C6713 DSP, which includes four fetch stages, two decode stages, and up to ten execution stages.

There are drawbacks and limitations to the pipelining technique. One drawback is the hardware and programming complexity required by it, for instance in terms of capabilities needed in the compiler and the scheduler. This is especially true in the case of deep pipelines. A limitation in the effective instruction execution throughput is given by situations that prevent the pipeline from being fully-loaded. These situations include pipeline flushes due to changes in the program flow, such as code branches or interrupts. In this case, the DSP does not know which instructions it should execute next until the branch instruction is executed. Other situations are data hazards, namely when one instruction needs the result of a previous instruction to be executed. Apart from a reduced throughput,

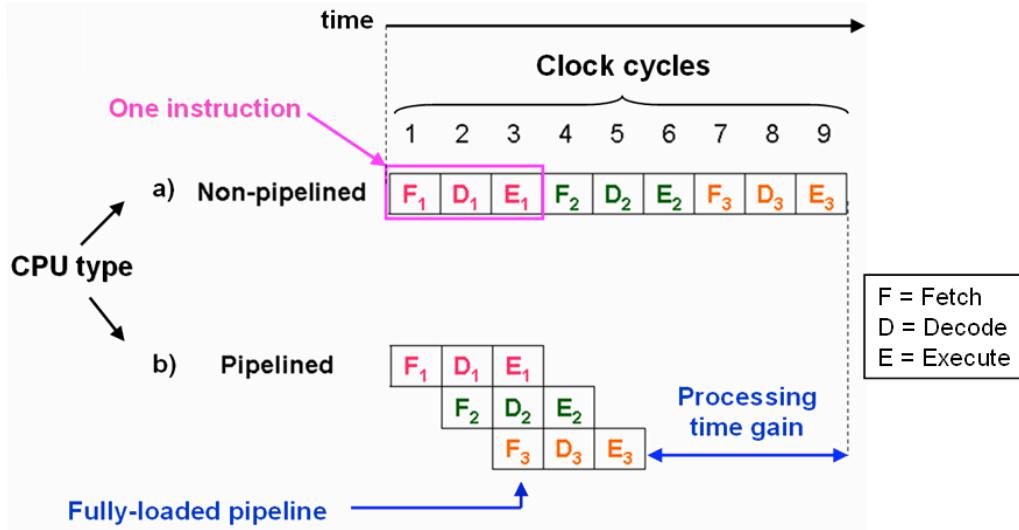


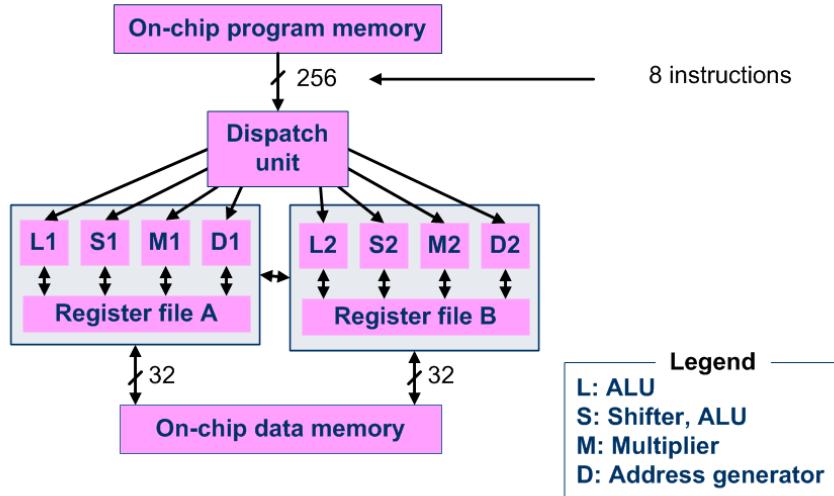
Fig. 14: Instruction execution and processing time gain of a pipelined CPU (plot b) with respect to a non-pipelined one (plot a)

these pipeline limitations cause a more difficult prediction of the worst-case scenario. Techniques not described here are available to provide the DSP programmer with a pipeline control; they include time-stationary pipeline control, data-stationary control, and interlocked pipeline.

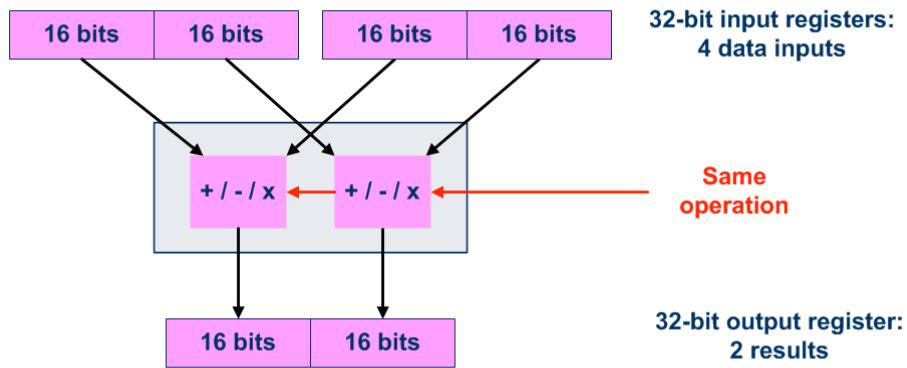
3.3.3 Parallel architectures

The DSP performance can be increased by an increased parallelism in the instructions execution. Parallel-enhanced DSP architectures started to appear on the market in the mid 1990s and were based on instruction-level parallelism, data-level parallelism, or a combination of both. These two approaches are called Very Long Instruction Word (VLIW) and Single-Input Multiple-Data (SIMD), respectively and are discussed below. The reader is referred to Refs. [17] and [18] for more information on the subject.

VLIW architectures are based upon instruction level parallelism, i.e., many instructions are issued at the same time and are executed in parallel by multiple execution units. As a consequence, DSPs based on this architecture are also called ‘multi-issue’ DSP. This is an innovative architecture that was first used in the TI TMS320C62xx DSP family. Figure 15 shows an example of the VLIW architecture: eight, 32-bit instructions are packed together in a 256-bit wide instruction which is fed to eight separate execution units. Characteristics of VLIW architectures include simple and regular instruction sets. Instruction scheduling is done at compile-time and not at run-time so as to guarantee a deterministic behaviour. This means that the decision on which instructions have to be executed in parallel is done when the program is compiled, hence the order does not change during the program execution. A run-time scheduling would instead make the scheduling dependent on data and resources availability, which could change for different program executions. An important advantage of the VLIW architecture is that it can increase the DSP performance for a wide range of algorithms. Additionally, the architecture is potentially scalable, i.e., more execution units could be added to allow a higher number of instructions to be executed in parallel. There are disadvantages as well, such as the high memory use and power consumption required by this architecture. From a programmer’s viewpoint, writing assembly code for VLIW architecture is very complex and the optimization is often better left to the compiler.

**Fig. 15:** TI TMS320C6xxx family VLIW architecture

SIMD architectures are based on data-level parallelism, i.e., only one instruction is issued at a time but the same operation specified by the instruction is performed on multiple data sets. Figure 16 shows the example of a DSP based upon the SIMD architecture: two 32-bit input registers provide four, 16-bit each, data inputs. They are processed in parallel by two separate execution units that carry out the same operation. The two, 16-bit data outputs are packed into a 32-bit register. Typical SIMD architecture can support multiple data width and is most effective on algorithms that require the processing of large data chunks. The SIMD operation mode can be switched ON or OFF, for instance in the ADI SHARC DSP. An advantage of the SIMD architecture is that it is applicable to other architectures; an example is the ADI TigerSHARC DSP that comprises both VLIW and SIMD characteristics. SIMD drawbacks include the fact that SIMD architectures are not useful for algorithms that process data serially or that contain tight feedback loops. It is sometimes possible to convert serial algorithms to parallel ones; however, the cost is in reorganization penalties and in a higher program-memory usage, owing to the need to re-arrange the instructions.

**Fig. 16:** Simplified schematics for ADI SHARC DSP as an example of SIMD architecture

3.4 Numerical fidelity

Arithmetic operations such as additions and multiplications are the heart of DSP systems. It is thus essential that the numerical fidelity be maximized, i.e., that errors due to the finite number of bits used in the number representation and in the arithmetic operations be minimized. DSPs have many ways to obtain this, ranging from the numeric representation to dedicated hardware features.

As far as the number representation is concerned, DSPs can be divided into two categories: fixed point and floating point.

Fixed-point DSPs perform integer as well as fractional arithmetic, and can support data widths of 16, 24 or 32 bits. A fixed-point format can represent both signed and unsigned integers and fractions. Fractional numbers can take values in the $[-1.0, 1.0]$ range and are often indicated as $Qx.y$, where ‘ x ’ indicates the number of bits located before the binary point and ‘ y ’ the number of bits after it. Figure 17(a) shows how 16-bit signed fractional point numbers are coded. Signed fractional numbers with 24-bit and 32-bit data width are coded in an equivalent way as Q1.23 and Q1.31, respectively. They can take values in the same $[-1.0, 1.0]$ range, however, their resolution is higher than the 16-bit implementation. An example of extended precision fixed-point can be found in Ref. [19].

Floating-point DSPs represent numbers with a mantissa and an exponent, nowadays following the IEEE 754 [20] standard shown in Fig. 17(b) for a 32-bit number. The mantissa dictates the number precision and the exponent controls its dynamic range. Numbers are scaled so as to use the full word-length available, hence maximizing the attainable precision. The reader is referred to Ref. [21] for more information on the subject.

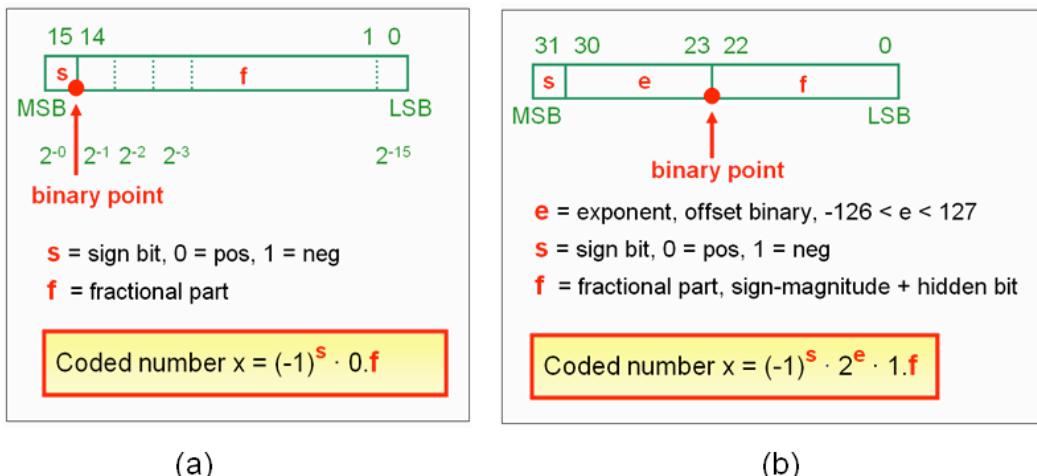


Fig. 17: (a): 16-bit signed fractional point, often indicated as Q1.15. (b): IEEE 754 normalized representation of a single precision floating point number.

Floating-point numbers provide a higher dynamic range, which can be essential when dealing with large data sets and with data sets whose range cannot be easily predicted. The dynamic range for a 32-bit number represented as fixed-point and as floating-point is shown in Fig. 18.

$$\text{Dynamic range}_{\text{dB}} = 20 \log_{10} \left[\frac{\text{largest value}}{\text{smallest value}} \right] \begin{cases} \text{Fixed point } \sim 180 \text{ dB} \\ \text{Floating point } \sim 1500 \text{ dB} \end{cases}$$

Fig. 18: Dynamic range for 32-bit data, represented as 32-bit signed fractional point and IEEE 754 normalized number

In addition to the different number formats available, DSPs provide hardware ways to improve numerical fidelity. One example is represented by the large accumulator registers, used to hold intermediate and final results of arithmetic operations. These registers are several bits (at least four) wider than the normal registers in order to prevent overflow as much as possible during accumulation operations. The extra bits are called guard bits and allow one to retain a higher precision in

intermediate computation steps. Flags to indicate that an overflow/underflow has happened are also available. These flags are often connected to interrupts, thus allowing exception-handling routines to be called. Another means DSPs have to improve numerical fidelity is saturated arithmetic. This means that a number is saturated to the maximum value that can be represented, so as to avoid wrap-around phenomena.

3.5 Fast-execution control

Here we show two important examples of how DSP can fast-execute control instructions. The first example is the zero-overhead hardware loop and refers to the program flow control in loops. The second example refers to how DSPs react to interrupts.

Looping is a critical feature in many digital signal processing algorithms. An important DSP feature is the implementation by hardware of looping constructs, referred to as ‘zero-overhead hardware loop’. This allows DSP programmers to initialize loops by setting a counter and defining the loop bounds, without spending any software overhead to update and test loop counters or branching back to the beginning of the loop.

The capability to service interrupts very quickly and in a deterministic way is an important DSP characteristic. Interrupts are internal (for instance generated by internal timers) or external (brought to the DSP code via pins) events that change the DSP execution flow when they are serviced. The latency is the time elapsed from when the interrupt event is triggered and when the DSP starts to execute the first instruction of the corresponding Interrupt Service Routine (ISR). When an interrupt is received and if the interrupt has a sufficiently-high priority, the DSP must carry out the following actions:

- stop its current activity;
- save the information related to the interrupted activity (called context) into the DSP stack;
- start servicing the interrupt.

The context corresponding to the interrupted activity can be restored when the ISR has been executed and the previous activity is continued.

Table 6: Interrupt dispatchers available on the ADI ADSP21160M DSP. The instruction cycle is 12.5 µs, hence the number of cycles can easily be converted to time.

Interrupt dispatcher	Cycles before ISR	Cycles after ISR
Normal	183	109
Fast	40	26
Super-fast (with alternate registers set)	34	10
Final	24	15

More than one interrupt dispatcher is typically available in a DSP; this means that the user can select the amount of context to be saved, knowing that a higher number of saved registers implies a longer context switching time. An interesting feature available in some DSPs, such as the ADI SHARC AD21160 [22], is the presence of two register sets, called ‘primary’ and ‘alternate’ for all the CPU’s key registers. When an interrupt occurs, the alternate register set can be used, thus allowing a very fast context switch. Table 6 shows the four interrupt dispatchers available on the ADSP21160M DSP and their corresponding latency (‘Cycles before ISR’) and context restore time (‘Cycles after ISR’). The ‘Final’ dispatcher is intended for use with user-written assembly functions or C functions that have been compiled using ‘*#pragma interrupt*’. In particular, this dispatcher relies on the compiler (or assembly routine) to save and restore all appropriate registers.

3.6 DSP core example: TI TMS320C6713

Figure 19 shows TI's TMS320C6713 DSP [23] core architecture, as an example of modern VLIW architecture implementing many of the characteristics described in Section 3. This DSP is that used in the laboratory companion of the lectures upon which this paper is based.

Boxes inside the yellow square belong to the DSP core architecture, which here is considered to include the cache memory as well as the DMA controller. The white boxes are components common to all C6000 devices; grey boxes are additional features on the TMS320C6713 DSP.

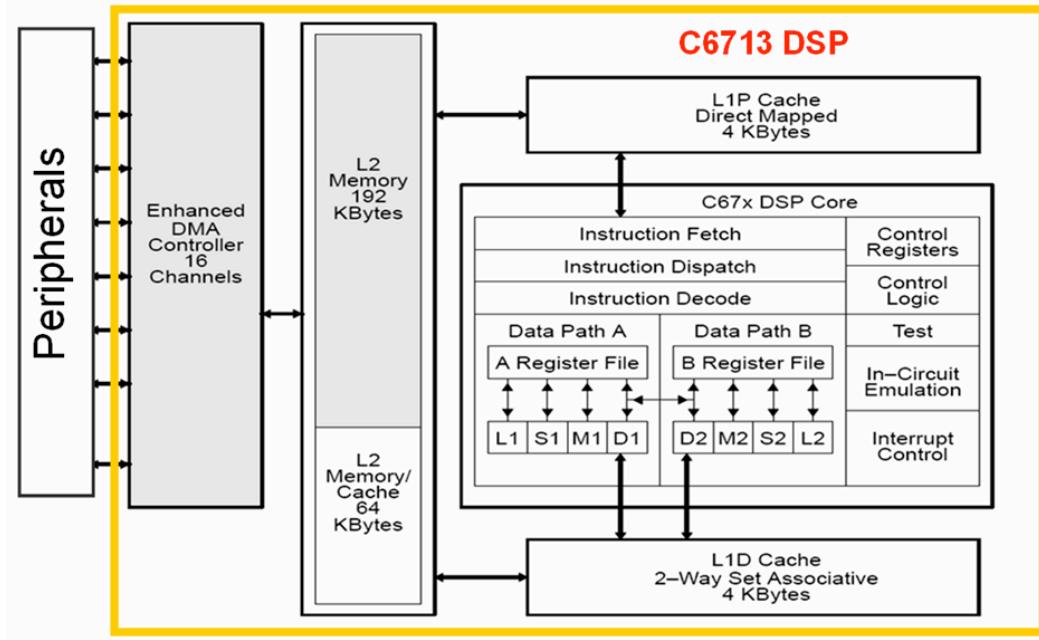


Fig. 19: TI TMS320C6713 DSP core architecture. Picture courtesy of TI [23].

The TMS320C6713 DSP is a floating point DSP with VLIW architecture. The internal program memory is structured so that a total of eight instructions can be fetched at every cycle. To give a numerical example, with a clock rate of 225 MHz the C6713 DSP can fetch eight, 32-bit instructions every 4.4 ns. Features of the C6713 include 264 kBytes of internal memory: 8 kB as L1 cache and 256 kB as L2 memory shared between program and data space. The processing of instructions occurs in each of the two data paths (A and B), each of which contains four functional units (.L, .S, .M, .D). An Enhanced DMA (EDMA) controller supports up to 16 EDMA channels. Four of the sixteen channels (channels 8–11) are reserved for EDMA chaining, leaving twelve EDMA channels available to service peripheral devices.

4 DSP peripherals

4.1 Introduction

The available peripherals are an important factor for the DSP choice. Peripherals are here considered as belonging to two categories:

- interconnect, discussed in Section 4.2;
- services, such as timers, PLL and power management, discussed in Section 4.3.

DSP developers must in fact carefully evaluate the needs of their system in terms of interconnect and services required, to avoid bottlenecks and reduced system performance.

Modern DSPs often have several peripherals integrated on-chip, such as UARTs, serial, USB and video ports. There are benefits in using embedded peripherals, such as fast performance and reduced power consumption. There are, however, drawbacks, in that embedded peripherals can be less flexible across applications and their unit cost might be higher.

The evolution of DSP-supported peripherals has been terrific over the last 20 years. From the original few parallel and serial ports, DSP can now support a wide peripherals range, including those needed by audio/video streaming applications. Often the DSP chip does not have pins to allow using all supported peripherals at the same time. To overcome this limitation, the pins are multiplexed, i.e., the DSP developer must select at boot time which peripherals he/she needs to have available. An example of pin multiplexing referred to TI's TMS320C6713 DSP is given in Section 4.4.

An overview of interconnect and DSP services is given in Sections 4.2 and 4.3, respectively. Hints on different interfacing possibilities to external memories and data converter memories are provided in Sections 4.5 and 4.6, respectively. Finally, a brief outline of the DSP booting process is given in Section 4.7.

4.2 Interconnect

The amount of supported interconnect and data I/O is huge, so only a few examples are given below, divided per interconnect type.

Serial interfaces

- a) Serial Peripheral Interface (SPI): this is an industry-standard synchronous serial link that supports communication with multiple SPI compatible devices. The SPI peripheral is a synchronous, four-wire interface consisting of two data pins, one device select pin, and a gated clock pin. With the two data pins, it allows for full-duplex operation to other SPI compatible devices. An example of DSP fitted with a SPI port is ADI's Blackfin ADSP-BF533 [24].
- b) Multichannel Buffered Serial Ports (McBSP) [25] on TI's DSPs: this serial interface is based upon the standard serial port found in TMS320C2x and TMS320C5x DSPs.
- c) Multichannel Audio Serial Port (McASP) [26] on TI's DSPs: this is a serial port optimized for the needs of multichannel audio applications. Each McASP includes transmit and receive sections that can operate synchronized as well as completely independent, i.e., with separate master clocks, bit clocks, and data stream formats.

Parallel interfaces

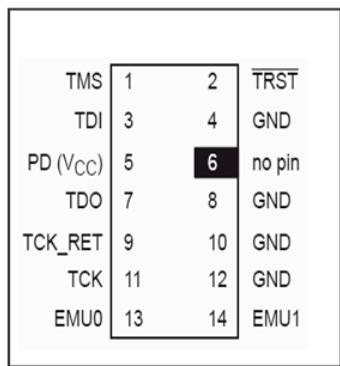
- a) ADI's linkports [27] are parallel interfaces that allow DSP–DSP as well as DSP–peripheral connection. An example of their use for inter-DSP communication to build multi-DSP systems is given in Sub-section 9.3.1.
- b) Parallel Peripheral Interface (PPI) [28] on ADI's Blackfin DSP: this is a multifunction parallel interface, configurable between 8 and 16 bits in width. It supports bidirectional data flow and it includes three synchronization lines and a clock pin for connection to an externally-supplied clock. The PPI can receive data at clock speeds of up to 65 MHz, while transmit rates can approach 60 MHz.

Other interfaces commonly found, for instance in TI DSPs, are Peripheral Component Interconnect (PCI) [29], Inter-Integrated Circuit (I2C) [30], Host-Port Interface (HPI) [31] and General-Purpose Input/Output (GPIO) [32].

4.3 Services

System services provide functionality that is common to embedded systems; the on-chip hardware is generally accompanied by an API that allows one to easily interface to them. A few examples of services are given below.

- Timers: DSPs are typically fitted with one or more general-purpose timers that are used to time or count events, generate interrupts to the CPU, or send synchronization events to a DMA/EDMA controller. More information on timers for TI's TMS320C6000 DSPs can be found in Ref. [33].
- PLL controller: it generates clock pulses for the DSP code and the peripherals from internal or external clock signals. More information on PLL controllers for TI's TMS320C6000 DSPs can be found in Ref. [34].
- Power Management: the power-down logic allows the reduction of clocking so as to reduce power consumption. In fact, most of the operating power of CMOS logic dissipates during circuit switching from one logic state to the other. Significant power can be saved by preventing some of these level switches. More information on power management logic of TI's TMS320C6000 DSPs can be found in Ref. [35].
- Boot configuration: a variety of boot configurations are often available in DSPs. They are user-programmable and determine what actions the DSP performs after it has been reset to prepare for the initialization. These actions include loading the DSP code load from external memory or from an external host. Some boot modes are outlined in Section 4.7. More information on boot modes, device configuration, and available boot processes for TI TMS320C62x/67x is available in Ref. [36].
- JTAG: this interface implements the IEEE standard 1149.1 and allows emulation and debugging. A detailed description of its use can be found in Section 7.2. Figure 20 shows a typical JTAG connector and corresponding signals [37].



Signal	Description	Emulator state	Target state
TMS	Test mode select	OUT	IN
TDI	Test data input	OUT	IN
TDO	Test data output	IN	OUT
TCK	Test clock: 10.368 MHz clock source from emulation cable pod, that can be used to drive the system test clock.	OUT	IN
TRST	Test reset	OUT	IN
EMU0	Emulation pin 0	IN	IN/OUT
EMU1	Emulation pin 1	IN	IN/OUT
PD(Vcc)	Presence detect: it indicates that the emulation cable is connected and that the power is powered up	IN	OUT
TCK_RET	Test clock return, input to the emulator	IN	OUT
GND	Ground		

Fig. 20: Fourteen-pin JTAG header and corresponding signals. Picture courtesy of TI [37].

4.4 TI C6713 DSP example

The peripherals available on TI's TMS320C6713 DSP are shown in Fig. 21 as boxes encircled by a yellow shape. The white boxes are components common to all C6000 devices, while grey boxes are additional features on the TMS320C6713 DSP.

Many peripherals are available on this DSP; however, there are pins that are shared by more than one peripheral and are internally multiplexed. Most of these pins are configured by software via a configuration register, hence they can be programmed to switch functionality at any time. Others (such as the HPI pins) are configured by external pullup/pulldown resistors at DSP chip reset; as a consequence, only one peripheral has primary control of the function of these pins after reset.

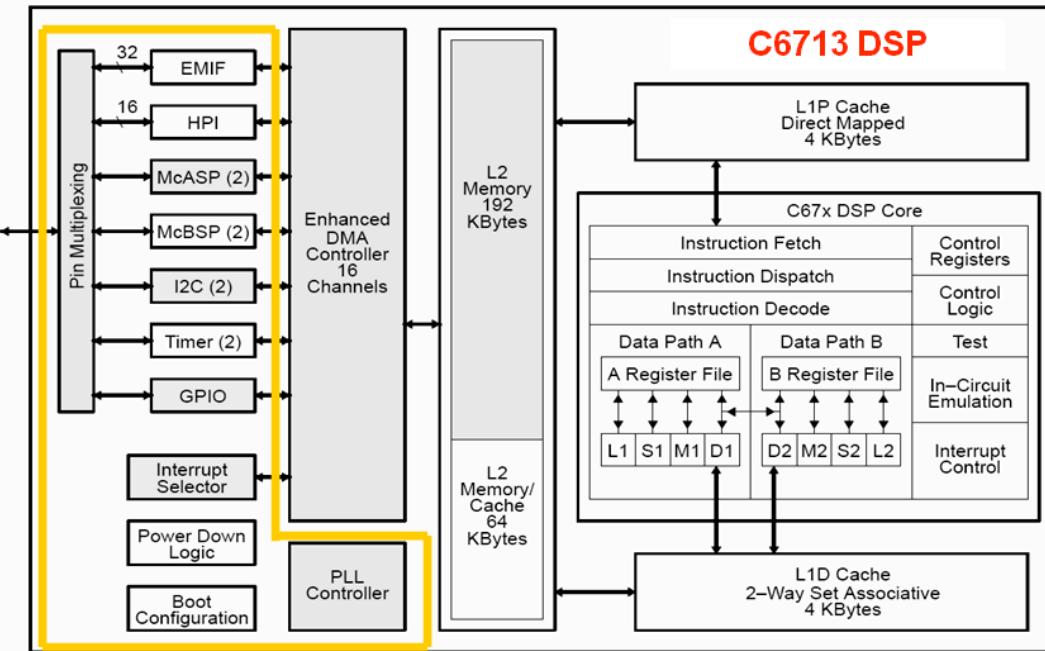


Fig. 21: TI TMS320C6713 DSP available peripherals. Picture courtesy of TI [23].

4.5 Memory interfacing

DSPs often have to interface with external memory, typically shared with host processors or with other DSPs. The two main mechanisms available to implement the memory interfacing are to use hardware interfaces already existing on the DSP chip or to provide external hardware that carries out the memory interfacing. These two methods are briefly mentioned below.

Hardware interfaces are often available on TI as well as on ADI DSPs. An example is TI External Memory Interface (EMIF) [38], which is a glueless interface to memories such as SRAM, EPROM, Flash, Synchronous Burst SRAM (SBSRAM) and Synchronous DRAM (SDRAM). On the TMS320C6713 DSP, for instance, the EMIF provides 512 Mbytes of addressable external memory space. Additionally, the EMIF supports memory width of 8 bits, 12 bits and 32 bits, including read/write of both big- and little-endian devices.

When no dedicated on-chip hardware is available, the most common solution for interfacing a DSP to an external memory is to add external hardware between memory and DSP, as shown in Fig. 22. Typically this is done by using a CPLD or an FPGA which implements address decoding and access arbitration. Care must be taken when programming the access priority and/or interleaved memory access in the CPLD/FPGA. This is essential to preserve the data integrity. Synchronous mechanisms should be preferred over asynchronous ones to carry out the data interfacing.

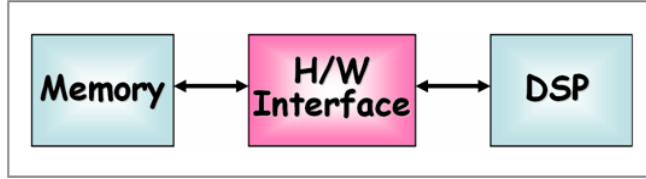


Fig. 22: Generic DSP–external memory interfacing scheme. Very often the h/w interface consists of a CPLD or an FPGA.

4.6 Data converter interfacing

DSPs provide a variety of methods to interface with data converters such as ADCs. On-chip peripherals are a very convenient data transfer mechanism, since data converters are typically much slower than the DSPs they are interfaced with, hence asking the DSP core to directly retrieve data from the converters is a waste of valuable processing time.

Serial interfaces are often available in TI’s DSPs: peripherals such as McBSP and McASP plus the powerful DMA allow an easy interface to many data converter types [39, 40]. Another possible solution for TI DSPs is to use the EMIF in asynchronous mode together with the DMA.

In addition to serial interfaces, ADI Blackfin DSP provides a parallel interface, namely the PPI interface mentioned in Section 4.2, as a convenient way to interact with many converters. This interface typically allows higher sampling rates than the serial interfaces.

A general solution for implementing the DSP–data converter interface is to use an FPGA between DSP and converter, so as to re-buffer the data. An example of this hardware implementation for ADI Blackfin DSPs used in wireless portable terminals is given in Ref. [41]. Additional pre-processing, such as filtering or down-conversion, can also be carried out in the FPGA. This is the case for instance in CERN’s LEIR LLRF system [42], where converters such as ADCs and DACs are hosted on daughtercards. Powerful FPGAs located on the same daughtercards carry out pre-processing and diagnostics actions under full DSP control.

Finally, mixed-signal DSPs, i.e., DSPs with embedded ADCs and/or DACs, are also available. An example of mixed-signal DSP is ADI’s ADSP-21990, containing a pipeline flash converter with eight independent analog inputs and sampling frequency of up to 20 MHz.

4.7 DSP booting

The actions executed by the DSP immediately after a power-down or a reset are called DSP boot and are defined by a certain number of configurable input pins. This paragraph will focus on how the executable file(s) is uploaded to the DSP after a power-down or reset. Two methods are available, which typically correspond to differently built executables. More information on the code building process and on the many file extensions can be found in Section 6.4.

The first method is to use the JTAG connector to directly upload to the executable in the DSP. Upon a DSP power-down the code will typically not be retained in the DSP and another code upload will be necessary. This method is used during the system debugging phase, when additional useful information can be gathered via the JTAG.

On operational systems the DSP loads the executable code without a JTAG cable. Many methods are available for doing this, depending on the DSP family and manufacturer; some general ways are described below.

- No-boot. The DSP fetches instructions directly from a pre-determined memory address, corresponding to EPROM or Flash memory and executes them. On SHARC DSPs, for instance, the pre-defined start address is typically 0x80 0004.

- b) Host-boot. The DSP is stalled until the host configures the DSP memory. For TI TMS320C6xxx DSPs, for instance, this is done via the HPI interface. When all necessary memory is initialized, the host processor takes the DSP out of the reset state by writing in a HPI register.
- c) ROM-boot. A boot kernel is uploaded from ROM to DSP at boot time and starts executing itself. The kernel copies data from an external ROM to the DSP by using the DMA controller and overwrites itself with the last DMA transfer. After the transfer is completed the DSP begins its program execution. Figure 23 visualizes the TI DSP process of booting from ROM memory: the program (shown in green) has been moved from ROM to L2 and L1 Program (L1P) cache via EMIF and DMA.

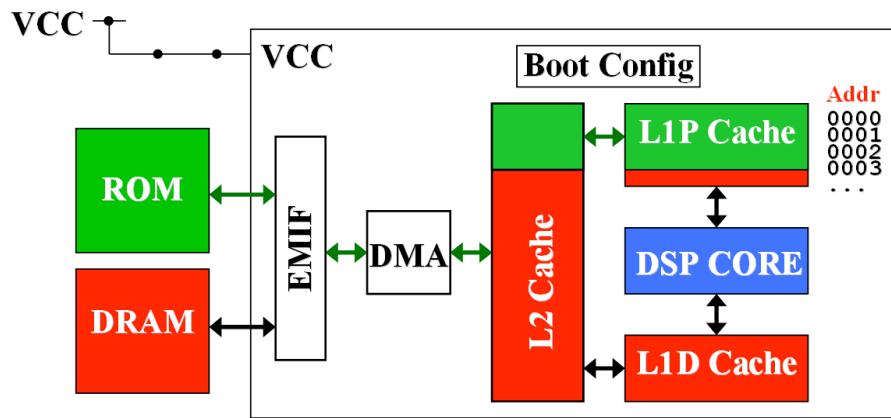


Fig. 23: Example of TI TMS320C6x DSP booting from ROM memory. The picture is courtesy of TI [43].

5 Real-time design flow: introduction

Figure 24 shows a time-ordered view of the various activities or phases that a real-time system developer may be required to carry out during a new system development. These activities will be treated in this document in a didactic rather than in a time-related order, to allow even the unexperienced reader to build up the knowledge needed at each step. It should be underlined that the real-time design flow may be not totally forward-directed, and at each step the developer may have to go back to a previous phase to make modifications or carry out additional tests.

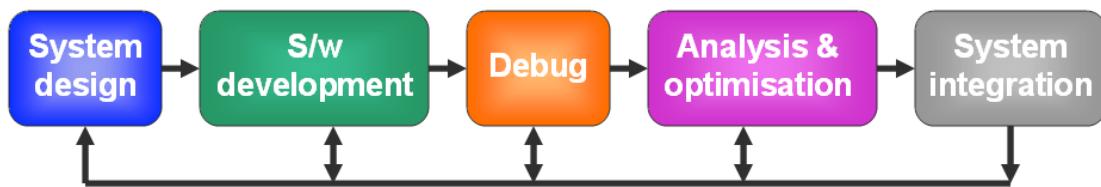


Fig. 24: Activities typically required to develop a new, DSP-based system

The ‘system design’ phase may include both hardware and software design. For hardware design, the developer must make choices such as the DSP type to use, the hardware architecture/interfaces, and so on. For software design, choices such as the code structure, the data flow and data exchange interfaces must be made. This phase is treated in Section 9.

The ‘software development’ phase includes creating the DSP project and writing the actual DSP code. Basic and essential information for this phase is given in Section 6.

The ‘debug’ phase is a very critical one, where the developer must verify that the code executes what it was meant to. Some debugging techniques as well as different methodologies available (such as simulation and emulation) are described in Section 7.

The ‘analysis and optimization’ phase allows the developer to optimize the system for different goals, such as speed, memory, input/output bandwidth, or power consumption. Analysis and optimization tools are described in Section 8, together with some optimization guidelines.

Finally, the ‘system integration’ is the essential phase where the system is integrated within the existing infrastructure and is therefore made fully operational. It is not possible to give precise details on this phase owing to the many existing control infrastructures. However, general guidelines and good practices are discussed in Section 10.

6 Real-time design flow: software development

DSPs are programmed by software via a cross-compilation. This means that the executable is created in a platform (such as a Windows- or a SUN-based machine) different from the one that it runs on, i.e., the DSP itself. One reason for this is that DSPs have limited and dedicated resources, hence it would not be convenient or even possible to run a file system with a user-friendly development environment.

The choice of programming languages is vast, including native assembly language as well as high-level languages such as C, C++, C extensions and dialects, Ada and so on. High-level software tools such as MATLAB and National Instruments allow one to automatically generate code files from graphical interfaces, thus providing rapid prototyping methods.

The code-building tools are very often provided by the DSP manufacturers themselves. Compilers and Integrated Development Environments (IDEs) are also available from other sources, such as Green Hills Software. The trend is now towards more powerful and user-friendly tools, capable of taming and using in the best possible way the underlying hardware and software complexity.

6.1 Development set-up and environment

DSP executables are developed by using Integrated Development Environments (IDEs) provided by DSP manufacturers; they integrate many functions, such as editing, debugging, project files management, and profiling. Very often the licences are bought on a ‘per-project’ basis, even if ADI provides also floating (i.e., networked) licences. The development environment for TI and ADI DSPs are called ‘Code Composer Studio’ and ‘VisualDSP++’, respectively; they provide very similar functionalities. It should be underlined that TI has recently made available free of charge the compiler, assembler, optimizer and linker to non-commercial users. However, neither the IDE nor a debugger were included, thus the developer must still use the proprietary tools.

Figure 25 gives an example of a typical Code Composer screen. On the left-hand side there is the list of all files included in the software project. At the centre of the screen two windows show the code, as a C file (*process.c*) and as assembly code (Dis-assembly window). A breakpoint has been set and the execution is stopped there. Below the code windows, two memory windows are also visible, detailing the data present at addresses 0x80000000 and following, and at addresses 0x40000030 and following. Data at address 0x80000002 is of a different colour because its value changed recently. At the bottom of the IDE screen the following item are displayed: a) the Compile/Link window, which details the results from the last code compilation; b) the Watch window, which displays the value assumed by two C-language variables and c) the Register window, which details the contents of all DSP registers. On the right-hand side there are three graphs: the yellow ones show memory regions, while the green one shows the Fast Fourier Transform of data stored in memory as calculated by the

IDE. The reader can find more details about Code Composer Studio and its capabilities in Refs. [44]–[46].

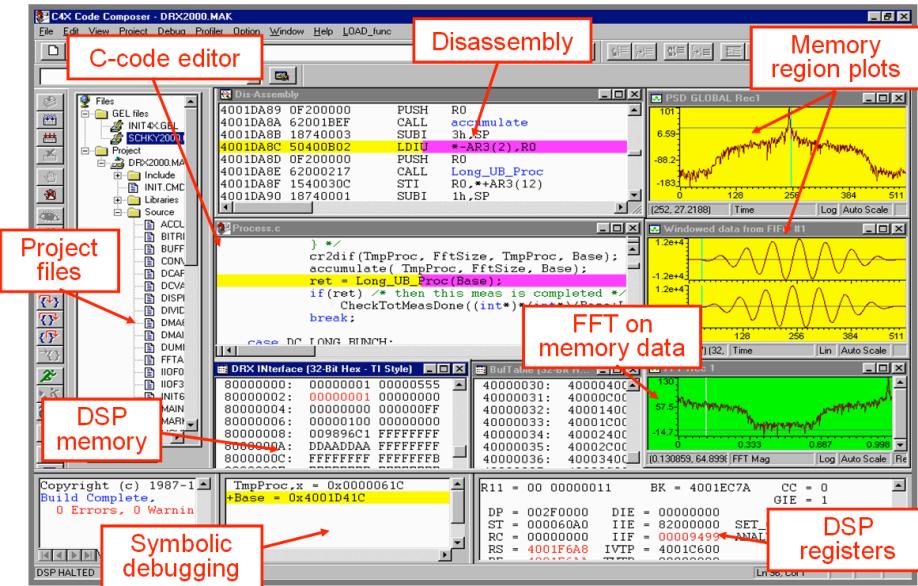


Fig. 25: Screenshot from Code Composer, i.e., the TI DSP IDE. The picture was taken in 1998 from the development of CERN's AD Schottky system.

Figure 26 shows a typical DSP-based system set-up. On the left-hand side the DSP IDE runs on a PC, which is connected to the DSP via a JTAG emulator and pod. This allows one to edit the code, compile it, download it to the hardware and retrieve debug information. On the right-hand side the system exploitation is shown whereby the DSP runs its program and a PowerPC board, running LynxOS and acting as master VME, controls the DSP actions, downloads the control parameters, and retrieves the resulting data. The example shown is that of CERN's AD Schottky measurement system [47].

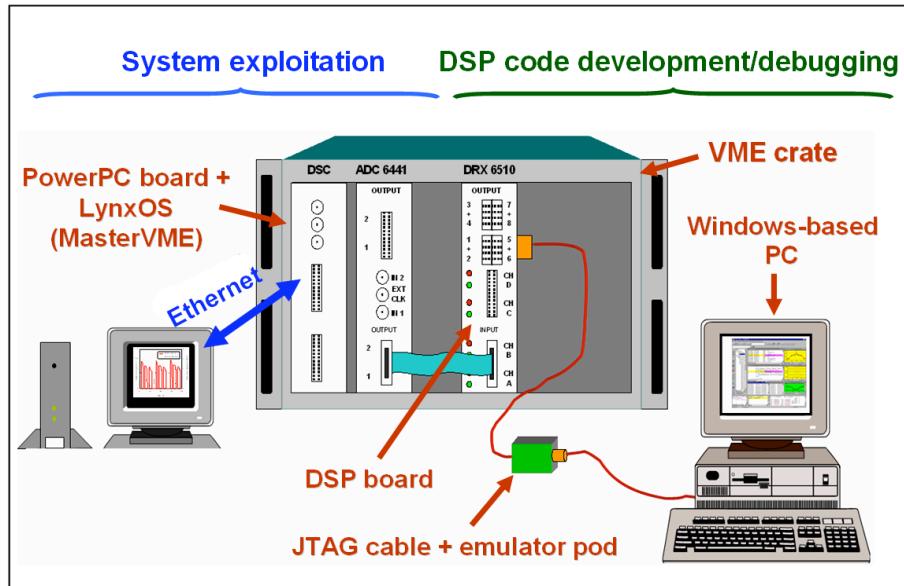


Fig. 26: Typical system exploitation (on the left-hand side) and code development (on the right-hand side) set-ups

6.2 Languages: assembly, C, C++, graphical

The choice of the language(s) to be used for the DSP development is very important and depends mainly on the selected DSP, as different DSPs may support different languages. Often a DSP system will include both assembly and high-level languages; the language choice or the chosen balance between the languages depends also on the required processor workload, i.e., on how much the code should be optimized to satisfy the requirements. The language choice is nowadays much larger than in the past, mainly thanks to the improvements of compilers. Additionally, the increased complexity of DSP hardware (see Section 3), such as deep pipelining, makes the hand-optimization much more difficult. The main language choices include: a) assembly language; b) high-level languages such as C, C dialects/extensions and C++; c) graphical languages such as Matlab. These three choices are discussed below.

6.2.1 Assembly language

The assembly language is very close to the hardware, as it explicitly works with registers and it requires a detailed knowledge of the inner DSP architecture. To write assembly code typically takes longer than to write high-level languages; additionally, it is often more difficult to understand other people's assembly programs than to understand programs written in high-level languages. The assembly grammar/style and the available instruction set/peripherals depend not only on the DSP manufacture, but also on the DSP family and on the targeted DSP. As a consequence, it might be difficult or even impossible to port assembly programs from one DSP to another. For instance, for DSPs belonging to the TI C6xxx family there is about an 85% assembly code compatibility, i.e., when going from a C62x to a C64x DSP there are no issues but if moving from a C64x to a C62x one might have to introduce some changes in the code owing to the different instruction set.

DSP applications have typically very demanding processing requirements. The need to obtain the maximum processing performance has often led DSP programmers to use assembly programming extensively. Nowadays the improvements in code compilers and the increasing difficulty in hand-optimizing assembly code have prompted DSP developers to use high-level languages more often. However, in some DSPs there are still features available only in assembly, such as the super-fast interrupt dispatcher for ADI's ADSP21160M DSP shown in Table 6. Very often, the bulk of the DSP code is written in high-level languages and the parts needing a better performance may be written in assembly.

Different manufacturers adopt different assembly styles, which have also evolved over the years. Table 7 shows a comparison between a traditional assembly style, adopted for instance by TI C40 DSPs, and the algebraic assembly, adopted by ADI SHARC DSPs.

Table 7: Comparison of assembly code styles. The traditional assembly style was adopted for instance by TI TMC320C4x DSPs, while the algebraic assembly is used in ADI.

Operation	Traditional assembly	Algebraic assembly
Move registers contents	mov R7, R0	R7 = R0
Addition	add R0, R1, R2	R0 = R1 + R2
Conditional jump	beq R1, R2, _loc	comp (R1,R2); if eq jump _loc;

Figure 27 gives an example of how one line of C code is converted to the corresponding assembly code for the TI C6317 DSP. The upper window shows part of the ‘SIN_to_output_RTDX.c’ file, which was included in the DSP laboratory companion of the lectures described in this document; the lower ‘Disassembly’ window shows the resulting assembly code.

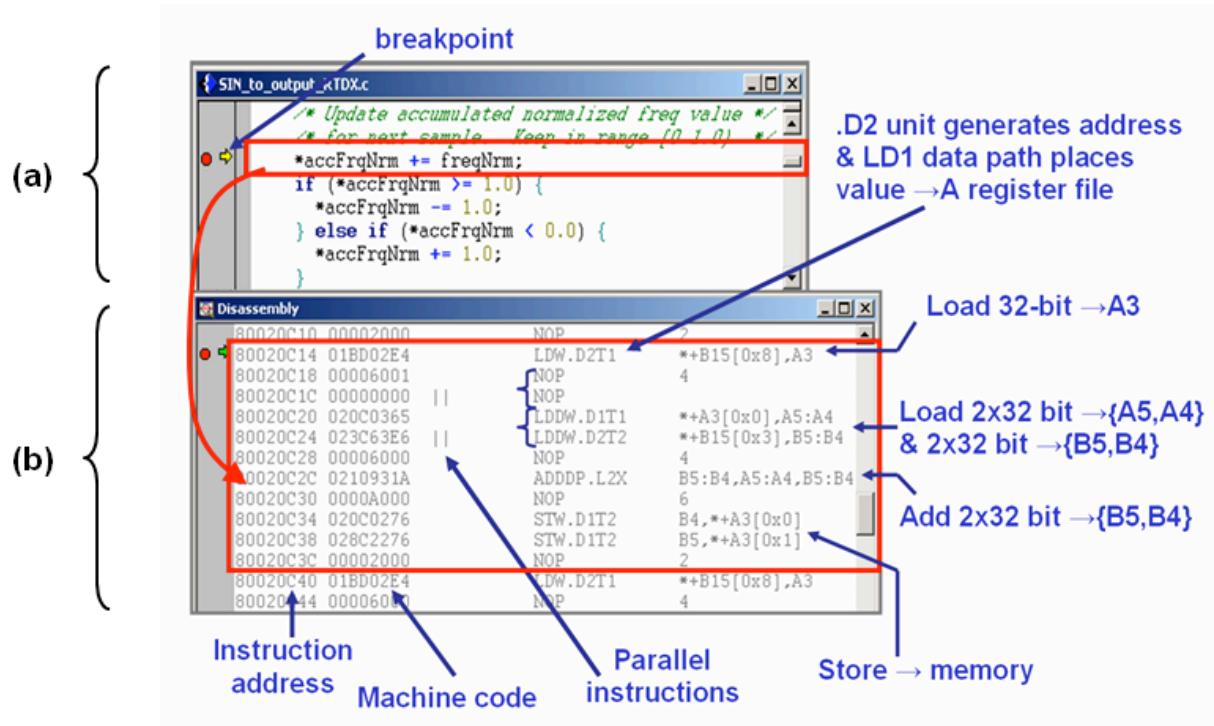


Fig. 27: C and assembly language examples for the TI C6713 DSP. Window (a): C source code. Window (b): assembly code resulting from the first C-code line in window (a).

6.2.2 High-level languages: C

The C language was developed in the early 1970s; three main standards exist, referred to as ANSI, ISO, and C99 respectively. There are many reasons why it is convenient to use the C language in DSP-based systems. The C language is very popular and known by engineers and software developers alike; it is typically easier to understand and faster to develop than assembly code. It supports features useful for embedded systems such as control structures and low-level bit manipulations. All DSPs are provided with a C compiler, hence it may be possible to port the C code from one DSP to another.

There are, however, drawbacks to the use of standard C languages in DSP-based systems. First, the executable resulting from a C-language source code is typically slower than that derived from optimized assembly code and has a larger size. The ANSI/ISO C language does not have support for typical DSP hardware features such as circular buffers or non-flat memory spaces. Additionally, the address at which data must be aligned can vary between different DSP architectures: on some DSPs a 4-byte integer can start at any address, but on other DSPs it could start for instance at even addresses only. As a consequence, the data alignment obtained with ANSI/ISO C compilers may be incompatible with the data alignment required by the DSP, thus leading to deadly bus errors. In the standard C language there is no native support for fixed-point fractional variables, a serious drawback for many DSPs and signal processing algorithms. Finally, the standard C compiler data-type sizes are not standardized and may not fit the DSP native data size, leading for instance to the replacement of fast hardware implementations with slower software emulations. For instance, 64-bit double operations are available in ADI’s TigerSHARC as software emulations only; hence the declaration of

variables as double and not as float will result in slower execution. Table 8 shows how data-type sizes can vary for different DSPs.

Table 8: Examples of data-type size for different DSPs

char size	Processor	int size	Processor
8	ADI Blackfin	16	ADI '21xx, TI 'C54, C55
16	ADI '21xx, TI 'C54, 'C55	24	Freescale 56x
24	Freescale 56x	32	ADI Blackfin, TI 'C6x
32	ADI Blackfin, TI 'C6x	32	ADI SHARC, TigerSHARC
32	ADI SHARC, TigerSHARC		

(a)

(b)

Table 9 shows the data-type sizes and number format for the TI C6713 DSP. The 2's complement and binary formats are used for signed and unsigned numbers, respectively.

Table 9: Data-type sizes and number format for the TI C6713 DSP

TI 'C6713 DSP		
Data type	# bits	Representation
char	8	ASCII
short	16	2's complement / binary
int	32	2's complement / binary
long	40	2's complement / binary
float	32	IEEE 32-bit
double	64	IEEE 64-bit

There are two main approaches to adapting the C language to specific DSPs hardware and to the needs of signal processing applications. The first approach is the definition of ‘intrinsic’ functions, i.e., of functions that map directly to optimized DSP instructions. Table 10 shows some examples of intrinsic functions available in TI C6713 DSPs. The second approach is to ‘extend’ the C-language so as to include specialized data types and constructs. Of course, the drawback of the latter approach is a reduced portability of the resulting C language.

Table 10: TI C6713 intrinsic functions – some examples.

Intrinsic	Description
double _rsqrtp(double src);	Returns approximate 64-bit double square root reciprocal
double _fabs(double src);	Returns absolute value of src
unit _enable_interrupts(void);	Returns previous interrupt state & enables interrupts

6.2.3 High-level languages: C++

The C++ programming language supports object-oriented programming and is the language of choice for many business computer applications. C++ compilers are often available for DSPs; some advantages of using it are the ability to provide a higher abstraction layer and the upwards compatibility with the C language. There are, however, several disadvantages, for instance the increased memory requirements due to the more general constructs. Additionally, many application programs and libraries rely on functions such as *malloc()* and *free()*, which need a heap.

While the way to adapt the C-language to DSPs is to add features, the C++ language is adapted by trimming its features. C++ characteristics typically removed are multiple inheritance and exception handling; the resulting code is more efficient and the executable is smaller.

6.2.4 Graphical languages

A trend which has developed over the last five to ten years is to use graphical programming to generate DSP code. Examples of programs and tools aimed at this are the MATLAB, Hypersignal RIDE (now acquired by National Instruments) and the LabVIEW DSP Module. These methodologies generate DSP executables that often are not highly optimized, therefore not suitable for the implementation of demanding DSP-based systems. However, they allow one to quickly move from the design to the implementation phase, thus providing a rapid prototyping methodology.

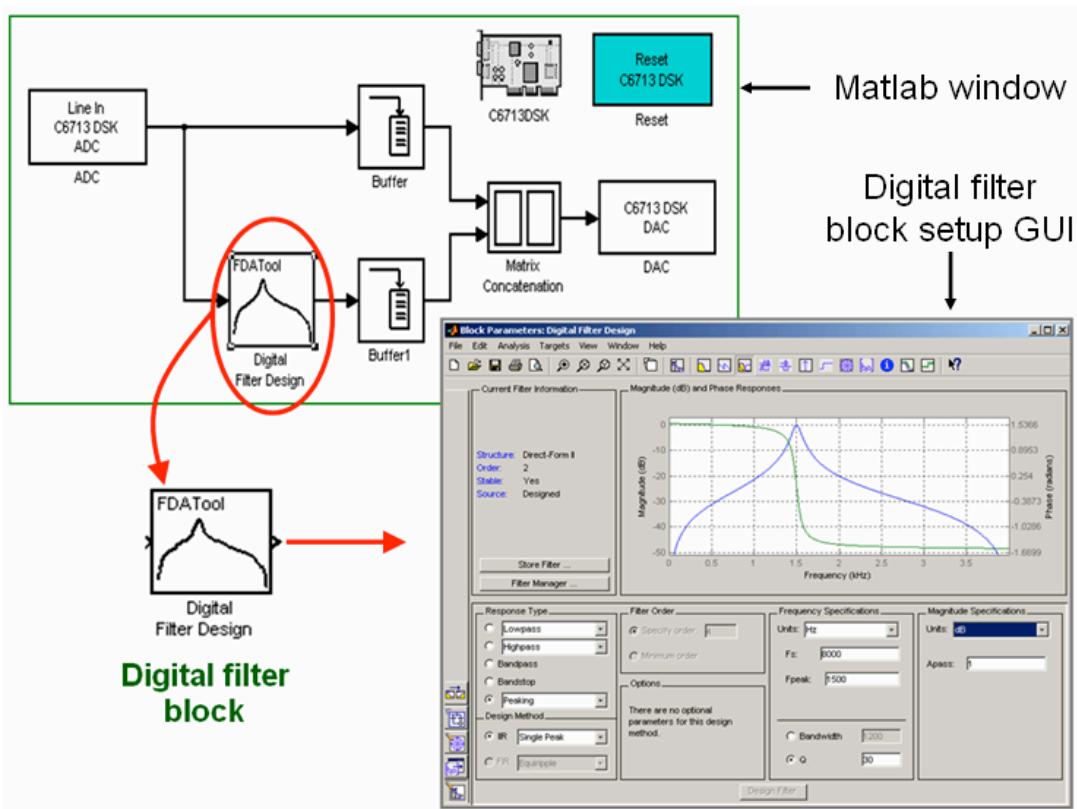


Fig. 28: MATLAB graphical programming used in the DSP laboratory companion in these notes. The digital filter block can easily be set up by using a user-friendly set-up GUI.

As an example, MATLAB provides tools such as Simulink, Real-Time Workshop, Real-Time Workshop Embedded Coder, Embedded Target for TI C6xxx DSPs and Link for Code Composer that allow generating embedded code for TI DSPs and downloading it directly into a DSP evaluation

board. These tools provide interfaces for the DSP peripherals, too. The DSP laboratory companion on these notes was based upon TI C6713 DSK and MATLAB tools. Figure 28 shows the MATLAB graphical program that constituted one of the laboratory exercises. MATLAB allows not only to interface immediately with the on-board CODEC by using the ADC and DAC blocks, but also to set up through a user-friendly GUI the digital filter to be implemented.

6.3 Real-time operating system

A Real-Time Operating System (RTOS) is a program that has real-time capabilities, is downloaded to the DSP at boot time, and manages all DSP programs, typically referred to as tasks. The RTOS interfaces tasks with peripherals such as DMA, I/O and memory, via an Application Program Interface (API), as shown in Fig. 29.

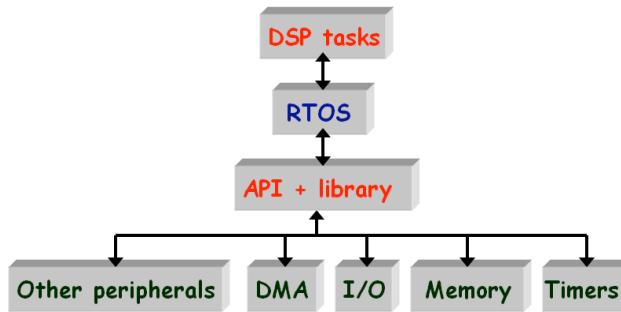


Fig. 29: Embedded DSP software components

A RTOS is typically task-based and supports multiple tasks (often referred to as threads) by time-sharing, i.e., by multiplexing the processor time over the active tasks set. Each task has a priority associated to it and the RTOS schedules which task should run depending on the priority. Very often this is done in a pre-emptive way, meaning that when a high-priority task becomes ready for execution, it pre-empts the execution of a lower-priority task, without having to wait for its turn in the regular re-scheduling. Finally, RTOS have a small memory footprint, so as not to have too negative an impact on the DSP executable size.

There are many advantages when using a RTOS to develop a DSP-based system. For instance, the API and library shown in Fig. 29 provide a device abstraction level between DSP hardware features and task implementation, thus allowing a DSP developer to focus on the task rather than the hardware interface's design and coding. The DSP developer may have to just call different interfacing functions in case the code should be ported to a different DSP, hence easing code portability. A RTOS manages the task's execution hence the developer can cleanly structure the code, define appropriate priority levels for each task, and insure that their execution meets critical real-time deadlines. System debug and optimization can be improved, and memory protection can often be provided. There are, however, drawbacks to the use of RTOS. As an example, a RTOS uses DSP resources, such as processing time and DSP timers, for its own functioning. Additionally, the RTOS turnover is typically quite high and royalties are often required from developers.

Many RTOS are available at any time, typically targeted to a precise DSP family or processor. Examples are TALON RTOS from Blackhawk, targeted at TI DSPs, and INTEGRITY RTOS from Green Hills Software or NUCLEUS RTOS from Accelerated Technology, targeted at ADI Blackfin DSPs. It is worth mentioning Linux-based OS, such as RT-Linux, RTAI and uLinux. Both RT-Linux and RTAI use a small real-time kernel that runs Linux as a real-time task with lower priority. The last RTOS listed above, uLinux, is a soft-time OS adapted to ADI Blackfin DSPs. uLinux cannot always guarantee RTOS capabilities such as a deterministic interrupt latency; however, it can typically satisfy the needs of commercial products, where time constraints are often on the millisecond order as dictated by the ability of the user to recognise glitches in audio and video signals.

Other RTOS worth mentioning are those provided and maintained by DSP manufacturers. Both TI and ADI provide royalties-free RTOS with similar characteristics, such as a small memory footprint, multi-tasks and multi-priority levels support. They are called DSP/BIOS [48] for TI and VisualDSP++ Kernel (VDK) for ADI, and can optionally be included in the DSP code. In particular, TI DSP/BIOS provides thirty priority levels and four classes of execution threads. The thread classes, listed in order of decreasing priority, are Hardware Interrupts (HWI), Software Interrupts (SWI), Tasks (TSK) and Background (IDL). Figure 30 shows how the processing time is shared between different threads in TI DSP/BIOS. In the vertical scale the different threads are ordered by priority, the higher up having more priority; in the horizontal scale the time is shown. Software interrupts can be pre-empted by a higher-priority software interrupt or by a hardware interrupt. Same-level interrupts are executed in a first-come, first-served way. Tasks are capable of suspension (see Task TSK2 in Fig. 30) as well as of pre-emption.

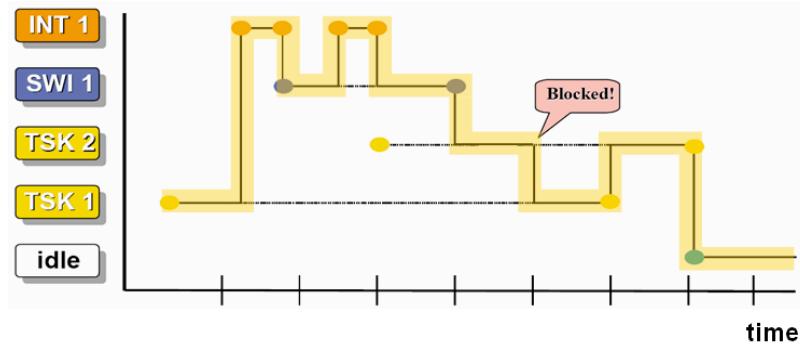


Fig. 30: DSP/BIOS prioritized thread execution example. Image courtesy of Texas Instruments [48].

6.4 Code-building process

The DSP code-building process relies on a set of software development tools, typically provided by DSP manufacturers.

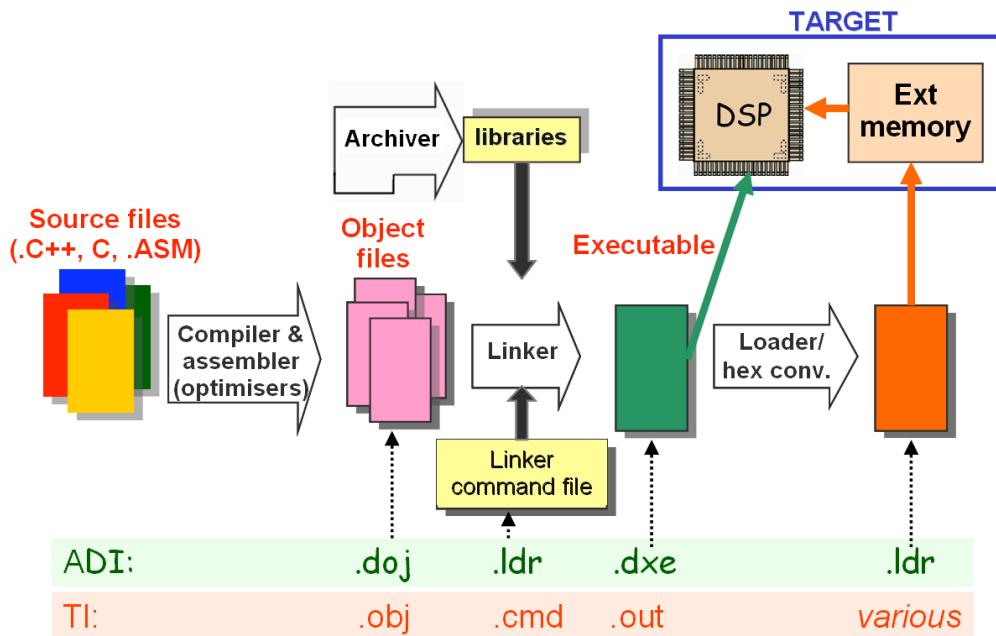


Fig. 31: Main elements of the code building process. Typical file extensions for ADI and TI DSPs are shown at the bottom of the picture.

Figure 31 shows the main elements and tools needed for the code-building process. Source files are converted to object files by the compiler and the assembler. Archiver tools allow the creation of libraries from object files; these libraries can then be linked to object files to create an executable. The executable can be directly downloaded from the IDE to the target DSP via a JTAG interface; as an alternative, the executable can be converted to a special form and loaded to a memory external to the DSP, from which the DSP itself will boot. The first approach is typically used during the DSP development phase, while the second approach is more convenient during system exploitation. Finally, the file extensions used at the different code-building process steps for ADI and TI DSPs are shown at the bottom of Fig. 31.

Three tools, namely compiler, assembler, and linker, are used to generate executable code from C/C++ or assembly source code. Figure 32 shows their use in the code-building process on TI DSPs. The tools' main characteristics are summarized in Sub-sections 6.4.1 to 6.4.3.

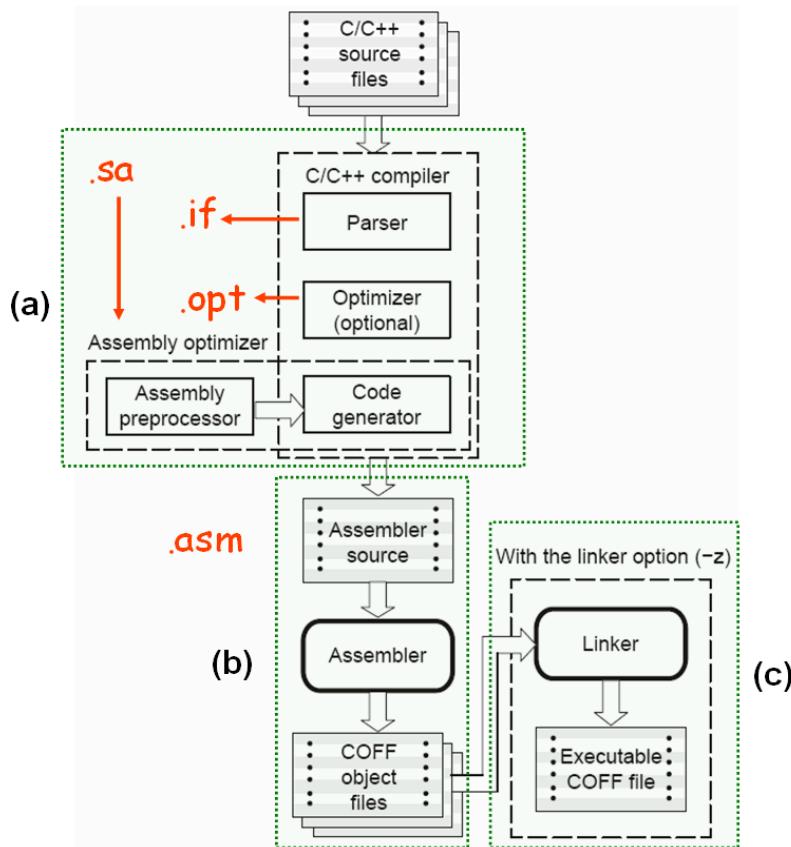


Fig. 32: Generic code-building processing: (a) compiler; (b) assembler; (c) linker. The picture is courtesy of TI [49].

6.4.1 C / C++ compiler for TI C6xxx DSPs [49]

The C/C++ compiler generates C6xxx assembler code (.asm extension) from C, C++ or linear assembly source files. The compiler can perform various levels of optimization: high-level optimization is carried out by the optimizer, while low-level, target-specific optimization occurs in the code generator. Finally, the compiler includes a real-time library which is non-target-specific.

6.4.2 Assembler for TI C6xxx DSPs [50]

The assembler generates machine language object files from assembly files; the object files format is the Common Object File Format (COFF). The assembler supports macros both as inline functions and

taken from a library; it also allows segmenting the code into sections, a section being the smaller unit of an object file. The COFF basic sections are

- a) text for the executable code;
- b) data for the initialized data;
- c) bss for the un-initialized variables.

6.4.3 *Linker for TI C6xxx DSPs [50]*

The linker generates executable modules from COFF files as input. It resolves undefined external references and assigns the final addresses to symbols and to the various sections. A DSP system typically includes many types of memory and it is often up to the programmer to place the most critical program code and data into the on-chip memory. The linker allows allocating sections separately in different memory regions, so as to guarantee an efficient memory access. An example of this is shown in Fig. 33.

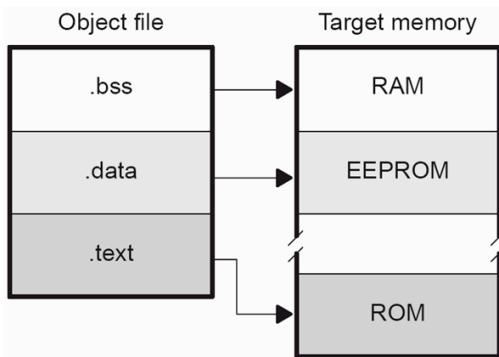


Fig. 33: Example of sections allocation into different types of target memory

The linker also allows one to clearly implement a memory map shared between DSP and host processor; this is essential for instance to exchange data between them.

7 Real-time design flow: debugging

The debugging phase is the most critical and least predictable phase in the real-time design flow, especially for large systems. The debugging capabilities of the development environment tools can make the difference between creating a successful system and spiralling into an endless search for elusive bugs.

The starting point of this phase is an executable code, i.e., a code without compilation and linker errors; the goal is to ascertain that the code behaves as expected. The debugging tools and techniques have a strong impact on the amount of time and effort needed to validate a DSP code.

There are many types of bugs: they can be repeatable or intermittent, the latter being much tougher to track down than the first ones. Bugs can be due to the code implementation, such as logical errors in the source code, or can derive from external problems, i.e., hardware misbehaviours. The approaches and the tools to debug a DSP code include simulation, emulation, and real-time debugging techniques. Simulation tools allow running the DSP code on a software simulator fitted with full visibility into DSP internal registers. Emulation tools embed debug components into the target to allow an information flow between target and host computer. Real-time debugging techniques allow a real-time data exchange between host and target without stopping the DSP. These techniques are described in detail in Sections 7.1. to 7.3.

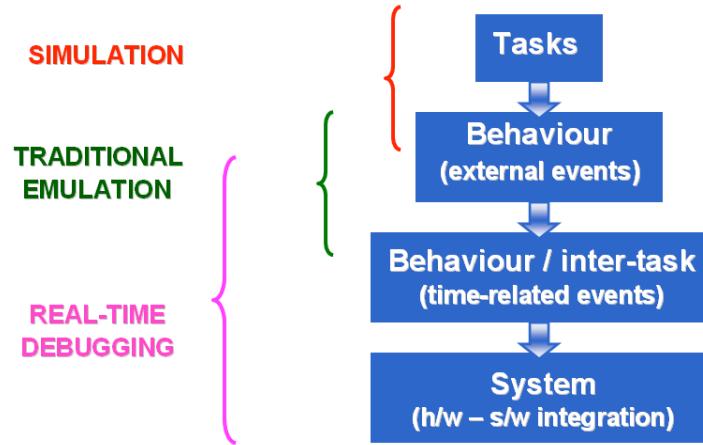


Fig. 34: Debug steps and their suggested sequencing. The debug tools suited to different steps are also shown.

The developer should not attempt to debug the DSP code as a whole, unless the code itself is relatively short and simple. He is instead recommended to debug the code in several steps: Fig. 34 shows an example of steps and of their sequencing, together with the appropriate debug tools and techniques. First, single tasks such as functions and routines should be validated; this step can be carried out via simulation only. Second, the behaviour of sub-systems or specific parts of the code can be tested with respect to external events, such as ISR triggering. This part can be carried out with the help of traditional emulation techniques. Third, the behaviour of many tasks can be validated with respect to real-time constraints, such as the proper frequency of ISR triggering. Once all system components have been validated, the whole system can be tested. These last two steps profit particularly from real-time debugging techniques.

7.1 Simulation

DSP software simulators have been available for more than fifteen years. They can simulate CPU instruction sets as well as peripherals and interrupts, thus allowing DSP code validation at a reduced cost and even before the hardware the code should run on is available. Simulators provide a high visibility into the simulated target, in that the user can execute the code step by step and look at the intermediate values taken by internal DSP registers. Large amount of data can be collected and analysed; resource usage can be evaluated and used for an optimized hardware design.

Simulators are highly repeatable, since the same algorithm can be run in exactly the same way over and over. The reader should note that this kind of repeatability is difficult to obtain with other techniques, such as emulation, as external events (for instance interrupts) are almost impossible to be precisely repeated with hardware. Simulators may also allow measurement of the code execution time, with limitations due to the type of simulator chosen. A useful feature available with the TI C5x and C6x simulators is the ‘rewind’ [51], which allows viewing the past history of the application being executed on the simulator.

The main limitation common to DSP simulators is their execution speed, several orders of magnitude slower than the target they simulate; in particular, the more accurate the modelling of the DSP chip and corresponding peripherals, the slower the simulation. DSP tool vendors have overcome this problem by providing different simulators for the same DSP, providing a different level of chip and peripherals modelling. Figure 35 shows some simulators available for TI DSPs. The reader should notice that TI provides up to three simulators for each DSP, namely:

- a) CPU Cycle Accurate Simulator: This simulator models the instruction set, timers, and external interrupts, allowing the debugging and optimization of the program for code size and CPU cycles.
- b) Device Functional Simulator: This simulator not only models instruction set, timers, and external interrupts, but also allows features such as DMA, Interrupt Selector, caches and McBSP to be programmed and used. However, the true cycles of a DMA data transfer are not simulated.
- c) Device Cycle Accurate Simulator: This simulator models all peripherals and caches in a cycle-accurate manner, thus allowing the user to measure the total device and stall cycles used by the application.

More information on TI simulators can be found in Ref. [52].

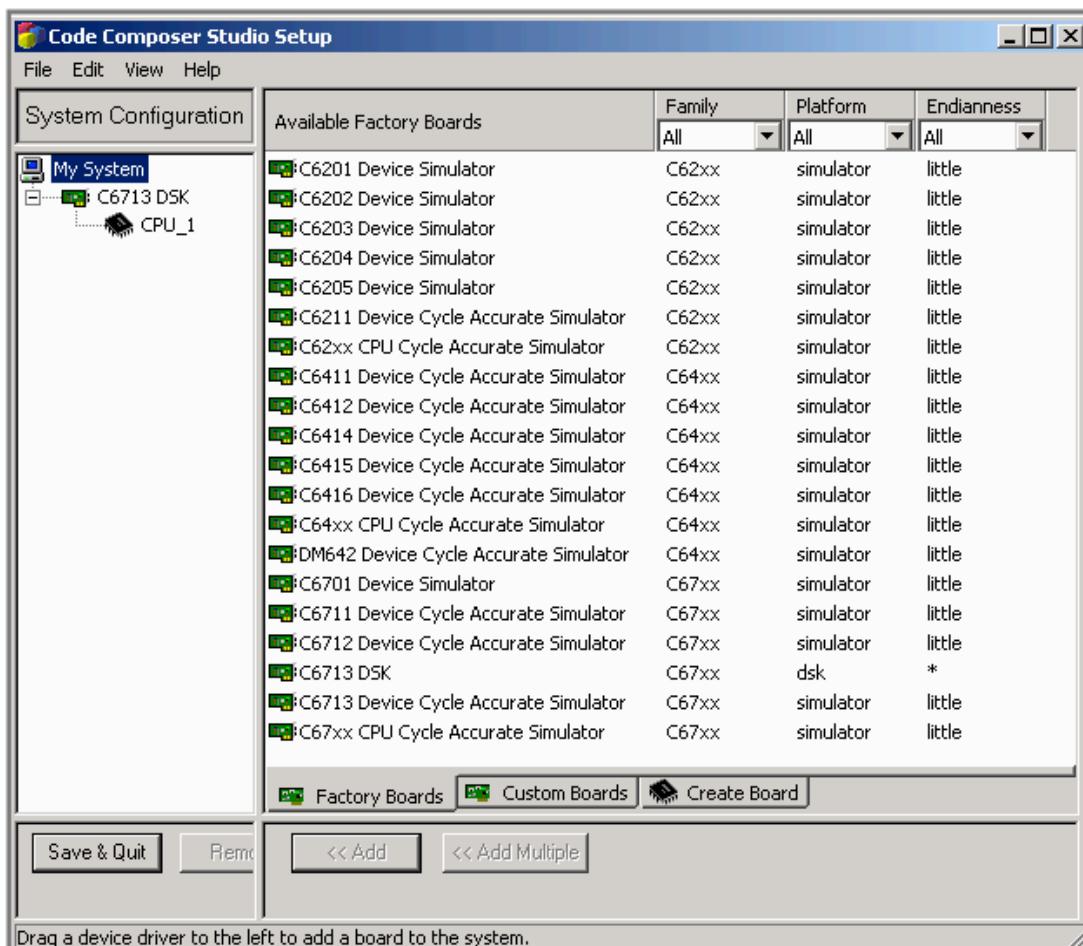


Fig. 35: Example of DSP simulators available with TI's Code Composer Studio development environment

7.2 Emulation

The integration of processor, memory, and peripherals in a single silicon chip is commonly referred to as System-On-a-Chip (SOC). This approach allows reducing the physical distance between components, hence devices become smaller in size, run faster, cost less to manufacture, and are typically more reliable. From a DSP code developer's viewpoint, the main disadvantage of this approach is the lack of access to embedded signals, often referred to as vanishing visibility. In fact,

many chip packages (e.g., ball grid array) do not allow probing the chip pins; additionally, internal chip busses are often not even available at the chip pins. Emulation techniques [53] restore the visibility needed for code debugging by embedding debug components into the chip itself.

There are three main kinds of emulation, namely:

- a) Monitor-based emulation: A supervisor program (called monitor) runs on the DSP and uses one of the processor's input-output interfaces to communicate with the debugger program running on the host. The debugging capabilities of this approach are more limited than those provided by the two other approaches; additionally, the monitor presence changes the state of the processor, for instance regarding the instruction pipeline. The advantage is that it does not require emulation hardware, hence its cost is lower.
- b) Pod-based In Circuit Emulation (ICE): The target processor is replaced by a device that acts like the original device, but is provided with additional pins to make accessible and visible internal structures such as internal busses. This emulation approach has the advantage of providing real-time traces of the program execution. However, replacing the target processor with a different and more complex device may create electrical loading problems. Additionally, this solution is quite costly, the hardware is different from the commercialized product and becomes quite difficult to implement at high processor speed.
- c) Scan-based emulation: Dedicated interfaces and debugging logic are incorporated into commercially-available DSP chips. This on-chip logic is responsible for monitoring the chip's real-time operations, for stopping the processor when for instance a breakpoint is reached, and for passing debugging information to the host computer. An emulation controller controls the flow of information to /from the target and can be located either on the DSP board or on an external pod. Many types of target-host interface exist. On the DSP board one can typically find a JTAG (IEEE standard 1149.1) connector. On the host computer, parallel or USB ports are often available.

The scan-based emulation technique has been widely preferred over the other two since the late 1980s and is nowadays available on the vast majority of DSPs. Figure 36 shows the TI XDS560 emulator, composed of a PC card, a cable with JTAG interface to the target, and an emulation controller pod. Many emulators are available on the market, with different interfaces and characteristics. As an example, it is worth mentioning Spectrum Digital's XDS510 USB galvanic JTAG emulator, which provides voltage isolation.

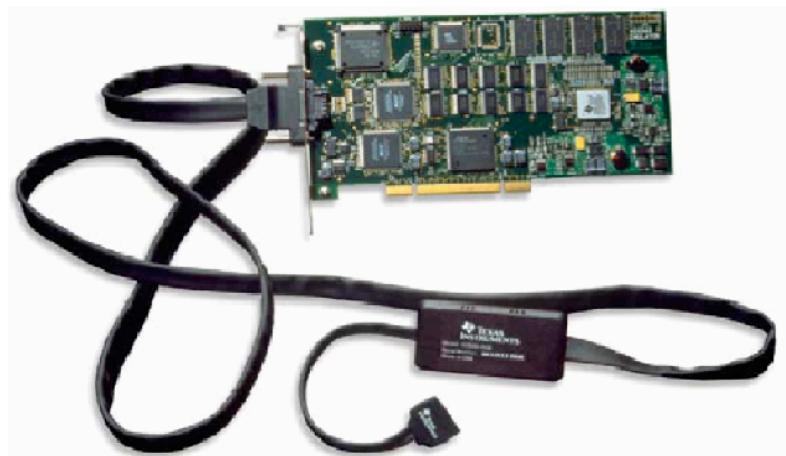


Fig. 36: TI XDS560 emulator, composed of a card to install on the host computer (PCI interface), a JTAG cable and an emulation controller pod

Capabilities of scan-based emulators include source-level debugging, i.e., the possibility to see the assembly instructions being executed and to access variables and memory locations either by name or by address.

Capabilities such as writing to the standard output are available. As an example, the *printf()* function allows printing DSP information on the debugger GUI; the reader should, however, be aware that this operation can be extremely time-consuming, and optimized functions (such as *LOG_printf()* for TI DSPs) should be preferred.

Another common capability supported by emulation technology is the breakpoint. A breakpoint freezes the DSP and allows the developer to examine DSP registers, to plot the content of memory regions, and to dump data to files. Two main forms of breakpoint exist, namely software and hardware. A software breakpoint replaces the instruction at the breakpoint location with one creating an exception condition that transfers the DSP control to the emulation controller. An hardware breakpoint is implemented by using custom hardware on the target device. The hardware logic can for instance monitor a set of addresses on the DSP and stop the DSP code execution when a code fetch is performed at a specific location. Breakpoints can be triggered also by a combination of addresses, data, and system status. This allows DSP developers to analyse the system when for instance it hangs, i.e., when the DSP program counter branches into an invalid memory address. Intermittent bugs can also be tracked down.

It is important to underline that the debugging capabilities provided by emulators allow mostly ‘stop-mode debugging’, in that the DSP is halted and information is sent to the host computer at that moment. This debugging technique is invasive and allows the developer to get isolated, although very useful, snapshots of the halted application. To improve the situation, DSP tool vendors have developed a more advanced debugging technology that allows real-time data exchange between target and host. This technique is described next.

7.3 Real-time techniques

Over the last ten years, DSP vendors have developed techniques for a real-time data exchange between target and host without stopping the DSP and with minimal interference on the DSP run. This provides a continuous visibility into the way the target operates. Additionally, it allows the simulation of data input to the target.

ADI’s real-time communication technology is called Background Telemetry Channel (BTC) [54]. This is based upon a shared group of registers accessible by the DSP and by the host for reading and writing. It is currently supported on Blackfin and ADSP-219s DSPs only.

TI’s real-time communication technology is called Real Time Data eXchange (RTDX) [55, 56]. Its main software and hardware components are shown in Fig. 37. A collection of channels, through which data is exchanged, are created between target and host. These channels are unidirectional and data can be sent across them asynchronously. TI provides two libraries, the RTDX target library and the RTDX host library, that have to be linked to target and host applications, respectively. As an example, the target application sends data to the host by calling functions in the RTDX target library. These functions buffer the data to be sent and then give the program flow control back to the calling program; after this, the RTDX target library transmits the buffered data to the host without interfering in the target application. RTDX is also supported when running inside a DSP simulator; to that end, the DSP developer should link the target application with the RTDX simulator target library corresponding to the chosen target. On the host side, data can be visualized and treated from applications interfacing with the RTDX host library. On Windows platforms a Microsoft Component Object Module (COM) interface is available, allowing clients such as VisualBasic, VisualC++, Excel, LabView, MATLAB and others.

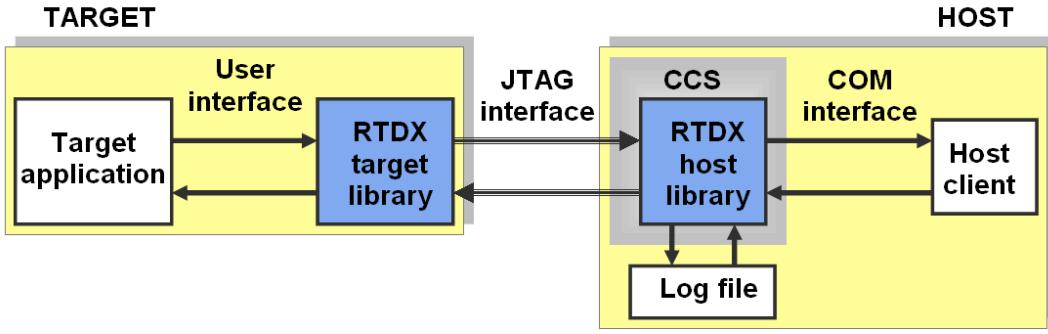


Fig. 37: TI's RTDX main components. The picture is courtesy of TI [56].

In 1998 TI implemented the original RTDX technology, which runs on XDS510-class emulators. A high-speed RTDX version was developed later that relies on additional DSP chip hardware features and on improved emulators, namely the XDS560 class. These emulators make use of two non-JTAG pins in the standard TI JTAG connector to increase RTDX bandwidth. They are also backwards compatible and can support standard RTDX, thus allowing higher data transfer speed. The high-speed RTDX is supported in TI's highest performance DSPs, such as the TMS320C55x, TMS320C621x, TMS320C671x and TMS320C64x families. Table 11 shows the data transfer speeds available with different combinations of RTDX and emulators. RTDX offers a bandwidth of 10 to 20 kbytes/s, thus enabling real-time debugging of applications such as CD audio and audio telephony. The high-speed RTDX with XDS560-class emulators provides a data transfer speed higher than 2 Mbytes/s, thus allowing real-time visibility into applications such as ADSL, hard-disk drives and videoconferencing [57].

Table 11: Data transfer speed as a function of the emulator type for TI's RTDX

Emulation type	Speed
RTDX + XDS510	10–20 kbytes/s
RTDX + USB (ex: 'C6713 DSK board)	10–20 kbytes/s
RTDX + XDS560	≤ 130 kbytes/s
High speed RTDX + XDS560	> 2 Mbytes/s

8 Code analysis and optimization

Most DSP applications are subject to real-time constraints and stress the available CPU and memory resources. As a consequence, code optimization might be required to satisfy the application requirements.

DSP code can be optimized according to one or more parameters such as execution speed, memory usage, input/output bandwidth, or power consumption. Different parts of the code can be optimized according to different parameters. A trade-off between code size and higher performance exists, hence some functions can be optimized for execution speed and others for code size.

Code development environments typically allow defining several code configuration releases, each characterized by different optimization levels. Figure 38 shows the project configurations available in TI Code Composer Studio. The 'Release' configuration comprises the higher optimization

level, while the ‘Debug’ configuration enables debug features, which typically increase code size. Finally, the user can specify a ‘Custom’ configuration where user-selectable debug and optimization features are enabled.

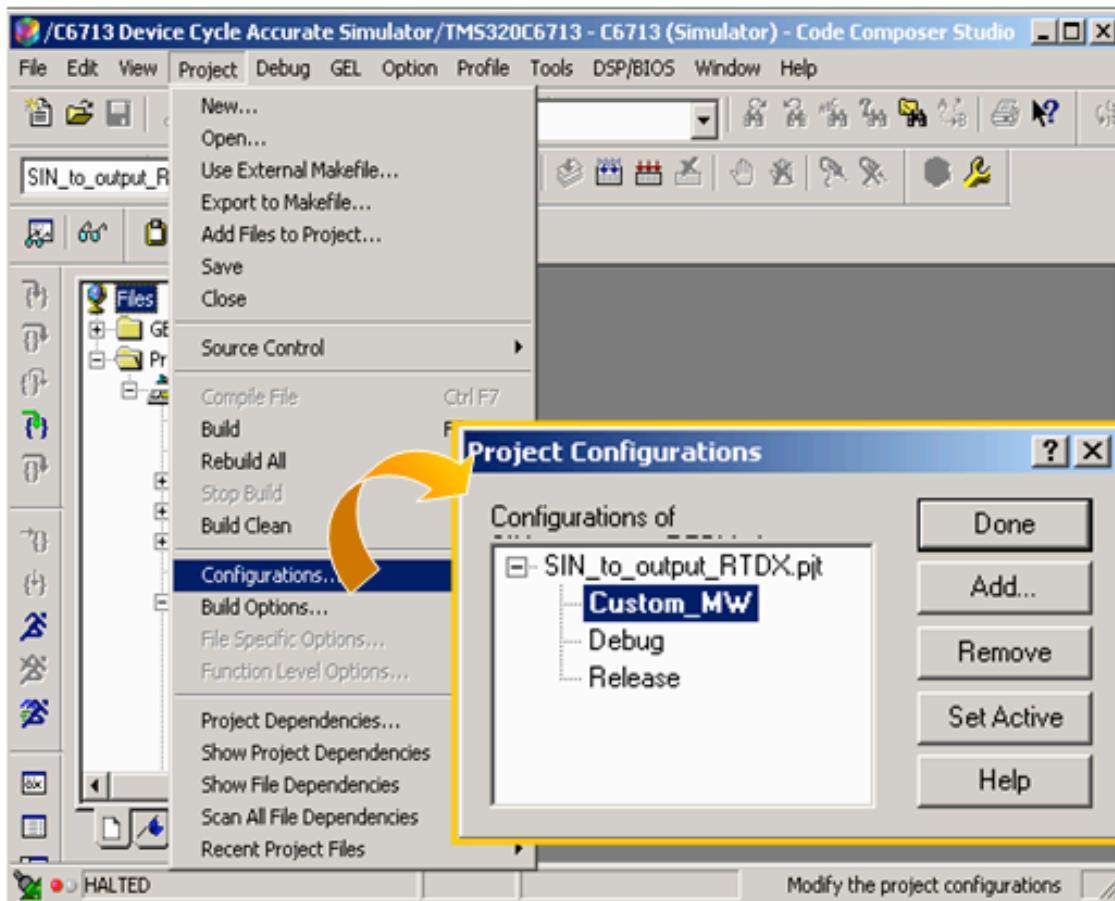


Fig. 38: Choice of the DSP code project configurations in TI Code Composer

It is important to underline that debug and optimization phases are different and often conflicting. In fact, an optimized code does not include any debug information; additionally, the optimizer can re-arrange the code so that the assembly code is not an immediate transposition of the original source code. The reader is strongly encouraged to avoid using the debug and the optimize options together; it is recommended instead to first debug the code and only then to enable the optimization.

8.1 Switching the code optimizer ON

Compilers are nowadays very efficient at code optimization, allowing DSP developers to write higher level code instead of assembly. To do this, compiles must be highly customized, i.e., tightly targeted to the hardware architecture the code will be running upon. However, current trends in software engineering include retargeting compilers to DSP specialized architectures [58].

As previously mentioned, many kinds of optimization can be required. An example is execution speed vs. executable size. Figure 39 shows how the user can select one or the other in the Code Composer Studio development environment.

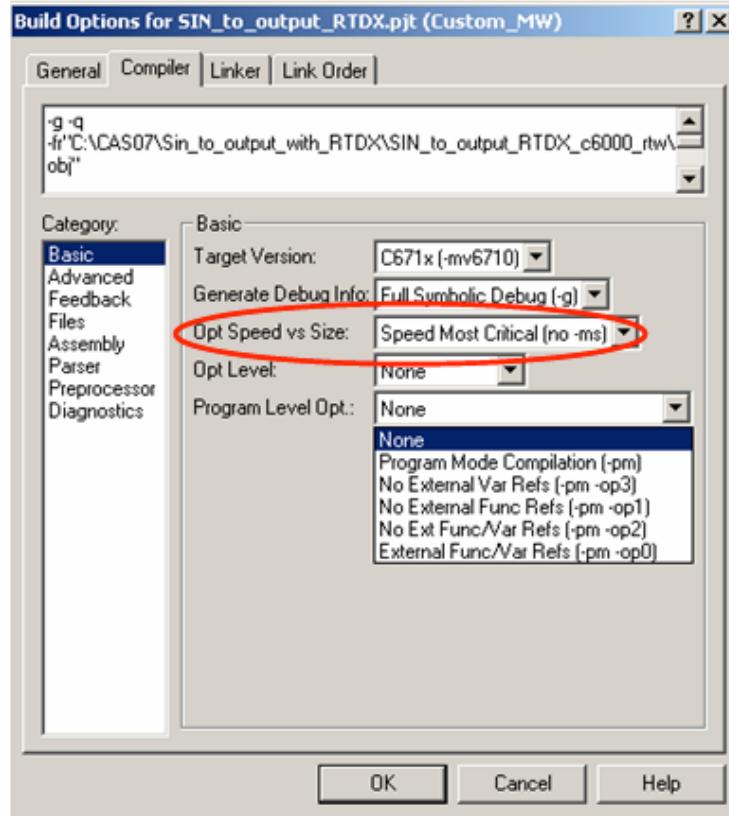


Fig. 39: Choice of optimization levels in TI Code Composer. The plot highlights execution speed vs. executable code size.

The reader should be aware that the optimizer can rearrange the code, hence the code must be written in a proper way. Failing this, the actions generated by the optimized code might be different from those desired and implemented by a non-optimized code. Figure 40 shows two code snippets where the value assumed by the memory location pointed to by *ctrl* determines the *while()* loop behaviour. In particular, the DSP exits the *while()* loop if the *ctrl* content takes the value 0xFF; the *ctrl* content can be modified by another processor or execution thread. Both code snippets will perform equally in case of non-optimization. However, in case of optimization the left-hand side code will not evaluate the *ctrl* content at every *while()* iteration, hence the DSP will remain forever in the loop. On the right-hand side snippet, the *volatile* keyword disables memory optimization locally, thus forcing the DSP to re-evaluate the *ctrl* content value at every *while()* loop iteration. This guarantees the desired behaviour even when the code is optimized. The number of *volatile* variables should be restricted to situations where they are strictly needed, as they limit the compiler's optimization.

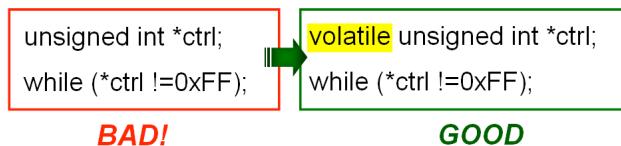


Fig. 40: Example of good and bad programming techniques. The left-hand side code would likely result in a programming misbehaviour.

The recommended code development flow is to first write high-level code, such as C or C++. This code can then be debugged and optimized, to comply with the specified performance. In case the code runs still slower than desired, the time-critical areas can be re-coded in linear assembly. If the

code is still too slow, then the DSP developer should turn to hand-optimized assembly code. Figure 41 shows a comparison of the different programming techniques, with corresponding execution efficiency and development effort.

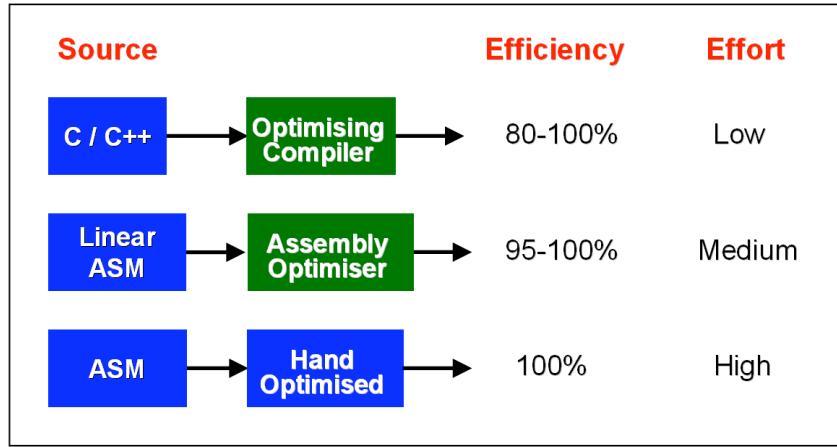


Fig. 41: Comparison of programming techniques with corresponding execution efficiency and estimated development effort. The picture is courtesy of TI [59].

8.2 Analysis tools

DSP code often follows the 20/80 rule, which states that 20% of the software in an application uses 80% of the processing time. As a consequence, the DSP developer should first concentrate efforts on determining where to optimize, i.e., on understanding where the execution cycles are mostly spent.

The best way to determine the parts of the code to optimize is to profile the application. Over the last ten years DSP development environments have considerably enlarged their offer of analysis tools. Some examples of TI's CCS analysis and tuning tools are:

- Compiler consultant. It analyses the DSP code and provides recommendations on how to optimize its performance. This includes compiler optimization switches and programs, thus allowing a quick improvement in performance. Figure 42 shows how to enable the compiler consultant in CCS.
- Cache tune. It provides a graphical visualization of memory reference patterns and memory accesses, thus allowing the identification of problem areas related for instance to memory access conflicts.
- Code size tune. It profiles the application, collects data on individual functions and determines the best combinations of compiler options to optimize the trade-off between code size and execution speed.
- Analysis ToolKit (ATK). It runs with DSP simulators only and allows one to analyse the DSP code robustness and efficiency. The reader can find more information on the ATK setup and use in Refs. [60, 61].

The DSP developer should not only know when to optimize, as described previously: he/she should also know when to stop. In fact, there is a law of diminishing returns in the code analysis and optimization process. It is thus important to take advantage of the improvements that come with relatively little effort, and leave as a last resort those that are difficult to implement and provide low-yield.

Finally, it is strongly recommended to make only one optimization change at the same time; this will allow the developer to exactly map the optimization to its result.

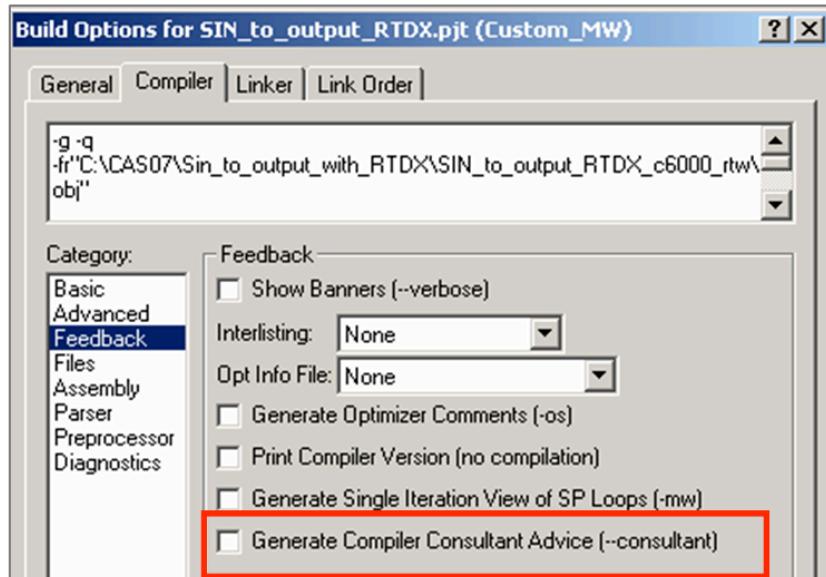


Fig. 42: How the ‘Compiler Consultant Advice’ can be enabled in TI’s CCS Development Environment

8.3 Programming optimization guidelines

This Section includes some general programming guidelines for writing efficient code; these guidelines are applicable to the vast majority of DSP compilers. DSP developers should, however, refer to the manuals of the development tools they are using for more precise information on how to write efficient code. The reference manual for TI TMS320C6xxx DSP can be found in Ref. [62].

Finally, it is strongly recommended to make only one optimization change at the same time; this will allow the developer to exactly map the optimization to its result.

- Guideline 1: Use the DMA when possible and allocate data in memory wisely

DMA controllers (see Sub-section 3.2.3) must be used whenever possible so as to free the DSP core for other tasks. The linker (see Sub-section 6.4.3) should be used for allocating data in memory so as to guarantee an efficient memory access. Additionally, DSP developers should avoid placing arrays at the beginning or at the very end of memory blocks, as this creates problems for software pipelining. Software pipelining is a technique that optimizes tight loops by fetching a data set while the DSP is processing the previous one. However, the last iteration of a loop would attempt to fetch data outside the memory space, in case an array is placed on the memory edge. Compilers must then execute the last iteration in a slower way (‘loop epilogue’) to prevent this address error from happening. Some compilers, such as the ADI Blackfin one, make available compiler options to specify that it is safe to load additional elements at the end of the array.

- Guideline 2: Choose variable data types carefully

DSP developers should know the internal architecture of the DSP they are working on, so as to be able to use native data type DSPs as opposed to emulated ones, whenever possible. In fact, operations on native data types are implemented by hardware, hence are typically very fast. On the contrary, operations on emulated data types are carried out by software functions, hence are slower and use more resources. An example of emulated data type is the double floating point format on ADI’s TigerSHARC floating point DSPs. Another example is the floating point format on ADI’s

Blackfin family of fixed-point processors [63]. In these DSPs the floating point format is implemented by software functions that use fixed-point multiply and ALU logic. In this last case a faster version of the same functions is available with non-IEEE-compliant data formats, i.e., formats implementing a ‘relaxed’ IEEE version so as to reduce the computational complexity. Table 12 shows a, execution times comparison of IEEE-compliant and non-IEEE-compliant functions in ADI’s Blackfin BF533.

Table 12: Execution time of IEEE-compliant vs. non-IEEE-compliant library functions for ADI’s Blackfin BF533

operation	fast-ft [cycles]	IEEE-ft [cycles]	ratio
multiply	93	241	0.4
add	127	264	0.5
subtract	161	329	0.5
divide	256	945	0.3
pow	8158	17037	0.5

- Guideline 3: Functions and function calls

Functions such as *max()*, *min()* and *abs()* are often single-cycle instruction and should be used whenever possible instead of manually coding them. Figure 43 shows on the right-hand side the *max()* function and on the left-hand side a manual implementation of the same function. The advantage in terms of code efficiency of using a single-cycle *max()* function is evident. Often more complex functions such as FFT, IIR, or FIR filters are available in vendor-provided libraries. The reader is strongly encouraged to use them, as their optimization is carried out at algorithm level.

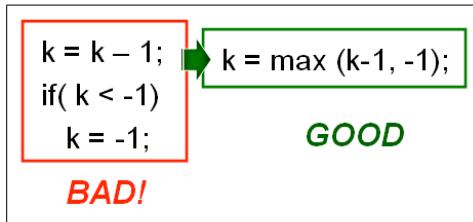


Fig. 43: Example of good and bad programming techniques

As few parameters as possible should be passed to a function. In fact, parameters are typically passed to functions by using registers. However, the stack is used when no more registers are available, thus slowing down the code execution considerably.

- Guideline 4: Avoid data aliasing

Aliasing occurs when multiple variables point to the same data. For example, two buffers overlap, two pointers point to the same software object or global variables used in a loop. This situation can disrupt optimization, as the compiler will analyse the code to determine when aliasing could occur. If it cannot work out if two or more pointers point to independent addresses or not, the compiler will typically behave conservatively, hence avoid optimization so as to preserve the program correctness.

- Guideline 5: Write loops code carefully

Loops are found very often in DSP algorithms, hence their coding can strongly influence the program execution performance. Function calls and control statements should be avoided inside a loop, so as to prevent pipeline flushes (see Sub-section 3.3.2). Figure 44 shows an example of good

and bad programming techniques referred to control statements inside a `for()` loop: by moving the conditional expression `if...else` outside the loop, as shown in the right-hand side code snippet, one can reduce the number of times the conditional expression is executed.

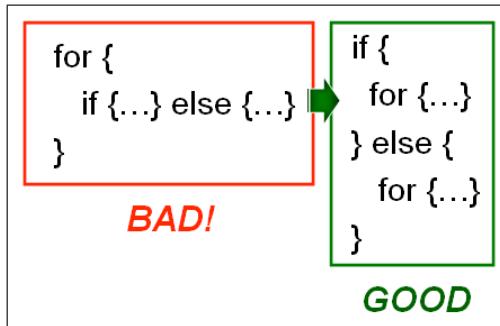


Fig. 44: Example of good and bad programming techniques

Loop code should be kept small, so as to fit entirely into the DSP cache memory and to allow a local repeat optimization. In case of many nested loops, the reader should be aware that compilers typically focus their optimization efforts on the inner loop. As a consequence, pulling operations from the outer to the inner loop can improve performance. Finally, it is recommended to use `int` or `unsigned int` data types for loop counters instead of the larger-sized data type `long`.

- Guideline 6: Be aware of time-consuming operations

There are operations, such as the division, that do not have hardware support for a single-cycle implementation. They are instead implemented by functions implementing iterative approximations algorithms, such as the Newton–Raphson. The DSP developer should be aware of that and try to avoid them when possible. For example, the division by a power-of-two operation can be converted to the easier right shift on unsigned variables. DSP manufacturers often provide indications on techniques to implement the division instruction more efficiently [64].

Other operations are available from library functions. Examples are `sine`, `cosine` and `atan` functions, very often needed in the accelerator sector for the implementation of rotation matrixes and for rectangular to polar coordinates conversion. If needed, custom implementations can be developed to obtain a favourable ratio between precision and execution time. Table 13 shows the comparison of different implementations of the same functions; in particular, the second column shows a custom implementation used in CERN’s LEIR accelerator. In this implementation, the `sine`, `cosine` and `atan` calculation algorithm has been implemented by a polynomial expansion of the seventh order instead of the usual Taylor series expansion.

Table 13: Execution times vs. different implementations of the same functions

Function	Execution time [μs]		
	CERN single-precision implementation	VisualDSP++ single-precision implementation	VisualDSP++ double-precision implementation
cosine	0.25	0.59	5.5
sine	(for a sine/cosine couple)	0.59	5.3
atan	0.4125	1.4	5.6

- Guideline 7: Be aware that DSP software heavily influences power optimization

DSP software can have a significant impact on power consumption: a software-efficient in terms of the required processor cycles to carry out a task is often also energy efficient. Software should be written so as to minimize the number of accesses to off-chip memory; in fact, the power required to access off-chip memory is usually much higher than that used for accessing on-chip memory. Power consumption can be further optimized in DSPs that support selective disabling of unused functional blocks (e.g., on-chip memories, peripherals, clocks, etc.). These ‘power down modes’ are available in ADI DSPs (such as Blackfin) as well as in TI DSPs (such as the TMS320C6xxx family [35]). Making a good use of these modes and features can be difficult; however, APIs and specific software modules are available to help. An example is TI’s DSP/BIOS Power Manager (PWRM) module [65], providing a kernel-level API that interfaces directly to the DSP hardware by writing and reading configuration registers. Figure 45 shows how this module is integrated in a generic application architecture for DSPs belonging to TI’s TMS320C55x family.

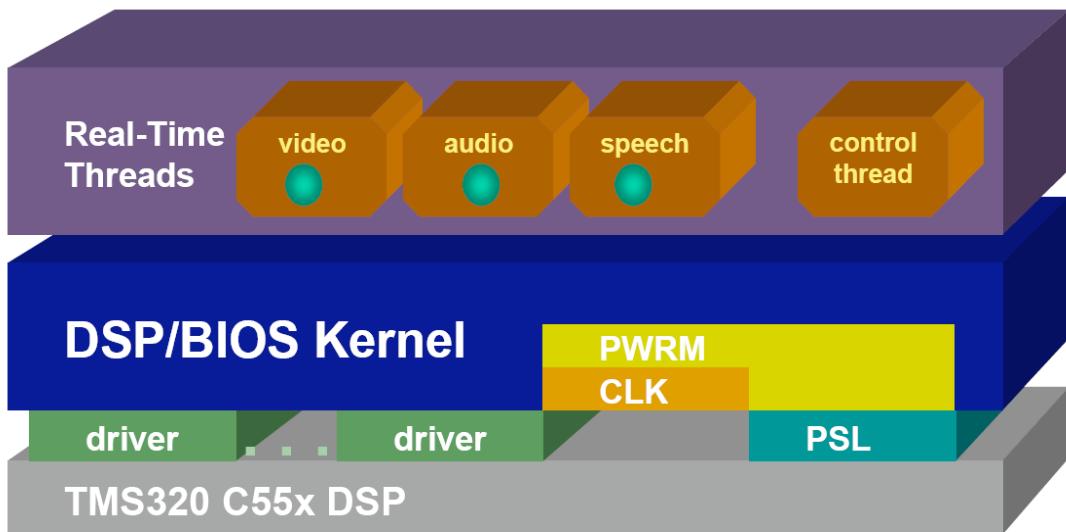


Fig. 45: TI’s DSP/BIOS Power Manager (PWRM) module in a general system architecture. Picture courtesy of Texas Instruments [65].

9 Real-time design flow: system design

This section deals with some aspects of digital systems design, particularly with software and hardware architectures. Here the assumption is that the system to be designed is based upon one or more DSPs. The reader should, however, be aware that in the accelerator sector there are currently three main real-time digital signal processing actors: DSPs, FPGAs and front-end computers. The front-end computers are typically implemented by embedded General Purpose Processors (GPPs) running a RTOS. Nowadays, the increase in clock speed allows GPPs to carry out real-time data processing and slow control actions; in addition, there is a tendency to integrate DSP hardware features and specialized instructions into GPPs, yielding GPP hybrids. One example of such processors is given in Fig. 46, showing the PowerPC with Motorola’s Altivec extension. The Altivec 128-bit SIMD unit adds up to 16 operations per clock cycle, in parallel to the Integer and Floating Point units, and 162 instructions to the existing RISC architecture.

Fundamental choices to make when designing a new digital system are which digital signal processing actors should be used and how tasks should be shared between them. This choice requires detailed and up-to-date knowledge of the different possibilities.

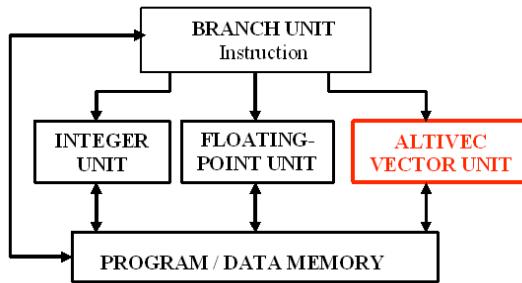


Fig. 46: Altivec technology: SIMD expansion to Motorola PowerPC (G4 family)

In industry the choice of the DSP to use is often based on the ‘4P’ law: Performance, Power consumption, Price and Peripherals. In the accelerator sector, the power consumption factor is typically negligible. Other factors are instead decisive, such as standardization in the laboratory, synergies with existing systems, and possibilities of evolution to cover different machines. Last but not least, one should consider the existing know-how in terms of tools and of hardware, which can be directly translated to a shorter development time.

In this section three design aspects are considered and briefly discussed, namely:

- DSP choice in Sections 9.1 and 9.2.
- System architecture in Sections 9.3 to 9.6.
- DSP code design in Sections 9.7 and 9.8.

9.1 DSP choice: fixed vs. floating-point DSPs

The reader can find a basic description of fixed- and floating-point number formats in Section 3.4.

Fixed-point formats can typically be implemented in hardware in a cheaper way, with better energy efficiency and less silicon than floating-point formats. Very often fixed-point DSPs support a clock faster than floating-point DSPs; as an example, TI fixed-point DSPs can currently be clocked up to 1.2 GHz, while TI floating-point DSPs are clocked up to 300 MHz.

Floating-point formats are easier to use since the DSP programmer can mostly avoid carrying out number scaling prior to each arithmetic operation. In addition, floating-point numbers provide a higher dynamic range, which can be essential when dealing with large data sets and with data sets whose range cannot be easily predicted. The reader should be aware that floating-point numbers are not equispaced, i.e., the gap between adjacent numbers depends on their magnitude: large numbers have large gaps between them, and small numbers have small gaps. As an example, the gap between adjacent numbers is higher than 10 for numbers of the order of $2 \cdot 10^8$. Additionally, the error due to truncation and rounding during the floating-point number scaling inside the DSP depends on the number magnitude, too. This introduces a noise floor modulation that can be detrimental for high-quality audio signal processing. For this reason, high-quality audio has been traditionally implemented by using fixed-point numbers. However, a migration of high-fidelity audio from fixed- to floating-point implementation is currently taking place, so as to benefit from the greater accuracy provided by floating point numbers.

The choice between fixed- and floating-point DSP is not always easy and depends on factors such as power consumption, price, and application type. As an example, military radars need floating-point implementations as they rely in finding the maximal absolute value of the cross-correlation between the sent signal and the received echo. This is expressed as the integral of a function against an exponential; the integral can be calculated by using FFT techniques that benefit from the floating point dynamic range and resolution. For radar systems, the power consumption is not a major issue. The

floating-point DSP additional cost is not an issue either, as the processor represents only a fraction of the global system cost. Another example is the mobile TV. The core of this application is the decoder, which can be MPEG-2, MPEG-4 or JPEG-2000. The decoding algorithms are designed to be performed in fixed-point; the greater precision of floating-point numbers is not useful as the algorithms are in general bit-exact.

It should be underlined that many digital signal processing algorithms are often specified and designed with floating-point numbers, but are subsequently implemented in fixed-point architectures so as to satisfy cost and power efficiency requirements. This requires an algorithm conversion from floating-point to fixed-point and different methodologies are available [66].

Finally, as mentioned in Section 8.3, some fixed-point DSPs make available floating-point numbers and operations by emulating them in software (hence they are slower than in a native floating-point DSP). An example is ADI's Blackfin [63].

The fact that floating-point numbers are not equispaced has already been mentioned. The reader might be interested in looking at some consequences of this with an example from the LHC beam control implementation. Figure 47 shows a zoom onto the beam loops part of the LHC beam control. The ‘Low-level Loops Processor’ is a board including a TigerSHARC DSP and an FPGA. The FPGA carries out some simple pre-processing and data interfacing, while the DSP implements the low-level loops. In particular, the DSP calculates the frequency to be sent to the cavities from the beam phase, radial position, synchrotron frequency, and programmed frequency; these calculations are carried out in floating-point format. The frequency to be sent to the cavities, referred to as F_{out} in Fig. 47, must be expressed as an unsigned, 16-bit integer. The desired frequency range to represent is 10 kHz, hence the needed resolution is 0.15 Hz. The LHC cavities work at a frequency of about 400.78 MHz but the spacing of a single-precision, floating-point number with magnitude of approximately $400 \cdot 10^6$ is higher than one. To avoid the use of slower, double-precision, floating-point format, the beam loop calculations are carried out as offset from 400.7819 MHz.

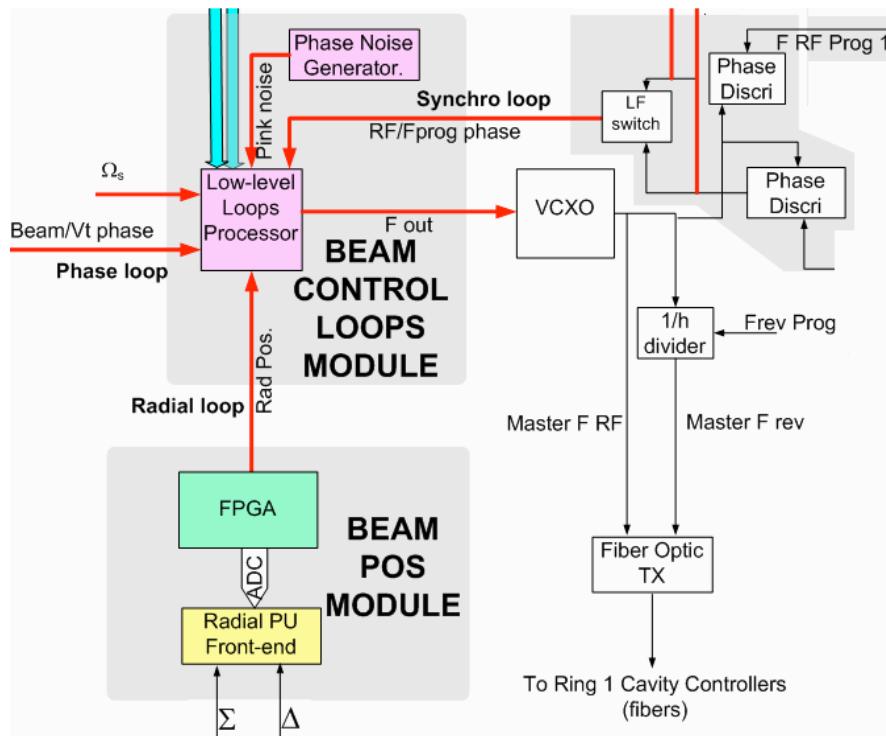


Fig. 47: LHC beam control – zoom onto the beam loops part

9.2 DSP choice: benchmarking

Benchmarking a DSP means evaluating it on a number of different metrics. Table 14 gives an example of some common metrics and corresponding units.

Table 14: Examples of DSP performance metric sets and corresponding units

Metric	Unit
Maximum clock frequency	MHz
Execution speed	Millions of Instructions Per Second (MIPS) Millions of Operations Per Second (MOPS) Number of Multiply-and-Accumulate operations per second
Memory bandwidth	Mbytes/s
Memory latency	Number of clock cycles
Power consumption	W or W/MIPS

Good benchmarks are important for comparing DSPs and allow critical business or technical decisions to be made. It should be underlined that benchmarks can be misleading, thus should be considered in a critical way. As an example, the maximum clock frequency of a DSP can be different from the instruction rates; hence this parameter might not be indicative of the real DSP processing power. Another example is the execution speed measured in MIPS: this metric is easy to measure but it is often too simple to provide useful information about how a processor would perform in a real application. VLIW architectures issue and execute multiple instructions per instruction cycle. These processors usually use simpler instructions that perform less work than the instructions typical of conventional DSPs. As a consequence, MIPS comparison between VLIW-based DSP and conventional ones is misleading.

More complex benchmarks are available; examples are the execution of application tasks (typically called kernel functions) such as IIR filters, FIR filters, or FFTs. Kernel function benchmarking is typically more reliable and is available from DSP manufacturers as well as from independent companies.

It is difficult to provide general guidelines to measure the efficacy of DSP benchmarks for DSP selection. Two general rules should be followed: first, the benchmark should perform the type of work the DSP will be expected to carry out in the targeted application. Second, the benchmark should implement the work in a way similar to what will be used in the targeted application.

9.3 System architecture: multiprocessor architectures

Multiprocessor architectures are those where two or more processors interact in real-time to carry out a task. Right from their early days, many DSP families have been designed to be compatible with multiprocessing operation; an example is the TI TMS320C40 family. Multiprocessing architectures are particularly suited for applications with a high degree of parallelism, such as voice processing. In fact, processing ten voice channels can be carried out by implementing a one-voice channel, then repeating the process ten times in parallel. Applications requiring multiprocessing computing to support processing of greater data flow include high-end audio treatment, 3D graphics acceleration, and wireless communication infrastructure, just to mention a few of them.

There is another reason to move to multiprocessing systems. For many years developers have been taking advantage of the steady progress in DSP performance. New and faster processors would be available, allowing more powerful applications to be implemented sometimes only for the price of porting existing code to the new DSP. This favourable situation was driven by the steady progress of the semiconductor industry that managed to pack more transistors into smaller packages and at higher clock frequencies. The increased performance was enabled by architectural innovations, such as VLIW, as well as added resources, such as on-chip memories. In recent years, however, progress in single-chip performance has been slowing down. The semiconductor industry has turned to parallelism to increase performance. This is true not only for the DSP sector, but in general for business computing. One example is the Intel Core Duo processors, including two execution cores in a single processor, now the established platform for personal computers and laptops.

Finally, the reader should be aware that development environments have evolved to provide support for debugging multiple processor cores connected in the same JTAG path [67]. An example is TI's Parallel Debug Manager [68], which is integrated within the Code Composer Studio IDE.

Of the many possible multiprocessing forms, the multi-DSP and multi-core approaches are considered and discussed in Sub-sections 9.3.1 and 9.3.2, respectively. Examples of embedded multi-processors and different approaches can be found in Ref. [69].

9.3.1 Multi-DSP architecture

Many separate DSP chips can co-operate to carry out a task providing an increased system performance. One advantage of this approach is the scalability, i.e., the ability to tune the system performance and cost to the required functionality and processing performance by varying the number of DSP chips used.

The reader should, however, be aware that multi-DSP designs involve different constraints than single-processing systems. Three key aspects must be taken into account.

- a) Tasks must be partitioned between processors. As an example, a single processor can handle a task from start to end; as an alternative, a processor can perform only a portion of the task, then pass the intermediate results to another processor.
- b) Resources such as memory and bus access must be shared between processors so as to avoid bottlenecks. As an example, additional memory may be added to store intermediate results. Organizing memory into segments or banks allows simultaneous memory accesses without contentions if different banks are accessed.
- c) A robust and fast inter-DSP communication means must be established. If the communication is too complex or takes too much time, the advantage of a multiprocessing can be lost.

Two examples of multi-DSP architectures based on ADI DSPs are shown in Fig. 48. The reader can find more detailed information in Refs. [70] and [71].

On the left-hand side (plot a) the point-to-point architecture is depicted, based upon ADI linkport interconnect cable standard [27]. Point-to-point interconnect provides a direct connection between processor elements. This is particularly useful when large blocks of intermediate results must be passed between two DSPs without involving the others. Read/write transactions to external memory are saved by passing data directly between two DSPs, thus allowing the use of slower memory devices. Additionally, the point-to-point interconnect can be used to scale a design: additional links can be added to have more DSPs interacting. This can be done either directly or by bridging across several links.

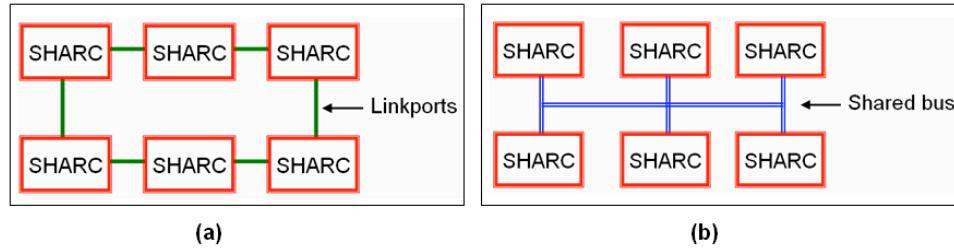


Fig. 48: Examples of multi-DSP configurations. (a) point-to-point, linkport-based and (b) cluster bus

On the right-hand side (plot b) the cluster bus architecture is depicted. A cluster bus maps internal memory resources, such as registers and processor memory addresses, directly onto the bus. This allows DSP code developers to exchange data between DSPs using addresses as if each processor possessed the memory for storing the data. Memory arbitration is managed by the bus master; this avoids the need for complex memory or data sharing schemes managed by software or by RTOS. The map includes also a common broadcast space for messages that need to reach all DSPs. As an example, Fig. 49 shows the TigerSHARC global memory map. The multiprocessing space maps the internal memory space of each TigerSHARC processor in the cluster into any other TigerSHARC processor. Each TigerSHARC processor in the cluster is identified by its ID; valid processor ID values are 0 to 7.

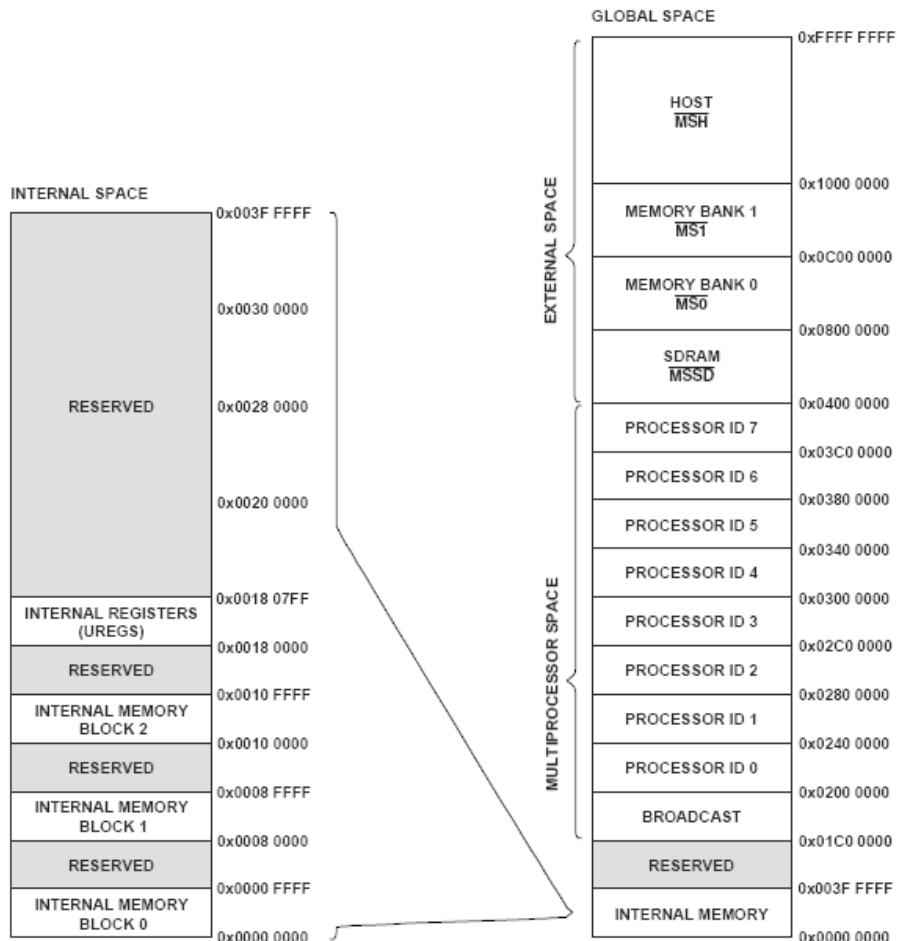


Fig. 49: ADI TigerSHARC TS101 global memory map. Picture courtesy of Analog Devices [72].

The reader should be aware that the two above-mentioned architectures, namely point-to-point and cluster bus, are not mutually exclusive; on the contrary, they can both be used in the same application as complementary solutions.

9.3.2 *Multi-core architecture*

In a multi-core architecture, multiple cores are integrated into the same chip. This provides a considerable increase of the performance per chip, even if the performance per core only increases slowly. Additionally, the power efficiency of multi-core implementations is much better than in traditional single-core implementations. This approach is a convenient alternative to DSP farms.

As the performance required by DSP systems keeps increasing, it is nowadays essential for DSP developers to devise a processing extension strategy. Multi-core architectures can provide it, in that the DSP performance is boosted without switching to a different core architecture. This has the advantage that applications can be based upon multiple instances of an already-proven core, rather than be adapted to new architectures.

DSP multi-core architectures have been commercialized only recently; however, the DSP market has relied for many years on co-processor technology (also called on-chip accelerators) to boost performance. Figure 50 shows the evolution of DSP architecture. From the initial single-core architecture (a), the single-core plus co-processor architecture soon emerged. The co-processor often runs at the same frequency as the DSP, therefore ‘doubling’ the performance for the targeted application. Co-processor examples are Turbo and Viterbi decoders for communication applications. Example of decoder coprocessors for TI’s TMS320C64x can be found in Refs. [73] and [74]. Finally, over the last few years the multi-core architecture shown in plot (c) has emerged, which still includes co-processors.

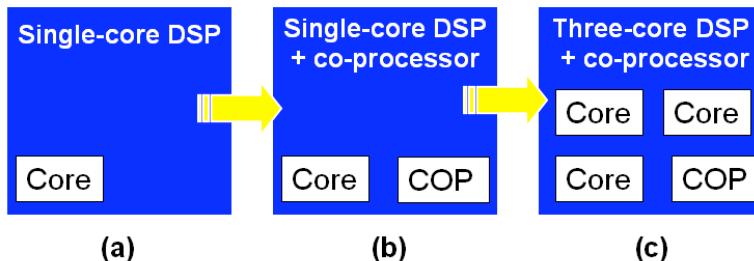


Fig. 50: Multi-core and co-processor DSP architectures evolution. Single-core DSP (a), single-core DSP plus coprocessor (b) and multi-core DSP plus coprocessor (c).

Multi-core architectures are available in two different flavours, namely Symmetric Multi-Processing (SMP) and Asymmetric Multi-Processing (AMP). SMP architectures include two or more processors which are similar (or identical), connected thorough a high-speed path and sharing some peripherals as well as memory space. AMP architectures combine two different processors, typically a microcontroller and a DSP, into a hybrid architecture.

It is possible to use a multi-core device in different ways. The different cores can operate independently or they can cooperate for task completion. An efficient inter-core communication may be needed in both cases, but it is particularly important when two or more cores work together to complete a task. As for the multi-DSP case discussed in Sub-section 9.3.1, it is important to decide how to share resources to avoid bottlenecks and deadlocks, and to ensure that one core does not corrupt the operation of another core. The resources must be partitioned not only at board level, like in the single-core case, but at device level, too, thus adding increase complexity. Figure 51 shows an example of multi-core bus and memory hierarchy architecture. L1 memories are typically dedicated to their own core as non-partitioned between cores, as it may be inefficient to access them from other

cores. The L2 memory is an internal memory shared between the different cores, as opposed to the single-core case where the L2 memory can be either internal or external. The multi-core architecture must make sure that each core can access the L2 memory and the arbitration must be such that cores are not locked out from accessing this resource.

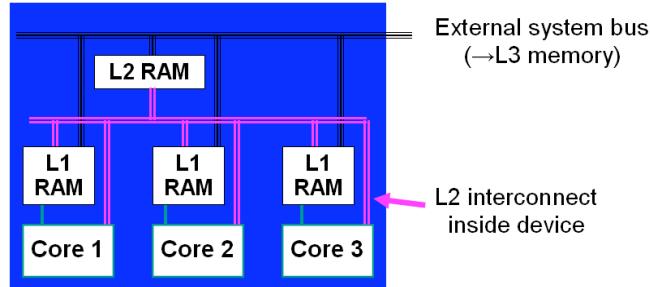


Fig. 51: Multi-core bus and memory hierarchy example

Figure 52 shows the TMS320C5421 DSP as an example of a multi-core, SMP DSP. The TMS320C5421 DSP is composed of two C54x DSP cores and is targeted at carrier-class voice and video end equipment. The cores are 16-bit fixed-point and the chip is provided with an integrated VITERBI accelerator. Four internal buses and dual address generators enable multiple program and data fetches and reduce memory bottlenecks.

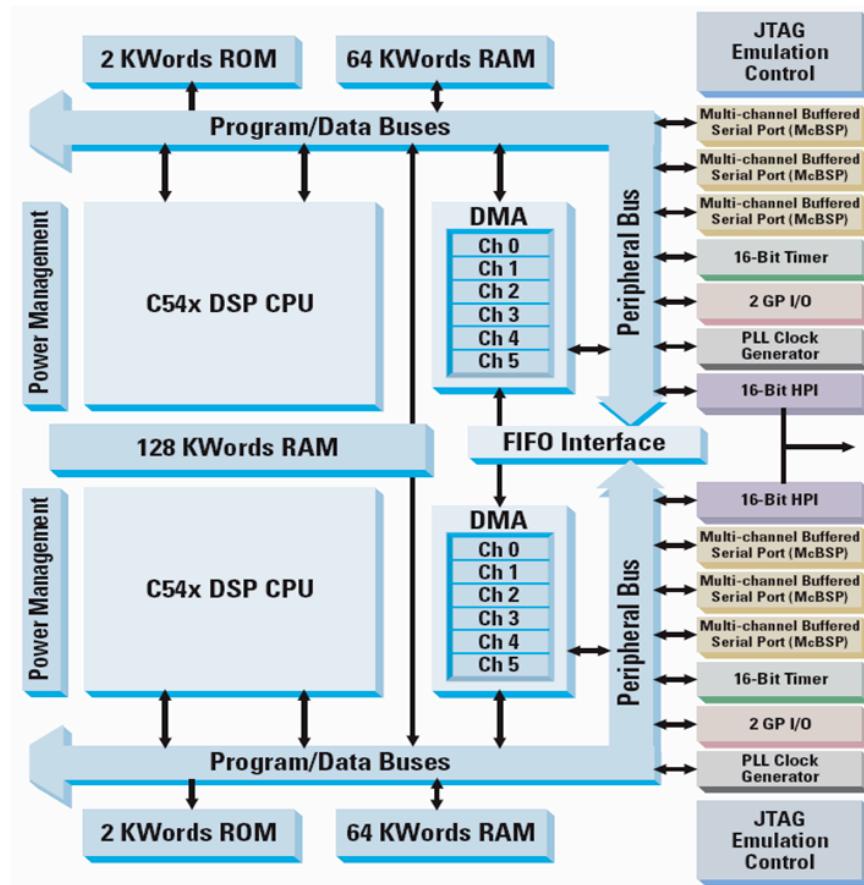


Fig. 52 TMS320C5421 multi-core DSP as an SMP example. Picture courtesy of Texas Instruments, DSP selection guide 2007, p. 48.

The programming of multi-core system is generally more complex than in the single-core case. In particular, the reader should be aware that multi-core code must follow the re-entrance rules, to make sure that one core's processing does not corrupt the data used by another core's processing. This approach is followed by single-core processors, too, when implementing multi-tasking operations.

An example of advantages and challenges a developer is dealing with when moving an audio application from single- to double-core architecture is given in Ref. [75].

9.4 System architecture: radiation effects

Single-Event Upset (SEU) events are alterations in the behaviour of electronic circuits induced by radiation. These alterations can be transient disruptions, such as changes of logic states, or permanent IC alterations. The reader is referred to Ref. [76] for more information on the subject.

Techniques to mitigate these effects in ICs can be carried out at different levels, namely:

- a) At device level, for instance by adding extra-doping layers to limit the substrate charge collection.
- b) At circuit level, for instance by adding decoupling resistors, diodes, or transistors in the SRAM hardening.
- c) At system level, with Error Detection And Correction (EDAC) circuitry or with algorithm-based fault tolerance [77]. An example of the latter approach is the Triple Module Redundancy (TMR) algorithm or the newer Weighted Checksum Code (WCC). The reader should, however, be aware that there are limitations to what these algorithms can achieve. For instance, the WCC method applied to floating-point systems may fail, as roundoff errors may not be distinguished from functional errors caused by radiation.

Neither ADI nor TI currently provide any radiation-hard DSP. Third-party companies have developed and marketed radiation-hard versions of ADI and TI DSPs. An example is Space Micro Inc., based in San Diego, California. This company devised the Proton 200k single-board computer based upon a TI C67xx DSP, fitted with EDAC circuitry and with a total dose tolerance higher than 100 krad.

The LHC power supply controllers [78, 79] are examples of mitigation techniques applied to DSP. They are based upon non-radiation-hard TI C32 DSPs and micro controllers. The memory is protected with EDAC circuitry and by avoiding the use of DSP internal memory, which cannot be protected. A watchdog system restarts the power supply controller in the event of a crash. Radiation tests [80] have been carried out to check that the devised protection strategy is sufficient for normal operation.

9.5 System architecture: interfaces

An essential step in the digital system design is to clearly define the interfaces between the different parts of the system. Figure 53 shows some typical building blocks that can be found in a digital system, namely DSP(s), FPGA(s), daughtercards, Master VME, machine timings, and signals.

The DSP system designer must define the interfaces between DSP(s) and the other building blocks. It is strongly recommended to avoid hard-coding in the DSP code the address of memory regions shared with other processing elements. On the contrary, the linker should be used to allocate appropriately the software structures in the DSP memory, as mentioned in Sub-section 6.4.3. Additionally, the DSP developer should created data access libraries, so as to obtain a modular hence more easily upgradeable approach.

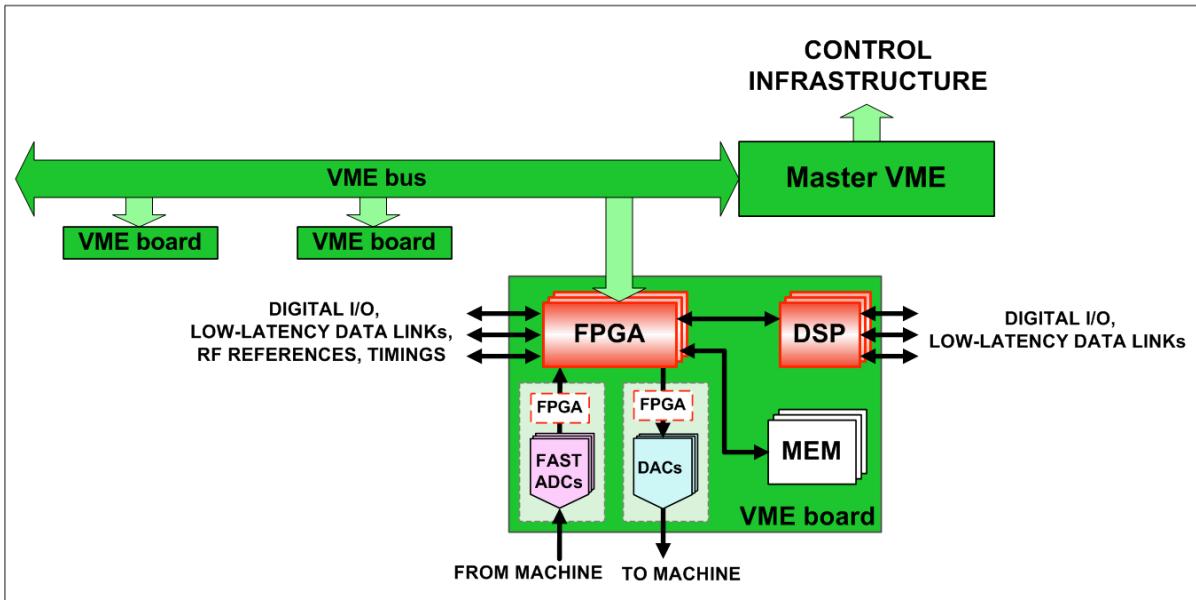


Fig. 53: Typical digital system building blocks and corresponding interfaces

9.6 System architecture: general recommendations

Basically all DSP chips present some anomalies on their expected behaviour. This is especially true for the first release of DSP chips, as discovered anomalies are typically solved on later releases. A list of all anomalies for a certain DSP release, which includes also workarounds when possible, is normally available on the manufacturer's website. The reader is strongly encouraged to look at those lists, so as to avoid being delayed by already-known problems.

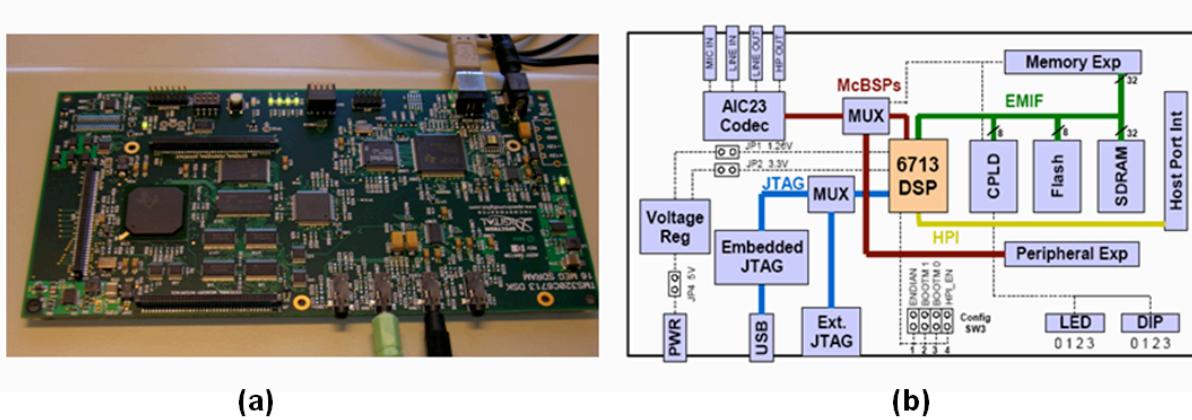


Fig. 54: TI C6713 DSK evaluation board – picture (a) and board layout (b)

A DSP system designer can gain useful software and hardware experience by using evaluation boards in the early stages of system design. Evaluation boards are typically provided by manufacturers for the most representative DSPs. They are relatively inexpensive and are typically fitted with ADCs and DACs; they come with the standard development environment and JTAG interface, too. The DSP designer can use them to solve technical uncertainties and sometimes can even modify them to quickly build a system prototype [81]. Figure 54 shows TI's C6713 DSK evaluation board (a) and corresponding board layout (b); this evaluation board was that used in the DSP laboratory companion of the lectures summarized in this paper.

9.7 DSP code design: interrupt-driven vs. RTOS-based systems

A fundamental choice that the DSP code developer must make is how to trigger the different DSP actions. The two main possibilities are via a RTOS or via interrupts.

An overview of RTOS is given in Section 6.3. RTOS can define different threads, each one performing a specific action, as well as the corresponding threads' priorities and triggers. RTOS-based systems have typically a clean design and many built-in checks. The disadvantage of using RTOS is a potentially slower response to external events (interrupts) and the use of DSP resources (such as some hardware timings and interrupts) for the internal RTOS functioning.

Interrupt-driven systems associate actions directly to interrupts. The resource use is therefore optimized. An example of interrupt-driven system is CERN's LEIR LLRF [42]. Figure 55 shows some of its software components: a background task triggered every millisecond carries out housekeeping actions, while a control task triggered every 12.5 μ s implements the beam control actions. Driving a system through interrupts is very efficient with a limited number of interrupts. For a high number of interrupts, the system can become very complex and its behaviour not easily predictable.

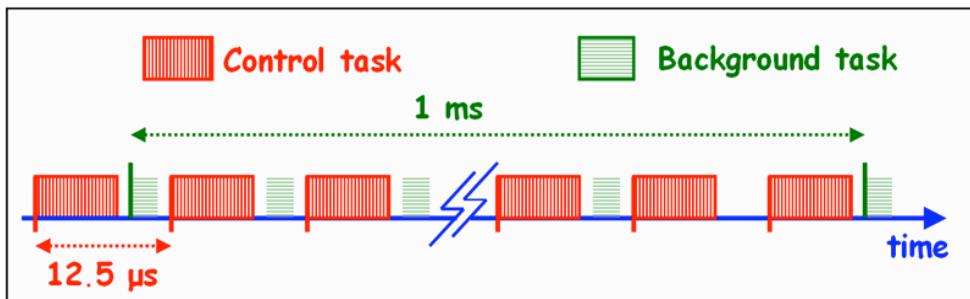


Fig. 55: Example of an interrupt-driven system. Control and background tasks are triggered by interrupts and are shown in red and green, respectively.

9.8 DSP code design: good practice

A vast amount of literature is available on code design good practice. Here just a few points are underlined, which are particularly relevant to embedded systems.

First, digital systems must not turn into tightly sealed black boxes. It is essential that designers embed many diagnostics buffers in the DSP code, so as to prevent this from happening. The diagnostics buffers could take many forms, such as post-mortem, circular or linear buffers. They might be user-configurable and must be visible from the application program. An example of a digital system including extensive diagnostics capabilities can be found in Ref. [42].

Second, every new DSP code release should be characterized by a version number, visible from the application level. The functionality and interface map corresponding to a certain version number should be clearly documented, so as to avoid painful misunderstandings between the many system layers. Source code control is essential for managing complex software development projects, as large projects require more than one DSP code developer working on many source files. Source code control tools make it possible to keep track of the changes made to individual source files and prevent files from being accessed by more than one person at a time. DSP software development environments can often support many source control providers. Code Composer Studio, for example, supports any source control provider that implements the Microsoft SCC Interface.

Finally, DSP developers should also add checks on the execution duration, to make sure the code does not overrun. This is particularly important for interrupt-driven systems (mentioned in Section 9.7), where one or more interrupts may be missed if the actions corresponding to an interrupt are not finished by the time the next interrupt occurs. As an example, the minimum and maximum

number of clock cycles needed for executing a piece of code can be constantly measured and monitored by the user at high level. All DSPs provide means to measure the number of clock cycles required to execute a certain amount of code; the number of clock cycles can then be easily converted into absolute time. Figure 56 shows a possible implementation on ADI SHARC DSPS of the execution duration of a code called ‘critical action’. SHARC processors have a set of registers called *emuclk* and *emuclk2* which make up a 64-bit counter. This counter is unconditionally incremented during every instruction cycle on the DSP and is not affected by factors such as cache-misses or wait-states. Every time *emuclk* wraps to zero, *emuclk2* is incremented by one. By determining the difference in the *emuclk* value between before and after the critical action, the DSP developer can determine the number of clock cycles — hence the time — to execute the code.

```
ustat3 = emuclk; // read time @execution start
CRITICAL ACTION
r0 = emuclk;      // read time @execution end
r1 = ustat3;
r1 = r0 - r1;     // calculate execution time
```

Fig. 56: Execution duration measurement with *emuclk* registers in the ADI SHARC DSP

10 Real-time design flow: system integration

The system integration is one of the final parts in the system development process. This phase is extremely important as it can determine the success or the failure of a whole system. In fact, a system which is well integrated can become operational, while a system only partially integrated will often remain a ‘machine development’ tool, easily forgotten.

During the system integration phase, the system is commissioned with respect to data exchange with the control infrastructure and the application program(s). Two or more groups, such as Instrumentation, Controls and Operation, can be involved in this effort, depending on the laboratory’s organization. As a consequence, a coordination and specification work is required.

Good system integration practices will depend on the laboratory’s organization as well as on the system architecture. There are, however, some guidelines that can be applied to most cases.

- Guideline 1: Work in parallel

All software layers needed in a system should be planned in parallel. Waiting until the low-level part is completed before starting with the specification and/or with the development of the other layers may result in unacceptable delays.

- Guideline 2: About interfaces

Section 9.5 summarized the many interfaces that can exist in a system. For a successful system integration it is essential that the interfaces are specified clearly, are agreed upon with all different parties and are fully documented. Recipes on how to set up different software components of the system or on how to interact with them can be really useful and speed up considerably system development as well as debugging. It is recommended that all documents be kept updated and stored on servers accessible by all parties involved. Remember: good fences make good neighbours!

- Guideline 3: Always include checks on the DSP inputs validity

The validity of all control inputs to the DSP should be checked. Alarms or warnings should be raised if a control value falls outside the allowed range. This mechanism will help the system integration part and could even prevent serious malfunctioning from happening.

- Guideline 4: Add spare parameters

It is strongly recommended to map spare parameters between the DSP and application program; they should have different formats for maximum flexibility. These spare parameters allow adding debugging features or making some small update without modifications to the intermediate software layers.

- Guideline 5: Code release and validation

The source code (and if possible the corresponding executable, too) should be saved together with a description of its features and implemented interfaces. This will allow going back to previous working releases in case of problems. Procedure and data sets should also be defined for code validation.

11 Summary and conclusions

This paper aimed at providing an overview of DSP fundamentals and DSP-based system design. The DSP hardware and software evolution was discussed in Sections 1 and 2, together with typical DSP applications to the accelerator sector. Section 3 showed the main features of DSP core architectures and Section 4 gave an overview of DSP peripherals. The real-time design flow was introduced in Section 5 and its steps were discussed in detail in Section 6 (software development), Section 7 (debugging), Section 8 (analysis and optimization), Section 9 (system design) and Section 10 (system integration).

Existing chip examples were often given and referenced to technical manuals or application notes. Examples of DSP use in existing accelerator systems were also given whenever possible.

The DSP field is of course very large and more information, as well as hands-on practice, is required to become proficient in it. However, the author hopes that this document and the references herein can be useful starting points for anyone wishing to work with DSPs.

References

- [1] T. Shea, Types of Accelerators and Specific Needs, these CAS proceedings.
- [2] M. E. Angoletta, Digital Signal Processing In Beam Instrumentation: Latest Trends And Typical Applications, DIPAC'03, Mainz, Germany, 2003.
- [3] M. E. Angoletta, Digital Low-Level RF, EPAC'06, Edinburgh, Scotland, 2006.
- [4] J. Eyre and J. Bier, The Evolution of DSP Processors, *IEEE Signal Proc. Mag.*, vol. 17, Issue 2, March 2000, pp. 44–51.
- [5] J. Glossner et al., Trends In Compilable DSP Architecture, Proceedings of IEEE Workshop on Signal Processing Systems (SiPS) 2000, November 2000, Lafayette, LA, USA, pp. 181–199, ISBN 0-7803-6488-0.
- [6] R. Restle and A. Cron, TMS320Cc30-IEEE Floating-Point Format Converter, Texas Instruments Application Report SPRA400, 1997.
- [7] E.A. Lee, Programmable DSP Architectures: Part I, *IEEE ASSP Mag.*, October 1988, pp. 4–19.

- [8] E.A. Lee, Programmable DSP Architectures: Part II, *IEEE ASSP Mag.*, January 1989, pp. 4–14.
- [9] J. Eyre, The Digital Signal Processor Derby, *IEEE Spectrum*, June 2001, pp. 62–68.
- [10] L. Geppert, High-Flying DSP Architectures, *IEEE Spectrum*, Nov. 1998, pp. 53–56.
- [11] P. Lapsley, J. Bier, A. Shoham and E. A. Lee, DSP Processor Fundamentals: Architectures and Features, IEEE Press, ISBN 0-7803-3405-1, 1997.
- [12] TMS320C621x/C671x DSP Two-Level Internal Memory Reference Guide, Texas Instruments Literature Number SPRU609A, November 2003.
- [13] M. Anderson, Advanced Processor Features And Why You Should Care, Part 1 And 2, talks ESC-404 and ESC-424, Embedded Systems Conference, Silicon Valley 2006.
- [14] Analog Devices Team, The Memory Inside: TigerSHARC Swallows Its DRAM, Special Feature SHARC Bites Back, COTS Journal, December 2003.
- [15] S. Srinivasan, V. Cuppu and B. Jacob, Transparent Data-Memory Organisations For Digital Signal Processors, Proceedings of CASES'01, International Conference on Compilers, Architecture and Synthesis for Embedded Systems, Caches and Memory Systems Session, Atlanta, Georgia, USA, 2001, pp. 44–48.
- [16] TMS320C620x/C670x DSP Program and Data Memory Controller/Direct Memory Access (DMA) Controller - Reference Guide, Texas Instruments Literature Number SPRU234, July 2003.
- [17] D. Talla, L.K. John, V. Lapinskii and B.L. Evans, Evaluating Signal Processing And Multimedia Applications On SIMD, VLIW And Superscalar Architectures, Proceedings of the International Conference on Computer Design, September 2000, Austin TX, USA, ISBN 0-7695-0801-4.
- [18] J. A. Fisher, P. Farabosci and C. Young, A VLIW Approach To Architecture, Compilers And Tools, Morgan Kaufmann Publisher, December 2004, ISBN-13 978-1558607668.
- [19] Extended-Precision Fixed-Point Arithmetic On The Blackfin Processor Platform, Analog Devices Engineer-to-Engineer Note EE-186, May 2003.
- [20] IEEE Standard For Radix-Independent Floating-Point Arithmetic, ANSI/IEEE Std 854–1987.
- [21] D. Goldber, What Every Computer Scientist Should Know About Floating-Point Arithmetic, *Comput. Surv.*, March 1991.
- [22] ADSP-21160 SHARC DSP – Hardware Reference, Revision 3.0, November 2003, Analog Devices Part Number 82-001966-01.
- [23] TMS320C6713, TMS320C6713B Floating-Point Digital Signal Processors, Texas Instruments Manual SPRS186I, December 2001, Revised May 2004.
- [24] ADSP-BF533 Blackfin Processor – Hardware Reference, Revision 3.1, June 2005, Analog Devices Part Number 82-002005-01.
- [25] TMS320C6000 DSP – Multichannel Buffered Serial Port (McBSP) – Reference Guide, Texas Instruments Literature Number SPRU580C, May 2004.
- [26] TMS320C6000 DSP – Multichannel Audio Serial Port (McASP) – Reference Guide, Texas Instruments Literature Number SPRU041C, August 2003.
- [27] R. Kilgore, Link Port Open Systems Interconnect Cable Standard, Analog Devices Engineer-to-Engineer Note EE-106, October 1999.
- [28] J. Kent and J. Sondermeyer, Interfacing ADSP-BF533/BF561 Blackfin Processors to High-Speed Parallel ADCs, Analog Devices Application Note AN-813.
- [29] TMS320C6000 DSP Inter-Integrated Circuit (I2C) Module – Reference Guide, Texas Instruments Literature Number SPRU581A, October 2003.

- [30] TMS320C6000 DSP Peripheral Component Interconnect (PCI) – Reference Guide, Texas Instruments Literature Number SPRUA75A, October 2002.
- [31] TMS320C6000 DSP Host Port Interface (HPI) – Reference Guide, Texas Instruments Literature Number SPRU578A, September 2003.
- [32] TMS320C6000 DSP General-Purpose Input/Output (GPIO) – Reference Guide, Texas Instruments Literature Number SPRU584A, March 2004.
- [33] TMS320C6000 DSP 32-Bit Timer – Reference Guide, Texas Instruments Literature Number SPRU582A, March 2004.
- [34] TMS320C6000 DSP Software-Programmable Phase-Locked Loop (PLL) Controller – Reference Guide, Texas Instruments Literature Number SPRU233B, March 2004.
- [35] TMS320C6000 DSP Power-Down Logic and Modes – Reference Guide, Texas Instruments Literature Number SPRU728, October 2003.
- [36] TMS320C620x/C670x DSP Boot Modes and Configuration – Reference Guide, Texas Instruments Literature Number SPRU642, July 2003.
- [37] TMS320C6000 Peripherals – Reference Guide, Texas Instruments Literature Number SPRU109D, February 2001.
- [38] TMS320C6000 DSP External Memory Interface (EMIF) – Reference Guide, Texas Instruments Literature Number SPRU266A, September 2003.
- [39] T. Kugelstadt, A Methodology Of Interfacing Serial A-to-D converters to DSPs, Analog Application Journal, February 2000, pp. 1–10.
- [40] Efficiently Interfacing Serial Data Converters To High-Speed DSPs, Analog Application Journal, August 2000, pp. 10–15.
- [41] J. Sondermeyer, J. Kent, M. Kessler and R. Gentile, Interfacing The ADSP-BF535 Blackfin Processor To High-Speed Converters (Like Those On The AD9860/2) Over The External Memory Bus, Analog Devices Engineer-to-Engineer Note EE-162, June 2003.
- [42] M. E. Angoletta et al., Beam Tests Of A New Digital Beam Control System For The CERN LEIR Accelerator, PAC ‘05, Knoxville, Tennessee, 2005.
- [43] D. Dahnoun, Bootloader, Texas Instruments University Program, Chapter 9, 2004.
- [44] Code Composer Studio IDE Getting Started Guide – Users Guide, Texas Instruments Literature Number SPRU509F, May 2005.
- [45] V. Wan and K-S. Lee, Automated Regression Tests And Measurements With The CCStudio Scripting Utility, Texas Instruments Application Report SPRAAB7, October 2005.
- [46] A. Campbell, K-S. Lee and D. Sale, Creating Device Initialization GEL Files, Texas Instruments Application Report SPRAA74A, December 2004.
- [47] M.E. Angoletta et al., The New Digital-Receiver-Based System for Antiproton Beam Diagnostics, PAC 2001, Chicago, Illinois, 2001.
- [48] D. Dart, DSP/BIOS Technical Overview, Texas Instruments Application Report SPRA780, August 2001.
- [49] TMS320C6000 Optimizing Compiler – User’s Guide, Texas Instruments Literature Number SPRU187L, May 2004.
- [50] TMS320C6000 Assembly Language Tools – User’s Guide, Texas Instruments Literature Number SPRU186N, April 2004.
- [51] Rewind User’s Guide, Texas Instruments Literature Number SPRU713A, April 2005.
- [52] TMS320C6000 Instruction Set Simulator – Technical Reference, Texas Instruments Literature Number SPRU600F, April 2005.
- [53] C. Brokish, Emulation Fundamentals for TI’s DSP Solutions, Texas Instruments Application Report SPRA439C, October 2005.

- [54] VisualDSP++ 4.5 – User’s Guide, Revision 2.0, April 2006, Analog Devices Part Number 82-000420-02.
- [55] B. Novak, XDS560 Emulation Technology Brings Real-time Debugging Visibility to Next Generation High-Speed Systems, Texas Instruments Application Report SPRA823A, June 2002.
- [56] H. Thampi, J. Govindarajan, DSP/BIOS, RTDX and Host-Target Communications, Texas Instruments Application report SPRA895, February 2003.
- [57] X. Fu, Real-Time Digital Video Transfer Via High-Speed RTDX, Texas Instruments Application report SPRA398, May 2002.
- [58] S. Jung, Y. Paek, The Very Portable Optimizer For Digital Signal Processors, Proceedings of CASES’01, International Conference on Compilers, Architecture and Synthesis for Embedded Systems, Compilers and Optimization Session, Atlanta, Georgia, USA, 2001, pp. 84–92.
- [59] D. Dahnoun, Linear Assembly, Texas Instruments University Program, Chapter 7, 2004.
- [60] Analysis Toolkit v1.3 for Code Composer Studio – User’s Guide, Texas Instruments Literature Number SPRU623D, April 2005.
- [61] V. Wan and P. Lal, Simulating RF3 to Leverage Code Tuning Capabilities, Texas Instruments Application Report SPRAA73, December 2004.
- [62] TMS320C6000 Optimizing Compiler – User’s Guide, Texas Instruments Literature Number SPRU197L, May 2004.
- [63] Analog Devices Team, Fast Floating-Point Arithmetic Emulation on Backfin Processors, Analog Devices Engineer-to-Engineer Note EE-185, August 2007.
- [64] Y-T. Cheng, TMS320C6000 Integer Division, Texas Instruments Application Report SPRA707, October 2000.
- [65] V. Wan and E. Young, Power Management in an RF5 Audio Streaming Application Using DSP/BIOS, Texas Instruments Application Report SPRAA19A, August 2005.
- [66] D. Menard, D. Chillet and O. Sentieys, Floating-To-Fixed-Point Conversion For Digital Signal Processors, EURASIP *J. Appl. Signal Proc.*, vol. 2006, Article ID 96421.
- [67] F. Culloch, Speeding the Development of Multi-DSP Applications, *Embedded Edge*, June 2001, pp. 22–29.
- [68] G. Cooper and J. Hunter, Configuring Code Composer Studio For Heterogeneous Debugging, Texas Instruments Application Report SPRA752, May 2001.
- [69] R. F. Hobson, A. R. Dyck, K. L. Cheung and B. Ressi, Signal Processing With Teams Of Embedded Workhorse Processors, EURASIP *J. Embedded Syst.*, vol. 2006, Article ID 69484.
- [70] M. Kokaly-Bannourah, Introduction To TigerSHARC Multiprocessor Systems Using VisualDSP++, Analog Devices Engineer-to-Engineer Note EE-167, April 2003.
- [71] M. Kokaly-Bannourah, Using The Expert Linker For Multiprocessor LDFs, Analog Devices Engineer-to-Engineer Note EE-202, May 2005.
- [72] ADSP-TS101 TigerSHARC Processor – Hardware Reference, Revision 1.1, May 2004, Analog Devices Part Number 82-001996-01.
- [73] TMS320C64x DSP Turbo-Decoder Coprocessor (TCP) – Reference Guide, Texas Instruments Literature Number SPRU534A, November 2003.
- [74] TMS320C64x DSP Viterbi-Decoder Coprocessor (VCP) – Reference Guide, Texas Instruments Literature Number SPRU533C, November 2003.
- [75] P. Cohrs, W. Powell and E. Williams, Creating a Dual-Processor Architectures for Digital Audio, *Embedded Edge*, June 2002, pp. 14–19.
- [76] P.E. Dodd and W.L. Massegill, Basic Mechanism of Single-Event Upset in Digital Microelectronics, *IEEE Trans. Nucl. Sci.*, vol. 50, No. 3, June 2003, pp. 583–602.

- [77] M. Vijay and R. Mittal, Algorithm-Based Fault Tolerance: A Review, *Microproc. Microsyst.*, vol. 21, No. 3, Dec. 1997, pp. 151–161.
- [78] Q. King et al., The All-Digital Approach To LHC Power Converter Current Control, CERN SL-2002-002 PO.
- [79] H. Schmickler, Usage Of DSP And In Large Scale Power Converter Installations (LHC), these CAS proceedings.
- [80] Q. King et al., Radiation Tests On The LHC Power Converter Control Electronics, Université Catholique De Louvain-La Neuve (UCL), CERN AB Note 2003-041 PO.
- [81] J. Weber et al., PEP-II Transverse Feedback Electronics Upgrade PAC05, Knoxville, 2005, p. 3928.

Introduction to FPGA design

J. Serrano

CERN, Geneva, Switzerland

Abstract

This paper presents an introduction to digital hardware design using Field Programmable Gate Arrays (FPGAs). After a historical introduction and a quick overview of digital design, the internal structure of a generic FPGA is discussed. We then describe the design flow, i.e., the steps needed to go from design idea to actual working hardware. Digital signal processing is an important area where FPGAs have found many applications in recent years. Therefore a complete section is devoted to this subject. The paper finishes with a discussion of important peripheral concepts essential for success in any project involving FPGAs.

1 Historical introduction

Digital electronics is concerned with circuits which represent information using a finite set of output states [1]. Most of the applications use in fact just two states, which are often labelled ‘0’ and ‘1’. Behind this choice is the fact that the whole Boolean formalism then becomes available for the solution of logic problems, and also that arithmetic using binary representations of numbers is a very mature field.

Different mappings between the two states and the corresponding output voltages or currents define different logic families. For example, the Transistor–Transistor Logic (TTL) family defines an output as logic ‘1’ if its voltage is above a certain threshold (typically 2.4 V). For the same family, if we set the input threshold for logic ‘1’ as 2 V, we will have a margin of 0.4 V which will allow us to interconnect TTL chips inside a design without the risk of misinterpretation of logic states. This complete preservation of information even in the presence of moderate amounts of noise is what has driven a steady shift of paradigm from analog to digital in many applications. Here we see as well another reason for the choice of binary logic: from a purely electrical point of view, having only two different values for the voltages or currents used to represent states is the safest choice in terms of design margins.

Historically, TTL chips from the 74 series fuelled an initial wave of digital system designs in the 1970s. From this seed, we shall focus on the separate branches that evolved to satisfy the demand for programmability of different logic functions. By programmability, we mean the ability of a designer to affect the logic behaviour of a chip after it has been produced in the factory.

A first improvement in the direction of programmability came with the introduction of gate arrays, which were nothing else than a chip filled with NAND gates that the designer could interconnect as needed to generate any logic function he desired. This interconnection had to happen at the chip design stage, i.e., *before* production, but it was already a convenient improvement over designing everything from scratch. We had to wait until the introduction of Programmable Logic Arrays (PLAs) in the 1980s to have a really programmable solution. These were two-level AND-OR structures with user-programmable connections. Programmable Array Logic (PAL) devices were an improvement in performance and cost over the PLA structure. Today, these devices are collectively called Programmable Logic Devices (PLDs).

The next stage in sophistication resulted in Complex PLDs (CPLDs), which were nothing else than a collection of multiple PLDs with programmable interconnections. FPGAs, in turn, contain a much larger number of simpler blocks with the attendant increase in interconnect logic, which in fact dominates the entire chip.

2 Basics of digital design

A typical logic design inside an FPGA is made of combinatorial logic blocks sandwiched in between arrays of flip-flops, as depicted in Fig. 1. A combinatorial block is any digital sub-circuit in which the current state of the outputs only depends, within the electrical propagation time, on the current state of the inputs. To this group belong all the well-known basic logic functions such as the two-input AND, OR and any combination of them. It should be noted, that logic functions of arbitrary complexity can be derived from these basic blocks. Multiplexers, encoders and decoders are all examples of combinatorial blocks. The input in Fig. 1 might be made of many bits. The circuit is also supplied with a clock, which is a simple square wave oscillating at a certain fixed frequency. The two flip-flops in the circuit, which might be flip-flop blocks in the case of a multi-bit input, are fed with the same clock and propagate the signals from their D inputs to their Q outputs every time there is a rising edge in the clock signal. Apart from that very specific instant in time, D is disconnected from Q.

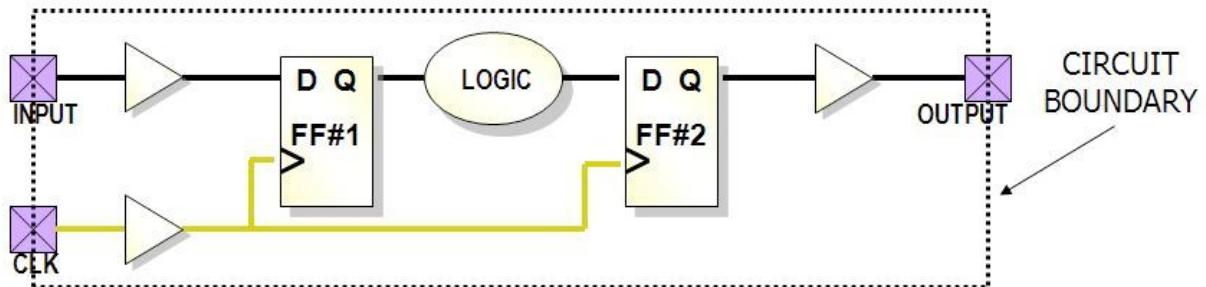


Fig. 1: A typical digital design

The structure of the circuit is thus very simple, and its application as a template covers the vast majority of digital design requirements in standard applications, such as the control of particle accelerators. The designer only needs to ensure that the worst-case propagation delay between any of the inputs to the combinatorial logic block and any of its outputs is less than one clock period. If that condition is satisfied, the inputs to the second stage of flip-flops will be stable by the time the next clock edge reaches them. As we shall see later, the process of ensuring this timing closure is nowadays automated, so the designer need only be concerned with the specification of the logic behaviour of the circuit.

The main merit of this design strategy, called *synchronous design*, is that the analysis of the possible timing failures and race conditions is greatly simplified. One might design otherwise, feeding for example the clock input of a flip-flop from the output of a combinatorial block, in what is called a ‘gated clock’ circuit, and that design might give good results during simulation. But then, slight changes in the different delays of the signals might result in circuit malfunction. Slight changes of delays in the synchronous design paradigm can be easily accommodated by taking some safety margin in the maximum allowed time for any combinatorial signal path. This simple recipe contrasts with the ‘many-body problem’ of examining effects of delay changes in asynchronous designs.

3 FPGA structure

A typical modern FPGA (see Fig. 2) provides the designer with programmable logic blocks that contain the pool of combinatorial blocks and flip-flops to be used in the design. In addition, vendors acknowledge the fact that logic is often used in conjunction with memory, and typically include variable amounts of static Random Access Memory (RAM) inside their chips. Clock conditioning has also become commonplace, and support in the form of Delay Locked Loops (DLLs) and Phase Locked Loops (PLLs) is also provided inside the same silicon chip. Finally, an FPGA chip does not lead a solitary life isolated from the rest of the world. It needs to be easily interfaced to other chips or external signals. In order to make this interfacing easier, FPGA vendors have invested a great deal of effort in enhancing the flexibility of the input/output blocks behind the chip pads. Each pad can serve as an input, an output, or both. The list of electrical standards supported is extensive, and novel techniques for maximizing bandwidth, such as clocking data in using both edges of the clock, are widely supported.

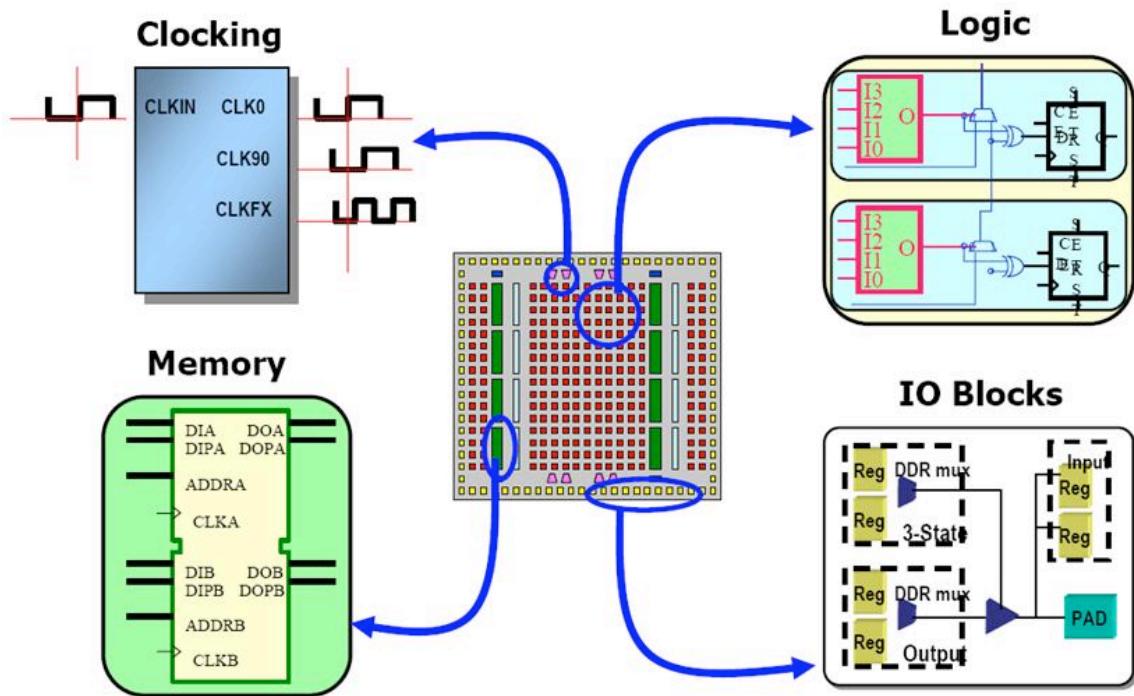


Fig. 2: Internal structure of a generic FPGA (courtesy Xilinx, Inc.)

All the components shown in Fig. 2, however, typically account for less than 20% of the silicon inside an FPGA chip. What is not shown is the large amounts of programmable interconnect and the auxiliary circuits which ‘program’ the generic blocks to become a well-defined piece of logic. This silicon inefficiency is the price to pay for programmability, and is also the reason why FPGAs have traditionally been more successful in high-end, low-volume applications in the past, with Application-Specific Integrated Circuits (ASICs) taking a leading role for high-volume applications. With Moore’s law, however, the line between high-end and low-end applications is continuously shifting, and FPGAs are more and more used in domains which used to be dominated by ASICs and Digital Signal Processors (DSPs).

To overcome the silicon inefficiency problem, FPGA vendors often include hardwired Intellectual Property (IP) cores inside the chips for functions identified as recurrent in many designs. These non-programmable blocks include general-purpose processors, high-speed serial interfaces, arithmetic blocks and Ethernet Medium Access Control (MAC) units.

4 Design flows

The designer facing a design problem must go through a series of steps between initial ideas and final hardware. This series of steps is commonly referred to as the ‘design flow’. First, after all the requirements have been spelled out, a proper digital design phase must be carried out. It should be stressed that the tools supplied by the different FPGA vendors to target their chips do not help the designer in this phase. They only enter the scene once the designer is ready to translate a given design into working hardware.

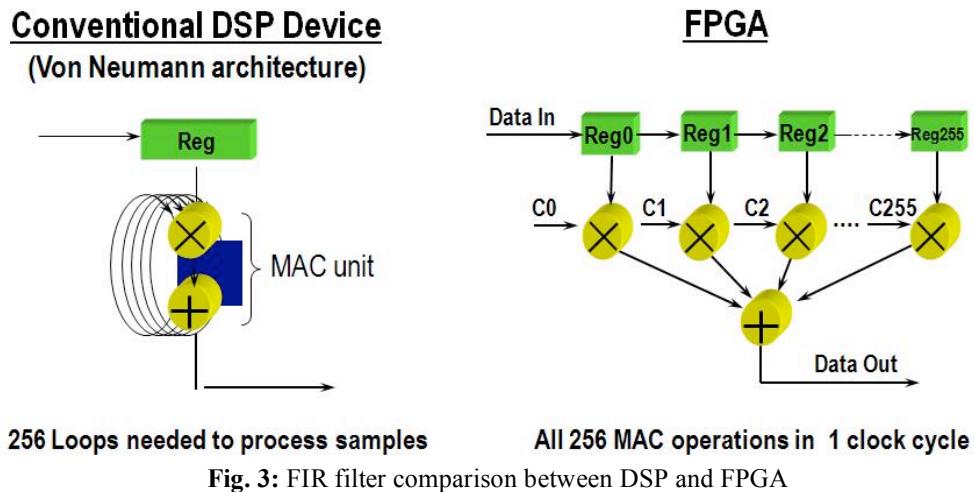
The most common flow nowadays used in the design of FPGAs involves the following subsequent phases:

- Design entry. This step consists in transforming the design ideas into some form of computerized representation. This is most commonly accomplished using Hardware Description Languages (HDLs). The two most popular HDLs are Verilog and the Very High Speed Integrated Circuit HDL (VHDL) [2]. It should be noted that an HDL, as its name implies, is only a tool to *describe* a design that pre-existed in the mind, notes, and sketches of a designer. It is not a tool to design electronic circuits. Another point to note is that HDLs differ from conventional software programming languages in the sense that they don’t support the concept of sequential execution of statements in the code. This is easy to understand if one considers the alternative schematic representation of an HDL file: what one sees in the upper part of the schematic cannot be said to happen before or after what one sees in the lower part.
- Synthesis. The synthesis tool receives HDL and a choice of FPGA vendor and model. From these two pieces of information, it generates a netlist which uses the primitives proposed by the vendor in order to satisfy the logic behaviour specified in the HDL files. Most synthesis tools go through additional steps such as logic optimization, register load balancing, and other techniques to enhance timing performance, so the resulting netlist can be regarded as a very efficient implementation of the HDL design.
- Place and route. The placer takes the synthesized netlist and chooses a place for each of the primitives inside the chip. The router’s task is then to interconnect all these primitives together satisfying the timing constraints. The most obvious constraint for a design is the frequency of the system clock, but there are more involved constraints one can impose on a design using the software packages supported by the vendors.
- Bit stream generation. FPGAs are typically configured at power-up time from some sort of external permanent storage device, typically a flash memory. Once the place and route process is finished, the resulting choices for the configuration of each programmable element in the FPGA chip, be it logic or interconnect, must be stored in a file to program the flash.

Of these four phases, only the first one is human-labour intensive. Somebody has to type in the HDL code, which can be tedious and error-prone for complicated designs involving, for example, lots of digital signal processing. This is the reason for the appearance, in recent years, of alternative flows which include a preliminary phase in which the user can draw blocks at a higher level of abstraction and rely on the software tool for the generation of the HDL. Some of these tools also include the capability of simulating blocks which will become HDLs with other blocks which provide stimuli and processing to make the simulation output easier to interpret. The concept of hardware co-simulation is also becoming widely used. In co-simulation, stimuli are sent to a running FPGA hosting the design to be tested and the outputs of the design are sent back to a computer for display (typically through a Joint Test Action Group (JTAG), or Ethernet connection). The advantage of co-simulation is that one is testing the real system, therefore suppressing all possible misinterpretations present in a pure simulator. In other cases, co-simulation may be the only way to simulate a complex design in a reasonable amount of time.

5 Digital signal processing using FPGAs

Clearly the main advantage of FPGAs over conventional DSPs to perform digital signal processing is their capability to exploit parallelism, i.e., replication of hardware functions that operate concurrently in different parts of the chip. Figure 3 shows how a Finite Impulse Response (FIR) filter could be implemented in both platforms. While the DSP needs 256 clock ticks to calculate an output sample, the FPGA generates a new sample for every clock cycle. Even if DSP chips can be clocked faster than FPGAs, the difference is in no case larger than a factor of 10. If one adds that many such filters can exist concurrently and interact inside the same FPGA, it is easy to see that DSPs are no match for FPGAs in high-performance signal processing applications [3].



Another advantage of FPGAs is the flexibility for trading off between area and speed until very late in the design cycle. Figure 4 shows three different implementations of a sum of products, from a full expansion using more silicon to a minimalist implementation which takes more clock cycles to generate a result.

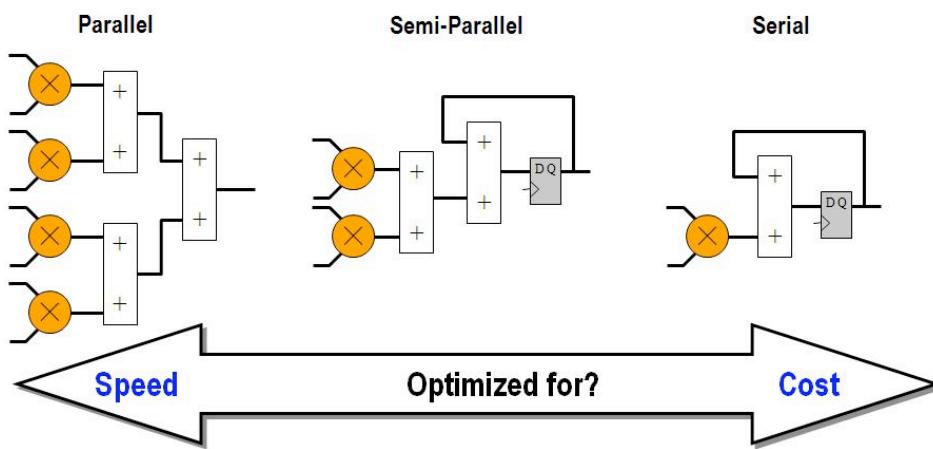


Fig. 4: Illustrating the speed/area trade-off in FPGAs

In this section, we give a basic introduction to fixed-point arithmetic and then touch briefly upon two interesting techniques for processing fixed-point signals: distributed arithmetic and the COordinate Rotation DIgital Computer (CORDIC) algorithm. The interested reader will find more details in the references at the end of this document.

5.1 Fixed-point arithmetic

In FPGA design, one typically uses a two's complement fixed-point representation of numbers. Floating point design is perfectly feasible, but the high-performance applications typically targeted by FPGAs can very often be served adequately by using enough bits in a fixed-point representation. This is another advantage of FPGAs: the possibility to tailor the bus widths in different parts of the design to satisfy the final precision requirements. Figure 5 shows an example of fixed-point two's complement representation, where we have taken three of the bits for the integer part and five for the fractional part. In reality, as we shall see, an adder or any other arithmetic circuit does not know about our decision on how many bits to interpret as fractional. This is purely an interpretational issue, so for all practical purposes, one can think of fixed-point arithmetic as integer arithmetic.

digit worth									decimal value
$-(2^2)$	2^1	2^0	\bullet	2^{-1}	2^{-2}	2^{-3}	2^{-4}	2^{-5}	
-4	2	1	\bullet	0.5	0.25	0.125	0.0625	0.03125	
0	0	0	\bullet	0	0	0	0	1	0.03125
0	0	0	\bullet	0	0	0	1	0	0.0625
1	0	1	\bullet	0	0	0	0	0	-3.0
1	1	0	\bullet	0	0	1	1	1	-1.78125
1	1	1	\bullet	1	1	1	1	1	-0.03125

Fig. 5: Fixed-point two's complement representation of signed numbers using three integer bits and five fractional bits

As an introductory example, let us see how one can make a circuit that performs simple addition or subtraction using logic gates. This will be a combinatorial circuit with nine inputs (say four per input and one for controlling if we add or subtract) and five outputs (the final result). Notice that we need to have one extra bit in the output because the addition/subtraction of two 4-bit numbers can result in a 5-bit number.

We start with the full adder circuit of Fig. 6. It is easy to see that this circuit made of AND, OR and XOR gates takes two bits and a carry (maybe from a preceding stage) and generates a proper sum and carry out.

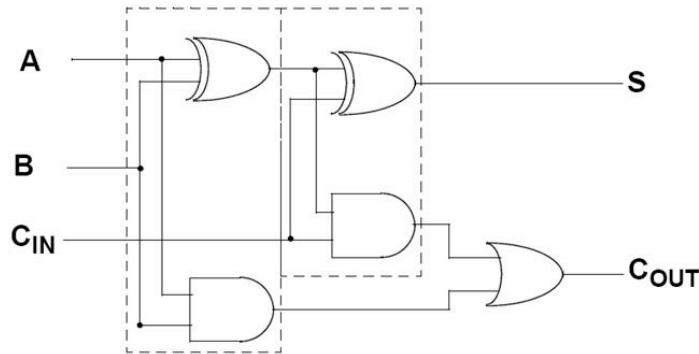
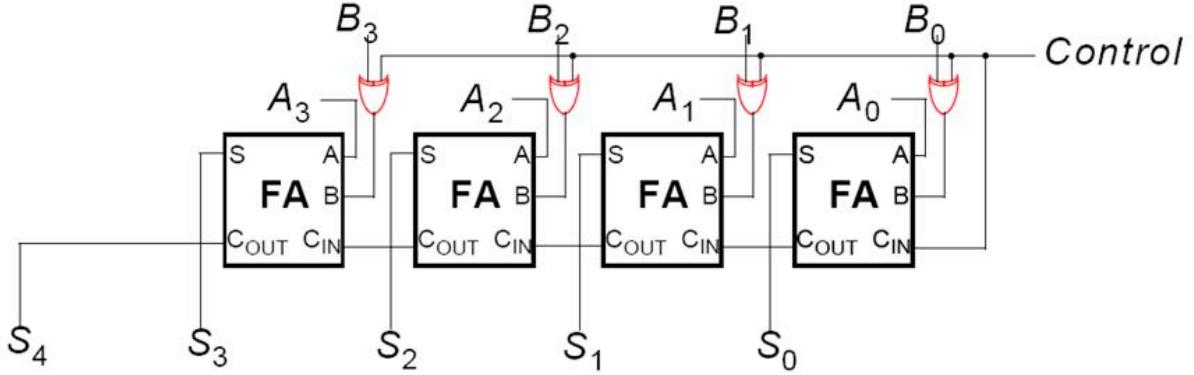
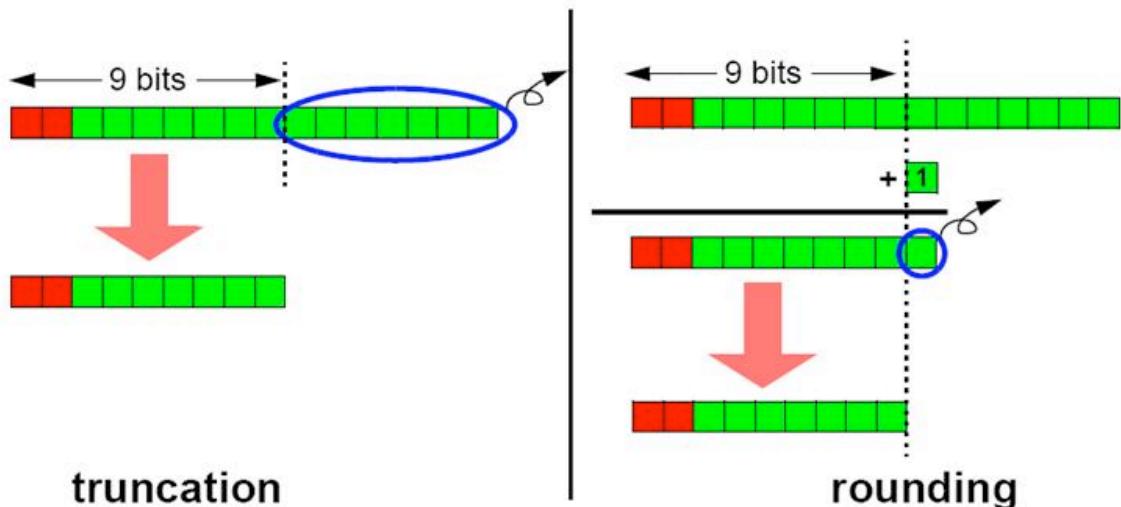


Fig. 6: A full adder circuit

Out of many of these Full Added (FA) cells, one can build the circuit of Fig. 7, which takes two 4-bit signed numbers and adds them together if the Control signal is '0'. Otherwise, a subtraction is performed. Although these days one would generate this circuit with a simple HDL statement, it is enlightening to understand what the underlying hardware looks like. More complicated blocks to multiply, divide, take a square root, etc. can be synthesized using these basic blocks and the reader is referred to the specialized literature for details [3].

**Fig. 7:** 4-bit full add/subtract

One question that arises immediately is what to do with this arithmetic bus that gets wider and wider as we cascade more and more operations one after the other. At some point, the bus width will become inconveniently large. We will be wasting bits with unnecessary information, and our timing constraints could be compromised as the combinatorial paths traverse more layers of logic before hitting the next flip-flop. One example could be a feedback system that generates an analog signal towards a Radio Frequency (RF) cavity using a 16-bit Digital to Analog Converter (DAC) fed by an FPGA. There would be no point in keeping an internal representation of say 50 bits and then collapse it all at the very end to feed the DAC with only 16 bits. The solution to this problem is to control the width after each operation by judiciously choosing a suitable number of bits to represent the intermediate results. Figure 8 shows two ways of doing this: truncation and rounding. In truncation, some of the fractional bits are taken out before feeding the result to the next stage. In rounding, a ‘1’ is added to the most significant bit (of the ones to be taken out) before truncation.

**Fig. 8:** Truncation vs. rounding in fixed-point representation

Notice that in two’s complement, truncation is a biased operation. The output of truncation will always be a smaller number than the input. If an unbiased scheme is needed, then rounding should be used at the expense of the extra addition involved. The loss in precision incurred by taking some of the bits out can be studied statistically by modelling rounding or truncation as a source of white noise with an amplitude dependent on the number of bits eliminated [4].

5.2 Distributed arithmetic

Digital signal processing is all about sums of products. For example, if a generic filter is fed with an input sequence $x[n]$, we can write its output as

$$y = \sum_{n=0}^{N-1} c[n] \cdot x[n] \quad (1)$$

where $c[n]$ are the filter coefficients. If these coefficients are constant, and assuming the input signal to be B bits wide, we can rearrange the terms in a sequence that will end up suggesting an alternative hardware implementation. We begin by re-writing Eq. (1) as

$$y = \sum_{n=0}^{N-1} \left(c[n] \cdot \sum_{b=0}^{B-1} x_b[n] \cdot 2^b \right) \quad (2)$$

where $x_b[n]$ is bit number b of $x[n]$, i.e., either ‘0’ or ‘1’. After rearranging:

$$y = \sum_{b=0}^{B-1} 2^b \cdot \left(\sum_{n=0}^{N-1} c[n] \cdot x_b[n] \right) \quad (3)$$

and the term in parentheses can be implemented as a Look Up Table (LUT) with N inputs, as suggested in Fig. 9.

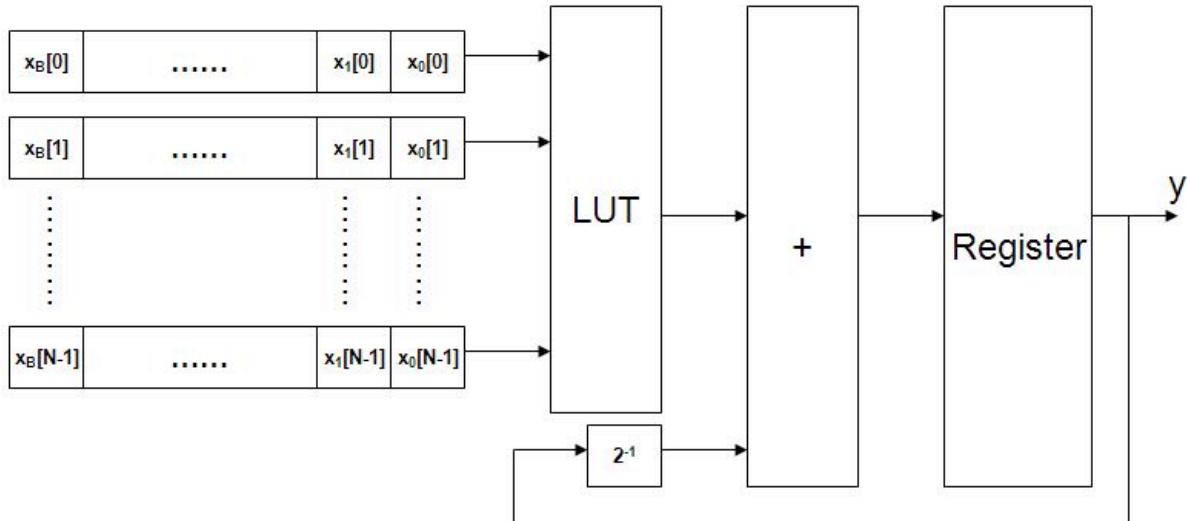


Fig. 9: Distributed arithmetic implementation of a filter

The filter implemented in this way has no need of hardware multipliers, and generates a result every B ticks, independent of the filter length N . By increasing the number of LUTs and replicating hardware, one can trade off latency versus area. The extreme case would be full parallelization: replicating B times to get one output sample per clock tick.

5.3 The CORDIC algorithm

The CORDIC is a circuit that iteratively rotates an input vector $(x^{(1)}, y^{(1)})$ and generates an output vector $(x^{(2)}, y^{(2)})$, where x and y are the Cartesian coordinates of the vectors. There are two modes of operation. In rotation mode, an angle accumulator is set to the desired rotation angle, and the CORDIC approximates that angle by performing elementary rotations of decreasing angles. The output is the input vector rotated by the specified angle. In vectoring mode, the CORDIC block rotates the input vector using the same table of decreasing angles until the resulting vector is aligned with the

horizontal axis. In this mode, the result is the angle accumulated throughout the whole rotation process. The trick in the CORDIC algorithm is to constrain the set of angles to those whose tangent can be expressed as 2^{-i} , i being the iteration index. Then the rotation operations for these angles do not need any specific multipliers, since a multiplication by 2^{-i} is just a right-shift by i places. This produces a very efficient hardware implementation in terms of area and speed. Each iteration generates roughly an extra bit of precision in the result. Among the several things one can calculate with a CORDIC, we can highlight the following:

- Magnitude of a vector: it is found on the x of the output vector after operating the CORDIC in vectoring mode.
- Sine and cosine of an angle: found by feeding an input vector with $x = 1$, $y = 0$ and setting the CORDIC to work in rotation mode with the specified angle.

More uses of the CORDIC as well as a detailed description on its internal features and ways to accelerate it can be found in Ref. [5].

6 FPGAs in real-world designs

This section is devoted to design aspects which are encountered in real projects. The FPGA designer will find none of these problems while simulating behavioural HDL in a computer, but they are paramount for the success of any FPGA project.

6.1 Performance-boosting techniques

We have already discussed the benefits of synchronous design. The place-and-route tool will analyse the timing constraints and optimize the placement and routing in such a way as to meet these constraints. But what if it cannot? If the delays due to propagation through individual gates are already higher than the specified clock period, there is nothing the tool can do to meet the specification. Remember, the tool's degrees of freedom are just related to where to place blocks and how to interconnect them. The interconnect delay will never be less than 0, and it has to be added to gate propagation delays which are fully determined by the synthesized netlist. So if the gate propagation delays already exceed the specified clock period, it's mission impossible for the place-and-route tool.

Delays in modern designs can be as much as 90% due to routing and 10% due to logic. The routing bit is due to long routes and capacitive loading on the nets. Many synthesis tools automatically insert buffers in some nets to provide more current to drive the capacitive loads, therefore decreasing routing delay, as depicted in Fig. 10.

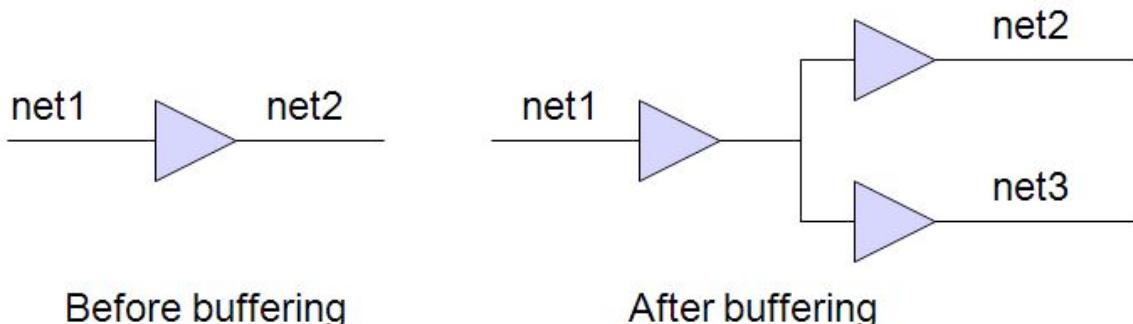


Fig. 10: Automatic buffer insertion example

The automatic replication of registers is another useful technique. This can be set as an option for those synthesis tools that support it, or it can be done by hand at the HDL level. Figure 11 illustrates the principle. The nets coming out of the flip-flop after the producer are going to four different destinations, potentially covering great lengths inside the chip. After the flip-flop is duplicated, each of the outputs only has to serve two destinations, so the timing constraints become easier. If there were combinatorial logic after the first flip-flop, it would also be replicated. The HDL specification is therefore fully respected.

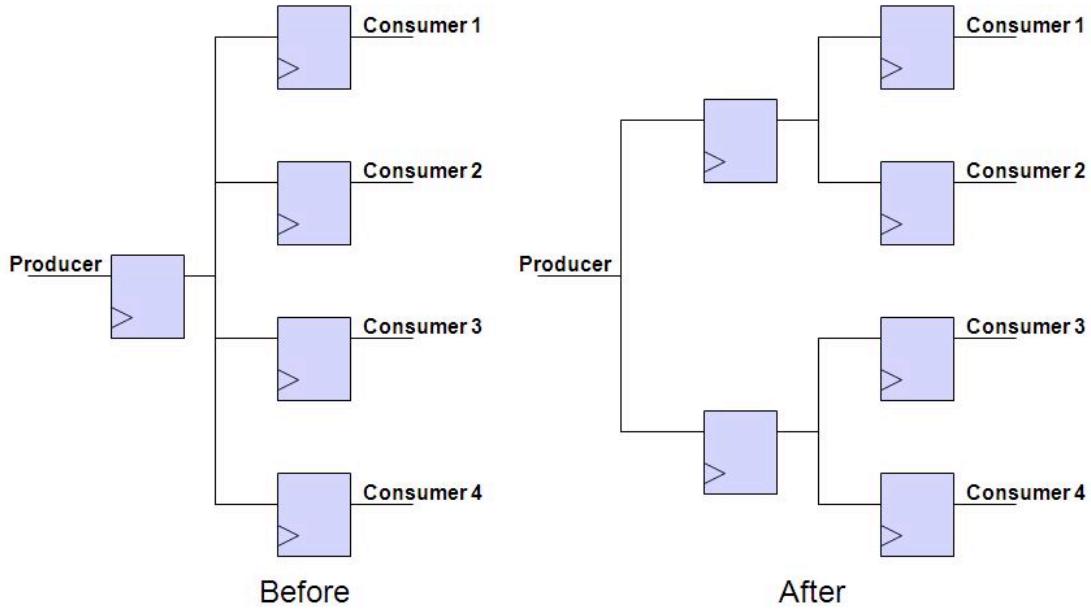


Fig. 11: Automatic replication of registers

Another problem case concerns long combinatorial delays between flip-flop stages. As we said earlier, there is nothing that the place-and-route tool can do in this case. The solution must come from the synthesis tool or the designer. Retiming — also known as register balancing — is a technique that can be used in these cases. Figure 12 shows how it works. Some of the combinatorial logic is passed to the next stage in the pipeline so that the maximum delay in each stage remains within the specifications.

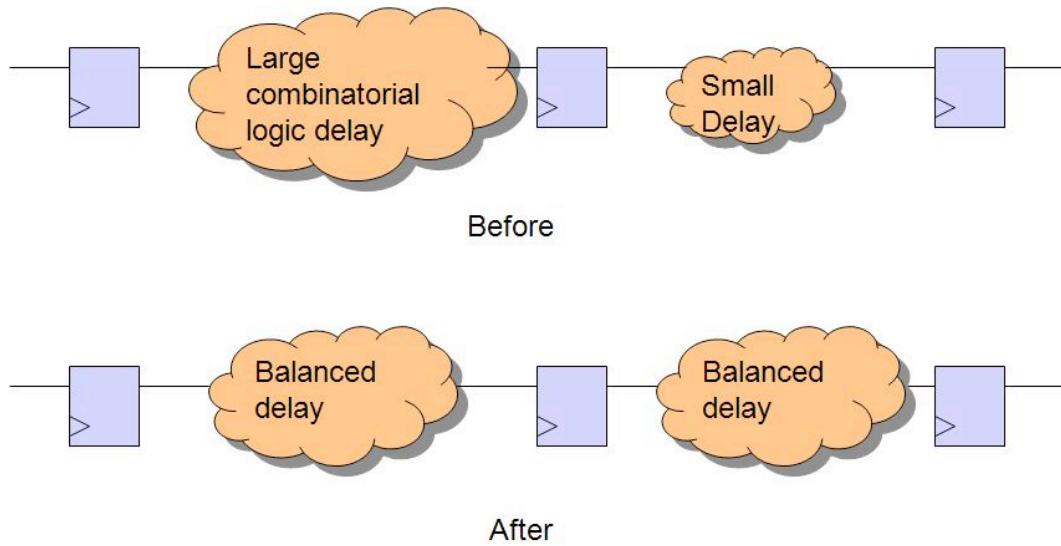


Fig. 12: Retiming

If retiming is not possible, one can always try pipelining, provided the circuit is not sensitive to the latency in the extra number of clock cycles. The principle is explained in Fig. 13. It consists in breaking up the large combinatorial delay by inserting flip-flop stages after intermediate results. In this case, it is better to modify the original design rather than using the synthesis tool, since it could lead to an incoherency between HDL sources and final hardware.

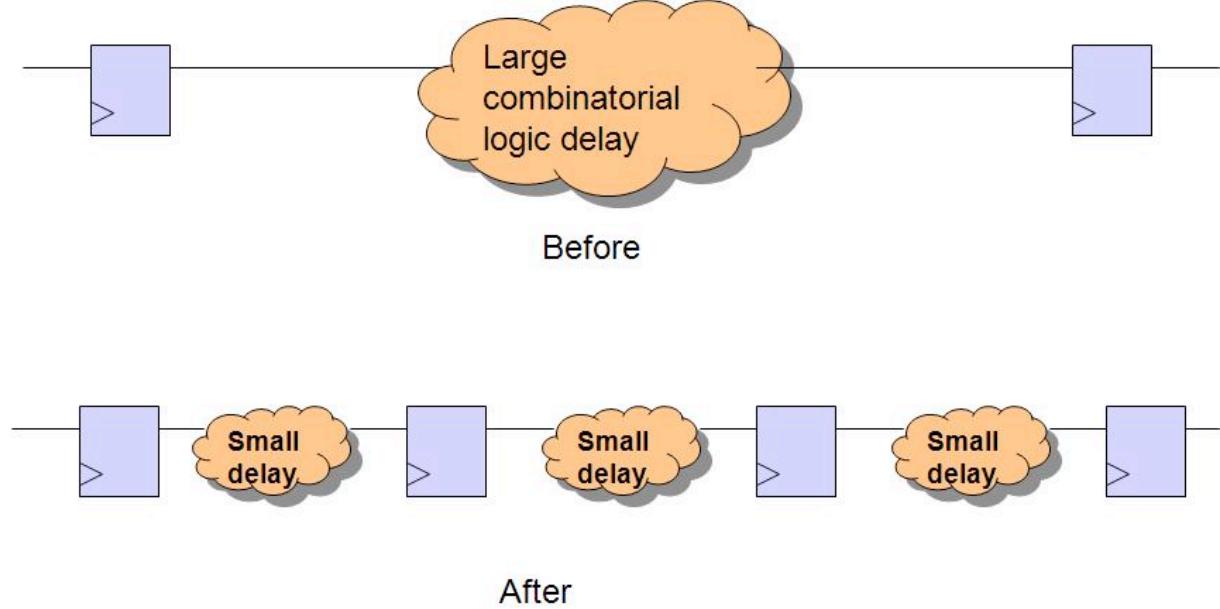


Fig. 13: Pipelining

Finally, time-multiplexing in conjunction with hardware replication can also be a powerful tool to prevent timing pathologies. The principle, depicted in Fig. 14, consists in splitting a data path in two, making each branch work at half the speed, and recombining the results at the end to regenerate a data flow at the design clock frequency.

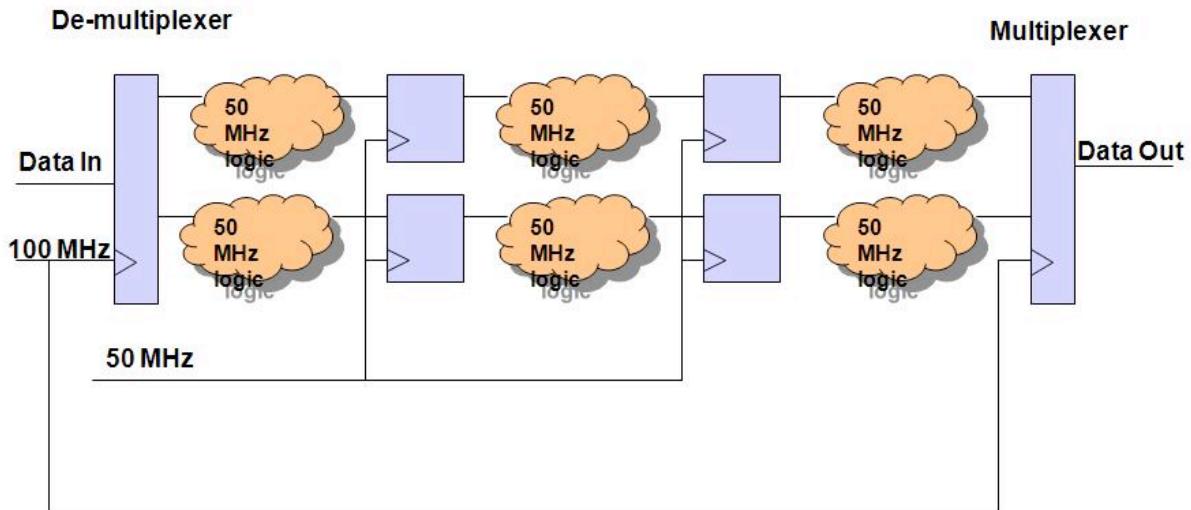
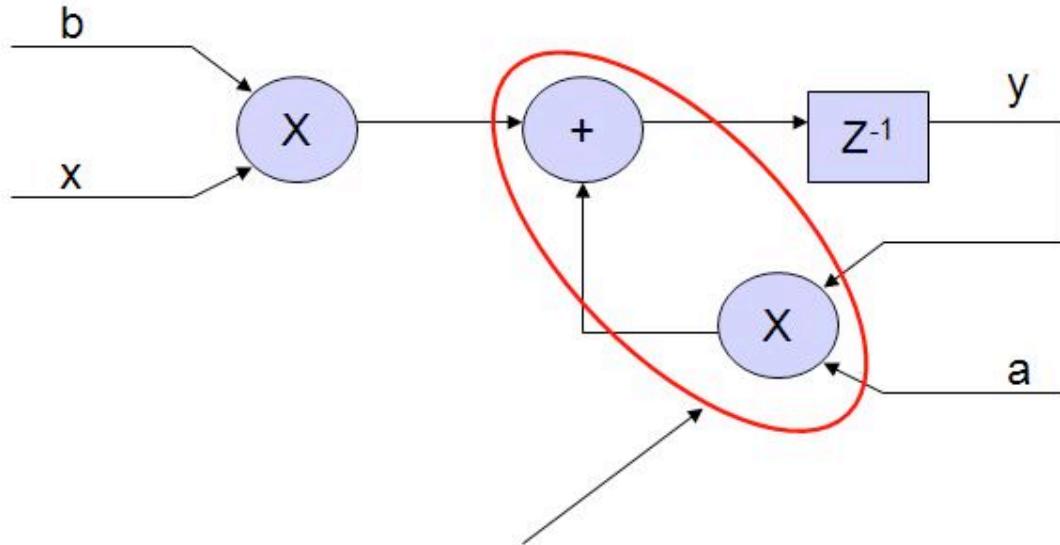


Fig. 14: Time-multiplexing

As an example of how these tools can be used in practical cases, let us examine a performance problem that arose in the phase filter of a PLL used to track bunch frequency in CERN's PS. Figure 15 shows the original filter design, a first-order Infinite Impulse Response (IIR) filter implementing the transfer function $y[n+1] = ay[n] + bx[n]$. Signal y goes back to the output flip-flop through a multiplier

and an adder, and these combinatorial delays are not compatible with the clock frequency. What can we do?



Performance bottleneck in the feedback path

Fig. 15: A simple IIR filter with a performance problem

We can calculate $y[n+2] = ay[n+1] + bx[n+1] = a^2y[n] + abx[n] + bx[n+1]$, and see what the resulting direct implementation would look like in Fig. 16.

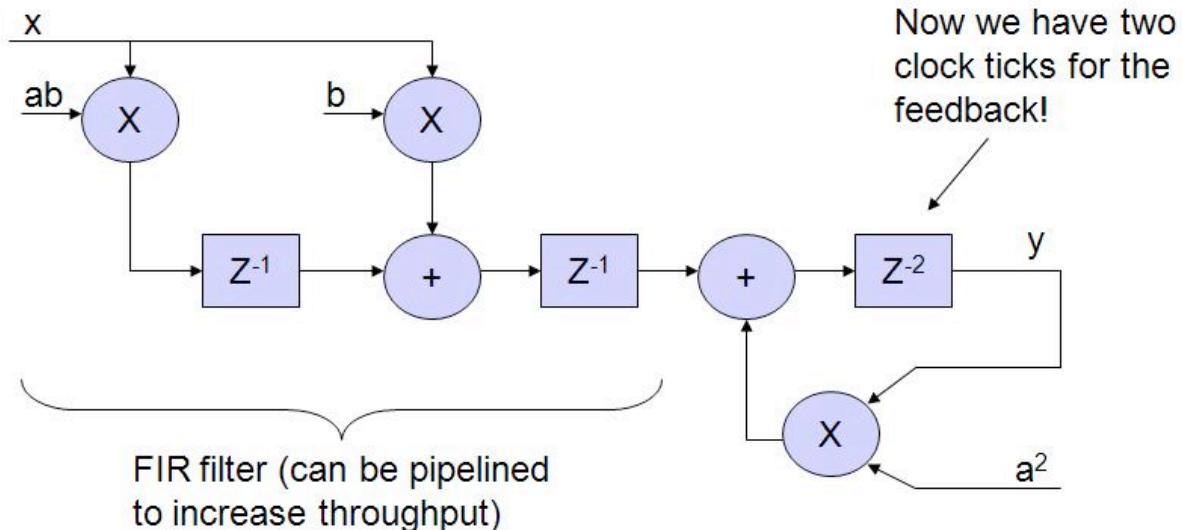


Fig. 16: Look-ahead scheme for IIR

The circuit now looks much more favourable for timing improvements. The leftmost part looks like an FIR and can be pipelined as much as necessary. The second part now contains two flip-flops in series in the feedback path, which can be used for retiming. The technique we used is called ‘look-ahead’ and is very common for boosting the speed of digital circuits.

6.2 Powering FPGAs

FPGAs are typically powered from various supply rails. They need different voltages for the internal logic, the Input/Output (I/O) circuitry, and some analog blocks like PLLs. Typical specifications

include a $\pm 5\%$ tolerance on the actual value of the voltage and monotonic ramping of the supplies during power-up. While it is not proven that ramping in a non-monotonic way would not work, FPGAs are not tested that way after manufacturing, so it is better to guarantee a monotonic ramp in order to avoid surprises. Devices also specify a minimum and a maximum ramping time for the voltage rails. Again, this is just how they are tested after production, and it is very wise to follow these guidelines.

An important aspect to bear in mind concerns in-rush current at power-up due to the decoupling capacitors on the power supply rails. If C is the total capacitance, $I_c = C * \Delta V / \Delta T$, so one might want to slow the ramping process down using a soft-start circuit in order to avoid the kick-in of protection mechanisms in regulators, which could in turn compromise monotonicity.

Sequencing of supply voltages, i.e., making one available, then another one and so on, was a required practice in old technologies, and nowadays it is only recommended. It seems sensible that the I/O stages get power only after the internal logic is properly configured. A Supply Voltage Supervisor (SVS) chip can be used to control the process. Sequencing is also good to make sure that the main (typically 5 V) rail feeding the regulators is well established (i.e., all capacitors charged) before they begin requesting current from it. Otherwise the 5 V protection could trip and spikes could appear in the output of the regulators.

The design of a proper bypassing network using capacitors is also a critical issue. A decoupling network should look like a short to ground for all the frequencies of power supply noise we want to reject. At high frequencies, like the ones of interest for this discussion, a capacitor chip can be modelled as an equivalent RLC circuit to take into account the various imperfections in its design. The parasitic inductance dominates at high frequencies, and is (almost) exclusively determined by the package type of the capacitor. The global frequency response presents a downward slope at low frequencies whose value depends on the capacitance, and an upward slope at high frequencies whose value depends on the parasitic inductance. The minimum of the curve thus depends on the capacitance value, and can be made arbitrarily wide by selecting a suitable set of capacitor values and placing them in parallel. High-value capacitors take care of low-frequency perturbations and can be placed relatively far away from the chip, while low values of capacitance (typically 10 nF), can be placed close to the chip — ideally below it — to take care of the fast perturbations. Reference [6] can be consulted for further details.

6.3 Interfacing to the outside world

Modern FPGAs have very versatile I/O blocks which make them easy to interface to other chips. In this section, we look in particular at issues which could appear when interfacing to Analog to Digital Converters (ADCs) or DACs.

Whenever a design deals with high-speed, high-pin-count parallel busses, as is the case often when interfacing FPGAs and ADCs/DACs, there is potential for noise problems. This is because the I/O drivers in the FPGAs commute state all at the same time, creating large current surges in the Power Distribution System (PDS). The PDS should be well decoupled using the appropriate mix of capacitors as discussed above, but it cannot filter all the noise at all frequencies. In addition, sampling many bits at a high frequency can pose synchronization problems. If the clock edge is very close to the transition of any of the data bits, a problem known as metastability — to be explained later — can arise. It is therefore desirable to avoid simultaneous fast-switching of large busses if possible. One example where this is possible is in the sampling of high frequency, low bandwidth analog signals. According to sampling theory, there is no need to sample them in their main Nyquist zone, i.e., with at least twice their frequency. It is sufficient to sample them at least faster than twice their bandwidth — which can be significantly slower. This can be a solution for systems where latency is more or less a secondary concern, but it might not be possible for feedback systems. Another possibility for mitigating noise problems is to choose ADC and DAC chips which use differential signalling for the

data bits and the clock. Currents in differential signals go in through one line and out of the other, without any net demand on the PDS. The use of differential signalling also creates negligible ground bounce. Ground bounce is caused by the fact that the impedance between the ground pins and the ground plane is not exactly zero. This can cause a perceived change in ground level, as seen by the digital chip, when it consumes significant current. This impedance has a resistive component but also a very important inductive component which will create voltage jumps as a function of dI/dt . Therefore, another way to improve the noise problem is to feed the digital outputs of the ADC with the lowest realistic supply voltage. Current generation ADCs can accept digital supplies in the range 2.0–2.5 V. In addition, current can be reduced by placing resistors in series with the ADC outputs.

Another aspect to bear in mind when designing FPGA systems is the state of I/O drivers during power-up. Most FPGA chips provide a possibility, through external jumpers, of selecting whether the I/Os will be pulled-up or tri-stated during the power-up process. The tri-state option lets the designer control the power-up state of each pin through external pull-up or pull-down resistors. This is important if glitch-free operation is requested during startup.

6.4 Clock domains and metastability

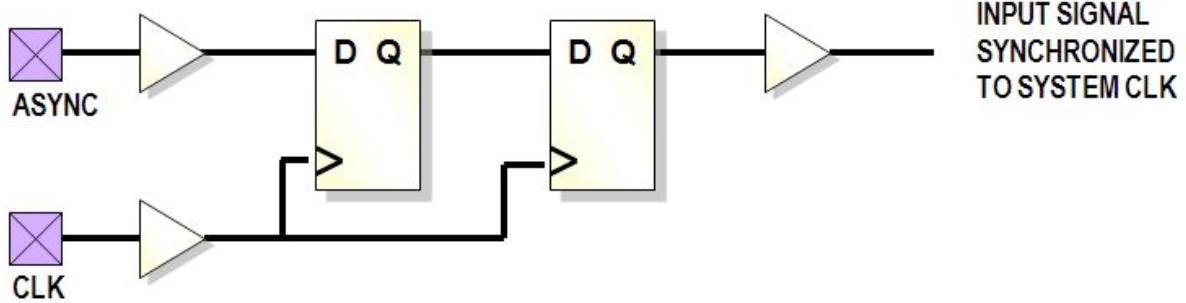
Oftentimes a designer is faced with an input signal that is not synchronized with the system clock, i.e., its rising and falling edges do not maintain a constant delay with respect to the rising edge of the clock signal. Let us imagine for example that we have a card where an FPGA is clocked by an on-board 100 MHz oscillator, and the card is fed with an external input representing the revolution frequency of a synchrotron. Let us also assume that the revolution tick has to be fed to two different state machines inside the FPGA, and that the correct functioning of this design relies on both state machines detecting the revolution tick during the *exact* same period of the system clock, which is used as the clock for the state machines.

A naïve design might split the revolution tick signal in two before feeding it to the state machines. The problem with this solution is that the revolution tick might eventually, after going through several layers of combinatorial logic, find itself at the D inputs of two different flip-flops inside the FPGA. But because the propagation delays of the revolution signal going through the two paths are different, one flip-flop might already clock it in as ‘1’ while the other still sees a ‘0’.

A less naïve designer would then propose to feed the revolution tick to the D input of a flip-flop to begin with, and only then split it in two. Indeed, the rate of failures would go down, but every now and then we would still see an incoherency between the two state machines. The culprit is an effect known as ‘metastability’ which afflicts flip-flops when a transition at their D input occurs too close in time to the rising edge in the clock input. In that case, their Q output hesitates until it finally settles to one of the two possible output values. The resolution time can be arbitrarily long as we push the two edges closer and closer in time. In our second design, from time to time the Q output of the synchronizing flip-flop will go metastable, with a resolution time such that — on the next system clock tick — one of the two subsequent flip-flops will already see a ‘1’ when the other one still sees a ‘0’.

While it seems that this could become a never-ending story, in fact, for all practical purposes, the circuit in Fig. 17 will solve the problem.

Now, for typical system clock and asynchronous input frequencies, the chances that the second flip-flop goes metastable after the first one did the same, one system clock tick earlier, are vanishingly small. One can easily design a circuit that will fail on average once every million years.

**Fig. 17:** Two-flip-flop synchronizer

The discussion above applies equally well to any design with more than one clock domain where data must be transferred from one domain to the other. By clock domain, we mean a part of the design which is clocked by a single common clock. Let us imagine that a designer needs to transfer a 16-bit number from one domain to another. Inserting a synchronizer for each bit would not help, since different flip-flops will see different set-up times of the data with respect to their clock. Indeed, these set-up times vary with time in a random way! The solution is to design the emitter block so that it asserts a data strobe when the data are ready, and holds the data stable for a suitable amount of time. The receiver circuit can then sample the strobe with a two-flip-flop synchronizer and clock the data in once it senses a ‘1’ on the strobe line. A variation on this circuit includes a handshake whereby the receiver sends back an acknowledge line to the emitter, which in turn synchronizes it into its clock domain. Once the emitter senses a ‘1’ on the acknowledge line, it knows it can change the state of the data lines to prepare the next transfer.

Sometimes the above scheme will not work because data comes in bursts at a speed which makes it impossible to perform the full handshake. In those cases, a First-In-First-Out (FIFO) block is needed with each of its sides clocked by a different clock.

6.5 Safe design

There is at least one asynchronous signal in almost all designs: the external reset. In many cases, it is very important to handle this signal properly in order to guarantee coherency of different parts of a design. If we return to the example of the two different state machines within an FPGA, both running this time off the same system clock, and we require that they both ‘wake up’ during the same clock tick after the de-assertion of the reset signal, we find ourselves with a need to treat the reset signal as we treated the revolution tick above, i.e., we need to feed it to a synchronizer before using it in the design.

The best reset strategy, not always possible, is to synchronize the reset to the system clock and then use it as a *synchronous* reset. This means that the reset line is treated as any other synchronous signal. It will enter some combinatorial block and affect its output, which will then be fed to the D input of a flip-flop. Chances are that the output of that combinatorial logic block will go to a predefined ‘reset’ state irrespective of other inputs if the reset line is active, but there is really nothing special about the reset line in this case from a topological point of view.

Things change if, for some reason like saving resources, the designer wants to use the asynchronous reset input present in all flip-flops. There is still a certain guarantee of coherency if the reset fed to these inputs has been properly synchronized in advance, but this will greatly depend on the clock period and the delay to reach each asynchronous reset input. Typical place-and-route tools will not include these paths in their timing analysis because they do not go from Q outputs to D inputs. It is really best if the asynchronous reset can be avoided altogether.

Another important topic in safe design is that of complete state coverage in state machines. If a state machine has five different states, it will need at least three signal lines to represent its current

state. But with three lines one can have eight different states, three of which will be illegal. It is the designer's responsibility to detect these states and take the state machine to a safe state if it goes to one of them. Now, how can a state machine go to an illegal state if no combination of inputs and states is supposed to take it there? The state vector is made of three lines, and each of these lines is — we can assume this without loss of generality — fed from the Q output of a flip-flop. High-energy particles crossing the FPGA can induce what is called a Single-Event Upset (SEU), flipping the state of one of the flip-flops to a new one, which might make the new three-bit combination illegal. With process geometries shrinking, one no longer needs a high-energy accelerator to produce SEUs. Atmospheric particles will produce them at a rate high enough to make it a major concern for safety-critical systems.

Sometimes in high-energy accelerator applications, an FPGA must live in a highly radioactive environment. This is a completely different game. Here are three techniques designers use frequently to make their designs more robust under such adverse conditions:

- Antifuse technology. An antifuse is an element that creates a short circuit when overheated, i.e., exactly the opposite of what a fuse does. FPGAs based on antifuse technology are inalterable at least as far as the configuration memory is concerned. But we know that, among the RAM bits of a typical FPGA, the vast majority of them are configuration bits, so antifuse technology is a major improvement in terms of resistance to radiation. The negative side is the price of antifuse chips and also their lower densities. These devices are roughly one generation behind in terms of silicon processes.
- Scrubbing. As we said, most of the FPGA RAM bits that are susceptible of being affected by a SEU are in fact configuration bits. One can read the configuration stream back repeatedly to check if there has been any corruption, and then take corrective action if needed. The device that scans the configuration bits must itself be more robust than the FPGA being checked. It is typically antifuse-based.
- Triple Mode Redundancy (TMR). This technique consists in replicating the same piece of logic three times and adding a set of voters to detect whether there was a SEU in one of the blocks. Figure 18 shows the principle for the simple example of a counter. If a mismatch is found, the losing counter will be informed and correct its value accordingly.

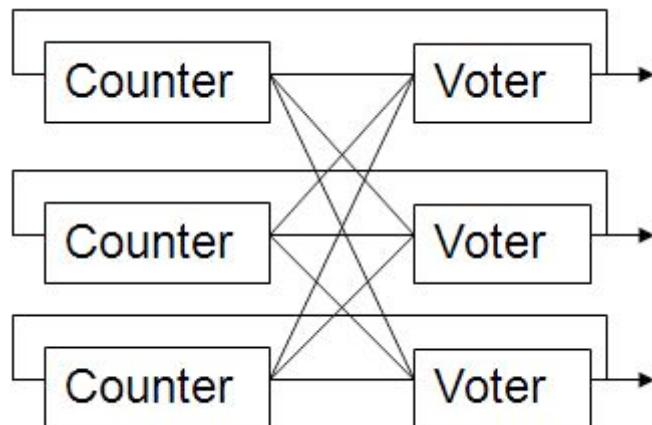


Fig. 18: TMR with state feedback

It is assumed that the probabilities to have a double upset that will affect two counters at the same time are negligible, but this may not be a very realistic assumption with the ever-diminishing process geometries. A single high-energy particle can indeed affect more than one transistor around the impact area. This is why some vendors are developing software that automatically generates TMR logic and places the different parts of the ensemble in different, geographically distant, areas of the

chip. Notice also that our FPGA now contains three counter outputs instead of one. With only one output, a SEU in one of the output transistors could defeat the whole TMR scheme. If the counter value is to be used outside the FPGA, decision logic must be implemented outside using a radiation-hard scheme in order to work out the current value of the counter.

For unmatched reliability, radiation-hard antifuse products are available. Every flip-flop in these devices is TMRed in silicon, with feedback TMR built in. Needless to say, these devices are extremely expensive and reserved for the most demanding applications in terms of radiation hardness. The extra logic needed for the TMR scheme and the state feedback are also a problem if excellent timing performance is required.

References

- [1] J.F. Wakerly, *Digital Design: Principles and Practices*, 4th ed. (Prentice Hall, Upper Saddle River, NJ, 2006).
- [2] A. Rushton, *VHDL for Logic Synthesis*, 2nd ed. (John Wiley & Sons, Chichester, 1998).
- [3] U. Meyer-Baese, *Digital Signal Processing with Field Programmable Gate Arrays*, 3rd ed. (Springer, Berlin, 2007).
- [4] J.G. Proakis and D.K. Manolakis, *Digital Signal Processing*, 4th ed. (Prentice Hall, Upper Saddle River, NJ, 2006).
- [5] R. Andraka, A survey of CORDIC algorithms for FPGAs, Proc. 1998 ACM/SIGDA 6th International Symposium on Field Programmable Gate Arrays, Feb. 22–24, Monterrey, CA, USA, pp. 191–200. URL: <http://www.andraka.com/files/crdesrvy.pdf>.
- [6] M. Alexander, Power Supply Distribution (PDS) Design: Using bypass/decoupling capacitors, Xilinx application note 623. URL: http://www.xilinx.com/support/documentation/application_notes/xapp623.pdf.

RF applications in digital signal processing

T. Schilcher

Paul Scherrer Institut, Villigen, Switzerland

Abstract

Ever higher demands for stability, accuracy, reproducibility, and monitoring capability are being placed on Low-Level Radio Frequency (LLRF) systems of particle accelerators. Meanwhile, continuing rapid advances in digital signal processing technology are being exploited to meet these demands, thus leading to development of digital LLRF systems. The first part of this course will begin by focusing on some of the important building-blocks of RF signal processing including mixer theory and down-conversion, I/Q (amplitude and phase) detection, digital down-conversion (DDC) and decimation, concluding with a survey of I/Q modulators. The second part of the course will introduce basic concepts of feedback systems, including examples of digital cavity field and phase control, followed by radial loop architectures. Adaptive feed-forward systems used for the suppression of repetitive beam disturbances will be examined. Finally, applications and principles of system identification approaches will be summarized.

1 Introduction

In this lecture we shall examine various aspects of Low-Level Radio Frequency (LLRF) systems for particle accelerators. These LLRF systems typically collect measurements of amplitude, phase, and frequency, perform various signal processing and computational processes on that data, and then use the outcome to monitor, control, and regulate RF fields in particle accelerators. The LLRF applications can be classified into monitoring, feedback, and feed-forward systems. The range of typical RF frequencies in particle accelerators varies widely starting from a few MHz (e.g., in ion accelerators) and going up to tens of GHz (e.g., high-gradient acceleration structures operating at 30 GHz). The variety of accelerators demands LLRF systems with different requirements. The boundary conditions for the LLRF systems depend on their field of application, e.g., linear or circular machines, normal or super-conducting RF systems, pulsed or CW operation, or even electron or hadron accelerators. A short incomplete list of typical LLRF applications is given in Table 1. The rapid advances in digital technology over the last decade have prepared the ground for applications of digital systems in the field of LLRF. The question arises why implementations of digital systems have found their way into LLRF applications to such a large extent. There are some disadvantages with respect to their analog counterparts but on the other hand, they offer many advantages. A comparison of digital and analog LLRF systems is given in Table 2 [1]. In feedback systems, the biggest advantage of analog systems is their short loop latency. Whenever high loop bandwidths are required, analog systems still continue to be the only realistic alternative.

Table 1: Typical low-level RF applications

Application	Task
Cavity field loops	Control amplitude and phase of accelerating RF fields in cavities
Tuner loops	Control resonant frequencies of RF cavities
Radial and phase loops	Control radial beam positions and phases in circular machines
Klystron loops	Control amplitude and phase of klystrons
RF gymnastics	Control bunch splitting and merging in circular machines

Table 2: Comparison of digital versus analog RF applications

	Digital	Analog
implementation	learning curve + software effort	easier / known
latency	longer	short
data acquisition/control	(<i>I/Q</i>) sampling (also direct) or Digital Down Conversion (DDC)	amplitude/phase, <i>IF</i> down conversion
algorithms	sophisticated state machines, exception handling	simple linear, time-invariant (example: PID control)
multi-user	full	limited
remote control + diagnostics	easy, often no additional hardware necessary	difficult, often extra HW necessary
flexibility/reconfigurability	high, easy upgrades	limited
drift/tolerance	no drifts, repeatability	drift (temperature, etc.), component tolerance
signal transport distance without distortions	longer	short
radiation sensitivity	high	small

However, the increasing processing speed of digital systems now often allows them to cover the loop bandwidth demands and therefore supersede analog systems.

To characterize digital LLRF systems, we can break them down into four main building blocks which are shown in Fig. 1. RF signals from the accelerator, e.g., from probe antennas or directional couplers, are down-converted (if necessary) and conditioned in the first block, which typically includes filtering, amplification and/or attenuation. The second block comprises the digitization process, while

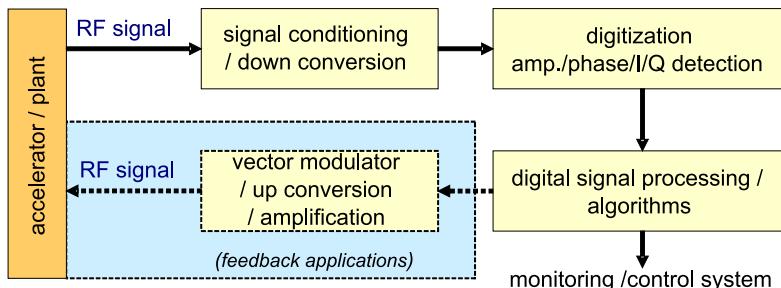


Fig. 1: Main blocks of RF applications. Feedback and feed-forward systems close the loop (blue box) while monitoring applications provide the processed RF field information to other sub-systems or to the control system.

the third block deals with the digital signal processing of the sampled RF fields. Depending on the hardware and algorithms, the extracted information is supplied to the control system for monitoring purposes or to any other sub-system requiring this information. In case of feedback applications, which close the signal path, the correction signals resulting from the control algorithms usually need to be converted back to analog RF signals of precise amplitudes, phases, and frequencies. This can be achieved by digital-to-analog converters and various up-conversion schemes—very often so-called vector modulators are used. Basic principles of these techniques will be described in the section covering the fourth building block. These four basic building blocks which form LLRF systems also form the basis of many other accelerator applications, e.g., diagnostic applications like beam-position monitoring measurements, or

bit feedbacks, and bunch-by-bunch feedback systems. Sharing of digital hardware platforms provides grounds for common development. As in every feedback system, the ultimate remaining error is dominated by the measurement process which includes systematic errors, accuracy, linearity, repeatability, stability, resolution, and noise. In the following sections, we shall first discuss these four building blocks before proceeding on to an introduction to feedback systems, adaptive feed-forward algorithms, and system identification.

2 Signal conditioning and down conversion

Digital systems require the transformation of analog signals into the digital domain. This is achieved by analog-to-digital converters (ADC, A/D converters). In most cases, analog preprocessing has to be applied prior to the A/D conversion because the sampling frequencies, analog input bandwidths, and the bit resolution of ADCs are limited. This preprocessing typically consists of amplitude scaling (amplification, attenuation) along with filtering and frequency translation to an intermediate frequency or to baseband. The digitization of high-frequency carrier signals is very often not possible or reasonable. Although today's ADCs already reach the giga-samples per second domain with even higher analog input bandwidths, the requirements for ADC clock and aperture jitter become more and more stringent. This is even more critical with under-sampling schemes in which signals are sampled outside the first Nyquist zone, which means any clock and/or aperture jitter has a large impact on the resulting error. The system designer always has to evaluate the trade-off between sampling speed and dynamic range of an ADC. Each additional effective bit of an ADC provides 6 dB more in terms of dynamic range. Very often it is better to use analog circuits in conjunction with ADCs to implement automated gain control (AGC) functions to ensure that the signal to be sampled falls within the ideal dynamic range of the chosen ADC. Taking into account these technical limits, frequency translations to lower intermediate frequencies are essential in many RF applications. This is usually achieved by RF mixers, whose basic functionality will presently be introduced. Since frequency conversions are non-linear operations, RF mixers cannot be realized by linear-time-invariant (LTI) components or circuits. Instead, the translation is achieved by either time varying or non-linear circuits, e.g., diodes. An ideal mixer is usually represented by a multiplier symbol (see Fig. 2) with two input ports and one output port. The signal at the output port is the vector

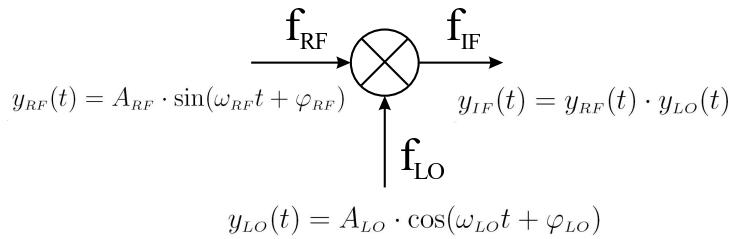


Fig. 2: Ideal RF mixer with input signals $y_{RF}(t)$ and $y_{LO}(t)$

multiplication of the signals at the two input ports. One of the input signals is the reference signal, the local oscillator (LO). Using some trigonometric product-to-sum identities and assuming two sinusoidal input signals with amplitudes A_{RF} , A_{LO} and frequencies f_{RF} , f_{LO} , the multiplied output signal y_{IF} can be represented as

$$\begin{aligned}
 y_{IF}(t) &= y_{RF}(t) \cdot y_{LO}(t) \\
 &= \frac{1}{2} A_{LO} A_{RF} \cdot \left(\underbrace{\sin[(\omega_{RF} - \omega_{LO})t + (\varphi_{RF} - \varphi_{LO})]}_{\text{lower sideband}} \right. \\
 &\quad \left. + \underbrace{\sin[(\omega_{RF} + \omega_{LO})t + (\varphi_{RF} + \varphi_{LO})]}_{\text{upper sideband}} \right) . \tag{1}
 \end{aligned}$$

The input signal is translated to two output frequencies, the upper ($f_{RF} + f_{LO}$) and the lower ($f_{RF} - f_{LO}$) sideband signals. Mixers are used for down- and up-conversion. In the case of down-conversion, the RF and LO signals are high-frequency inputs while the resulting output signal is the intermediate frequency signal [Fig. 3(a)]. Likewise, an intermediate frequency can be used as input to perform an up-conversion [Fig. 3(b)]. Usually, either the upper or the lower sideband of the mixing process is selected by filtering

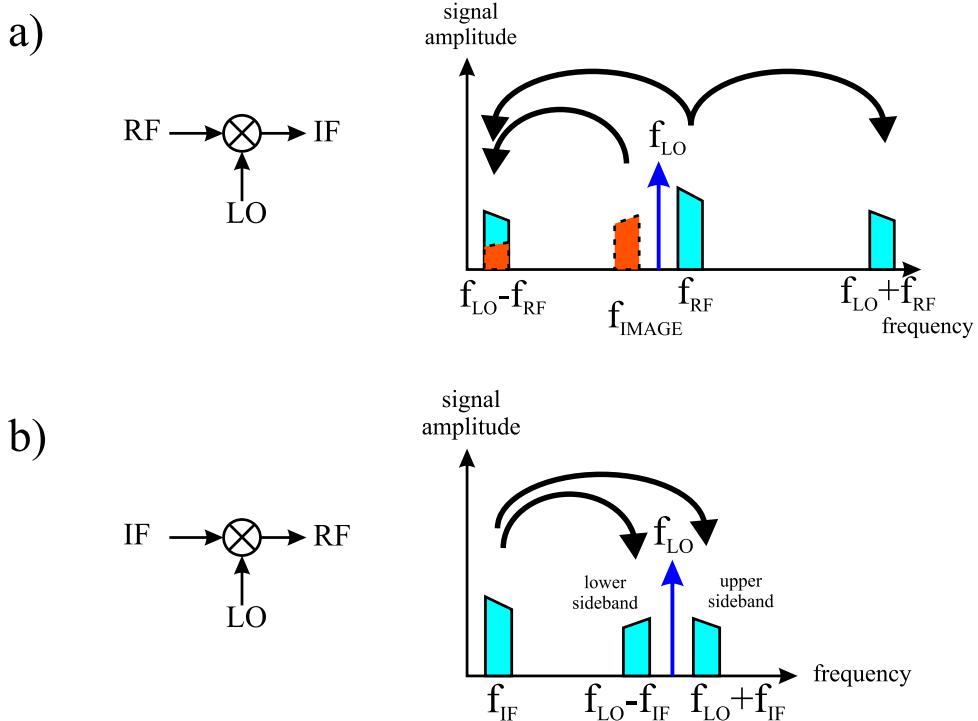


Fig. 3: (a) down conversion and (b) up conversion schemes with RF mixers

the output signal. Note that for a given LO frequency, signals at frequencies $LO \pm IF$ are converted to the same IF frequency. Therefore care has to be taken by applying proper filtering in order to avoid the conversion of any noise or signal at the image frequency to the IF frequency [Fig. 3(a)], otherwise it can degrade system performance, often severely. For receiver applications, to which particular attention will now be given, the lower sideband is extracted by low-pass filtering. With Eq. (1) this yields for the IF signal

$$y_{IF}(t) = A_{IF} \cdot \sin \left(\omega_{IF} t + \varphi_{IF} \right)$$

with the frequency, amplitude and phase definition

$$\begin{aligned} \omega_{IF} &= \omega_{RF} - \omega_{LO}, \\ A_{IF} &= \frac{1}{2} A_{LO} A_{RF} \sim A_{RF} \quad \text{with constant } A_{LO}, \\ \varphi_{IF} &= \varphi_{RF} - \varphi_{LO} \sim \varphi_{RF} \quad \text{with constant } \varphi_{LO}. \end{aligned}$$

With a constant LO signal, i.e., with a constant A_{LO} and φ_{LO} , the amplitude and phase of the down-converted signal are directly proportional to those of the input signal, in other words, all of the basic properties of an RF signal are conserved in the frequency conversion process. In this context, two important facts should be highlighted: first of all, a phase change or phase jitter at f_{RF} is exactly the same phase change at the IF frequency f_{IF} . As an example, a 1° phase change at 3 GHz corresponds also to a 1° phase change at any arbitrary IF frequency. Secondly, timing jitter of a given magnitude at a higher frequency leads to a bigger phase variation than at a lower frequency. This has to be taken into account

when deciding whether an RF signal should be directly sampled or down-converted first. ADC clock jitters are therefore much more critical in direct sampling applications and tougher requirements have to be applied. As an illustration, a 10 ps clock jitter at 500 MHz corresponds to 1.8° phase variation while the same clock jitter results in only 0.18° phase variation at 50 MHz.

So far, ideal mixers have been considered. As pointed out earlier, real mixers are non-linear devices. In addition to the desired ideal mixing product, their output spectra contain many undesired spurious signals which are located at $mf_{RF} \pm nf_{LO}$. Figure 4 depicts the output spectrum of a down-conversion mixer. The image frequency is denoted by IM . Usually, the signal at the mixer output is

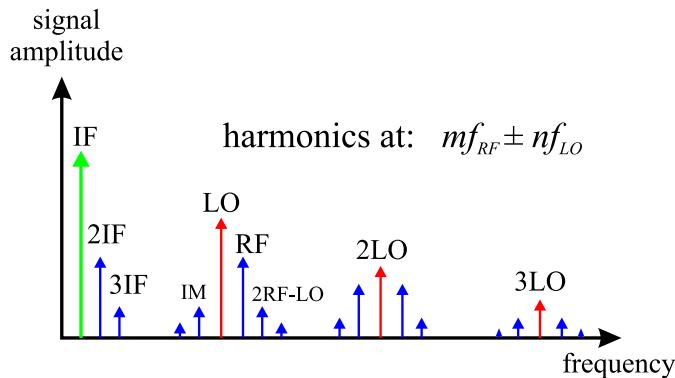


Fig. 4: Output spectrum of a real mixer

filtered by a low-pass filter in order to suppress the undesired spurs sufficiently. Introducing low-pass filters with a steep roll-off, however, will mean a trade-off between good suppression and increased group delay which is very important for low latency and feedback applications. Other common ways to reduce spurious outputs are the use of double-balanced mixers and image rejection mixers to minimize image mixing.

3 Detection of amplitude and phase in digital systems

Several techniques can be applied to detect amplitude and phase of RF signals in digital systems, an overview of which is shown in Fig. 5. Common amplitude detectors like Schottky diodes and phase detectors [2] provide amplitude and phase information as a continuous analog signal which can be digitized by ADCs for further processing [Fig. 5(a)]. In contrast to this, the polar representation of an RF signal can also be decomposed into its Cartesian representation (I/Q , see Section 3.1) by analog IQ demodulators. The individual I/Q components can then be digitized [Fig. 5(b)]. A third alternative is the so-called digital IQ sampling and Digital Down Conversion (DDC) where an RF signal is first down-converted to an intermediate frequency and then directly digitized. Depending on the frequency, the down-conversion process can be omitted. The I/Q information is extracted from these samples in a purely digital way. These techniques will be further discussed in Section 3.1. Since the focus of this lecture is on digital signal processing, the first two detection schemes will not be dealt with any further.

3.1 IQ sampling

The terminology ' IQ ' originates from the representation of an RF signal which can be in either polar (amplitude/phase representation) or in Cartesian coordinates. Any sinusoidal RF signal

$$y(t) = A \cdot \sin(\omega t + \varphi_0) \quad (2)$$

can be modelled as a phasor which is a rotating vector with amplitude A , frequency ω , and an initial phase φ_0 . A positive frequency corresponds to an anticlockwise rotating phasor, which can be decomposed into

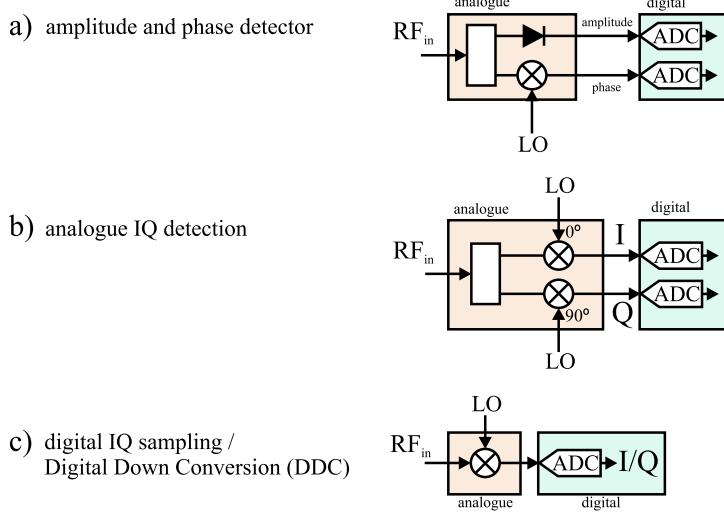


Fig. 5: different schemes of amplitude and phase detection in digital systems

its sin and cos components using basic trigonometric functions.

$$\begin{aligned} y(t) &= \underbrace{A \cos \varphi_0}_{=:I} \sin \omega t + \underbrace{A \sin \varphi_0}_{=:Q} \cos \omega t \\ y(t) &= I \cdot \sin \omega t + Q \cdot \cos \omega t. \end{aligned} \quad (3)$$

The amplitude of the sine component can be defined as the *in-phase* component (I), while the amplitude of the cosine component is called the *quadrature-phase* component (Q). This definition is somewhat arbitrary and can therefore be found inverted in the literature. The Cartesian representation of an RF signal (I/Q) is widely used in numerical applications. The well-known relationship between Cartesian and polar coordinates [Eq. (4)] makes switching between both representations possible—if necessary, and if computing resources allow the time-consuming calculation of square roots and trigonometric functions. Sampling of a sinusoidal RF signal with an ADC delivers only one component of the rotating phasor

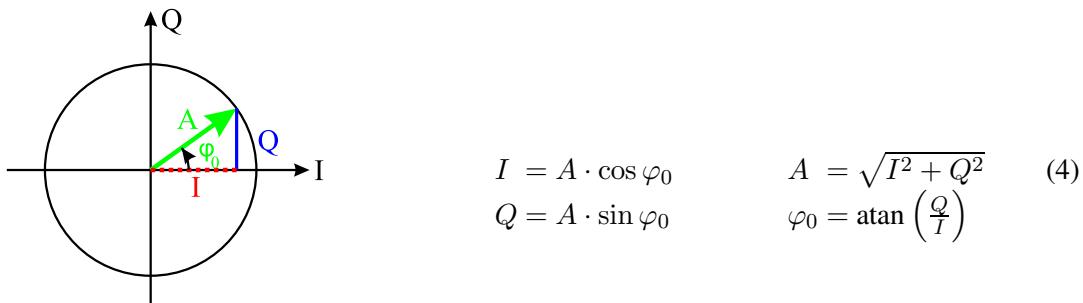


Fig. 6: Phasor representation of RF signal

at a time. Nevertheless it is both easy and possible to extract I/Q information based on the sampled data stream, as will now be shown. We assume that the vertical component is measured by the ADC (this assumption is also arbitrary but it has no consequence on the following results). The goal of sampling the RF or IF signal is to extract its amplitude/phase or I/Q information, respectively. If I/Q of the rotating phasor is known initially and measured again later at a well-defined time, an algorithm can rotate the phasor back to an initial reference phasor. A comparison of these two measurements will then reveal whether the incoming RF signal has changed its amplitude and phase. The usual IQ sampling is

achieved if the sampling frequency f_s and the intermediate frequency f_{IF} (or in general, the incoming RF signal) are related by

$$f_s = 4 \cdot f_{IF}. \quad (5)$$

In this case, the phase advance between two samples amounts to 90° . With Eq. (3) the digitization at four consecutive sampling times results in

$$\begin{aligned} \omega t_0 &= 0 : & y(t_0) &= Q \\ \omega t_1 &= \pi/2 : & y(t_1) &= I \\ \omega t_2 &= \pi : & y(t_2) &= -Q \\ \omega t_3 &= 3\pi/2 : & y(t_3) &= -I. \end{aligned} \quad (6)$$

This is illustrated in Fig. 7. At each sampling time, the I and Q components of the phasor can be determined by two consecutive samples, $y(t_i)$ and $y(t_{i+1})$. These vectors have to be rotated backwards

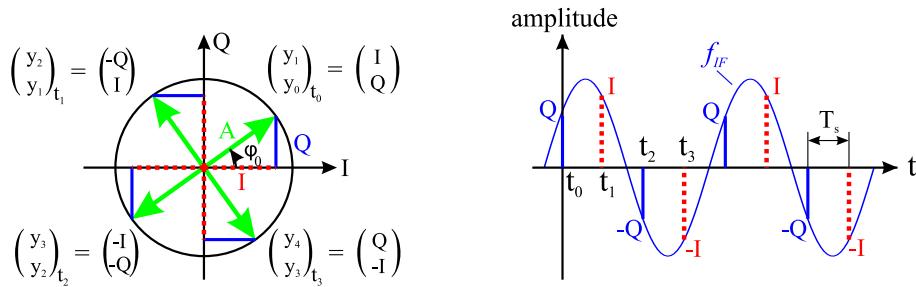


Fig. 7: IQ sampling with 90° phase advance between consecutive samples

by $\Delta\varphi_i = 0^\circ, -90^\circ, -180^\circ$ and -270° in order to compare them with the initial vector. The underlying rotation algorithm is

$$\begin{pmatrix} I \\ Q \end{pmatrix}_{t_i} = \begin{pmatrix} \cos \Delta\varphi_i & -\sin \Delta\varphi_i \\ \sin \Delta\varphi_i & \cos \Delta\varphi_i \end{pmatrix} \cdot \begin{pmatrix} y_{i+1} \\ y_i \end{pmatrix}. \quad (7)$$

The rotation algorithm is carried out at the sample rate f_s . It is based on the assumption that the incoming RF signal does not change its amplitude and phase substantially between two samples since the components of the (I/Q) phasor are based on two successive samples. If we were to choose to measure the horizontal component of the phasor with an ADC, the result would be exactly the same, which can easily be seen in Fig. 7. Instead of the sequence $Q, I, -Q, -I$ we would measure the sequence $I, -Q, -I, Q$ which only corresponds to a phase shift of 90° .

The restriction to choose a sampling frequency four times the carrier frequency simplifies the calculation since the elements of the rotation matrix only consist of 1, 0 and -1 . Despite this fact, it is possible in principle to choose the sampling frequency such that it is a multiple of the IF frequency.

$$f_s/f_{IF} = m, \quad m : \text{integer}.$$

In this case, the phase difference between two samples amounts to

$$\Delta\varphi = \frac{2\pi}{m}. \quad (8)$$

As long as the input signal does not change substantially, each IF period is sampled at the same location on the circle in the IQ plane. With two consecutive samples ($y_n = Q_n, y_{n+1} = Q_{n+1}$) and the rotation of the I/Q vector at time t_{n+1} to the vector at time t_n by the angle $-\Delta\varphi$, a set of linear equations is obtained.

$$\begin{pmatrix} I_n \\ Q_n \end{pmatrix} = \begin{pmatrix} \cos \Delta\varphi & \sin \Delta\varphi \\ -\sin \Delta\varphi & \cos \Delta\varphi \end{pmatrix} \cdot \begin{pmatrix} I_{n+1} \\ Q_{n+1} \end{pmatrix}.$$

Solving this set of equations to eliminate I_{n+1}, Q_{n+1} yields

$$\begin{pmatrix} I_n \\ Q_n \end{pmatrix} = \frac{1}{\sin \Delta\varphi} \cdot \begin{pmatrix} 1 & -\cos \Delta\varphi \\ 0 & \sin \Delta\varphi \end{pmatrix} \cdot \begin{pmatrix} y_{n+1} \\ y_n \end{pmatrix}.$$

Similar to the case where the IF frequency is sampled four times per period, the phasor (I_n, Q_n) has to be rotated backwards to the initial phasor (I_0, Q_0) by a rotation angle $-n\Delta\varphi$ where the phase advance $\Delta\varphi$ is defined in Eq. (8).

$$\begin{pmatrix} I_0 \\ Q_0 \end{pmatrix} = \frac{1}{\sin \Delta\varphi} \cdot \begin{pmatrix} \cos n\Delta\varphi & -\cos(n+1)\Delta\varphi \\ -\sin n\Delta\varphi & \sin(n+1)\Delta\varphi \end{pmatrix} \cdot \begin{pmatrix} y_{n+1} \\ y_n \end{pmatrix}.$$

In the most general case, the initial phasor can also be rotated by a user defined angle $-\varphi$ in order to match initial conditions. Including this last rotation the (I, Q) vector can be obtained by

$$\begin{pmatrix} I \\ Q \end{pmatrix} = \frac{1}{\sin \Delta\varphi} \cdot \begin{pmatrix} \cos(\varphi + n\Delta\varphi) & -\cos(\varphi + (n+1)\Delta\varphi) \\ -\sin(\varphi + n\Delta\varphi) & \sin(\varphi + (n+1)\Delta\varphi) \end{pmatrix} \cdot \begin{pmatrix} y_{n+1} \\ y_n \end{pmatrix}. \quad (9)$$

The calculation of I and Q belongs to the class of FIR filters of first order since the values are based on two consecutive samples. This sequence of processing is illustrated in Fig. 8. Although the phase

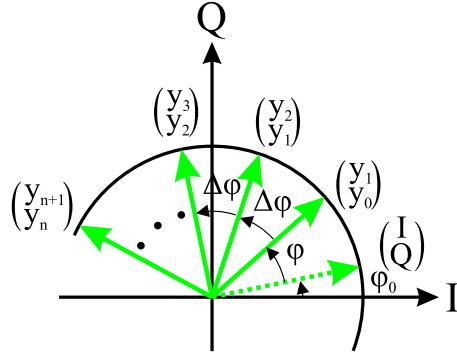


Fig. 8: IQ sampling with a phase advance of $\Delta\varphi = 2\pi/m$ between consecutive samples

advance can be chosen as illustrated in Eq. (8), the calculation of I and Q becomes extremely sensitive to noise and sampling errors if the phase advance is far away from 90° or 270° [see Eq. (9)].

There are a number of potential problems associated with this typical mode of IQ sampling ($\Delta\varphi = 90^\circ$). DC offsets at the input of the ADC or phase advances between samples (which differ from the expected 90°) systematically result in a ripple with frequency f_{IF} . These errors are easily detectable since their signatures reveal their origin. Other error sources are much more difficult to identify at first glance. Non-linearities of mixers and differential non-linearities of ADCs generate higher harmonics of the input signal frequency f_{IF} . Sampling this signal four times per period maps the second harmonic on the Nyquist frequency, while the third harmonic aliases directly on the IF frequency again. In general, all odd harmonics alias to the carrier frequency while even harmonics alias to DC or to the Nyquist frequency, respectively (see Fig. 9). Any harmonics which map onto the IF frequency are indistinguishable from the carrier by standard IQ sampling. As the carrier frequency and phase changes, the distortion changes, resulting in false measurement data.

3.2 Non-IQ sampling

A solution to the problems described is to change the sampling frequency in such a way that the following relation holds:

$$f_s = \frac{N}{M} \cdot f_{IF} \iff N \cdot T_s = M \cdot T_{IF} \quad \text{with } M, N: \text{integer}.$$

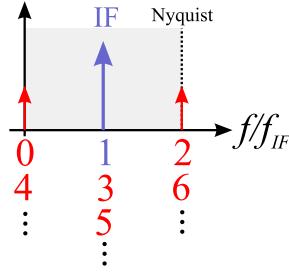


Fig. 9: Spectrum of IQ sampling with a phase advance of 90° between consecutive samples

The above is equivalent to taking N samples in M successive IF periods. Depending on the values chosen for M, N the circle in the IQ plane is now sampled at different locations where the samples only repeat after M IF periods. The phase advance between two consecutive ADC readings is

$$\Delta\varphi = \omega_{IF}T_s = 2\pi \frac{T_s}{T_{IF}} = 2\pi \frac{M}{N}.$$

Expressing the N successive samples with Eq. (3), the result is a set of linear equations [3], [4].

$$\begin{aligned} y_0 &= I \cdot \sin \varphi_0 + Q \cdot \cos \varphi_0 \\ y_1 &= I \cdot \sin \varphi_1 + Q \cdot \cos \varphi_1 \\ y_2 &= I \cdot \sin \varphi_2 + Q \cdot \cos \varphi_2 \quad \text{with } \varphi_i = i \cdot \Delta\varphi = i \cdot 2\pi \frac{M}{N}, \quad i: \text{integer} . \\ \dots \\ y_{(N-1)} &= I \cdot \sin \varphi_{(N-1)} + Q \cdot \cos \varphi_{(N-1)} \end{aligned} \quad (10)$$

In the ideal case of no measurement errors, any two equations of this set would give a solution common to both for I, Q . Under real conditions with noise, quantization errors, and clock jitter, the over-constrained system of equations can be solved by applying the Least Mean Square algorithm (LMS). The basis for this algorithm is to minimize the function

$$f(I, Q) = \sum_{i=0}^{N-1} (I \cdot \sin \varphi_i + Q \cdot \cos \varphi_i - y_i)^2. \quad (11)$$

The conditions for a minimum are vanishing derivatives with respect to I and Q .

$$\frac{\partial f}{\partial I} = 0, \quad \frac{\partial f}{\partial Q} = 0.$$

This yields

$$\begin{aligned} \frac{\partial f}{\partial I} &= 2I \sum_{i=0}^{N-1} \sin^2 \varphi_i + 2Q \sum_{i=0}^{N-1} \sin \varphi_i \cos \varphi_i - 2 \sum_{i=0}^{N-1} y_i \sin \varphi_i = 0 \\ \frac{\partial f}{\partial Q} &= 2Q \sum_{i=0}^{N-1} \cos^2 \varphi_i + 2I \sum_{i=0}^{N-1} \sin \varphi_i \cos \varphi_i - 2 \sum_{i=0}^{N-1} y_i \cos \varphi_i = 0. \end{aligned} \quad (12)$$

With the definitions

$$p_{11} = \sum_{i=0}^{N-1} \sin^2 \varphi_i, \quad p_{12} = p_{21} = \sum_{i=0}^{N-1} \sin \varphi_i \cos \varphi_i, \quad p_{22} = \sum_{i=0}^{N-1} \cos^2 \varphi_i$$

$$s_1 = \sum_{i=0}^{N-1} y_i \sin \varphi_i, \quad s_2 = \sum_{i=0}^{N-1} y_i \cos \varphi_i,$$

Eq. (12) can be written as

$$\begin{aligned} p_{11} \cdot I + p_{12} \cdot Q &= s_1 \\ p_{21} \cdot I + p_{22} \cdot Q &= s_2. \end{aligned}$$

The coefficients p_{12} and p_{21} can be simplified using the definition for φ_i of Eq. (10).

$$p_{12} = p_{21} = \frac{1}{2} \sum_{i=0}^{N-1} \sin 2\varphi_i = \frac{1}{2} \underbrace{\sum_{i=0}^{N-1} \sin 4\pi M \frac{i}{N}}_{=0} = 0,$$

Likewise, the coefficients p_{11} and p_{22} can be calculated to

$$p_{11} = \frac{1}{2} \sum_{i=0}^{N-1} (1 - \cos 2\varphi_i) = \frac{N}{2}, \quad p_{22} = \frac{1}{2} \sum_{i=0}^{N-1} (1 + \cos 2\varphi_i) = \frac{N}{2}.$$

Therefore the I and Q values can be simply calculated by

$$I = \frac{2}{N} \cdot \sum_{i=0}^{N-1} y_i \cdot \sin(i \cdot \Delta\varphi) \quad (13)$$

$$Q = \frac{2}{N} \cdot \sum_{i=0}^{N-1} y_i \cdot \cos(i \cdot \Delta\varphi). \quad (14)$$

It should be noted that the sine and cosine terms can be precalculated and stored in lookup tables since the phase advance is constant and defined in Eq. (10). Owing to the non- IQ sampling, most of the harmonics no longer line up with the carrier frequency, which can be seen in Fig. 10. Digital filtering can now be

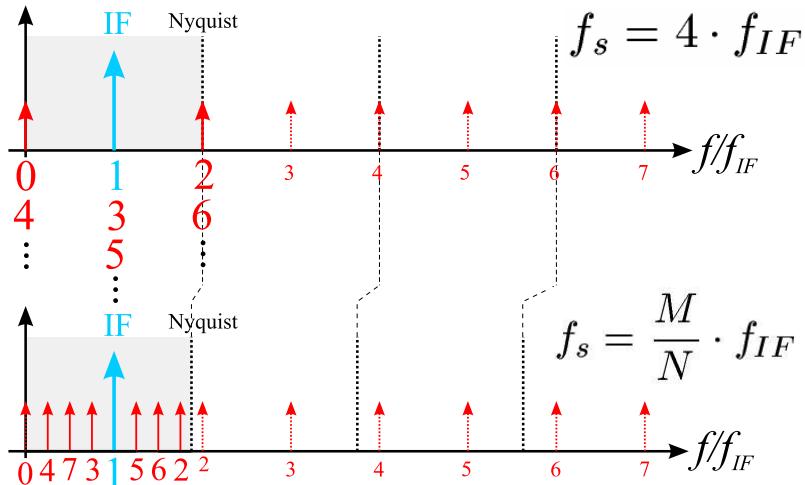


Fig. 10: Spectrum of non- IQ sampling for the case $M = 15, N = 4$

applied to separate the carrier from the harmonics. Because of the sampling of the carrier frequency at several locations on the circle in the IQ plane, errors from ADC non-linearities, quantization, DC offsets, and clock jitter are reduced. In contrast to IQ sampling, the calculation of I, Q by Eq. (13) and Eq. (14) results in more latency. Special care has to be taken if a digital filter is introduced to distinguish the carrier from the harmonics because of the resulting additional group delay. A trade-off between improved linearity and low latency has to be made.

3.3 Digital Down Conversion—DDC

Another possible way of extracting amplitude and phase information from a sampled carrier signal is by means of Digital Down Conversion (DDC), sometimes also referred to as Digital Drop Receiver (DDR). In many RF applications, the band of interest of an RF signal lies around a carrier or IF frequency. The frequency band of interest is very often narrow compared to the carrier frequency itself. According to the Nyquist theorem, the sampling rate of a band-limited signal must be at least twice the highest analog frequency component in order to be able to reconstruct the signal fully. This leads to high data rates and imposes high demands on subsequent digital processing stages. However, Nyquist–Shannon’s theorem states that the sampling frequency only has to be at least twice the maximum information bandwidth. Digital down conversion is a technique which takes a band-limited high-sample-rate digitized signal, shifts the band of interest to a lower frequency and reduces the sample rate while retaining all the information. The basic function of DDC is shown in Fig. 11. To illustrate the DDC principle, we assume

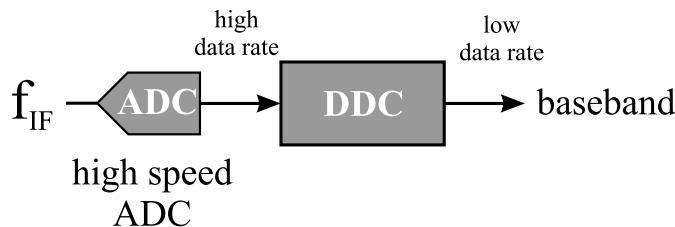


Fig. 11: Basic principle of Digital Down Conversion (DDC)

a 40 MHz carrier (IF signal). A band-limited signal with a bandwidth of 1 MHz is located around the carrier frequency. If oversampling is applied, sampling has to be carried out with a frequency of at least 82 MHz. However, an output sample rate of just 2 MHz would be sufficient to meet the requirement of Shannon’s theorem if the band of interest is first shifted to baseband.

Although DDC can be implemented on Digital Signal Processors (DSPs), the main application area is on Application Specific Integrated Circuits (ASICs) or Field Programmable Gate Arrays (FPGAs). DDC can be divided into two classes, wideband and narrowband, based on their decimation ratio. The sampling rate of wideband signals is typically reduced by a smaller amount (decimation ratio < 32). This can be achieved by FIR or so-called multi-rate FIR filters. Narrowband DDC provides large decimation ratios of more than 32, consequently their implementation on FPGA with FIR filters would take too many gates. A better approach is the use of Cascaded Integrator Comb (CIC) filters (see later in this lecture). A digital down converter is composed of three major building blocks (see Fig. 12) which will be addressed in the following sub-sections. The digital mixers convert the incoming digitized RF signal

DDC building blocks:

- mixer (digital mixer)
- local oscillator
- (numerically controlled oscillator, NCO)
- decimating low-pass filter (LPF)

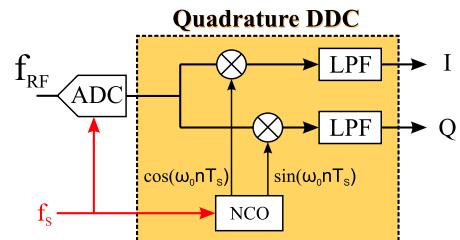


Fig. 12: Main building blocks of Digital Down Conversion

down to baseband and are realized as ideal multipliers, just like their ideal analog counterpart. The local oscillator inputs are supplied by numerically controlled oscillators (NCOs). As shown in Section 2, the resulting output consists of the sum and the difference frequency with respect to the input frequency. The purpose of the subsequent low-pass filter is to suppress the sum frequency component and to provide

anti-aliasing filtering. This is necessary in order to limit the signal spectrum prior to decimation. The decimating anti-aliasing filter is often implemented as CIC filter and/or FIR filter.

3.3.1 Numerically Controlled Oscillator—NCO

The numerically controlled oscillator in a DDC is implemented as a quadrature digital oscillator which generates a stream of sine and cosine samples. If the frequency of the NCO is chosen to match the carrier frequency, the difference signals at the outputs of the digital mixers represent the I and Q components of the input signal. One of the advantages of the NCO is that the two sine and cosine output signals are perfectly synchronized, since they are derived from a common clock. Moreover, the signals have a very precise 90° phase shift. There are several ways to implement a NCO. Its basic functionality is that of a phase accumulator and a phase-to-amplitude conversion block (see Fig. 13). A programmable

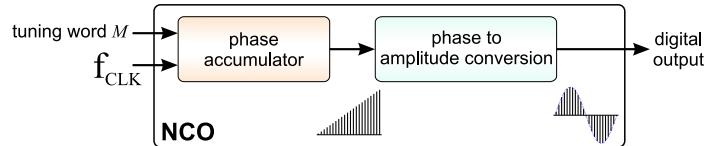


Fig. 13: Basic principle of a Numerically Controlled Oscillator (NCO)

phase increment is added within the phase accumulator at each clock cycle. This phase increment is determined by the tuning word M . The resulting total phase is then converted to the corresponding amplitude value taken from a memory-based lookup table in which a sine wave is stored. A phase roll-over is achieved easily if the table size is chosen to be a binary multiple. In this case, the roll-over takes place automatically. Numerically controlled oscillators have several advantages compared to their analog counterparts. First of all, the tuning word is programmable, which means that almost every frequency up to nearly half the clock frequency (Nyquist frequency) can be chosen by means of the software. NCOs have extremely fast hopping speeds in tuning the output frequency, which means they are extremely frequency agile. The frequency change occurs immediately after loading a new phase increment into the tuning word M . In addition, this technique provides a phase-continuous frequency hop without over- or undershoot and without the loop settling time anomalies encountered in analog circuit variable oscillators. The sine lookup table can contain a full sine wave or just one quarter of one; this is sufficient for reconstruction of the sine wave and reduces the amount of memory space required (however, the additional calculations necessary to end up with the correct amplitude have to be traded off against the saving of the memory space). The total size of the memory is dependent on the desired resolution (output amplitude bit-width) of the entries and on the number of entries stored in the table. The achievable output frequency f_{out} is a function of the clock frequency f_{CLK} , the tuning word M , the bit width of the phase accumulator N , and is given by

$$f_{out} = M \cdot \frac{f_{CLK}}{2^N}.$$

NCOs can reach very high frequency resolutions if the word length of the phase accumulator is chosen properly. As an example, if a clock frequency of 50 MHz is assumed and the phase accumulator width is set to 32-bits, the resulting frequency tuning resolution will be $\Delta f = 12$ mHz. On the other hand, the question that needs to be addressed is whether the same number of entries in the lookup table is needed as phase outputs are possible. Sticking to our example of 32-bit phase accumulation, and assuming an amplitude bit width of 8-bits, we would require a 4 GByte lookup table. One way out of such an impractical implementation is the method of *phase truncation*. Before the resulting phase of the accumulator is looked up in the table, it is truncated to a defined number of upper bits while the phase in the accumulator is preserved. This reduces the need for large memories tremendously. However, one consequence of this approach is the introduction of errors in the phase-to-amplitude conversion process. Regardless of the

chosen tuning word, these truncation errors are periodic and appear as line spectra in the frequency domain (although certain tuning words result in no phase truncation errors at all, which should be obvious). These lines in the spectra are known as phase truncation spurs which reduce the Spurious Free Dynamic Range (SFDR) of the output spectrum.

Although NCOs for digital down conversion do not require more elements than are shown in the block diagram shown in Fig. 13, we shall briefly extend this chapter to *Direct Digital Synthesis* for the sake of completeness. Direct Digital Synthesis or DDS is based on the same building blocks as used for NCO but is extended by a DAC to generate freely programmable, high-frequency, analog waveforms (see Fig. 14). The staircase-like output of the DAC (zero-order-hold function) implies that the output

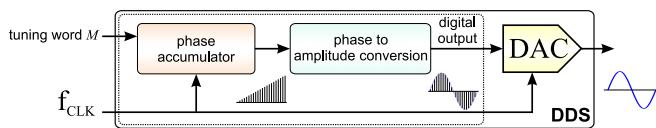


Fig. 14: basic principle of Direct Digital Synthesis (DDS)

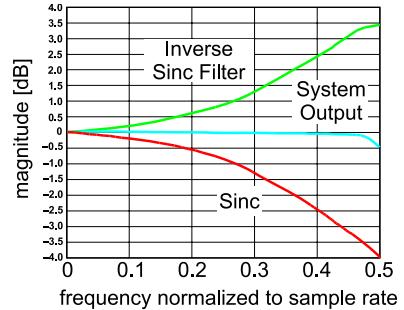


Fig. 15: inverse $\sin(x)/x$ correction at the output of DDS

amplitude spectrum follows a $\sin(x)/x$ function. This roll-off can be quite significant if higher output frequencies are desired. A solution to this problem is to pre-compensate the roll-off by digital inverse $\sin(x)/x$ filters (also called inverse sinc filters) before the data is sent to the DAC (Fig. 15). In this way, flat output amplitudes within ± 0.1 dB over a bandwidth of 80% of the Nyquist frequency can be obtained. In modern DDS chips, the $\sin(x)/x$ correction is already built in.

3.3.2 Cascaded Integrator Comb filter—CIC

As the final blocks in the DDC application, Cascaded Integrator Comb (CIC) filters are well suited as decimating low-pass filters. They belong to the class of multi-rate FIR filter which performs decimation or interpolation. This type of filter was introduced by Eugene Hogenauer in 1981 [5]. A CIC filter is a computationally efficient implementation of a narrowband low-pass filter; they require no multiplication, as we shall now see. The two basic building blocks of a CIC filter are an integrator and a comb (see Fig. 16). The integrator is a one-pole IIR filter with a unity feedback coefficient. The difference equation

basic integrator

$$y[n] = y[n - 1] + x[n]$$

transfer function:

$$H_I(z) = \frac{1}{1 - z^{-1}}$$

basic comb

$$y[n] = x[n] - x[n - D]$$

transfer function:

$$H_C(z) = 1 - z^{-D}$$

Fig. 16: Basic elements of a CIC filter

of the integrator and the corresponding transfer function are given in Fig. 16. As usual in digital signal

processing theory, the *delay by one clock* operator (z^{-1}) in the z -domain has been introduced. In a comb filter, a delayed version of the input is subtracted from itself, which causes constructive and destructive interference at the output. The resulting frequency response consists of a series of equally spaced spikes, which resemble a comb. The difference equation of the comb filter and its transfer function are also given in Fig. 16. The parameter D is often referred to as *differential delay*. In general, a cascade of M integrators and the same number of comb filters make up an M stage CIC filter. Additionally, a rate changer is inserted between integrator and comb, which changes the data rate by a factor of R [see Fig. 17(a)]. The decimation operation $\downarrow R$ means to discard all but every R^{th} sample. The integrators

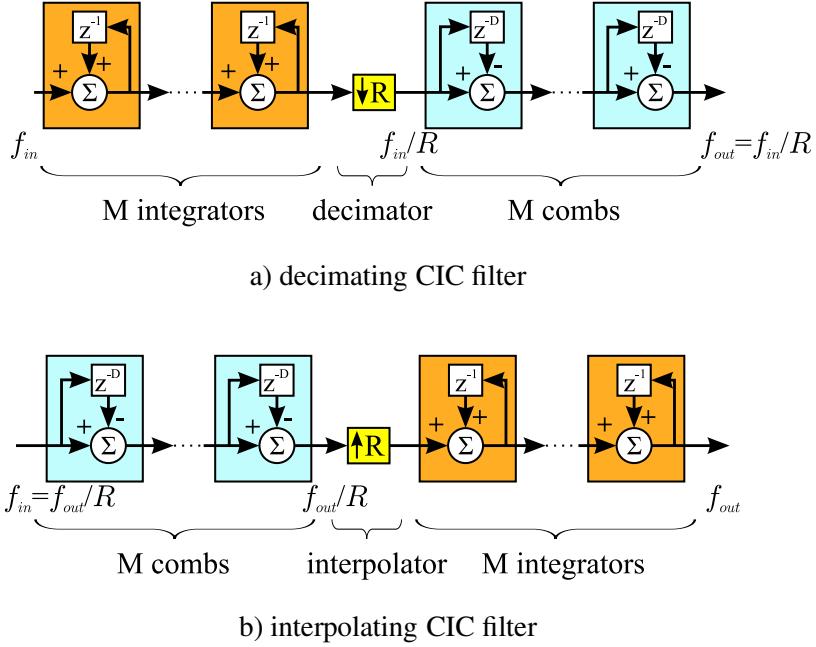


Fig. 17: Basic structure of a decimating (a) and interpolating (b) CIC filter

push date at a rate f_{in} while the comb filter stages operate at a reduced rate $f_{out} = f_{in}/R$, which minimizes power consumption in the high-speed digital hardware. Likewise, an interpolating CIC filter is composed of the same building blocks as a decimating CIC filter, only the sequence of integrators and combs are exchanged and a rate expander is inserted in between [Fig. 17(b)]. Thus this structure of CIC filters clearly reveals that only additions and subtractions are required for their implementation. The total transfer function of the decimating CIC filter is obtained by multiplying the individual transfer functions of the building blocks. However, care has to be taken with the comb transfer function since the comb stages subsequently operate at a reduced clock rate. If we refer the comb transfer function to the high input rate f_{in} , we have to replace the differential delay D in Eq. (15) by $R \cdot D$, since a delay of D at the reduced clock rate f_{out} corresponds to $R \cdot D$ clock cycles with respect to f_{in} . The transfer function of the basic comb filter after the rate changer is then given by

$$H_C(z) = 1 - z^{-RD}.$$

The total transfer function of an M stage CIC filter is therefore expressed by

$$H(z) = (H_I)^M (H_C)^M = \frac{(1 - z^{-RD})^M}{(1 - z^{-1})^M} = \left(\sum_{k=0}^{RD-1} z^{-k} \right)^M, \quad (16)$$

where the geometric sum is used to express $H(z)$ as a cascade of M FIR filters. Although initially it may have seemed that a CIC filter could be unstable—since it contains poles from the integrator—it is always stable because it is a pure FIR filter.

Up to now, the building blocks of CIC filters have been introduced without regard to any other particular motivation. An alternative approach to understanding the structure of CIC filters can be found by looking at recursive running-sum filters, which are well-known. In fact, CIC filters are derived from the concept of those filters [6]. In the following, we shall begin with a moving average filter (or box-car filter) of length N and demonstrate how this structure very closely resembles a CIC filter of first order. The difference equation of a moving average filter is given by

$$y[n] = \frac{1}{N} (x[n] + x[n-1] + \dots + x[n-N+1]) = \sum_{k=n-N+1}^n \left(\frac{1}{N} \cdot x[k] \right). \quad (17)$$

This is a standard FIR filter with equal coefficients. The corresponding transfer function can be easily deduced from this equation and can be expressed as

$$H(z) = \frac{1}{N} (1 + z^{-1} + \dots + z^{-(N-1)}) = \frac{1}{N} \sum_{k=0}^{N-1} z^{-k} = \frac{1}{N} \frac{1 - z^{-N}}{1 - z^{-1}}. \quad (18)$$

Here the formula for a geometric series has been used to represent the sum in a more convenient way. An alternative implementation of this box-car filter is the recursive running-sum. Instead of summing up all past N elements, it is sufficient to add only the latest scaled new sample $x[n]$ to the previous average value $y[n-1]$ and to subtract the oldest scaled sample $x[n-N]$.

$$\text{recursive running-sum: } y[n] = y[n-1] + \frac{1}{N} (x[n] - x[n-N])$$

The recursive running-sum can also be calculated in a slightly different, cascaded way. Rewriting Eq. (17) to

$$y[n] = \frac{1}{N} \left(\sum_{k=-\infty}^n x[k] - \sum_{k=-\infty}^{n-N} x[k] \right).$$

and defining

$$w[n] := \sum_{k=-\infty}^n x[k]$$

yields the set of equations

$$w[n] = w[n-1] + x[n] \quad (19)$$

$$y[n] = \frac{1}{N} (w[n] - w[n-N]). \quad (20)$$

In many applications, the box-car filter is followed by decimation of factor $R = N$ which means that only after N samples is the average value passed on for further processing. The cascaded box-car implementation given by Eq. (19) and Eq. (20) with decimation is visualized in Fig. 18. Since all operations

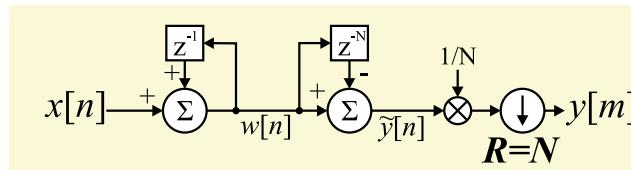


Fig. 18: Alternative representation of a recursive running-sum filter with decimation by factor N

are linear, it is possible to move the decimation from the end to just after the first integrator, i.e., decimate

$w[n]$. Only the *delay by N samples* operator (z^{-N}) has to be changed to z^{-D} with $D = N/R$ because of the rate change. If we compare the resulting transfer function

$$H_{\text{box-car}}(z) = \frac{1}{N} \frac{1 - z^{-RD}}{1 - z^{-1}}$$

with the one which we obtained for a first order CIC ($M = 1$) in Eq. (16), it becomes clear that box-car or recursive running-sum filters have the same transfer function as a first-order CIC—apart from the $1/N$ scaling factor and a more general differential delay D in CIC filters.

Cascaded integrator comb filters are typically employed in applications which have a large excess sample rate, i.e., in systems where the system sample rate is much larger than the bandwidth occupied by the signal. The purpose for which CIC filters are used in these applications are anti-aliasing filtering and decimation. To see the low-pass characteristic, we need to examine the frequency response. This is obtained by evaluating the transfer function along the unity circle in the z -plane by inserting

$$z = e^{i\omega T_S} = e^{i2\pi f_S}, \quad f_S = f_{in}: \text{sampling or input frequency}$$

into Eq. (16). The variable f_S is the sampling frequency or the input frequency f_{in} to the CIC filter. As a result, the magnitude response of a CIC filter can be calculated to

$$|H(e^{i\omega T_S})| = \left| \frac{\sin(\pi RD \frac{f}{f_S})}{\sin(\pi \frac{f}{f_S})} \right|^M.$$

Since the decimation factor R is freely programmable, it is more convenient to express the magnitude response with respect to the output frequency

$$f_{out} = \frac{f_S}{R} = \frac{f_{in}}{R}$$

which yields

$$|H(f)| = \left| \frac{\sin(\pi D \frac{f}{f_{out}})}{\sin(\pi \frac{f}{R f_{out}})} \right|^M. \quad (21)$$

The differential delay D , the decimation factor R , and the number of CIC stages M are the filter design parameters, chosen in such a way so as to fulfil the desired passband characteristics. The output spectrum has zeros, if the frequency is a multiple integer of f_{out}/D .

$$\text{filter zeros at } f = k \cdot \frac{f_{out}}{D} \quad k : \text{integer}.$$

Another property of CIC filters is their gain. When we compared CIC filters with box-car filters, it already became clear that they are almost identical apart from a scaling factor. The DC gain for CIC decimators can be calculated as

$$\lim_{f \rightarrow 0} |H(f)| = |H(0)| = (RD)^M. \quad (22)$$

Because of this fact, the magnitude response of a CIC filter is usually plotted relative do the DC magnitude. In other words, the magnitude is very often given as $|H(f)|/|H(0)|$. As a consequence of the DC gain, an internal register growth in the CIC path occurs. Each additional integrator must add the required number of bit width to account for the register growth which is equal to $\log_2(RD)$. As an example, if a differential delay $D = 1$ and a decimation of $R = 8$ is chosen, a register bit width of three additional bits is required at each integrator. In addition, overflows occur at each integrator but they are of

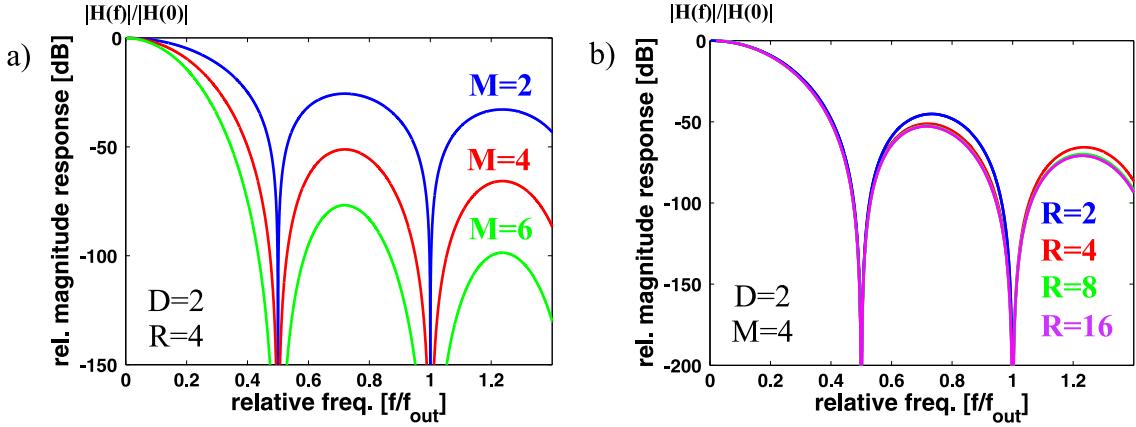


Fig. 19: Frequency response of CIC filter as a function of (a) the integrator and comb stages M ; (b) the decimation ratio R

no consequence so long as the filter is implemented with two's complement (non-saturating) arithmetic. The frequency response of a CIC filter as a function of the integrator and comb stages M is shown in Fig. 19(a), and as a function of the decimation ratio R in Fig. 19(b). In both cases, the differential delay is held constant at $D = 2$. There are several properties to note. First of all, the differential delay D sets the number of zeros in the frequency band up to f_{out} . Secondly, the shape of the filter is not influenced very much by the decimation ratio R . For values of R above approximately 16, changes in the filter shape are negligible. Thirdly, the passband attenuation is a function of the integrator and comb stages M . As can be seen in Fig. 19(a), the first side lobe is attenuated by 26 dB for $M = 2$ and increases with the increasing number of CIC stages M ; this acts to improve alias rejection. On the other hand, increasing M also increases the passband droop, which might not be acceptable in some applications. A solution to this problem could be to compensate the droop by using an additional non-CIC-based filter. For decimating CIC filters, the compensation filter is placed after the CIC, thus operating at a reduced clock speed (left plot in Fig. 20). In contrast to this, a pre-compensation filter is placed before the CIC

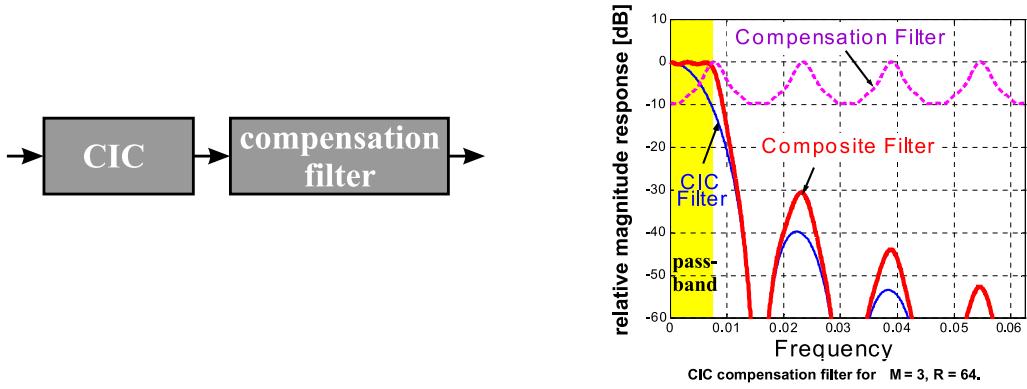


Fig. 20: Compensation filter for CIC to compensate the passband droop [7]

stage for interpolating CIC filters in order to work at the lower clock speed again. On account of the compensation filter, the composite filter can provide a sufficiently flat response within the passband. As an example, Fig. 20 shows a three-stage CIC filter (blue line) with a decimation ratio of $R = 64$ combined with a $x/\sin(x)$ -shaped compensation filter (pink dotted line); the frequency response of the composite filter (thick red line) is sufficiently flattened within the desired passband (yellow shaded box).

3.3.3 Comparison of IQ sampling and DDC

After having introduced digital down conversion and *IQ* and non-*IQ* sampling, we shall now briefly compare the different approaches. In fact, both schemes have lots in common. The data stream of *IQ* sampling [see Eq. (6)], $Q, I, -Q, -I$, can also be regarded as the result of a mixing process with $f_{LO} = f_S = 4 \cdot f_{IF}$ which corresponds to the NCO frequency. The output of the mixing process contains the difference and the sum frequencies, $f_S - f_{IF}$ and $f_S + f_{IF}$. The two frequencies represent the 3rd and the 5th harmonic which are mapped on the IF frequency due to aliasing. In contrast to DDC, in which the NCO is usually set to the IF frequency and the sum frequency has to be filtered by the successive low-pass filter, with *IQ* sampling this is not required, owing to the aliasing of both components to the IF frequency. Some basic properties of DDC and *IQ* sampling are provided in Table 3. DDC has advantages

Table 3: Basic properties of DDC and IQ demodulation

DDC	<i>IQ</i> demodulation
<ul style="list-style-type: none"> – long group delay (depending on clock speed and number of taps in the CIC/FIR filters) – very flexible (NCO can follow f_{IF} over a broad range) – data reduction and good S/N ratio 	<ul style="list-style-type: none"> – low latency, simple implementation – f_S is locked to $4 \cdot f_{IF}$ – sensitive to clock jitter and non-linearities – non-<i>IQ</i> sampling provides better S/N ratio on cost of latency

in applications with large, varying, IF frequencies and where a good signal-to-noise ratio is required, but at the cost of reasonable latency times. *IQ* demodulation, on the other hand, is mainly used in feedback applications where very short latency is important and where the sampling frequency can be locked to the IF frequency.

4 Up-conversion

In applications where the control of RF signals is required (like in feedback or in feed-forward systems), it is necessary to convert a digital baseband signal to a real passband signal. Ideally, a single DAC could generate the desired RF signal directly (Direct Digital Synthesis, DDS, see Fig. 14). However, today's DACs do not yet meet the precision and speed requirements for RF signals in the range of several hundreds of MHz. A series of DACs and mixers have to be used to provide the conversion into the RF domain. A brief overview about different up-conversion schemes will now be given. Basically, up-conversion techniques can be split into two approaches, homodyne (direct) and heterodyne.

4.1 Homodyne up-conversion

In homodyne up-conversion (also called direct or baseband up-conversion), the baseband I and Q signals are converted to analog signals and are then mixed with the in-phase and quadrature-phase components of a local oscillator (LO). The LO signal is split with a 90° hybrid in order to generate the required phase shift between the two paths. The translated baseband signals are then summed to generate the final RF signal (see Fig. 21 with the corresponding frequency spectrum). This device is generally known as *vector modulator*. The LO frequency is located at the desired RF carrier frequency and no intermediate frequency is required. The direct up-conversion is widely used because of its simplicity and cost effectiveness. The mixers in the vector modulator are operated as amplitude control elements since the I and Q inputs (baseband) directly control the amplitude of the two signal paths. We remember that in Eq. (2) and Eq. (3) we have seen that an arbitrary RF signal can be decomposed into its sine and cosine

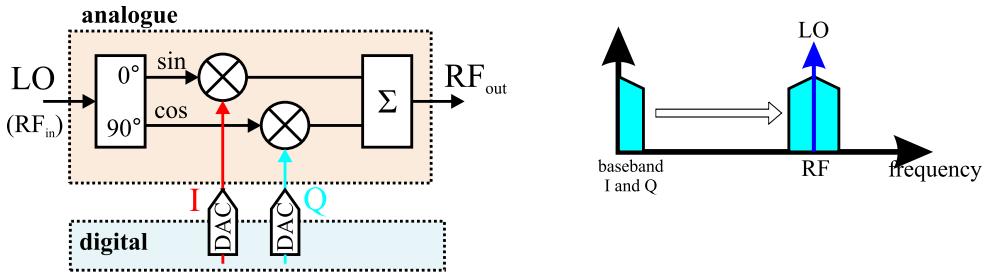


Fig. 21: Homodyne up-conversion approach (vector modulator)

components.

$$\begin{aligned} RF_{out}(t) &= I \cdot A_{LO} \cdot \sin \omega t + Q \cdot A_{LO} \cdot \cos \omega t \\ &= A_{out} \cdot \sin(\omega t + \varphi_0). \end{aligned}$$

And vice versa, if the I and Q amplitudes are controlled, the amplitude and phase of the outgoing carrier signal can be changed. The corresponding output amplitude and phase are given by

$$A_{out} = A_{RF} \sqrt{I^2 + Q^2} \quad \varphi_0 = \text{atan} \left(\frac{Q}{I} \right). \quad (23)$$

With this approach, a pure amplitude or a pure phase modulation can be implemented very easily. For amplitude modulation, the I and Q inputs are modulated with a common time-varying amplitude function $A_0(t)$.

$$\begin{array}{ll} \text{amplitude modulation:} & I(t) = A_0(t) \cdot \cos \varphi_0 \\ & Q(t) = A_0(t) \cdot \sin \varphi_0. \end{array}$$

For pure phase modulation, the I and Q inputs follow a common time-varying phase function $\varphi_0(t)$ while the amplitude is kept constant.

$$\begin{array}{ll} \text{phase modulation:} & I(t) = A_0 \cdot \cos \varphi_0(t) \\ & Q(t) = A_0 \cdot \sin \varphi_0(t). \end{array}$$

Although direct up-conversion is often preferred due to its simplicity, the approach is prone to several sources of errors. The use of analog components with their part-to-part variation and non-linearities introduces many errors like DC offsets, gain imbalance, LO phase noise, and quadrature skew. Any DC offsets in either I or Q will lead to carrier leakage, while a gain imbalance between the two signal paths results in phase-to-amplitude modulation. Ideally, the split LO signal is exactly 90° out of phase. In reality, the quadrature skew—the deviation from the 90° phase shift—introduces coupling between I and Q . Note that digital signal processing can be employed to pre-distort the input of the vector modulator in order to minimize the quadrature skew:

$$\begin{pmatrix} I_{out} \\ Q_{out} \end{pmatrix} = \frac{1}{\cos \varphi_S} \cdot \begin{pmatrix} \cos \varphi_S & -\sin \varphi_S \\ 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} I_{in} \\ Q_{in} \end{pmatrix}.$$

The skew phase is denoted by φ_S . The ideal inputs I_{in} , Q_{in} are pre-distorted to I_{out} , Q_{out} in order to counteract the skew caused by the analog vector modulator components.

4.2 Heterodyne up-conversion

Heterodyne up-conversion (also called IF up-conversion) applies the concept of digital up-conversion of I and Q signals to an IF frequency which is converted to an analog signal by a single DAC. The analog IF

signal is then mixed by means of at least one additional local oscillator (single-stage mixing) or several mixers (multistage mixing) to translate the IF signal to the desired RF frequency. The LO frequency has to be chosen such that the desired RF frequency corresponds to the sum of the LO and IF frequencies. In addition to the sum frequency, the mixing process also generates an image frequency at $f_{LO} - f_{IF}$. A schematic of single-stage heterodyne up-conversion along with the corresponding frequency spectrum is shown in Fig. 22. This approach is also called double-sideband modulation because of the generation

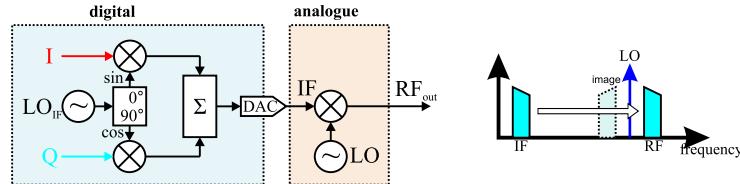


Fig. 22: Heterodyne up-conversion approach (IF up-conversion) as double-sideband modulator

of both frequency bands, i.e., the desired RF along with its image frequency band. Using this method requires filtering to ensure that the image frequency band is sufficiently suppressed and does not interfere with the RF frequency band. This concept is also known as the *filtering method*. In contrast to the homodyne up-conversion approach, the translation of I and Q to an IF frequency takes place digitally, thus it is not susceptible to gain imbalance and quadrature skews. However, the required DACs have to provide a higher bandwidth and are thus more subject to errors such as harmonic distortion and passband ripple.

An alternative for avoiding the image band is the single-sideband modulation scheme which is shown in Fig. 23. The digitally up-converted I and Q signals are converted to analog sine and cosine

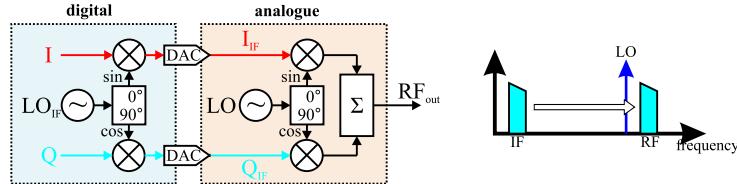


Fig. 23: Single-sideband modulator scheme

waveforms which are then applied to a vector modulator (direct quadrature modulator). Mixing the I_{IF} and Q_{IF} signals with the frequency f_{LO} generates frequencies at $f_{LO} \pm f_{IF}$. The translated I frequency $f_{LO} - f_{IF}$ is 180° out of phase with respect to the translated Q frequency $f_{LO} + f_{IF}$. These two frequencies will cancel each other out (at least in the ideal case), thus leaving a single sideband at the output of the vector modulator. A graphical representation of the image cancellation is shown in Fig. 24. In reality, gain imbalance and quadrature skew prevent complete cancellation; this results in

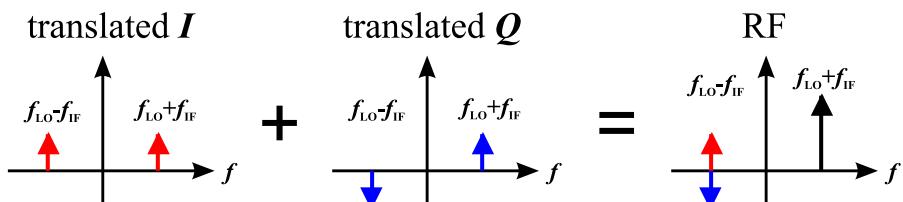


Fig. 24: Graphical representation of the image cancellation in the single-sideband modulator scheme

the unsuppressed sideband at $f_{LO} - f_{IF}$. The amplitude ratio between the desired single sideband at $f_{LO} + f_{IF}$ and the unwanted image sideband is called sideband suppression and is measured in [dBc].

5 Algorithms in RF applications

Following this discussion of signal conditioning/down conversion, digitization, *IQ* detection, as well as the up-conversion process (see Fig. 1), we shall now turn to common algorithms used in RF applications. In this respect the algorithms will be divided into feedback and adaptive feed-forward algorithms. Finally, a brief introduction to system identification will be given at the end.

5.1 Feedback algorithms

Among the many algorithms applied in the field of RF, we shall concentrate on only two particular cases for the purposes of this lecture: amplitude and phase feedback algorithms for RF cavity control, and phase and radial loops as applied in circular accelerators.

5.1.1 Amplitude and phase feedback for RF cavities

The task of these amplitude and phase feedback systems, generally known as Low-Level RF systems (LLRF), is to precisely maintain the phase and amplitude of the accelerating fields used to accelerate charged particle beams. There are many different types of RF cavities in use today (travelling wave structures, standing wave structures, spoke cavities, RFQ, ferrite loaded cavities, etc.), and their operating frequencies cover the range of some several MHz up to tens of GHz. Amplitude and phase feedback systems range from the control of normal and superconducting cavities (which are operated in pulsed or CW mode) up to the control of single and multi-cell cavities, with one or more cavities per RF amplifier (vector sum control). During the design process, one has to decide whether an analog, a digital, or a combined feedback system will be used. Likewise, the pros and cons of *IQ* versus amplitude/phase control have to be considered. The typical requirements for relative amplitude stability are in the range of $10^{-2} - 10^{-4}$ and for phase stability in the range of $10^{-2} - 10^{-4}$ rad, whereas the toughest requirements originate from FEL linac facilities. Apart from the pure amplitude and phase control, the systems very often have to fulfil additional tasks such as exception handling along with providing automated calibration routines and extensive diagnostics capabilities. If large bandwidths are required, analog systems—with their usually shorter loop latencies—have a big advantage, but more and more digital systems have been developed and are now in operation at accelerator facilities.

In the following section, we shall use standard control theory to analyse accelerator amplitude and phase feedback systems in more detail. A basic feedback loop of a system (e.g., an RF plant) with transfer matrix $G(s)$ and a controller with transfer matrix $C(s)$ is depicted in Fig. 25. The output of

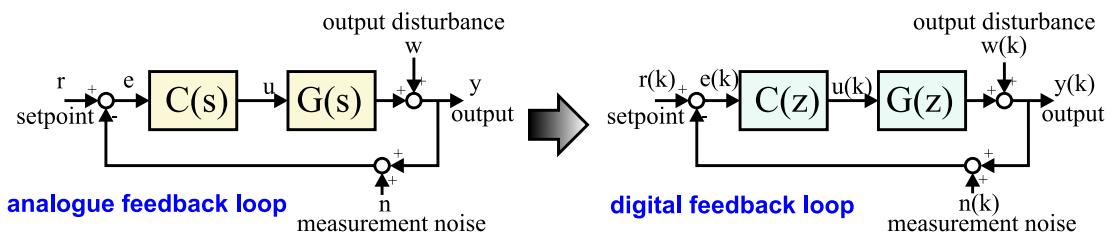


Fig. 25: Translation of a basic analog to a digital feedback loop

the system is described by y , the set-point by r , and the controller output by u . In addition, the output disturbance w and the measurement noise n are included. In order to describe a digital feedback loop, all continuous variables need to be transferred to the discrete domain. Mathematically speaking, the transfer matrices have to be transformed from the Laplace domain into the z -domain. While the controller can be directly designed in digital form, the transformation of the continuous plant into the z -domain requires a zero-order-hold block to be inserted. This since a digital controller only outputs discrete values (the latter are kept constant between two samples by a DAC, in order to deliver continuous inputs to the plant). The cascade of zero-order-hold block and continuous transfer matrix has to be transformed from the Laplace

into the time domain by the inverse Laplace operator \mathcal{L}^{-1} and then to the z -domain by \mathcal{Z} . The resulting rule is

$$\begin{aligned} G(z) &= \mathcal{Z}\left\{\mathcal{L}^{-1}\left\{G(s) \cdot H_{ZOH}(s)\right\}\Big|_{t=kT_S}\right\} & T_s : \text{sampling period, } k : \text{integer} \\ G(z) &= \frac{z-1}{z} \mathcal{Z}\left\{\mathcal{L}^{-1}\left\{\frac{G(s)}{s}\right\}\Big|_{t=kT_S}\right\}. \end{aligned} \quad (24)$$

The continuous transfer matrix $G(s)$ includes all sub-systems such as modulators, pre-amplifiers, cavities, klystrons, etc. In order to design the controller for the feedback loop, it is necessary to model the plant sufficiently well. For the sake of simplicity, we reduce the feedback system to a Single Input-Single Output (SISO) system. In this case, the transfer matrices reduce to transfer functions. Based on the system shown in Fig. 25, the output y and the tracking error $e_T = r - y$ can be expressed by

$$\begin{aligned} y &= \frac{GC}{1+GC}[r-n] + \frac{1}{1+GC}w \\ e_T &= \frac{GC}{1+GC}n + \frac{1}{1+GC}[r-w]. \end{aligned}$$

The variable (z) has been omitted for clarity. The transfer function GC denotes the open loop transfer function. With the commonly defined functions

$$\begin{aligned} S &:= \frac{1}{1+GC} && \text{sensitivity function} \\ T &:= \frac{GC}{1+GC} && \text{complementary sensitivity function (closed loop transfer function),} \end{aligned}$$

it becomes obvious that the sum of sensitivity and complementary sensitivity function is equal to 1.

$$S(z) + T(z) = 1. \quad (25)$$

There are several things to note here. Any measurement error n (e.g., IQ detection error) behaves like a change in the set-point r . In order to keep the influence of n on the output y and on the tracking error e_T small, $T(z)$ should be kept small; this improves robustness. On the other hand, the output y should be insensitive to frequency output disturbances w over the range of interest. Very often it is the low-frequency range where $S(z)$ should be minimized for performance reasons. Since $S(z)$ and $T(z)$ are complementary [see Eq. (25)], a trade-off between robustness and performance has to be made. Moreover, it is not possible to minimize $S(z)$ over the entire frequency range. This fact is expressed by the Bode integral formula [8, 9]. If we suppose that the open loop transfer function $G(s)C(s) = n(s)/d(s)$ is a rational function with two more poles than zeros (i.e., the degree($n(s)$) – degree($d(s)$) ≥ 2) and $G(s)C(s)$ is a linear time-invariant system which has no unstable poles (no poles in the right hand plane), then the Bode integral theorem states

$$\int_{-\infty}^{\infty} \ln |S(i\omega)| d\omega = 0. \quad (26)$$

Likewise, the Bode integral formula can also be written for linear discrete time-invariant systems. If $G(z)C(z) = n(z)/d(z)$ is a rational function with one more pole than the number of zeros, where no poles lie outside the unity circle (unstable poles), then the Bode integral theorem is expressed by

$$\int_{-\pi}^{\pi} \ln |S(e^{i\omega})| d\omega = 0. \quad (27)$$

This means the integral of the sensitivity is conserved. Low sensitivity (i.e., good disturbance rejection of the feedback system) over a certain frequency range must be compensated for by a sensitivity with

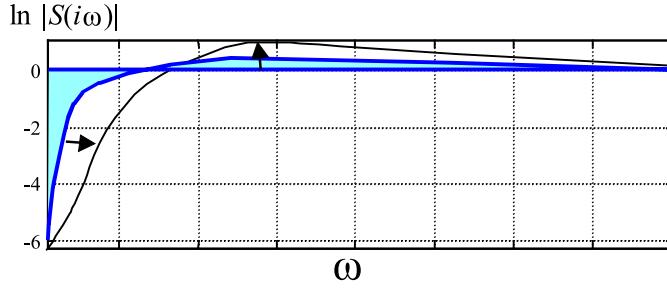


Fig. 26: Principle of Bode's integral theorem (waterbed effect); lowering the sensitivity in one frequency range increases it somewhere else

$|S| > 1$ across another range. This effect is sketched in Fig. 26. If the gain of a feedback loop is increased in order to increase the suppression bandwidth in the low-frequency range, the potential to amplify disturbances in the high-frequency range is also increased. This effect is frequently referred to as the *waterbed effect*. Care has to be taken to make sure that no dominant disturbance lines are located in the frequency range where the absolute value of the sensitivity function is greater than one.

In order to apply control theory to cavity amplitude and phase feedback systems, a model of the RF plant is required. Once all devices have been modelled (i.e., transfer matrices have been obtained), the total RF plant transfer matrix is the result of the transfer matrix products of the individual components. First of all, we shall deduce the transfer matrix of an RF cavity. Resonant modes in cavities can be described by means of resonant LCR circuits [10]. In the following we restrict ourselves to the simple case of a single-passband mode of a cavity. A simplified model of a LCR circuit fed by a generator current \tilde{I}_g and a beam current I_b is shown in Fig. 27. The true generator current I_g which is supplied by

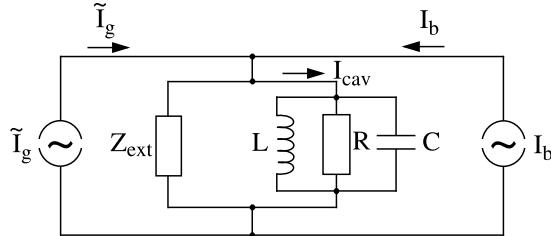


Fig. 27: Simplified model of resonant RF cavity with generator current \tilde{I}_g and beam current I_b

an RF source and transported to the cavity via a transmission line is transformed into the cavity system by means of the cavity input coupler. This transformation is taken into account in Fig. 27 by representing the generator current as \tilde{I}_g [11]. Based on the model depicted in Fig. 27, the differential equation of a driven LCR circuit can be obtained. The current $\mathbf{I}(t)$ is the sum of the generator and beam currents. For RF

$\mathbf{V}(t)$:	cavity voltage
$\mathbf{I}(t)$:	driving current (from generator and beam)
ω_0 :	resonance frequency of undamped cavity
Q_L :	loaded quality factor of cavity
R_L :	cavity resistance incl. external load

applications, the driving current and the cavity voltage are harmonic oscillations with time dependence $e^{i\omega t}$, where ω is the frequency of the driving current. Therefore, we separate fast RF oscillations from the slowly changing amplitude and phase variations.

$$\mathbf{V}(t) = \begin{pmatrix} V_r(t) \\ V_i(t) \end{pmatrix} \cdot e^{i\omega t}, \quad \mathbf{I}(t) = \begin{pmatrix} I_r(t) \\ I_i(t) \end{pmatrix} \cdot e^{i\omega t}. \quad (28)$$

In this formula, the notation of real and imaginary parts of the current and voltage have been chosen instead of the I and Q representation. Inserting Eq. (28) into the cavity differential equation results in the description of the cavity field envelope as a matrix equation.

$$\frac{d}{dt} \begin{pmatrix} V_r \\ V_i \end{pmatrix} = \begin{pmatrix} -\omega_{1/2} & -\Delta\omega \\ \Delta\omega & -\omega_{1/2} \end{pmatrix} \cdot \begin{pmatrix} V_r \\ V_i \end{pmatrix} + \begin{pmatrix} R_L \omega_{1/2} & 0 \\ 0 & R_L \omega_{1/2} \end{pmatrix} \cdot \begin{pmatrix} I_r \\ I_i \end{pmatrix}. \quad (29)$$

The parameter $\omega_{1/2}$ is the cavity bandwidth of the loaded cavity, and $\Delta\omega$ is the difference frequency between the resonance frequency and the frequency of the driving source. Both parameters are defined by

$$\omega_{1/2} = \frac{\omega_0}{2Q_L}, \quad \Delta\omega = \omega_0 - \omega. \quad (30)$$

In order to transform the differential Equation (29) into an algebraic equation, Laplace transformation can be applied which yields

$$\underbrace{\begin{pmatrix} V_r(s) \\ V_i(s) \end{pmatrix}}_{V(s)} = \underbrace{\frac{\omega_{1/2}}{\Delta\omega^2 + (s + \omega_{1/2})^2} \begin{pmatrix} s + \omega_{1/2} & -\Delta\omega \\ \Delta\omega & s + \omega_{1/2} \end{pmatrix}}_{H_{cav}(s)} \cdot \underbrace{\begin{pmatrix} R_L \cdot I_r(s) \\ R_L \cdot I_i(s) \end{pmatrix}}_{U(s)}. \quad (31)$$

$$H_{cav}(s) = \begin{pmatrix} H_{11}(s) & H_{12}(s) \\ H_{21}(s) & H_{22}(s) \end{pmatrix}$$

Having done so, we have obtained the continuous 2×2 transfer matrix of a resonant cavity. A few properties of this matrix will now be highlighted. If the cavity is operated on resonance ($\Delta\omega = 0$), then the off-diagonal elements of the transfer matrix vanish. The cavity acts like a low-pass filter of first order with a roll-off frequency of $\omega_{1/2}$.

$$\Delta\omega = 0 \text{ (cavity on resonance)} : \quad H_{11}(s) = H_{21}(s) = \frac{\omega_{1/2}}{s + \omega_{1/2}} \quad (32)$$

$$H_{12}(s) = H_{21}(s) = 0.$$

In this case, the real and imaginary parts of the cavity voltage (or the I and Q components), and thus amplitude and phase, are completely decoupled. If the cavity is detuned ($\Delta\omega \neq 0$), coupling between I and Q occurs. Especially for superconducting cavities, which are subject to Lorentz force detuning at high gradients, the cavity resonance frequency and therefore $\Delta\omega$ becomes a function of the accelerating field in the resonator. If the cavity is operated in a pulsed mode, then the cavity filling causes a transition of the resonance frequency. This leads to $\Delta\omega = \Delta\omega(t)$. Thus the transfer function $H_{cav}(s)$ becomes time variant, therefore complicating the control task. The stability of the feedback loop has to be guaranteed under these parameter changes, which may require modern control theory approaches to synthesize an *optimal controller*.

As a next step, the transformation from the continuous into the discrete cavity model has to be carried out according to Eq. (24). Although an analytical expression of $H_{cav}(z)$ could be given, the calculation is tedious. Most of the time it is sufficient to use computer programs (*octave*, *Scilab*, *matlab*, etc.) and to analyse the feedback loop with its discrete transfer matrices. An alternative to using the transfer matrix approach is the analysis of amplitude and phase feedback systems in the state space formalism [12] where the differential equation (29) can be written as

$$\dot{\mathbf{x}}(t) = \mathbf{A} \cdot \mathbf{x}(t) + \mathbf{B} \cdot \mathbf{u}(t) \quad (33)$$

$$\mathbf{y}(t) = \mathbf{C} \cdot \mathbf{x}(t) \quad (34)$$

with the definition

$$\mathbf{x}(t) = \begin{pmatrix} V_r(t) \\ V_i(t) \end{pmatrix}, \quad \mathbf{u}(t) = \begin{pmatrix} I_r(t) \\ I_i(t) \end{pmatrix}$$

$$\mathbf{A} = \begin{pmatrix} -\omega_{1/2} & -\Delta\omega \\ \Delta\omega & -\omega_{1/2} \end{pmatrix}, \quad \mathbf{B} = \begin{pmatrix} R_L\omega_{1/2} & 0 \\ 0 & R_L\omega_{1/2} \end{pmatrix}, \quad \mathbf{C} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}.$$

In this lecture we shall not explore the approach of state space formalism in more detail. The transfer matrix analysis of the closed amplitude and phase feedback loop in the digital domain requires the loop delay (which is inherent due to digital signal processing) to be taken into account. Also all cable delays must be considered. This delay of N sample periods is expressed by the z^{-N} transfer matrix in Fig. 28. In most cases, the digital controller for amplitude and phase feedback for RF cavities is typically imple-

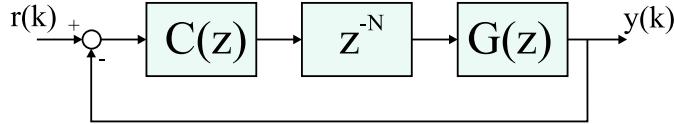


Fig. 28: Block diagram of a digital feedback loop with loop delay of N sample periods

mented using a PID controller. Nevertheless, studies are currently underway in many laboratories that aim at improving the feedback performance by synthesizing an optimal controller. Although PID controllers have been analysed thoroughly in many publications, we shall nevertheless investigate the loop performance with this type of controller in more detail, since PID controllers play such an important role in RF cavity field control. A standard PID control algorithm is expressed by

$$u(k) = u(k-1) + (K_p + K_i + K_d) \cdot e(k) - (K_p + 2K_d) \cdot e(k-1) + K_d \cdot e(k-2) \quad (35)$$

with the error value $e(k)$ at sample time t_k , and the proportional, integral, and derivative gains K_p, K_i, K_d respectively. To simplify matters, we regard the RF plant as consisting only of the cavity, neglecting all other components such as vector modulators, pre-amplifiers, klystrons, pickups, etc. Even with this simplified example, some important properties of the feedback design will be revealed. The plant $G(z)$ in Fig. 28 is described by the transfer function given in Eq. (31). As a standard rule of thumb, control theory recommends a gain margin of at least between 6–8 dB and a phase margin of between 40° and 60°. With a constant integral gain ($K_i = 0.1$), the proportional gain is adjusted in the above-mentioned example such that a gain margin of 8 dB is achieved. Depending on the total loop delay t_D , different maximum proportional gains can be set which yield different loop bandwidths. The sensitivity function as a result of this simulation is shown in Fig. 29. The achievable maximum bandwidth strongly depends on the total loop delay and is very sensitive if the total delay is on the order of the sampling period. Thus the loop delay is an important parameter which has to be minimized. High-speed digital technology is required to keep the delay in the digital signal processing part as short as possible. However, despite ever more powerful digital technology, the larger delays typical of digital feedback systems (compared to their analog counterparts) are still the limiting factor in their application. This becomes most evident if superconducting and normal conducting cavity field feedback systems are compared (see Table 4). In

Table 4: Comparison of basic feedback parameters of normal and superconducting cavities

Superconducting cavity	Normal conducting cavity
$Q_L: \approx \text{few } 10^5\text{--}10^7$	$Q_L: \approx 10\text{--}10^5$
$f_{1/2}: \approx \text{few } 100 \text{ Hz}$	$f_{1/2}: \approx 100 \text{ kHz}$
$\tau_{cav}: \approx \text{few } 100 \mu\text{s}$	$\tau_{cav}: \approx \text{few } \mu\text{s}$
feedback loop delay small compared to τ_{cav}	feedback loop delay in the order of τ_{cav}

this table, the cavity bandwidth $f_{1/2}$ is given by Eq. (30) and the cavity time constant (standing wave cavity) is the time constant of a resonator which is defined by

$$\tau_{cav} = \frac{1}{\omega_{1/2}} = \frac{Q_L}{\pi f_{RF}}. \quad (36)$$

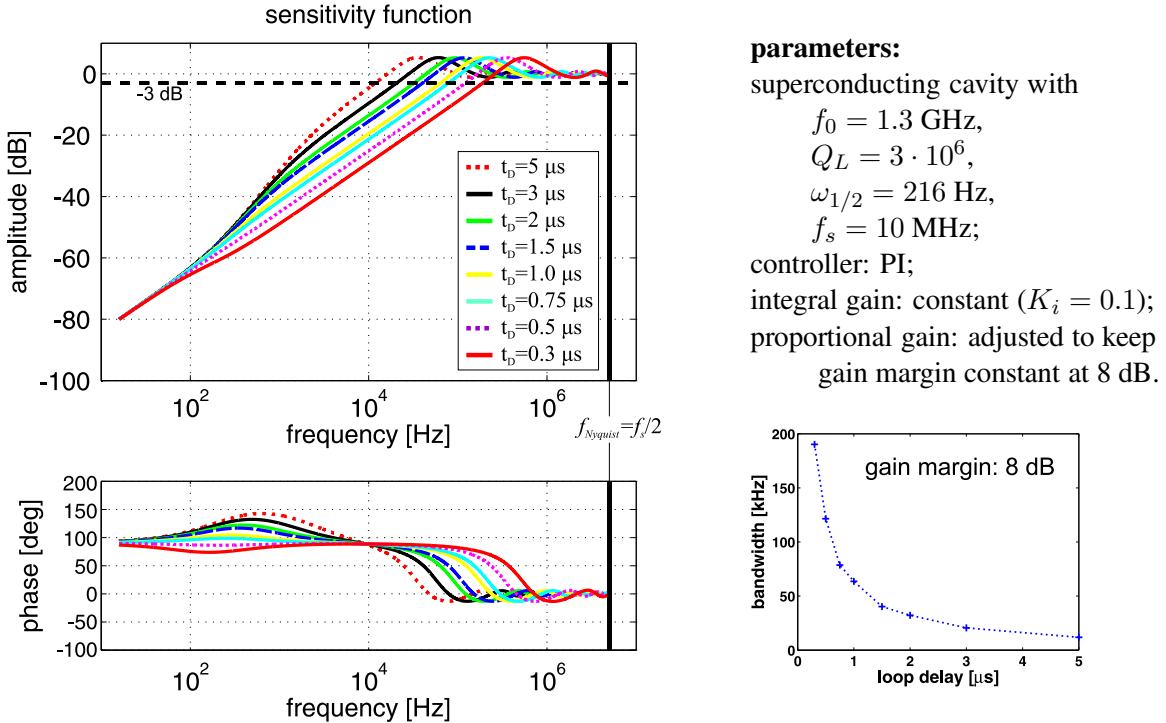


Fig. 29: Bode plot of the sensitivity function of a cavity-field feedback loop for different loop delays

Since the loaded quality factors Q_L of normal conducting cavities are some order of magnitudes smaller than those of superconducting cavities, the bandwidths of the normal conducting cavities are usually in the range of some hundred kHz. As a result, the cavity time constants are comparable with the loop delays which are typical of digital systems nowadays. The consequence is that loop gains are limited because unity gain is already reached with moderate gains at frequencies where the loop phase (dominated by the phase advance of the loop delay) approaches 180° . The achievable gains are highlighted in the open loop Bode plot (simplified model which consists of a series of cavity and loop delay transfer function) in Fig. 30. If the bandwidth provided by a digital feedback system is insufficient, analog or analog/digital

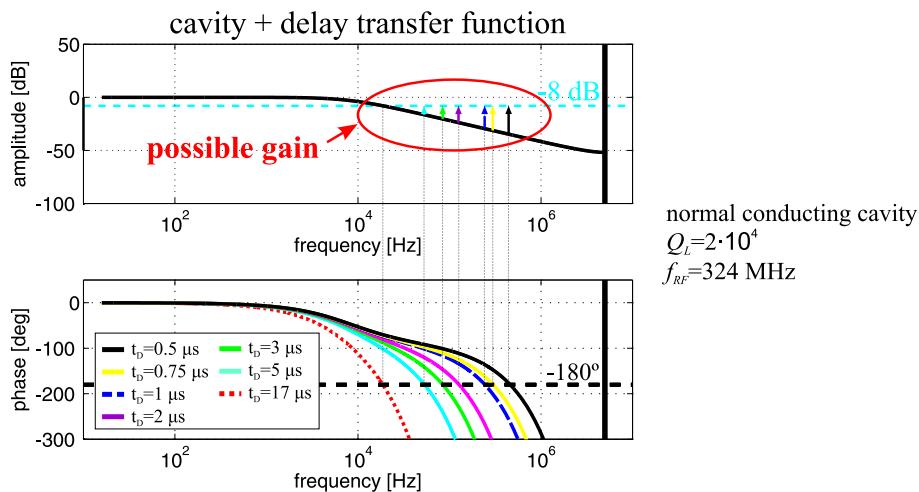


Fig. 30: Open loop Bode plot of a normal conducting RF cavity for different loop delays

hybrid systems might offer an alternative. As a final example, a digital cavity amplitude and phase feedback will now be shown. A schematic layout of the J-PARC LLRF system for the normal conducting

proton linac (Drift Tube Linac, DTL, and Separate-type Drift Tube Linac, SDTL) is depicted in Fig. 31 [13, 14]. The linac is operated in pulsed mode with a repetition rate of 12.5 Hz or 25 Hz, respectively,

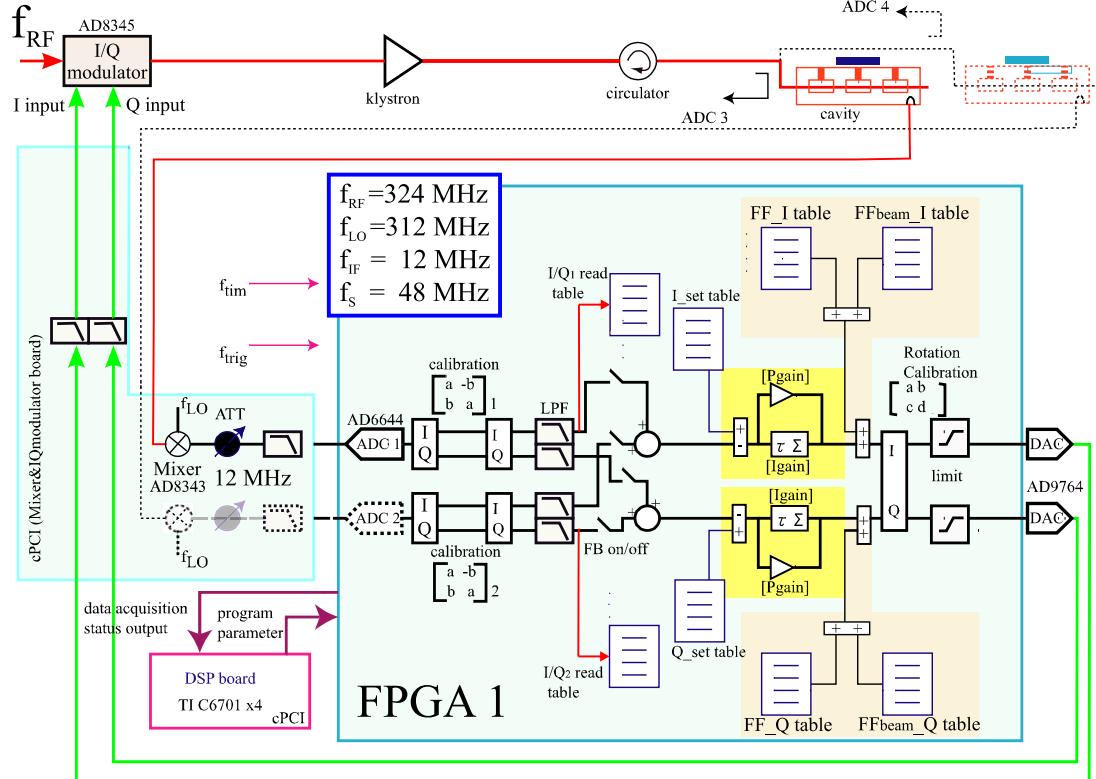


Fig. 31: Example of digital cavity amplitude and phase feedback system (J-PARC linac)

and a pulse length between $500 \mu\text{s}$ to $650 \mu\text{s}$. One klystron feeds two drift tube structures. Sampling the down-converted IF frequencies of 12 MHz with 48 MHz sampling frequency ADCs deliver a stream of I and Q data. The successive rotation matrices perform the 90° phase rotation, including an absolute phase shift, which accounts for the individual cable delays from the cavity pickups to the ADCs. The digital LLRF system calculates and controls the vector sum using a PI control algorithm. In addition, feed-forward is applied by means of lookup tables which are adapted from pulse to pulse. The cavity amplitude and phase control in the form of I/Q control is implemented in a single FPGA, while Digital Signal Processors (DSPs) are used for tuner control, communication and configuration of the FPGA. The normal conducting cavities along the linac have loaded quality factors Q_L between $8\,000$ and $3 \cdot 10^5$ and typical cavity time constants of around $100 \mu\text{s}$ due to the low operating frequency of 324 MHz . It has been demonstrated that the required amplitude and phase stabilities of $<\pm 1\%$ and $<\pm 1^\circ$, respectively, have been superseded by a factor of six. The total loop delay amounts to about 500 ns and the loop bandwidth reaches 100 kHz with moderate proportional gains of 10 and integral gains of 0.01 .

5.1.2 Radial and phase loops

Booster synchrotrons for hadron or ion acceleration are often designed to accelerate a variety of particles with different charge-to-mass ratios. These machines must be able to capture and very often adiabatically rebunch the injected beam, and then accelerate it to the desired extraction energy. The different species of particles require very different acceleration cycles. The relation between the necessary magnetic field change dB/B in a synchrotron, the change of the revolution frequency df/f , the change of the mean

radial position dR/R and the relative particle energy γ is given by [15]

$$\frac{dB}{B} = \gamma^2 \frac{df}{f} + (\gamma^2 - \gamma_{tr}^2) \frac{dR}{R} \quad (37)$$

where γ_{tr} is the transition energy. The LLRF drive system must be very flexible and fast to support the dynamic changes throughout the ramp cycle. The beam control system's task is to calculate the revolution frequency as a function of dipole field, control the cavity voltages, keep the mean radial position of the beam on the desired value, and lastly, to maintain the synchronous phase Φ_s between beam and cavity field. In addition, it must synchronize the RF phase to a master RF clock in order to provide synchronization of the beam transport to other accelerator rings. The RF frequency can change up to a factor of ten during the ramp. The values typically vary from several hundreds of kHz to several tens of MHz. In order to support these large frequency changes, low Q cavities are used which are often ferrite-loaded cavities or magnetic alloy based cavities. In contrast to electron machines, no damping of synchrotron oscillations occurs in hadron machines. As a result, any particles deviating from the synchronous phase Φ_s will lead to oscillations. Moreover, all errors due to phase noise, imperfections in the magnetic field B , power supply ripple, etc., will cause phase and radial errors of the accelerated beam with respect to their design values. Hence radial and phase loop feedback systems are required. To summarize, the beam control system consists of the following main components.

- *Frequency program*

It calculates the frequency based on the B field and based on desired radial position, it optimizes the frequency ramp to improve injection efficiency, and it generates dual harmonic RF drive signals for cavities in order to provide bunch shaping capabilities.

- *Beam phase loop*

It forces the cavity RF phase to follow the beam phase, i.e., it maintains the synchronous phase. Thus it damps coherent synchrotron oscillations from injection errors (energy, phase), bending magnet noise, and low-frequency synthesizer phase noise.

- *Radial loop*

It keeps the beam to its design radial position during acceleration and therefore forces the beam to have the synchronous energy.

- *Cavity amplitude loop*

It compensates for imperfections in the cavity amplifier chain and provides the capability to control the amplitude according to a ramping function.

- *Synchronization loop*

It locks the RF phase of the synchrotron to a master RF clock frequency in order to synchronize different accelerator rings which in turn ensures proper beam transfer.

The required frequency signals were commonly generated by analog methods, mainly by mixers and voltage controlled oscillators (VCOs), which were used to accommodate the large frequency changes during the acceleration cycle. The disadvantage of VCOs is their lack of absolute accuracy. Their stability limitations become evident if frequency tuning is necessary over a broad range. With the recent large advances in digital technology, VCOs were replaced more and more by Direct Digital Synthesis (DDS) in the late 1980s. The agility, reproducibility, and precision achieved with DDS supersedes the application of analog technology in most cases. Since DDS devices are completely digital except for the output DAC, their stability is completely defined by an external, highly stable fixed frequency oscillator. However, the spectral purity depends on the clock frequency and the operating frequency. After the digital implementation of the frequency program, the phase and radial loops were also more and more often implemented as digital feedback systems. In recent years, fully digital beam control systems have been implemented at machines like LEIR, AGS, RHIC, etc. A basic system layout is shown in Fig. 32. All RF signals such as those from cavity pickups, phase pickups (e.g., Wall Current Monitor, WCM), and

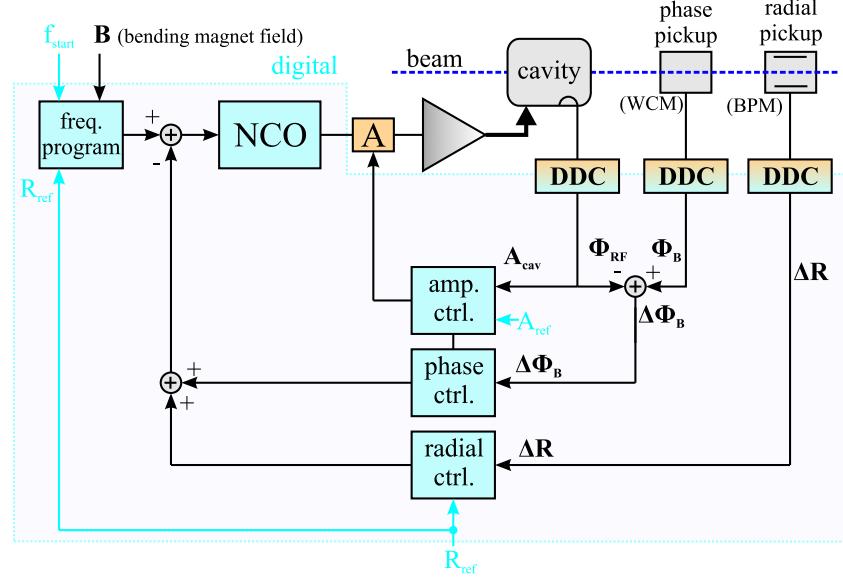


Fig. 32: Basic structure of digital radial and phase loops for hadron/ion synchrotrons

from radial pickups (Beam-Position Monitor, BPM), are digitized by means of I/Q sampling or Digital Down Conversion (DDC). The latter method is usually used since it provides the better possibility to track the changing frequencies as has been shown in Section 3.3. While the amplitude loop acts directly on the amplitude of the drive signal, the radial and phase loop apply a correction to the calculated frequency from the frequency program. The feedback gains of the purely digital control loops can be a function of the beam parameters in order to maintain the same loop performance throughout the acceleration cycle. To demonstrate this statement in more detail, the transfer functions of the radial and phase feedback loops will now be examined. The open loop model of both feedback systems is depicted in Fig. 33 [16]. The output of the DDS is amplified in the simple case as a pure

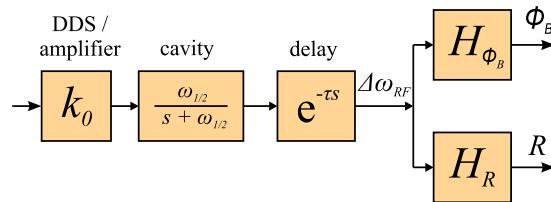


Fig. 33: Open loop model of radial and phase loops in hadron/ion synchrotrons

amplification with factor k_0 . The transfer function of the broadband cavity is given in Eq. (32) (cavity on resonance) and is followed by the loop delay τ . It can be shown by beam dynamics calculation that the transfer function from an RF frequency change $\Delta\omega_{RF}$ to beam phase difference $\Delta\phi_B = \phi_B(s) - \phi_{RF}(s)$ is given by [16, 17]

$$H_{\phi_B}(s) = \frac{\Delta\phi_B}{\Delta\omega_{RF}} = \frac{s}{s^2 + \omega_S^2}, \quad (38)$$

which is based on the model of a lossless oscillation around the synchronous phase Φ_S (linearized synchrotron oscillations). The frequency ω_S is the synchrotron frequency and can be calculated to

$$\omega_S = f_{rev} \cdot \sqrt{\frac{2\pi e V_{RF} \cos \Phi_S h |\eta|}{E}},$$

where f_{rev} is the asymptotic revolution frequency, V_{RF} the amplitude of the cavity voltage, h the harmonic number, E the particle energy, and $|\eta| = |1/\gamma^2 - 1/\gamma_{tr}^2|$ with γ_{tr} the transition energy. Likewise,

the transfer function from an RF frequency change $\Delta\omega_{RF}$ to a radial position change (or to the radial position itself) can be modelled to

$$H_R(s) = \frac{dR}{\Delta\omega_{RF}} = \frac{b}{s^2 + \omega_S^2} \quad (39)$$

with b as a function of energy and synchronous phase.

$$b = \frac{ceV_{RF} \cos \Phi_S}{2\pi\beta\gamma_{tr}E}.$$

We shall not go further into the details of beam dynamic calculations here, but just emphasize that both transfer functions, $H_{\phi_B}(s)$ and $H_R(s)$, are functions of energy which increases during the acceleration cycle. Hence the underlying model is time varying. These models are usually difficult to analyse and are treated in the frame of Linear Parameter Varying (LPV) models. Controllers for LPV models are often scheduled, i.e., the controller parameters are adjusted over time to meet the changing plant dynamics and thus guarantee constant loop performance and stability. As an example, the implementation of a radial loop is shown in Fig. 34. It is a test system implementation for LEIR (Low Energy Ion Ring) which has

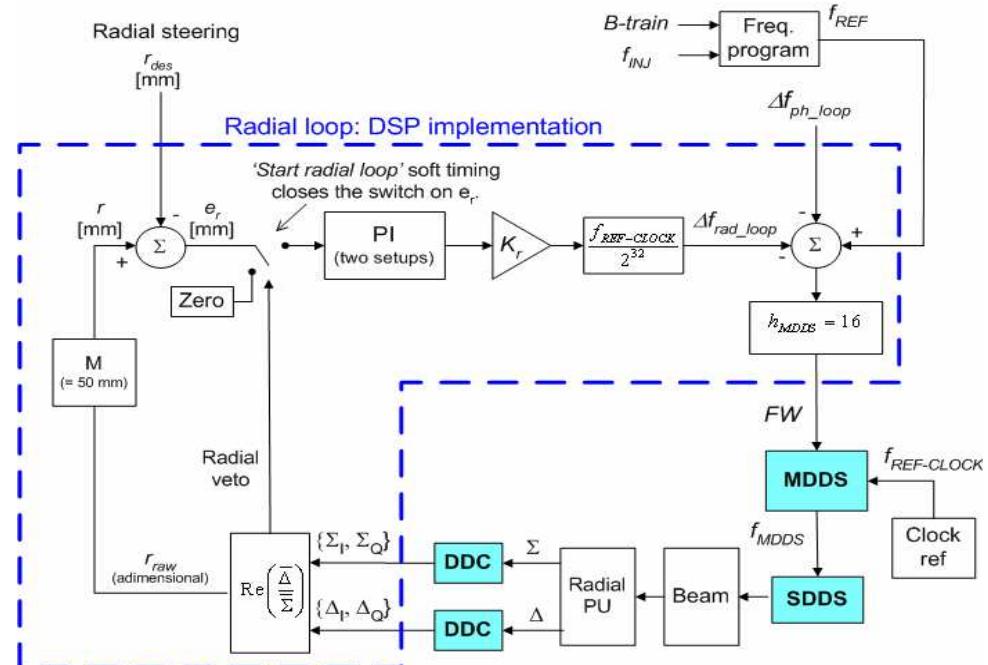


Fig. 34: Example of a radial loop implementation: LEIR system test at the PS Booster at CERN

been successfully tested at the PS Booster at CERN. Parts of the loop run on digital signal processors (feedback loop) while processing-intensive tasks like DDC have been implemented on FPGAs. Apart from the PI controller—which contains two different sets of control parameters (gain scheduling) to match the different parameters at injection and extraction—many of the concepts which have been presented in this course have been applied. The frequency is calculated by the *frequency program* based on the ‘B-train’ input, derived from a field measurement in a reference magnet which is connected in series with the bending magnets in the ring. Corrections from the radial and phase loop are added to the frequency output and applied to a Master DDS (MDDS). The distributed frequency is received by so-called Slave DDS cards (SDDDSs) which locally generate the RF drive frequency for their respective RF cavities. The signals from the beam-position monitor pickups (sum and difference signal) are down-converted by DDCs in which a 1st order CIC with decimation factor $R = 64$ has been implemented. Based on the digitized sum and difference values (I, Q), the position is calculated with a geometric scaling factor M . Finally, a NCO stage follows the PI controller.

5.2 Adaptive feed-forward

The performance of RF feedback systems with repetitive or well-known errors can be improved by the application of adaptive feed-forward algorithms; the goal of these algorithms is to unburden the feedback and/or to suppress known disturbances in advance. The use of feed-forward schemes to compensate beam loading in pulsed accelerating structures is a particularly common practice. The beam arrival time and duration is known in advance, but the induced voltage of the beam can only be compensated by feedback after a certain time, owing to the loop delay. Feed-forward systems have proven effective for suppressing the transients which normally occur if only feedback is used. Since the repetitive disturbances can slowly vary over time, the feed-forward signal has to be adapted by means of adaptive algorithms. The standard feedback loop shown in Fig. 25 is extended by the adaptive feed-forward scheme (see Fig. 35). The

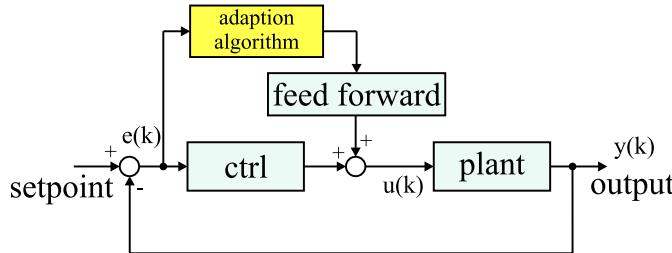


Fig. 35: Basic structure of an adaptive feed-forward in addition to a standard feedback loop

question remains as to how the feed-forward correction should be obtained. Recording the error signal $e(k)$, shifting it in time to compensate for the loop delay, then applying the opposite correction $-e(k)$ will not work, since this approach does not take into account the plant dynamics. What is necessary is to generate the proper additional input to the plant which then results in the required output $-e(k)$. In other words, an inverse model of the plant is necessary to extract the input which leads to the output $-e(k)$. However, this is not always easy in reality. A model to describe the dynamic behaviour of the system is either not known or not precise enough. In addition, stochastic errors are superimposed on the measured total error signal. Averaging methods are necessary in order to separate repetitive from stochastic errors. There are various approaches to attack this problem. One possibility is to obtain a system model by system identification (see Section 5.3), another way is to measure the system response with test input signals (step inputs), record the output, and calculate the system response matrix [18]. Assuming a linear time-invariant Single Input–Single Output (SISO) system, an input $u(k) = u(t_k)$ will cause an output $y(k) = y(\tau_k)$. Note that the definition of the sampling time is made such that the output occurs not earlier than

$$\tau_k = t_k + t_D \quad (40)$$

due to the loop delay t_D . If we change the input by small increments $\Delta u(t_k)$ around the chosen working point with input $u(t_k)$, the output changes by $\Delta y(\tau_k)$ for linear systems (or for sufficient small changes in a first order approximation). The first change can be written as

$$\Delta y(\tau_1) = R_{11} \cdot \Delta u(t_1),$$

where R_{11} describes the linear system response to the input $\Delta u(t_1)$. The second sampled output change is a linear combination of the previous two test input signals, expressed by

$$\Delta y(\tau_2) = R_{21} \cdot \Delta u(t_1) + R_{22} \cdot \Delta u(t_2).$$

Applying successive test input signals $\Delta u(t_k)$ to the system results in a sequence of output changes which can be written in a matrix equation.

$$\begin{pmatrix} \Delta y(\tau_1) \\ \vdots \\ \Delta y(\tau_p) \end{pmatrix} = \begin{pmatrix} R_{11} & 0 \\ \vdots & \ddots \\ R_{p1} & \dots & R_{pp} \end{pmatrix} \cdot \begin{pmatrix} \Delta u(t_1) \\ \vdots \\ \Delta u(t_p) \end{pmatrix} \iff \Delta \vec{y} = \mathbf{R} \cdot \Delta \vec{u}. \quad (41)$$

For reasons of causality, the upper half of the matrix is zero, since an input at time t_k cannot be detected earlier than $\tau_k = t_k + t_D$. Owing to the time definition in Eq. (40), the matrix \mathbf{R} does not contain zero rows or columns. Hence, the response matrix \mathbf{R} can be inverted in the general case and Eq. (41) solved for the required input which produces a measured output change.

$$\Delta \vec{u} = \mathbf{T} \cdot \Delta \vec{y} .$$

Once the inverted response matrix $\mathbf{T} = \mathbf{R}^{-1}$ has been measured, it is possible to calculate the necessary additional input $\Delta \vec{f}$ (feed-forward) to cancel the recorded repetitive error vector \vec{e} .

$$\Delta \vec{f} = \mathbf{T} \cdot (-\vec{e}) = \mathbf{T} \cdot (\vec{y} - \vec{r}) . \quad (42)$$

The vector \vec{r} denotes the set-point. As mentioned earlier, the repetitive error has to be separated from stochastic errors before being applied to the feed-forward adaption algorithm. A successful example of adaptive feed-forward in a pulsed cavity field feedback system (for compensation of repetitive errors due to beam loading and Lorentz force detuning) has been demonstrated at the TESLA Test Facility (now called FLASH) [19, 20]. The basic principle of the algorithm is shown in Fig. 36. The indicated feed-

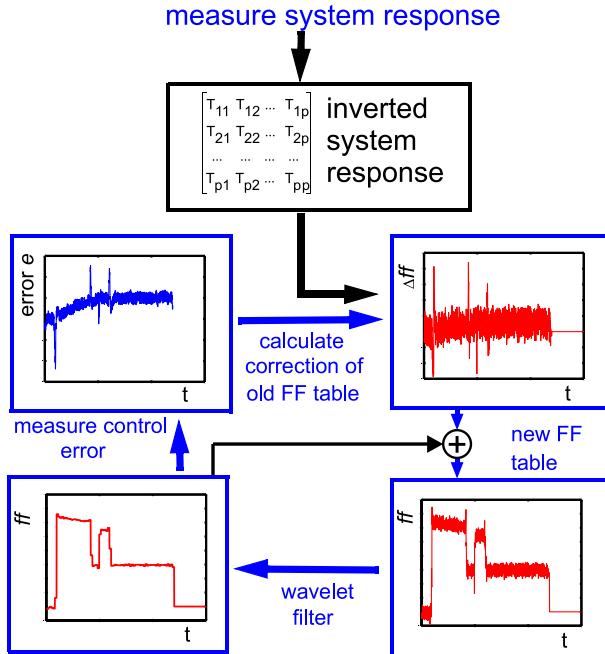


Fig. 36: Structure of an adaptive feed-forward algorithm at the TESLA Test Facility to compensate beam loading and Lorentz force detuning

forward table output forms one of the DAC output values to the vector modulator. Before the calculated feed-forward table is applied, it is wavelet filtered in order to remove stochastic noise introduced by the measurement process. Although this algorithm works fine, it turned out that any change of the working point (gradient and/or phase in the cavities) required the response matrix to be re-measured, since the whole system is highly non-linear. The response matrix measurement and the following adaption process tended to be too slow for the dynamic changes during machine operation. The need for a fast and robust adaptive feed-forward algorithm has led to the development of the so-called *time-reversed filtering* to generate the feed-forward tables; this is described in more detail in Ref. [21]. This approach of generating the feed-forward tables has been successfully demonstrated at FLASH and at the Spallation Neutron Source (SNS) [22] but is only applicable for pulsed systems for which the entire set of measurement data recorded during the pulse is available for the adaption process.

5.3 System identification

The field of system modelling and identification has advanced at a rapid pace during the past decades. System identification refers to the general process of extracting information about a system from measured input–output data. However, it is not possible to treat this subject to anything near its full extent in this lecture. We shall concentrate only on basic concepts and show an example of an application to a real RF plant. More detail about system identification is given in the lectures published in Ref. [12]. The interest in applying system identification to the area of RF control is to aid in the design or synthesis of high-performance controllers. The synthesis of a controller is based upon a model of the system. If a model exists, one can use it to develop a model-based controller. In case a model is not available but a sufficient amount of input–output data is accessible, the data can be used to develop a model of the system first, and then use that identification model to design the controller. Also, the implementation of efficient adaptive feed-forward algorithms requires a model of the system in order to cancel repetitive errors as shown in Section 5.2. The starting point for system identification is the choice of the underlying model type which describes the plant and its dynamics. The general model structure is based on differential equations and transfer functions. In practice, several model structures are used and can be classified by their attributes (linear/non-linear, time variant/time invariant, non-parametric/parametric like linear parameter varying (LPV) models etc.). Common model structures are AR (Auto-Regressive model), ARX (Auto-Regressive model with eXogenous inputs), ARMAX (Auto-Regressive Moving Average with EXogenous Input), Box-Jenkins (combination of moving average and autoregressive approaches), Output Error (OE), or FIR. System identification is based on the following main steps:

1. record output data with proper input signal

The input signal has to excite all relevant frequencies of the system in order to provide significant output data.

2. choose model structure

- grey box model

based on a physics model; preserves known structures with a number of unknown free parameters

- black box model

no underlying physics model, parameters have no direct physical meaning

3. estimate model parameter

The error $e(k)$ has to be minimized by finding an appropriate parameter set θ for the model.

4. validate the model using a set of data different from that employed in the identification process

Attempts to identify the dynamics of superconducting cavities with Lorentz force detuning have been made [23]. As an example, the Output Error model applied to an RF cavity is illustrated in Fig. 37. In this model, no parameters are used to describe the disturbance characteristics $d(t)$ and are therefore

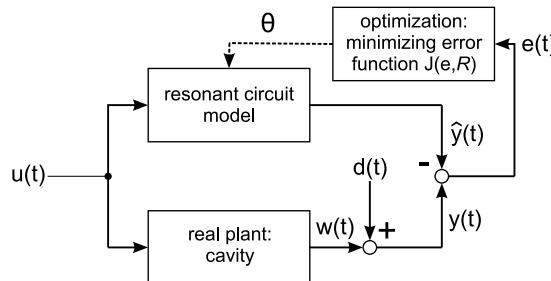


Fig. 37: Structure of the Output Error (OE) model for system identification

simply added to the system output. The model describes the system dynamics separately. To identify the

dynamics, the same inputs $u(k)$ are applied to the model as to the real plant. The resulting differences in the output $e(t) = y(t) - \hat{y}(t)$ are used to optimize the parameter set θ which describes the model. This is achieved by defining an error function with weighting matrix R . Apart from the goal of synthesizing an optimal controller and obtaining a model for feed-forward generation, system identification can also give insights about the system dynamics and provide information about not directly measurable states. Examples for these states are Lorentz force detuning $\Delta\omega(t)$ during an RF pulse [see Eq. (29)], the beam phase with respect to the RF field, or the cavity bandwidth, which is mainly determined by the external coupling [24]. The principle of this approach is illustrated in Fig. 38. As is indicated, either the grey

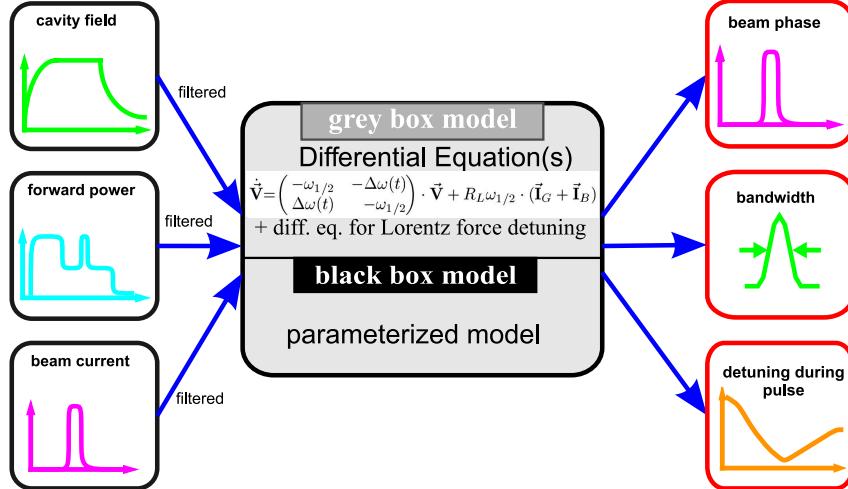


Fig. 38: Example: system identification in pulsed high-gradient superconducting cavities with Lorentz force detuning

box model (based on the cavity model of a LCR circuit in conjunction with differential equations for the Lorentz force detuning [25]) can be chosen, or a *black box model* is applied, in which case the determination of the appropriate model structure and order is typically a trial-and-error process.

6 Conclusion

The development of digital systems in the field of RF applications for accelerators has expanded significantly in the past decade due to the availability of greatly improved high-speed digital electronic hardware. Fast feedback systems, which were usually the domain of analog systems, have been replaced more and more by digital systems if the required regulation bandwidths are not too high. Digital low-level RF systems have much in common with many other accelerator RF sub-systems such as beam diagnostics. Common platforms for both areas are possible, but with dedicated analog front-end electronics. The extensive diagnostics capabilities in digital RF systems allow the implementation of fully automated procedures and calibration schemes for those complex systems. Because of high flexibility and ease of reconfiguration of software-based digital systems, more sophisticated algorithms than currently used are expected to emerge in coming years.

References

- [1] M.E. Angoletta, *Digital Low Level RF*, EPAC'06, Edinburgh, Scotland, 2006, p. 1847.
- [2] R. Garoby, *Low Level R.F. Building Blocks*, CAS CERN Accelerator School, RF Engineering for Particle Accelerators, Oxford, 1991, CERN-92-03, pp. 428–457.
- [3] M. Grecki, T. Jezynski and A. Brandt, *Estimation of IQ Vector Components of RF Field - Theory and Implementation*, MIXDES 2005, Cracow, Poland, 2005, pp. 783–788.

- [4] L. Doolittle, H. Ma and M.S. Champion, *Digital Low-Level RF Control Using Non-IQ Sampling*, LINAC'06, Knoxville, Tennessee, USA, 2006, p. 568.
- [5] E.B. Hogenauer, *An Economical Class of Digital Filters for Decimation and Interpolation*, IEEE Trans. Acoust. Speech Signal Process., vol. ASSP-29, no. 2, April 1981, pp. 155–162.
- [6] M.E. Frerking, *Digital Signal Processing in Communication Systems* (Van Nostrand Reinhold, New York, USA, 1994).
- [7] Xilinx Intellectual Property Center, datasheet, *Cascaded Integrator-Comb (CIC) Filter*, California USA, March 2002.
- [8] B.F. Wu and E.A. Jonckheere, *A Simplified Approach to Bode's Theorem for Continuous-Time and Discrete-Time Systems*, IEEE Trans. Autom. Control, vol. 37, Issue 11, Nov. 1992, pp. 1797–1802.
- [9] B.F. Wu and E.A. Jonckheere, *Chaotic Disturbance Rejection and Bode Limitation*, American Control Conference, Chicago, Illinois USA, June 24–26, 1992, TP1, pp. 2227–2231.
- [10] C.G. Montgomery, R.H. Dicke and E.M. Purcell, *Principles of Microwave Circuits*, Radiation Laboratory Series Vol. 8 (McGraw-Hill, New York, 1942), Ch. 7.
- [11] T. Schilcher, *Vector Sum Control of Pulsed Accelerating Fields in Lorentz Force Detuned Superconducting Cavities*, DESY Print, TESLA 1998-20, 1998.
- [12] S. Simrock, *Lecture on Control Theory*, these proceedings, 2007.
- [13] S. Michizono *et al.*, *Digital Feedback System for J-Parc Linac RF Source*, LINAC'04, Luebeck, Germany, 2004, p. 742.
- [14] S. Michizono *et al.*, *Performance of a Digital LLRF Field Control System for the J-PARC Linac*, LINAC'06, Knoxville, Tennessee, USA, 2006, p. 574.
- [15] J. Bellemans, V. Chohan, J.L. Gonzalez, S. Johnston, E. Schulte and E. Thivent, *Measurement of the Mean Radial Position of a Lead Ion Beam in the CERN PS*, EPAC'96, Sitges, Spain, 1996.
- [16] E. Onillon and J.M. Brennan, *The New BNL AGS Phase, Radial and Synchronization Loops*, EPAC'96, Sitges, Spain, 1996.
- [17] D. Boussard, *Design of a Ring RF System*, CAS CERN Accelerator School, RF Engineering for Particle Accelerators, Oxford, 1991, CERN-92-03, pp. 474–500.
- [18] R. Zhang, I. Ben-Zvi and J. Xie, *A Self Adaptive Feedforward RF Control System for Linacs*, Nucl. Instrum. Methods Phys. Res. A **324** (1993), pp. 421–428.
- [19] M. Liepe and S. Simrock, *Adaptive Feed Forward for the Digital RF Control System at the TESLA Test Facility*, EPAC'98, Stockholm, Sweden, 1998, pp. 1735–1737.
- [20] M. Liepe, *Regelung supraleitender Resonatoren mit Strahlbelastung am TESLA-Test-Linearbeschleuniger*, diploma thesis, University of Hamburg, 1998.
- [21] A. Brandt, *Development of a Finite State Machine for the Automated Operation of the LLRF Control at FLASH*, PhD thesis, University of Hamburg, 2007.
- [22] H. Ma, private communication, ORNL, Tennessee USA, 2007.
- [23] G. Koch, *Modelling of an Accelerator Based X-ray Free Electron Laser System for Controller Design*, diploma thesis, Technical University of Hamburg-Harburg, 2005.
- [24] S. Simrock, *Digital Low-Level RF Controls for Future Superconducting Linear Colliders*, PAC'05, Knoxville, Tennessee USA, 2006, pp. 515–519.
- [25] M. Hüning, S. Simrock, *System Identification for the Digital RF Control System at the TESLA Test Facility*, EPAC'98, Stockholm, Sweden, 1998, pp. 1732–1734.

Multi-bunch feedback systems

M. Lonza

Elettra Synchrotron Light Laboratory, Sincrotrone Trieste S.C.p.A., Trieste, Italy

Abstract

Coupled-bunch instabilities excited by the interaction of the particle beam with its surroundings can seriously limit the performance of circular particle accelerators. These instabilities can be cured by the use of active feedback systems based on sensors capable of detecting the unwanted beam motion and actuators that apply the feedback correction to the beam. The advances in electronic technology now allow the implementation of feedback loops using programmable digital systems. Besides important advantages in terms of flexibility and reproducibility, digital systems open the way to the use of novel diagnostic tools and additional features.

The lecture will first introduce coupled-bunch instabilities analysing the equation of motion of charged particles and the different modes of oscillation of a multi-bunch beam, showing how they can be observed and measured. Different types of feedback systems will then be presented as examples of real implementations that belong to the history of multi-bunch feedback systems. The main components of a feedback system and the related issues will also be analysed. Finally, we shall focus on digital feedback systems, their characteristics and features, as well as on how they can be concretely exploited for both the optimization of the feedback performance and for beam dynamics studies.

1 Coupled-bunch instabilities

1.1 Introduction

The demand for high brightness in synchrotron light sources and luminosity in circular colliders require the storage of intense particle beams with many bunches. The interaction of these beams with the environment in which they move can give rise to collective effects called ‘coupled-bunch instabilities’. Collective instabilities can be seen as follows: charged particles in a storage ring are subject to transverse (betatron) and longitudinal (synchrotron) oscillations that are normally damped by a natural damping mechanism. The electromagnetic field created by the bunches can interact with the surrounding metallic structures generating ‘wake fields’, which act back on the beam producing growth of the oscillations. If the growth is stronger than the damping the oscillation becomes unstable. Since the electromagnetic fields created by the bunches are proportional to their charge, the onset of instabilities and their amplitude are normally current dependent.

Large-amplitude instabilities can cause beam loss, thus limiting the maximum stored current. If non-linear effects prevent beam loss by saturating the oscillation amplitude, instabilities degrade the beam quality, and reduce the brightness or the luminosity of the accelerator.

1.2 Sources of coupled-bunch instabilities and cures

1.2.1 Cavity High Order Modes (HOM)

High-Q spurious resonances of the accelerating cavity structure are excited by the bunched beam and act back on the beam itself. Each bunch affects the following bunches through the wake fields excited in the cavity. In this way the cavity HOM can couple with a beam oscillation mode having the same frequency and give rise to instability. Both transverse and longitudinal multi-bunch modes can be excited.

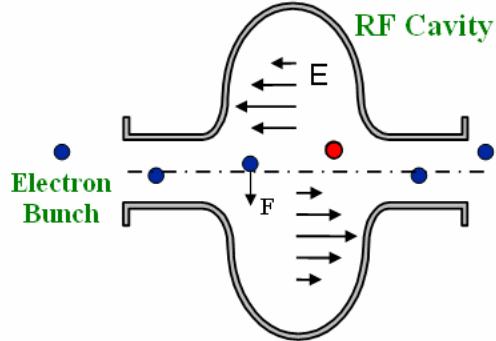


Fig. 1: RF cavity HOMs excited by a particle beam

Excitation of cavity HOMs can be avoided by a thorough design of the RF cavity or by the use of mode dampers made of antennas and resistive loads. Tuning of HOM frequencies through plungers or by changing the cavity cooling water temperature is also currently used in many accelerators [1].

1.2.2 Resistive wall impedance

Resistive wall transverse instabilities are generated by the interaction of the beam with the vacuum chamber through the so-called ‘skin effect’ [2]. They are particularly strong where low-gap chambers and in-vacuum insertion devices (undulators and wigglers) are used.

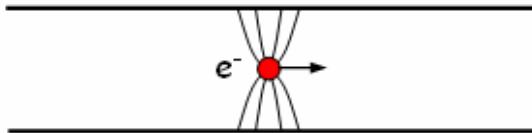


Fig. 2: Interaction of a bunch with the vacuum chamber

This effect can be reduced with an optimization of the vacuum chamber geometry and by using vacuum pipes made of low-resistivity materials.

1.2.3 Interaction with vacuum chamber objects

Coupled-bunch instabilities in both the transverse and longitudinal planes can be created by discontinuities of the vacuum chamber or small cavity-like structures located for example in BPMs, vacuum pumps, bellows, etc.

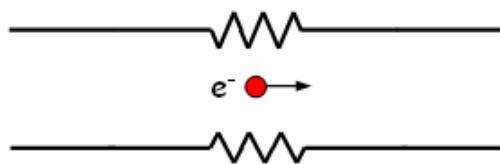


Fig. 3: Discontinuity of the vacuum chamber created by a bellow

The detrimental effect on coupled-bunch instabilities can be reduced with a proper design of the vacuum chamber and of the various installed objects [3].

1.2.4 *Ion instabilities*

Molecules of gas in the vacuum chamber can be ionized by collision with the electron beam. The generated positive ions remain trapped in the negative electric potential and produce electron-ion coherent oscillations [4].

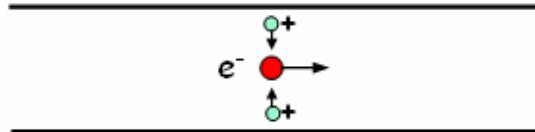


Fig. 4: Interaction of positive ions with electron bunches

A common practice currently used to counteract ion instabilities is the so-called ‘ion cleaning’, obtained by operating the machine with a gap in the bunch train.

1.2.5 *Generic countermeasures*

Besides passive cures that reduce unwanted effects by acting directly on the sources of instability, it is possible to counteract the onset of coupled-bunch instabilities by increasing the Landau damping, which can be obtained by enhancing the betatron/synchrotron tune spread. This can be accomplished, for example, by the use of higher harmonic RF cavities producing bunch lengthening [5], by modulating the amplitude or phase of the accelerating radio frequency [6], or by the use of octupole magnets. When all of the above-mentioned techniques are insufficient to reduce or eliminate coupled-bunch instabilities, active feedbacks are required [7–12].

1.3 Equation of motion of particle bunches

1.3.1 *Equation of motion of charged particles*

The motion of a charged particle in a storage ring can be described with the harmonic oscillator analogy following the equation of motion

$$\ddot{x}(t) + 2D \dot{x}(t) + \omega^2 x(t) = 0 \quad (1)$$

where x is the oscillation coordinate (transverse or longitudinal displacement), D the natural damping and ω the betatron/synchrotron tune frequency given by the $v\omega_0$ where v is the betatron/synchrotron tune and ω_0 the revolution frequency. In the transverse planes, oscillations are periodic horizontal or vertical displacements from an ideal trajectory, while in the longitudinal plane they are periodic ‘timing’ differences with respect to a reference ‘stable particle’ running in the accelerator at constant speed. Longitudinal oscillations are often referred to as ‘phase oscillations’.

If $\omega \gg D$, an approximated solution of the differential equation (1) is a damped sinusoidal oscillation

$$x(t) = k e^{-\frac{t}{\tau_D}} \sin(\omega t + \varphi) \quad (2)$$

where $\tau_D = 1/D$ is the ‘damping time constant’ (D is called ‘damping rate’).

Any excited oscillation (e.g. quantum excitation) is damped by the natural damping (e.g. due to radiation of synchrotron light). The oscillation of individual particles is uncorrelated and manifests itself as an emittance growth.

1.3.2 Coherent bunch oscillations

Coupling with other bunches through the interaction with surrounding metallic structures add a ‘driving force’ term $F(t)$ to the equation of motion

$$\ddot{x}(t) + 2D\dot{x}(t) + \omega^2x(t) = F(t) \quad (3)$$

Under given conditions, the oscillation of individual particles becomes correlated and the centroid of the bunch oscillates coherently with the other bunches giving rise to coherent bunch (coupled-bunch) oscillations. Each bunch oscillates according to the equation of motion:

$$\ddot{x}(t) + 2(D-G)\dot{x}(t) + \omega^2x(t) = 0 \quad (4)$$

where $\tau_G = 1/G$ is the ‘growth time constant’ (G is called ‘growth rate’). Similarly to Eq. (2), if $\omega \gg (D - G)$, an approximated solution of Eq. (4) is

$$x(t) = k e^{-\frac{t}{\tau}} \sin(\omega t + \varphi) \quad (5)$$

where $\frac{1}{\tau} = \frac{1}{\tau_D} - \frac{1}{\tau_G}$. If $D > G$ the oscillation amplitude decays, if $D < G$ it grows exponentially.

Since G is proportional to the stored beam current, if the latter is lower than a given current threshold the beam remains stable, if higher a coupled-bunch instability is excited.

1.3.3 Feedback action

The feedback action adds a damping term D_{fb} to the equation of motion

$$\ddot{x}(t) + 2(D - G + D_{fb})\dot{x}(t) + \omega^2x(t) = 0 \quad (6)$$

In order to damp the oscillation D_{fb} must be such that $D - G + D_{fb} > 0$.

A multi-bunch feedback system detects an instability by means of one or more Beam Position Monitors (BPMs) and acts back on the beam by applying electromagnetic ‘kicks’ to the bunches.

Since the bunch oscillation is sinusoidal, the turn-by-turn position of the one bunch measured at a given location is a sampled sinusoid. From Eq. (6), in order to introduce damping, the force applied by the feedback must be proportional to the derivative of the bunch oscillation. Consequently, the kick signal applied by the actuator to each bunch can be generated by shifting by $\pi/2$ the ‘sampled’ signal of the position of the same bunch when it passes through the ‘kicker’. This is the basic principle of ‘bunch-by-bunch’ feedback systems that will be treated in more detail in Section 2.

1.4 Multi-bunch modes

When a coupled-bunch instability is established, although each bunch oscillates at the tune frequency, there can be different modes of oscillation, called multi-bunch modes, depending on how each bunch oscillates with respect to the other bunches [9, 13].

Let us consider M bunches equally spaced around the ring. Each multi-bunch mode is characterized by a bunch-to-bunch phase difference of:

$$\Delta\Phi = m \frac{2\pi}{M} \quad (7)$$

where m is the multi-bunch mode number ($m = 0, 1, \dots, M-1$). Each multi-bunch mode is associated to a characteristic set of frequencies:

$$\omega = p M \omega_0 \pm (m+\nu) \omega_0 \quad (8)$$

where p is an integer number ($-\infty < p < \infty$), ω_0 is the revolution frequency, $\omega_{rf} = M\omega_0$ is the RF frequency (bunch repetition frequency) and ν is the tune. In turns, there are two sidebands at $\pm(m + \nu)\omega_0$ for each multiple of the RF frequency.

In the following sections we shall give some examples of transverse oscillation modes and examine the associated spectra.

1.4.1 Examples of transverse multi-bunch modes and their spectra

1.4.1.1 One single stable bunch

In this example a single bunch of particles is stored in the ring. Figure 5 shows the linear trajectory of the bunch during its run along the ring (red dotted line in the upper picture) and the time domain signal captured by a pickup (lower picture). The location of the pickup is identified in the figure by red triangles placed at longitudinal coordinate 0.

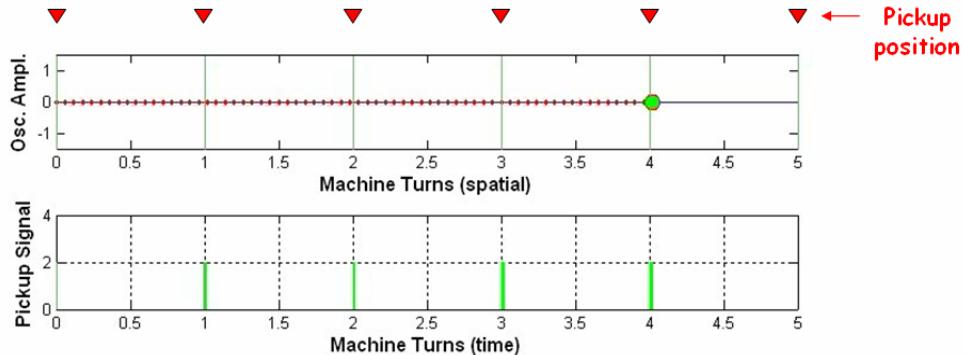


Fig. 5: Motion of a single stable bunch (upper picture) and time domain signal captured by a pickup (lower picture)

Every time the bunch passes through the pickup, an electrical pulse with constant amplitude is generated (green bars). If we think of it as a Dirac impulse, the spectrum of the pickup signal (Fig. 6) is a repetition of frequency lines at a multiple of the revolution frequency $p\omega_0$ for $-\infty < p < \infty$.

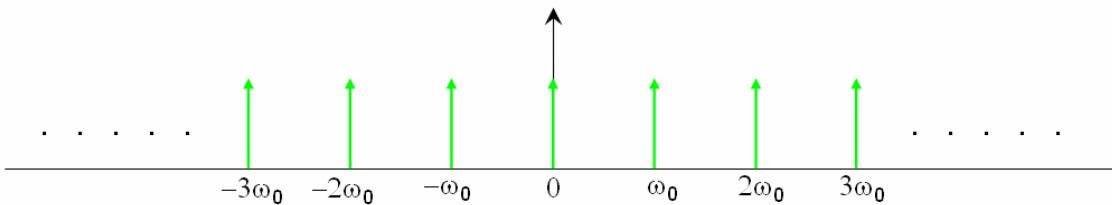


Fig. 6: Spectrum of the signal captured by the pickup with a single stable bunch

1.4.1.2 One single unstable bunch

In this case a single bunch stored in the ring oscillates at the tune frequency $\nu\omega_0$. For simplicity, in this example we consider a tune value lower than unity, $\nu = 0.25$. With a single bunch $M = 1$, only mode number 0 exists. The bunch performs one complete oscillation in four machine turns and the pickup signal is a sequence of pulses modulated in amplitude with frequency $\nu\omega_0$ (Fig. 7).

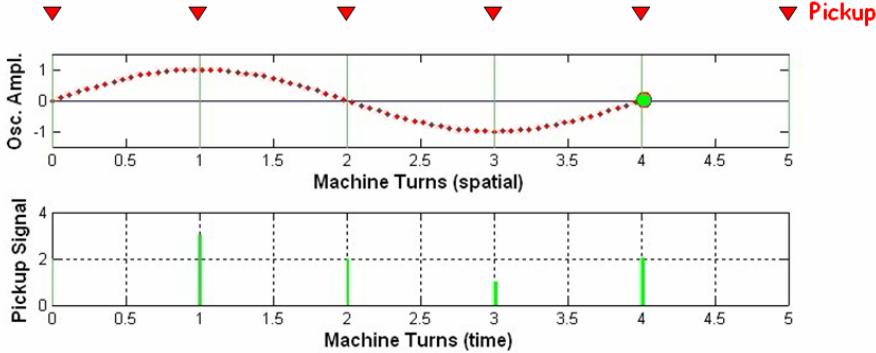


Fig. 7: A single bunch oscillating at the tune frequency and the corresponding pickup signal

In the spectrum two sidebands at $\pm\nu\omega_0$ appear at each of the revolution harmonics (Fig. 8).

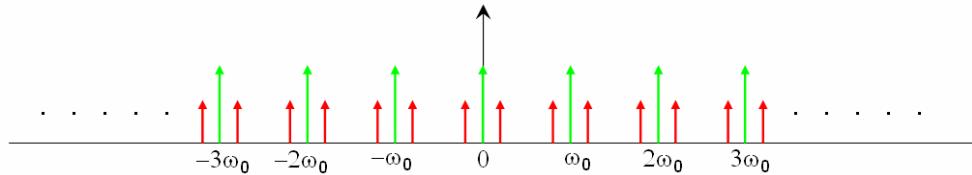


Fig. 8: Spectrum of the pickup signal with a single unstable bunch

1.4.1.3 Ten stable bunches

Let us consider now ten identical, equally spaced, stable bunches ($M = 10$). The pickup signal is a sequence of pulses at the bunch repetition frequency (RF frequency) $\omega_{rf} = 10 \omega_0$.

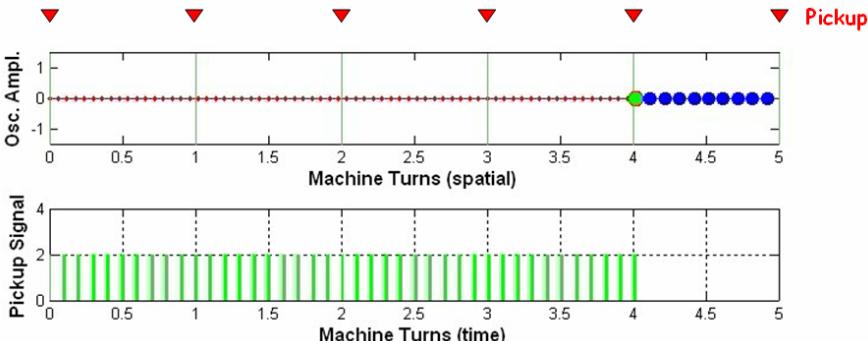


Fig. 9: Ten equally spaced stable bunches

The spectrum is a repetition of frequency lines at multiples of the bunch repetition frequency.

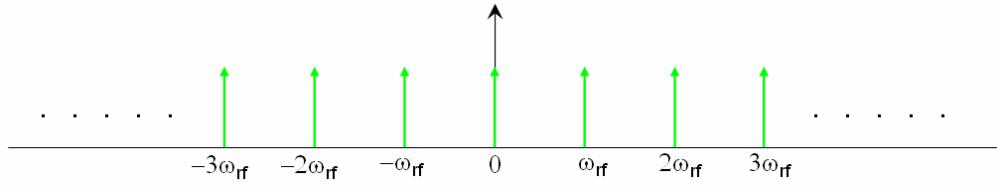


Fig. 10: Spectrum of the pickup signal with ten stable bunches

1.4.1.4 Ten unstable bunches, mode #0

In the case where the bunches oscillate at the tune frequency $\nu\omega_0$ with $\nu = 0.25$, since $M = 10$ there are ten possible modes of oscillation characterized by a phase difference between adjacent bunches of $\Delta\Phi = m \frac{2\pi}{10}$ with $m = 0, 1, \dots, 9$.

If $m = 0$ (mode number 0) then $\Delta\Phi = 0$, namely all the bunches oscillate with the same phase (Fig. 11). The pickup signal is a sequence of pulses at the bunch repetition frequency modulated in amplitude with frequency $\nu\omega_0$.

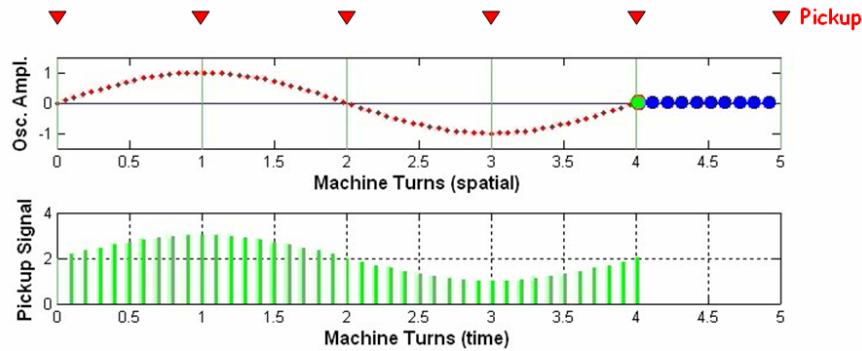


Fig. 11: Ten identical bunches oscillating in phase at the tune frequency. The red dotted line is the trajectory of one of them.

The spectrum is a repetition of frequency lines at multiples of the bunch repetition frequency with sidebands at $\pm\nu\omega_0$: $\omega = p\omega_{rf} \pm \nu\omega_0$ with $-\infty < p < \infty$.

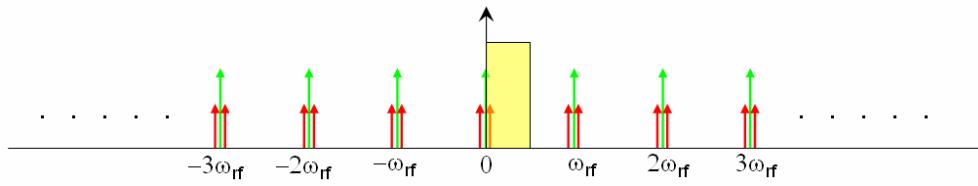


Fig. 12: Spectrum of ten unstable bunches: mode #0

Since the spectrum is periodic and the mode appears twice (lower and upper sideband) in a ω_{rf} frequency span centred on every harmonics of ω_{rf} , we can limit the spectrum analysis to a $0-\omega_{rf}/2$ frequency range. Figure 13 is a zoom of the frequency portion marked in yellow in Fig. 12.

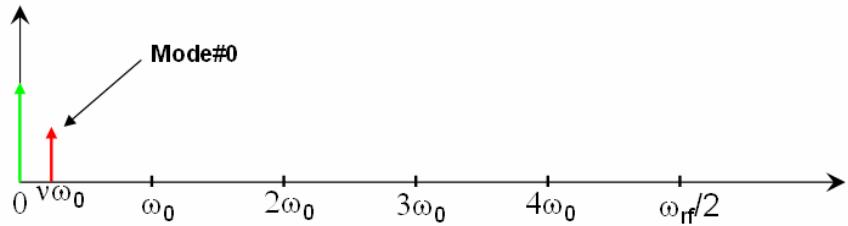


Fig. 13: Spectrum of mode #0 restricted to a $\omega_{rf}/2$ wide frequency range

1.4.1.5 Ten unstable bunches, mode #1

In mode #1, the phase difference of the bunches oscillating at the betatron frequency is $\Delta\Phi = 2\pi/10$ and the spectrum is characterized by frequency lines at $\omega = p\omega_{rf} \pm (v+1)\omega_0$, with $-\infty < p < \infty$.

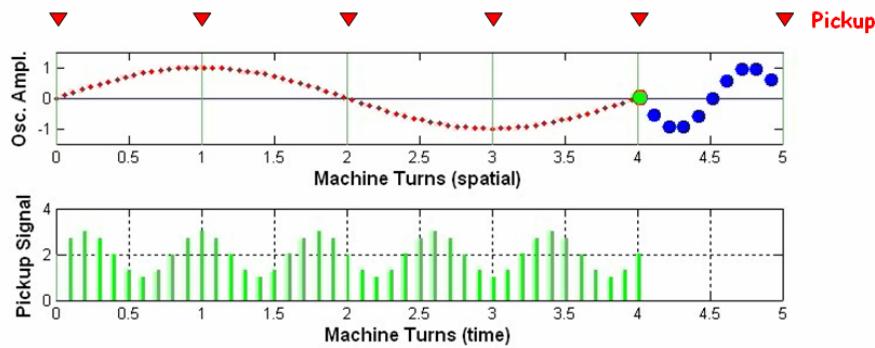


Fig. 14: Ten unstable bunches: mode #1

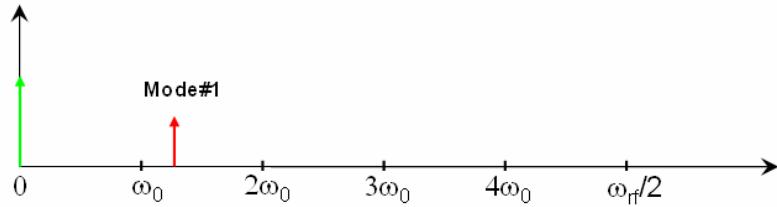


Fig. 15: Spectrum of mode #1

1.4.1.6 Ten unstable bunches, modes #2 to #9

In Figs. 16–23, the motion, time domain signal, and spectrum of multi-bunch modes #2 to #9 are shown.

MULTI-BUNCH FEEDBACK SYSTEMS

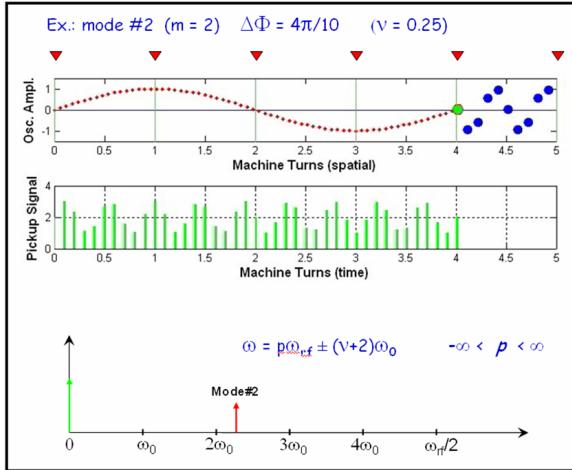


Fig. 16: Mode #2

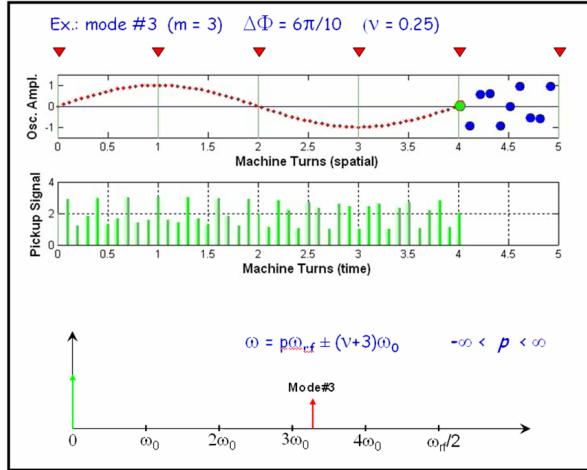


Fig. 17: Mode #3

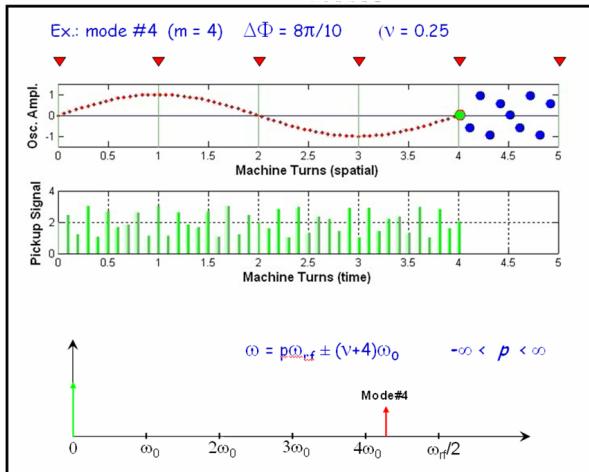


Fig. 18: Mode #4

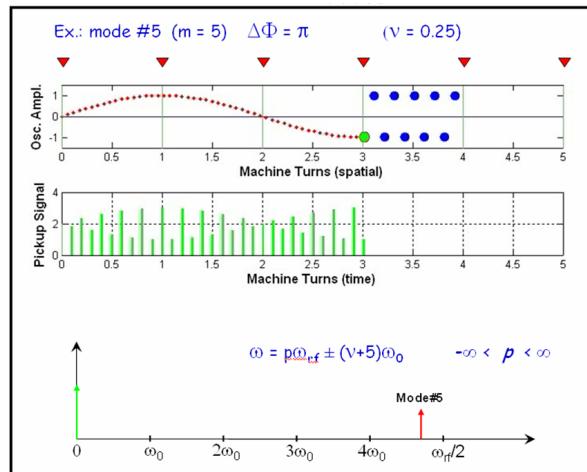


Fig. 19: Mode #5

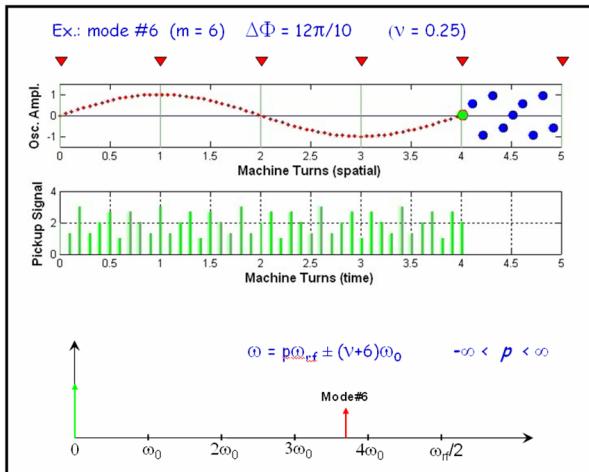


Fig. 20: Mode #6

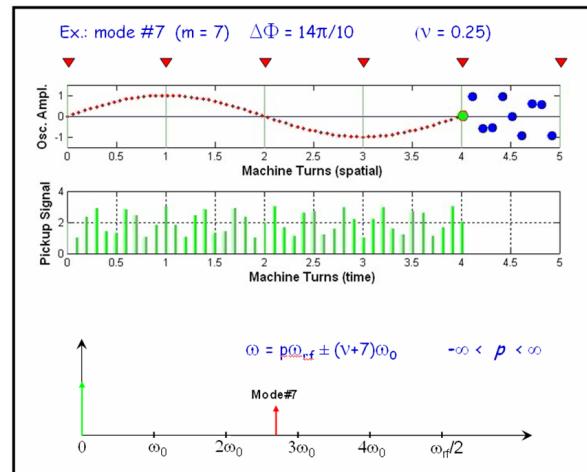


Fig. 21: Mode #7

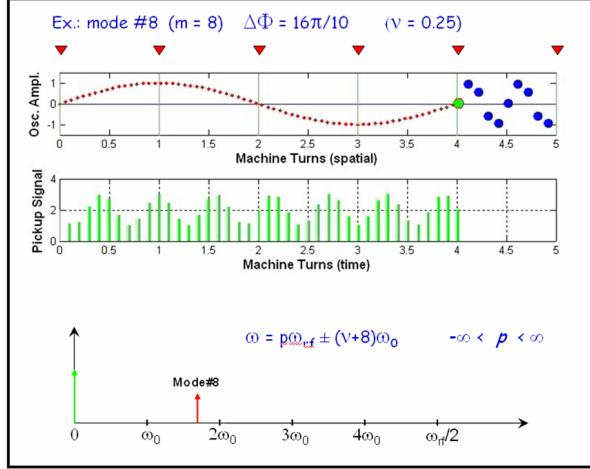


Fig. 22: Mode #8

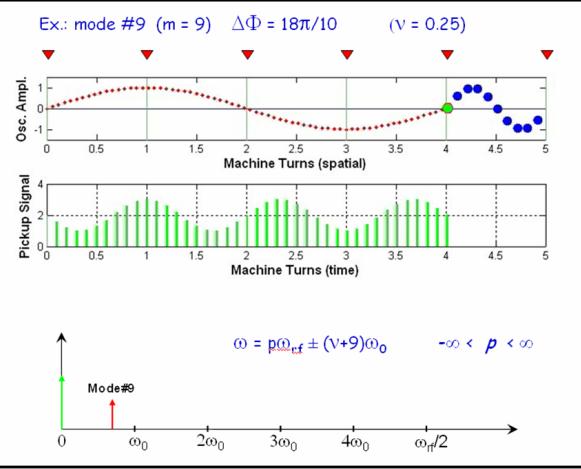


Fig. 23: Mode #9

1.4.2 Uneven filling

If the bunches do not have the same charge, i.e., the buckets are not equally filled (uneven filling), the spectrum has frequency components also at the revolution harmonics (multiples of ω_0). The amplitude of the revolution harmonics depends on the filling pattern of one machine turn.

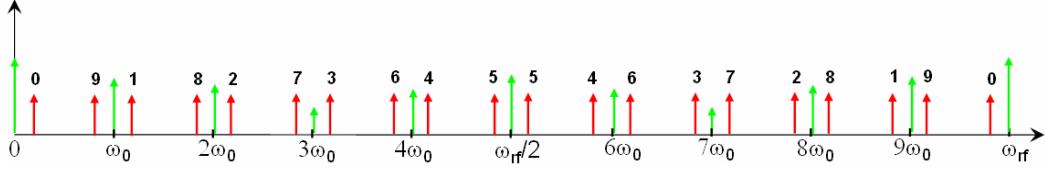
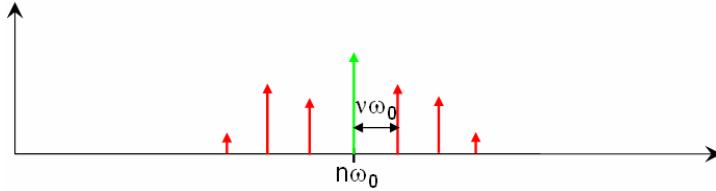


Fig. 24: Spectrum of a beam with uneven filled buckets

1.4.3 Longitudinal multi-bunch modes

Compared with transverse oscillations where multi-bunch instabilities produce amplitude modulation of the stable beam pickup signal, longitudinal multi-bunch modes produce phase modulation. Components at $\pm v\omega_0, \pm 2v\omega_0, \pm 3v\omega_0, \dots$ appear aside the revolution harmonics. Their amplitude depends on the depth of the phase modulation, namely the amplitude of the instability, according to Bessel series expansion (Fig. 25).

Fig. 25: Spectrum of a longitudinally unstable beam with components nearby a revolution harmonic at $\pm v\omega_0, \pm 2v\omega_0, \pm 3v\omega_0, \dots$

1.4.4 Coupled-bunch instability

Starting from the spectral representation of all potential multi-bunch modes, one of them can become unstable if one of its sidebands overlaps, for example, with the frequency response of a high order mode (HOM). The HOM couples with the sideband giving rise to a coupled-bunch instability, with consequent increase of the mode amplitude (Fig. 26).

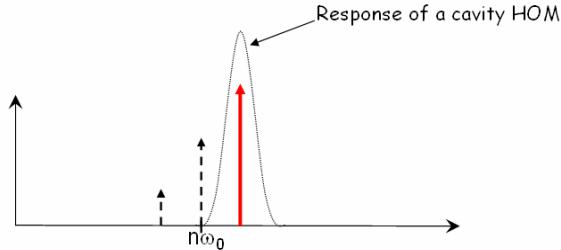


Fig. 26: Coupling of a cavity High Order Mode (HOM) with the sideband corresponding to a given multi-bunch mode

More generally, in order to characterize the interaction of the beam with the machine structures, the ‘impedance model’ is often used. The impedance is the frequency spectrum of the forces generated by wake fields acting back on the beam. It summarizes the electromagnetic effects of the machine components (vacuum chamber, cavities, bellows, BPMs, etc.) on the beam. The shape and amplitude of the machine impedance give us information about which multi-bunch modes could be excited and the strength of the excitation (growth time constant).

1.4.5 Effects of coupled-bunch instabilities

Besides the risk of beam loss or the limitation of the maximum stored current, the onset of instabilities produces an increase of the transverse beam size (also in the case of longitudinal instabilities because of dispersion) and thus of the effective emittance. This is detrimental for the beam quality since it lowers the brightness of the produced photon beams in synchrotron light sources and the luminosity in circular colliders. Besides these negative effects, however, the presence of multi-bunch instabilities is often associated with an increase of the beam lifetime. Saturation effects due to Landau damping, in fact, produce dilution of particles inside the bunches with consequent decrease of Touschek scattering.

1.4.6 Real examples of multi-bunch modes and their measurement

Let us consider two real examples of transverse and longitudinal instabilities. We refer to the Elettra synchrotron storage ring. The RF frequency is 499.654 MHz; having a harmonic number of 432, the revolution frequency is 1.157 MHz. The horizontal tune ν_{hor} is 12.30 (fractional tune frequency = 345 kHz), the vertical tune ν_{ver} is 8.17 (fractional tune frequency = 200 kHz) and the longitudinal tune ν_{long} is 0.0076 (8.8 kHz). Keeping in mind Eq. (8) ($\omega = p M \omega_0 \pm (m+i) \omega_0$) let us analyse two beam spectra measured with a spectrum analyser connected to a strip line pickup.

Example 1 In this example we analyse a spectral line at 512.185 MHz. It is a lower sideband of the second harmonic of the RF frequency, 200 kHz apart from the 443rd revolution harmonic at 512.355 MHz, denoting that it is a vertical instability. From Eq. (8) it can be easily calculated that it corresponds to the vertical multi-bunch mode #413.

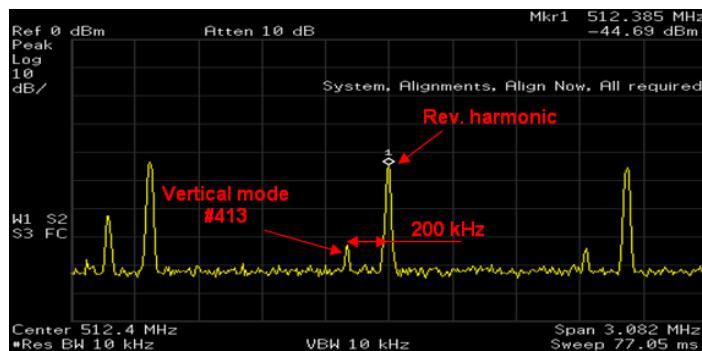


Fig. 27: Example of beam spectrum in the presence of vertical instabilities measured by a spectrum analyser

Example 2 In this case we analyse a spectral line at 604.914 MHz. It is an upper sideband of the RF frequency, 8.8 kHz apart from the 523rd revolution harmonic, denoting a longitudinal instability. From Eq. (8) we can calculate that it corresponds to the longitudinal multi-bunch mode #91.

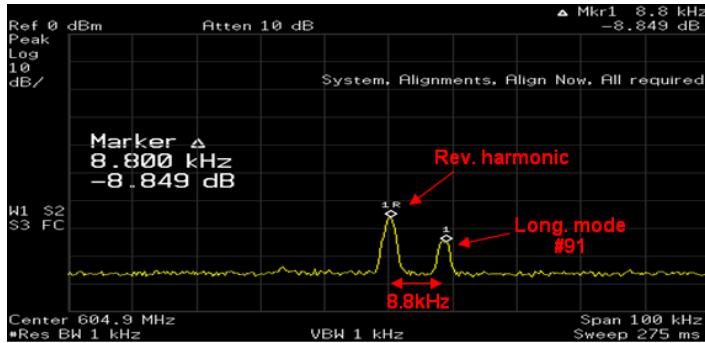


Fig. 28: Example of beam spectrum in the presence of a longitudinal instability measured by a spectrum analyser

2 Multi-bunch feedback systems

2.1 Introduction

As already seen in Section 1.3.3, a multi-bunch feedback system detects multi-bunch instabilities using one or more Beam Position Monitors (BPMs) and acts back on the beam through an electromagnetic actuator called ‘kicker’. A simplified block diagram of a feedback system is depicted in Fig. 29.

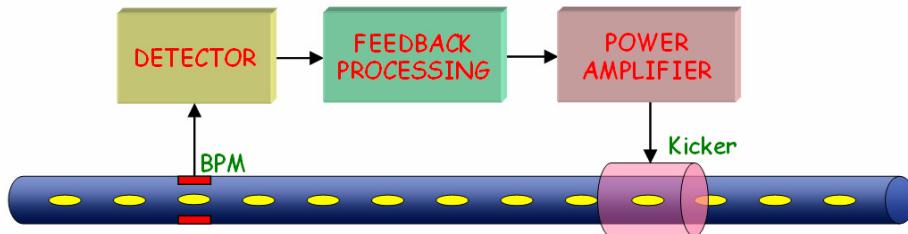


Fig. 29: Block diagram of a multi-bunch feedback system

BPMs and detectors measure the oscillation of the bunches, the processing unit generates the correction signal which is amplified by the power amplifier and sent to the kicker.

2.2 Types of feedback systems

Depending on how a feedback system detects and controls instabilities, there are two main feedback categories: ‘mode-by-mode’ and ‘bunch-by-bunch’ feedbacks. The former are often referred to as ‘frequency-domain’, the latter as ‘time-domain’ feedbacks [8]. Moreover, the feedback implementation can be analog or digital, depending on whether the signal is handled in the analogic domain or sampled and processed digitally. Despite the fact that most of the latest feedbacks are digital bunch-by-bunch systems, analogic or mode-by-mode implementations are still in use in a number of accelerators.

In the following sections the basic principles of the mentioned types of feedbacks will be illustrated; they are part of the history of multi-bunch feedback systems over a period of at least thirty years.

2.2.1 Mode-by-mode feedback

A mode-by-mode (frequency-domain) feedback acts separately on each of the controlled unstable modes [14]. A simplified block diagram is illustrated in Fig. 30.

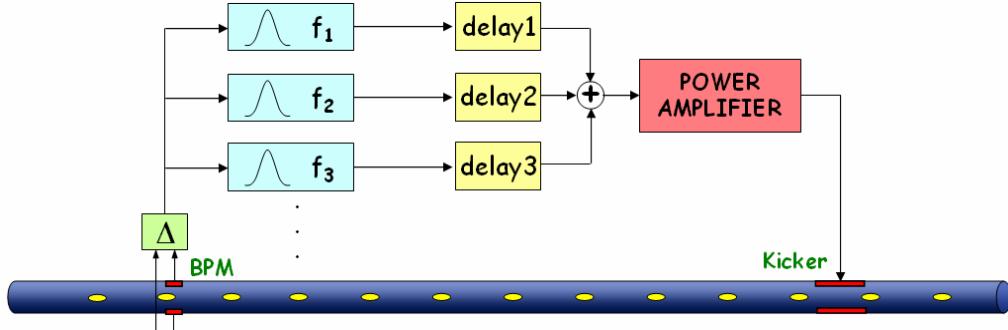


Fig. 30: Block diagram of a mode-by-mode feedback

An appropriate electronics generates the position error signal by combining the signals of the BPM buttons. A number of processing chains, each dedicated to one of the controlled modes, work in parallel. They are composed of a narrow band-pass filter that selects the proper frequency and by an adjustable delay line to phase shift the signal in order to produce a negative feedback. The channels are then combined, amplified, and sent to the kicker.

2.2.2 Bunch-by-bunch feedback

A bunch-by-bunch (time-domain) feedback individually steers each bunch by applying ‘small’ electromagnetic kicks every time the bunch passes through the kicker. The result is a damped oscillation lasting several turns. The basic blocks composing a bunch-by-bunch feedback are shown in Fig. 31.

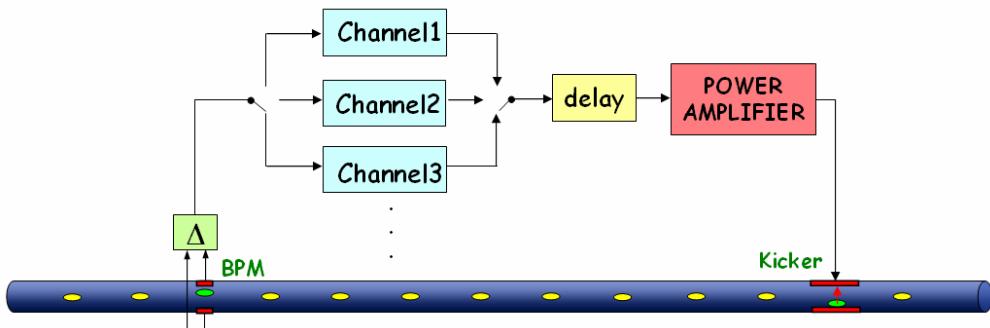


Fig. 31: Block diagram of a bunch-by-bunch feedback system

The correction signal of a given bunch is computed based on the motion of the same bunch. There are as many processing channels as the number of bunches. A demultiplexer provides each channel with the samples of the corresponding bunch, while a multiplexer recombines the samples at the end of the channels. Following the principle explained in Section 1.3.3, the main task of each channel is to properly phase shift the sampled position signal. A delay line assures that each bunch is kicked with its own correction samples.

Every bunch is measured and corrected at every machine turn but, owing to the delay of the feedback chain, the correction kick corresponding to a given measurement is applied to the bunch one or more turns later.

An important principle behind bunch-by-bunch feedbacks is that it is possible to stabilize all coupled-bunch modes by damping the oscillation of each bunch.

2.2.3 Analog bunch-by-bunch feedback

In the following sections we shall see some real analogic implementations of a transverse bunch-by-bunch feedback starting from a basic very simple architecture up to a system which has been deployed in many accelerators.

2.2.3.1 Feedback using one BPM

In a bunch-by-bunch feedback the correction signal applied to a given bunch must be proportional to the derivative of the bunch oscillation at the kicker. The correction signal is therefore a sampled sinusoid shifted by $\pi/2$ with respect to the oscillation of the bunch when it passes through the kicker. Starting from this principle, a very simple way to obtain such a phase shift is to have the BPM and kicker placed in points of the ring with the appropriate betatron phase difference and to use the detected signal to directly feed the kicker through an amplifier. The system is shown in Fig. 32.

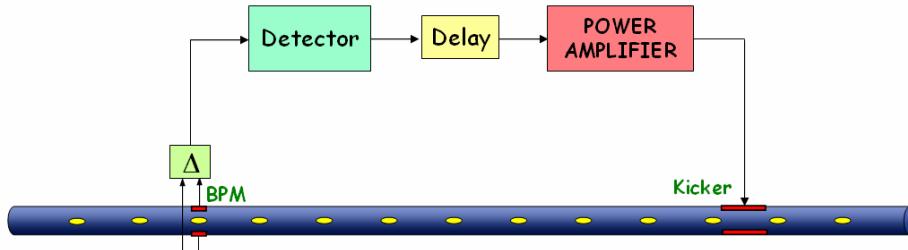


Fig. 32: Block diagram of a simple analog bunch-by-bunch transverse feedback

The detector down converts the high frequency (typically a multiple of the bunch frequency f_{rf}) BPM signal into base-band (range $0 - f_{rf}/2$). The delay line assures that the signal of a given bunch passing through the feedback chain arrives at the kicker when, after one machine turn, the same bunch passes through it.

2.2.3.2 Feedback using two BPMs

By using two BPMs, provided that they are separated by about $\pi/2$ in betatron phase, it is possible to place them in any ring position with respect to the kicker (Fig. 33).

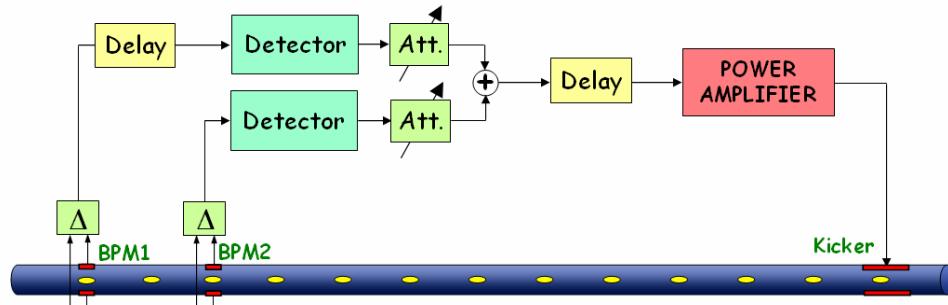


Fig. 33: Analog bunch-by-bunch transverse feedback using two BPMs

As shown in Fig. 34, if we opportunely combine the two detected signals using variable attenuators, it is possible to vary the phase of the resulting bunch correction signal. The attenuators have to be adjusted so that the bunch oscillation and the corresponding kick signal are in quadrature.

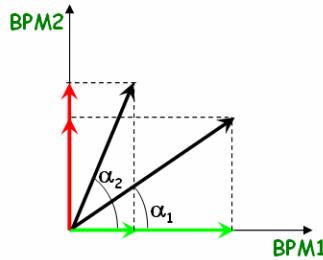


Fig. 34: If the two BPM bunch signals are in quadrature, the phase of the sum signal can be varied by adjusting the variable attenuators

2.2.3.3 Suppression of revolution harmonics

The revolution harmonics (frequencies at multiples of ω_0) are useless components that can be removed from the correction signal in order not to saturate the RF amplifier. A special module implementing notch filters at the harmonics of ω_0 can be placed before the amplifier to eliminate these components. This operation is also called ‘stable beam’ rejection. We shall see later a number of possible implementations of this module that are also used in digital feedback systems.

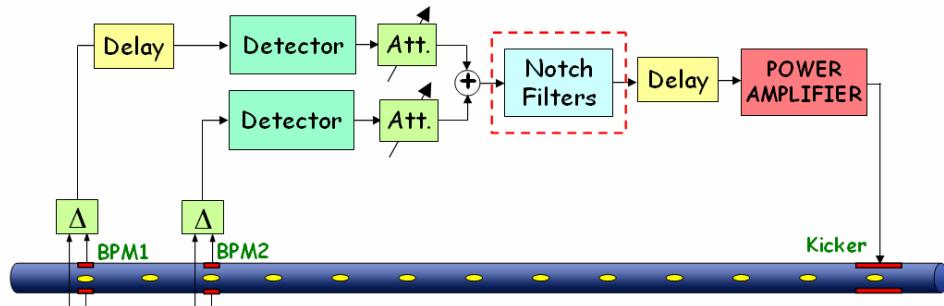


Fig. 35: Analog bunch-by-bunch transverse feedback with suppression of revolution harmonics

Architectures similar to the above examples have been used to build the ALS [15], Bessy II [16], PLS [17] and ANKA [18] transverse multi-bunch feedback systems.

2.2.4 Digital bunch-by-bunch feedback

Commercially available fast digital electronics (ADCs, DACs, DSPs, FPGAs, etc.) allow one nowadays to build digital feedback systems where the signal of the bunch positions is digitized, digitally processed to calculate the corrections, and re-converted to analog. The block diagram of a generic digital feedback is illustrated in Fig. 36. Thanks to the capabilities offered by digital signal processing, a digital feedback usually makes use of only one BPM and a kicker mounted in any ring position.

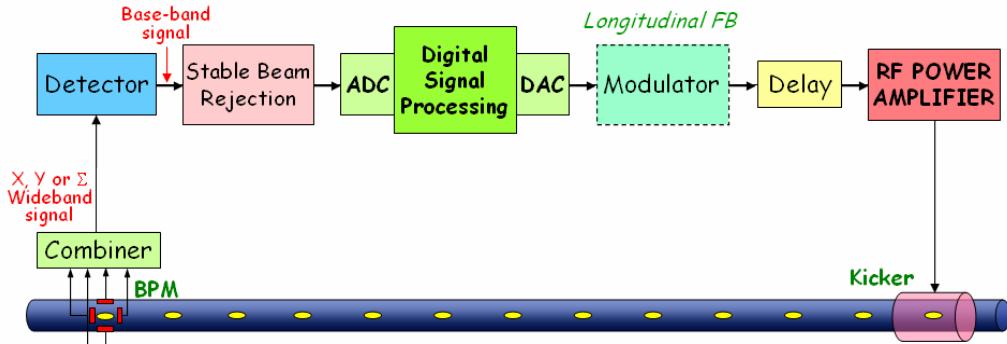


Fig. 36: Block diagram of a digital bunch-by-bunch feedback system

Starting from the BPM button signals, the combiner generates the wideband horizontal (X), vertical (Y) or sum (Σ) signal, which is then demodulated to base-band by the detector (also called ‘RF front-end’). Some feedback implementations feature a stable beam rejection module that removes useless stable beam components from the signal, which is eventually digitized, processed, and re-converted to analog by the digital processor. While in transverse feedback systems amplifier and kicker operate in base-band, longitudinal feedbacks require a modulator that translates the correction signal to the kicker operation frequency. The delay line adjusts the timing of the signal to match the bunch arrival time.

2.2.5 Digital vs. analog feedbacks

The following list summarizes the main advantages of digital feedback systems:

- **reproducibility**: when the signal is digitized it is not subject to temperature/environment changes or ageing;
- **programmability**: the implementation of processing functionalities is usually made using DSPs or FPGAs, which are programmable via software/firmware;
- **performance**: digital controllers feature superior processing capabilities with the possibility to implement sophisticated control algorithms not feasible in analog;
- **additional features**: possibility to combine basic control algorithms and additional useful features like signal conditioning, saturation control, down sampling, etc.;
- **implementation of diagnostic tools**, used for both feedback commissioning and machine physics studies (see Section 6);
- **easier and more efficient integration** of the feedback in the accelerator control system, important for feedback set-up and tuning, fast data acquisition, easy and automated operations, etc.

Among the disadvantages are the higher delay of the feedback chain (due to ADC, digital processing and DAC) with respect to analog feedbacks, although with the use of FPGAs this delay is reduced to acceptable values.

3 Components of feedback systems

In the following sections a more detailed description of components usually employed in (digital) feedback systems will be given, with practical issues, examples, and real implementations.

3.1 BPM and combiner

The four signals from a standard four-button BPM can be opportunely combined to obtain the wideband X, Y and Σ signals used, respectively, by the horizontal, vertical, and longitudinal feedback systems. Figure 37 shows an example of how this can be made.

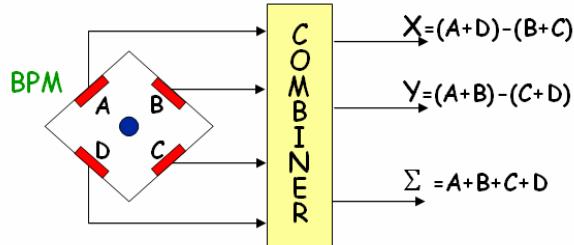


Fig. 37: BPM and combiner are used to provide X, Y and Σ wideband signals

Any $f_{rf}/2$ portion of the beam spectrum contains the information of all potential multibunch modes and can be used to detect instabilities and measure their amplitude. Usually BPM and combiner work in a frequency range around a multiple of f_{rf} (Fig. 38), where the amplitude of the overall frequency response of BPM and cables is maximum. Moreover, a higher f_{rf} harmonic is preferred for the longitudinal feedback because of the better sensitivity of the phase detection system.

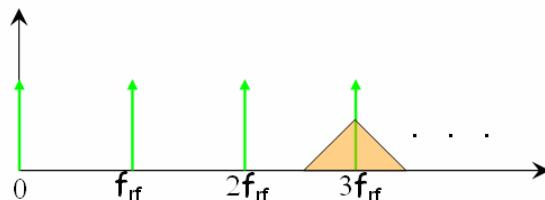


Fig. 38: Example of wideband signal centred on the third harmonic of the RF frequency

3.2 Detector (RF front-end)

3.2.1 Transverse case

The detector translates the wideband signal into base-band ($0-f_{rf}/2$ range): the operation is an amplitude demodulation (Fig. 39).

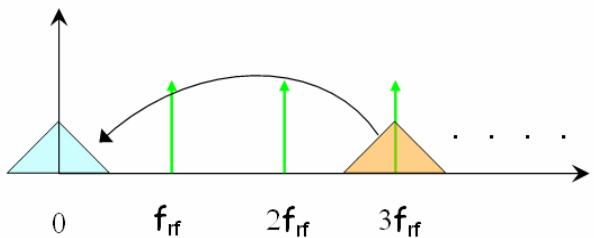


Fig. 39: The amplitude demodulation translates the wideband signal to base-band

A possible implementation of amplitude demodulation, widely used in telecommunication technology, is the ‘heterodyne’ technique illustrated in Fig. 40. The wideband signal is first band-pass filtered to select the desired frequency range and then mixed with the ‘local oscillator’ signal, which must be synchronous with the former. The local oscillator signal can be derived from the RF by multiplying its frequency by an integer number corresponding to the chosen harmonic of f_{rf} . The resulting signal is eventually low-pass filtered in the $0-f_{rf}/2$ frequency range.

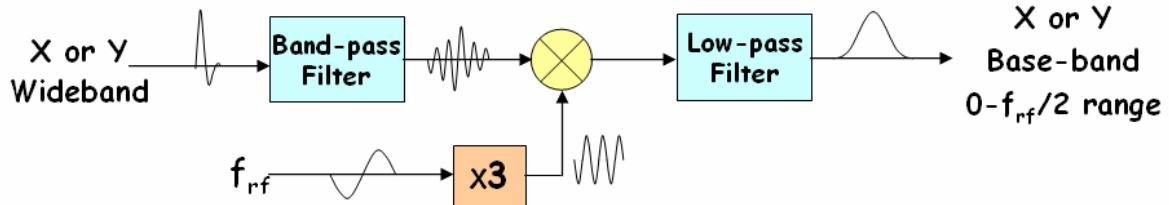


Fig. 40: Heterodyne amplitude demodulation using the third harmonic of the RF frequency

3.2.2 Longitudinal case

The wideband sum signal (Σ) contains only information about the phase (longitudinal position) of the bunches, since the sum of the four bunch signals has almost constant amplitude.

The base-band phase error signal ($0-f_{rf}/2$ range) can be generated by phase demodulation, which can be carried out with the same heterodyne technique used for amplitude demodulation but using a ‘local oscillator’ in quadrature with the wideband signal, namely shifted in phase by $\pi/2$ (Fig. 41).

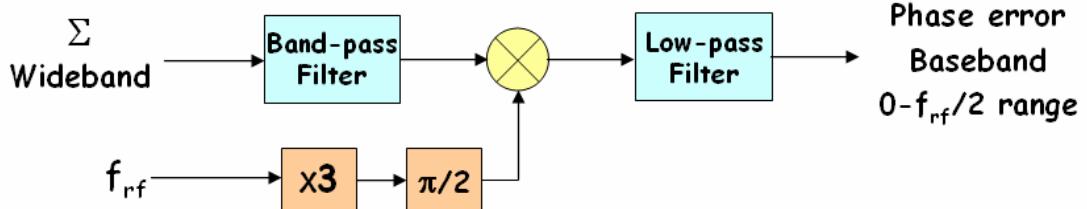


Fig. 41: Phase demodulation using the heterodyne technique

3.2.3 Amplitude and phase demodulation

The following are simple mathematic demonstrations that can help in understanding heterodyne amplitude and phase demodulations.

Amplitude demodulation

Let us consider a sinusoidal carrier $\sin(3\omega_{rf} t)$ modulated in amplitude by a signal $A(t)$. If we multiply it by the local oscillator signal $\sin(3\omega_{rf} t)$ (mixing operation) we obtain two frequency components. The component at higher frequency can be filtered out by a low-pass filter leaving a signal proportional to the modulating signal $A(t)$:

$$A(t) \sin(3\omega_{rf} t) \sin(3\omega_{rf} t) \propto A(t) (\cos(0) - \cos(6\omega_{rf} t)) \approx A(t). \quad (9)$$

Phase demodulation

In this case the sinusoidal carrier is phase modulated by a signal $\varphi(t)$. If the latter is small, by multiplying the modulated signal by a quadrature-phase local oscillator signal $\cos(3\omega_{rf} t)$, after low-pass filtering we obtain the modulating signal:

$$\sin(3\omega_{rf} t + \varphi(t)) \cos(3\omega_{rf} t) \propto \sin(6\omega_{rf} t + \varphi(t)) + \sin(\varphi(t)) \approx \varphi(t). \quad (10)$$

3.2.4 Time domain considerations

The base-band signal can be seen as a sequence of ‘pulses’ each with amplitude proportional to the position error (X , Y or Φ) and to the charge of the corresponding bunch. By sampling this signal with an A/D converter synchronous with the bunch frequency, one can measure X , Y or Φ (Fig. 42).

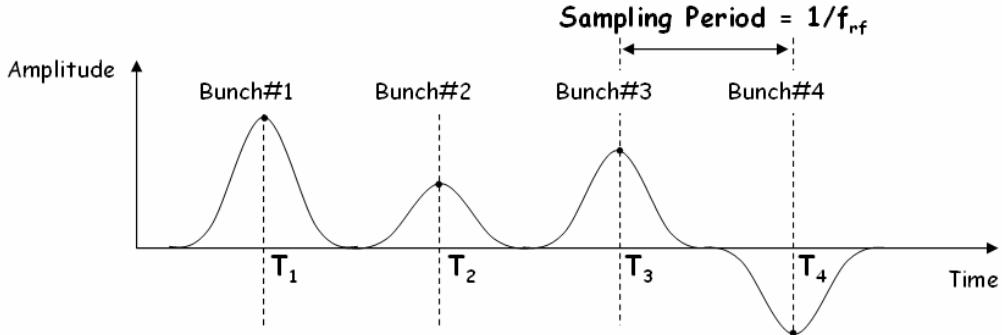


Fig. 42: Detector output signal

Note that the multi-bunch-mode number $M/2$, which is the one with the highest frequency (close to $f_{rf}/2$) in base-band, is characterized by a sequence of pulses with almost the same amplitude but alternating signs.

The design of the detector band-pass and low-pass filters affects the shape of the pulses that should have maximum flatness on the top (to reduce clock jitter noise) and minimum overlap with adjacent pulses (to reduce cross-talk between bunches) (Fig. 43). To achieve that, Bessel filters with linear phase response or a special filter called ‘comb generator’ [19] can be used.

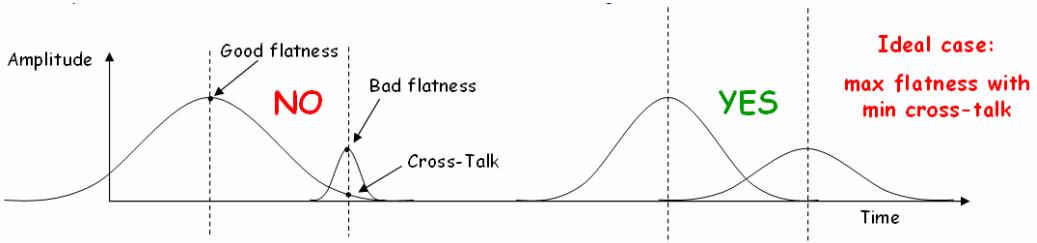


Fig. 43: Examples of badly and well filtered detector signals

3.3 Rejection of stable beam signal

The amplitude of the pulses in the base-band signal can have a constant offset called ‘stable beam signal’. In the transverse planes this offset can be due to an off-centre beam in the BPM or to unbalanced BPM electrodes or cables. In the longitudinal plane it can be generated by a mismatch of the local oscillator phase with respect to the bunch phases, which is difficult to avoid when the bunches have different synchronous phases due to beam loading.

In the frequency domain, the presence of a stable beam signal carries non-zero revolution harmonics.

The stable beam signal is useless for the feedback since it does not contain information about multi-bunch modes. It is therefore preferable to reduce it in order to avoid saturation of the ADC, DAC, or amplifier. A number of techniques have been adopted to reduce this signal; we shall mention three examples.

3.3.1 Balancing of BPM buttons

In the case of a transverse feedback, variable attenuators placed on the BPM electrodes can be used to equalize the amplitude of the signals. In this way closed-orbit offsets or unbalanced signals can be compensated.

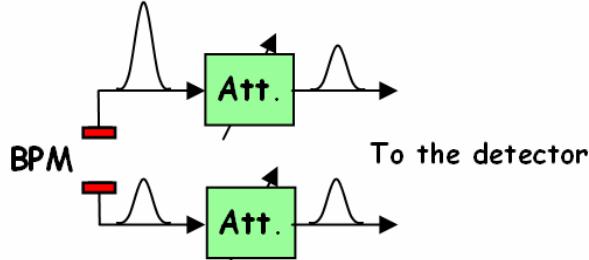


Fig. 44: Example of BPM signal balancing

3.3.2 Comb filter

A filter performing stable beam rejection through the reduction of the revolution harmonics can be simply implemented in the analog domain by means of delay lines and combiners, as shown in Fig. 45. Its frequency response, depicted in the same figure, shows a series of notches at multiples of ω_0 able to suppress all the revolution harmonics (DC included) from the detector output signal [18].

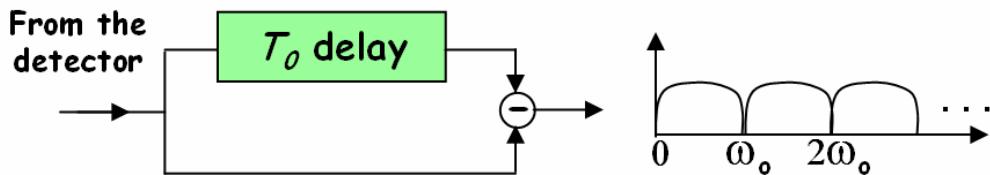


Fig. 45: Stable beam rejection using a comb filter: block diagram of the filter and its frequency response

3.3.3 Digital DC rejection

The stable beam signal can be eliminated by removing the DC component from the turn-by-turn signal of every bunch. This can be done digitally (Fig. 46).

The detector output signal is sampled by an ADC clocked at f_r . The turn-by-turn digital signal of each bunch is integrated by a digital processing unit (e.g. a FPGA), recombined with the other bunch signals, converted to analog and subtracted from the original signal. In this way the constant component of each bunch is eliminated [20].

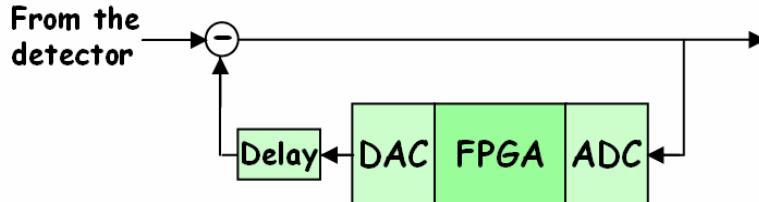


Fig. 46: Block diagram of a digital DC rejection system

3.4 Digital processor

The A/D converter samples and digitizes the detector signal at the bunch repetition frequency; each sample corresponds to the position error (X , Y or Φ) of a given bunch. Precise synchronization of the sampling clock with the bunch signal must be provided.

The digital samples are then de-multiplexed into M channels, M being the number of bunches in the ring. In each channel the turn-by-turn samples of a given bunch are processed by a dedicated digital filter to calculate the correction samples. The basic processing consists of DC component suppression (if not completely accomplished by the external stable beam rejection) and phase shift of the signal at the betatron/synchrotron frequency (see Section 1.3.3).

The correction sample streams are then recombined and converted to analog by the D/A converter (Fig. 47).

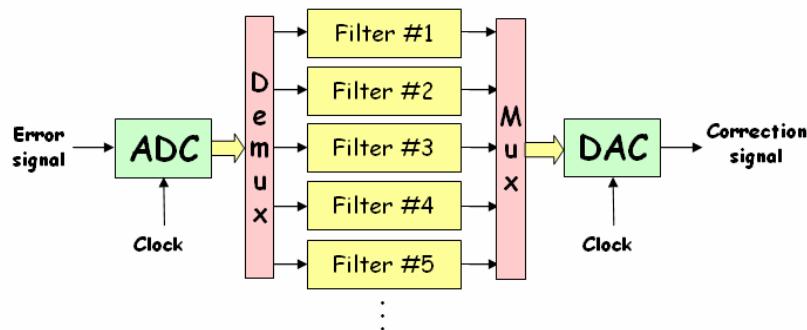


Fig. 47: Block diagram of the digital processor of a bunch-by-bunch feedback

3.4.1 Digital processor implementation

ADC: existing multibunch feedback systems usually employ 8-bit ADCs clocked at up to 500 Msample/s; some implementations use a number of ADCs with higher resolution (e.g. 14 bits) and lower clock rate working in parallel. ADCs with enhanced resolution have the advantage of a lower quantization noise (crucial for low-emittance machines) and a higher dynamic range, which allow one, in some cases, to avoid the external stable beam rejection.

DAC: the DACs usually employed convert samples at up to 500 Msample/s and 14-bit resolution.

Digital Processing: the feedback processing can be performed by discrete digital electronics (obsolete technology), DSPs (Digital Signal Processors), or FPGAs (Field Programmable Gate Arrays). The table in Fig. 48 summarizes advantages and disadvantages of the last two options. Although there are still some digital multibunch feedback systems in the world employing DSPs, the trend is towards solutions using FPGAs.

	Pros	Cons
DSP	<ul style="list-style-type: none"> ➢ Easy programming ➢ Flexible 	<ul style="list-style-type: none"> ➢ Difficult HW integration ➢ Latency ➢ Sequential program execution ➢ A number of DSPs are necessary
FPGA	<ul style="list-style-type: none"> ➢ Fast (only one FPGA is necessary) ➢ Parallel processing ➢ Low latency 	<ul style="list-style-type: none"> ➢ Trickier programming ➢ Less flexible

Fig. 48: Summary of advantages and disadvantages of DSPs and FPGAs

3.4.2 Examples of digital processors

The following are examples of digital processor implementation employed in a number of accelerators worldwide. We would like to point out that this list is probably incomplete and not up-to-date.

PETRA transverse and longitudinal feedbacks [21]: this is one of the first digital implementations of a bunch-by-bunch feedback. The digital processor is composed of an ADC, a digital processing electronics made of discrete components (adders, multipliers, shift registers, etc.) implementing a FIR filter, and a DAC.

ALS/PEP-II/DAΦNE longitudinal feedback (also adopted at SPEAR, Bessy II and PLS) [22, 16, 17]: the A/D and D/A conversions are performed by VXI boards, while the feedback processing is made by DSP boards hosted in a number of VME crates.

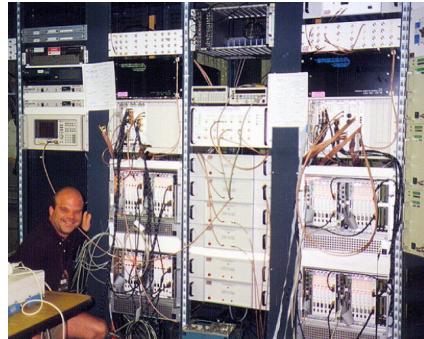


Fig. 49: The ALS/PEP-II/DAΦNE longitudinal feedback system

PEP-II transverse feedback [23, 24]: the digital part, made of two ADCs, a FPGA, and a DAC, features a digital delay and integrated diagnostics tools, while the rest of the signal processing is made analogically.

KEKB transverse and longitudinal feedbacks [25, 26]: the digital processing unit, made of discrete digital electronics and banks of memories, performs a two-tap FIR filter featuring stable beam rejection, phase shift and delay.



Fig. 50: The digital processing board of the KEKB feedback systems

Elettra/SLS transverse and longitudinal feedbacks [27, 28]: the digital processing unit is made of a VME crate equipped with one ADC, one DAC, and six commercial DSP boards with four microprocessors each. Further developments at SLS allowed the direct connection of ADC and DAC boards, exploiting the on-board FPGAs to perform the feedback processing [29].



Fig. 51: The Elettra horizontal, vertical, and longitudinal feedback processors

CESR transverse and longitudinal feedbacks [30]: they employ VME digital processing boards equipped with ADC, DAC, FIFOs and PLDs.

HERA-p longitudinal feedback [31]: it is made of a processing chain with two ADCs (for I and Q components), a FPGA, and two DACs.

HLS tranverse feedback [32]: the digital processor consists of two ADCs, one FPGA, and two DACs.

SPring-8 transverse feedback (also adopted at TLS, KEK-Photon-Factory and Soleil) [33, 34, 35, 36]: it employs a fast analog de-multiplexer that distributes analog samples to a number of slower ADC–FPGA channels. The correction samples are converted to analog by one DAC.



Fig. 52: The processing unit of the SPring-8 multibunch feedback system

ESRF transverse/longitudinal and Diamond transverse feedbacks [20, 37]: they adopt a commercial product called ‘Libera Bunch-by-Bunch’ (by Instrumentation Technologies [38]), which features four ADCs sampling the same analog signal opportunely delayed, one FPGA, and one DAC.



Fig. 53: ‘Libera Bunch-by-Bunch’, by Instrumentation Technologies

DAΦNE transverse and KEK-Photon-Factory longitudinal feedbacks [39]: these feedback systems rely on a commercial product called ‘iGp’ (by Dimtel [40]), featuring an ADC–FPGA–DAC chain.



Fig. 54: ‘iGp’, by Dimtel

3.5 Amplifier and kicker

The kicker is the feedback actuator. It generates a transverse/longitudinal electromagnetic field that steers the bunches with small ‘kicks’ as they pass through the kicker. The overall effect is the damping of the betatron/synchrotron oscillations. The power amplifier provides the kicker with the necessary RF power by amplifying the signal from the DAC (or from the modulator in the case of a longitudinal feedback).

An important parameter that measures the efficiency of the kicker is the ‘shunt impedance’ R , defined as the ratio between the squared voltage seen by the bunch and twice the power at the kicker input:

$$R = \frac{V^2}{2P_{\text{IN}}} . \quad (11)$$

Since the shunt impedance depends on the frequency of the input signal, it is normally represented as a function of frequency.

We shall use this definition later in this lecture when calculating the power necessary to damp coupled-bunch oscillations.

Amplifier and kicker need a bandwidth of at least $f_{rf}/2$. In a transverse feedback, for example, frequencies go from $\sim\text{DC}$ (all kicks of the same sign) to $\sim f_{rf}/2$ (kicks of alternating signs). The bandwidth of amplifier and kicker must be sufficient to correct each bunch with the appropriate kick without affecting the neighbouring bunches. The amplifier and kicker design has to maximize the kick strength while minimizing the cross-talk between corrections given to adjacent bunches.

Another issue is the group delay that should be constant in the working frequency range, namely the phase response should be linear. If this is not the case, the feedback efficiency is reduced when damping some multibunch modes and, under given conditions, the feedback can even become positive. Figure 55 shows an example of amplitude and phase response of an amplifier. Where the phase is 180° , the multibunch mode corresponding to that frequency is excited.

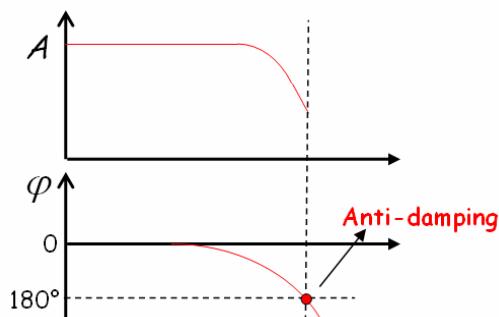


Fig. 55: Example of amplifier amplitude and phase response where, for a given frequency, the feedback performs anti-damping

3.5.1 Transverse feedback

A stripline geometry is usually employed for the transverse feedback kicker. Amplifier and kicker work in base-band, namely in the frequency range from \sim DC to $\sim f_{rf}/2$. Figure 56 shows an example of a transverse feedback back-end adopting two power amplifiers feeding the downstream ports of two striplines in counter phase. The upstream ports are connected to 50Ω power loads. Power low-pass filters can be optionally included to protect the amplifiers from peak voltages picked up by the kicker from the beam.

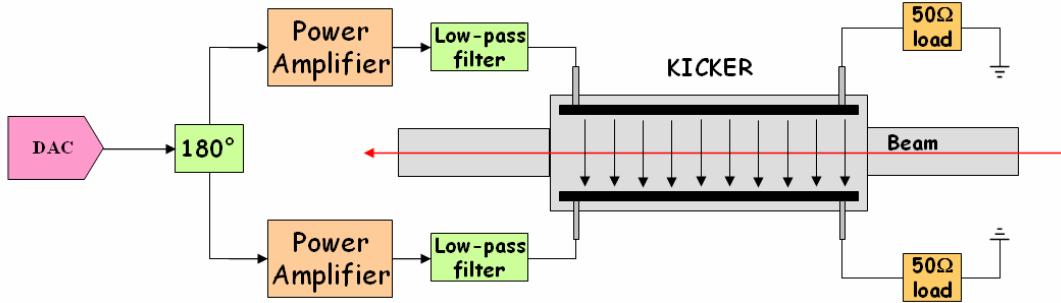


Fig. 56: Example of transverse feedback back-end using two power amplifiers to feed the kicker

The design of the Elettra/SLS transverse kickers [41] is depicted in Fig. 57 together with a picture of the kickers installed on the Elettra vacuum chamber, while the shunt impedance of the same kickers as a function of frequency is plotted in Fig. 58.

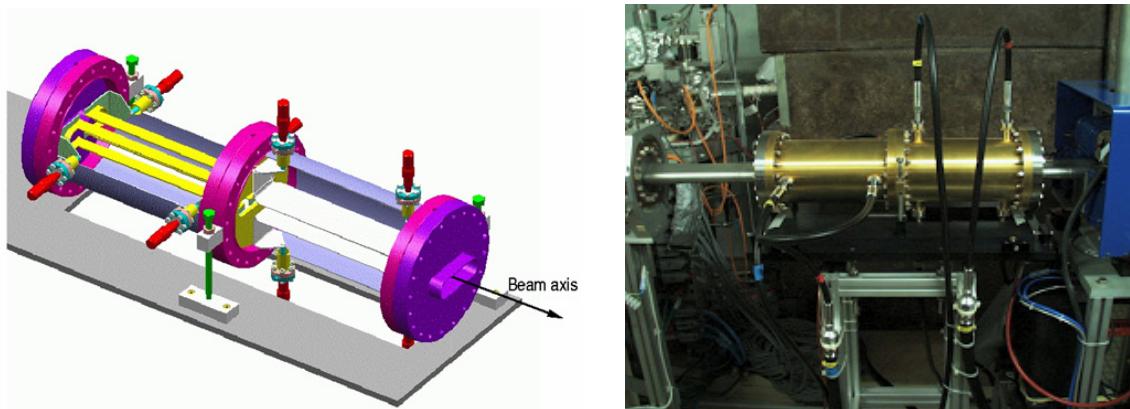


Fig. 57: The Elettra/SLS horizontal and vertical kickers

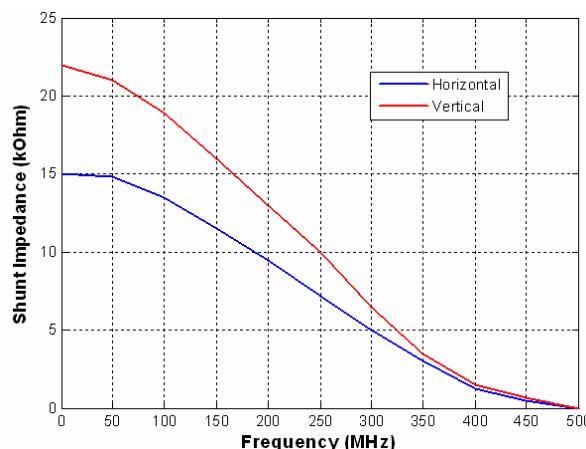


Fig. 58: Shunt impedance of the Elettra/SLS transverse kickers

3.5.2 Longitudinal feedback

For the longitudinal kicker, a ‘cavity-like’ structure is usually preferred because of the higher shunt impedance and smaller size. As in the case of the transverse kicker, the operating frequency range of the longitudinal kicker is $f_{rf}/2$ wide, but it is usually placed on one side of a multiple of f_{rf} (e.g. from $3f_{rf}$ to $3f_{rf}+f_{rf}/2$).

The base-band signal from the DAC has to be translated in frequency or, in other words, ‘modulated’ in order to overlap with the kicker shunt impedance (Fig. 59). A SSB (Single Side-Band) amplitude modulation or other techniques like for example QPSK (Quadrature Phase Shift Keying) modulation can be adopted. The modulated signal is then amplified and sent to the kicker. A circulator can be employed in this case to protect the amplifier from reverse power (Fig. 60) [42].

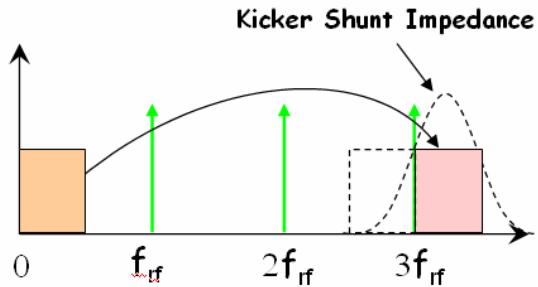


Fig. 59: The base-band signal is modulated to overlap with the kicker shunt impedance

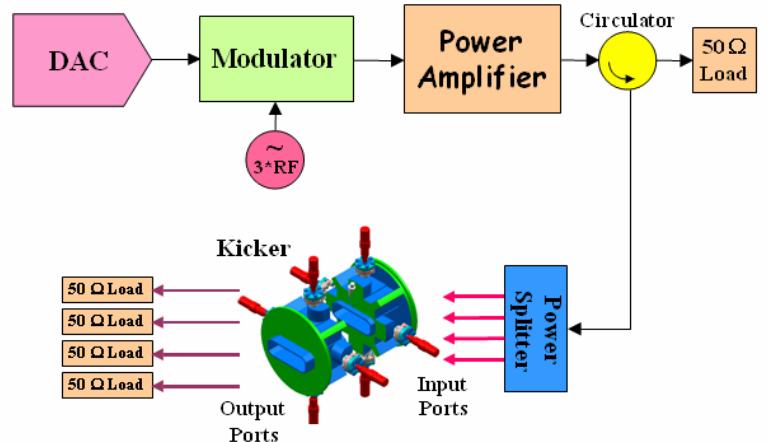


Fig. 60: Block diagram of a typical longitudinal feedback back-end

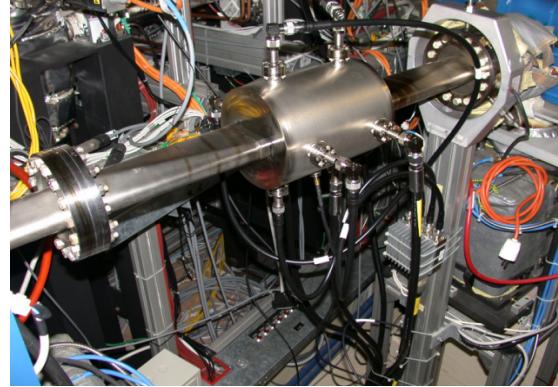
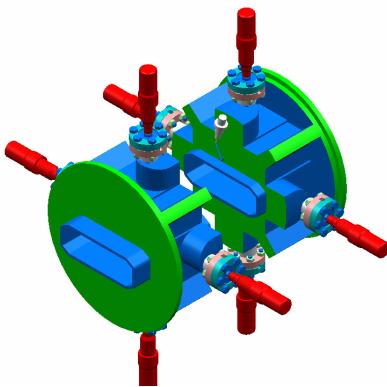


Fig. 61: The Elettra/SLS longitudinal kicker [43]

3.6 Control system integration

Each component of the feedback system which needs to be configured and adjusted should include an interface to the accelerator control system (Fig. 62). There should be the possibility to perform any

operation remotely to facilitate the system commissioning and the optimization of its performance. Moreover, as we shall see later, it is crucial to have an effective data-acquisition channel to provide fast transfer of large amounts of data for analysis of the feedback performance and for beam dynamics studies. It is also preferable to have direct access to the feedback system from a numerical computing environment and/or a script language like Matlab [44] or similar products (Octave, Scilab, Python, IGOR Pro, IDL, etc.) for quick development of measurement procedures using scripts as well as for data analysis and visualization.

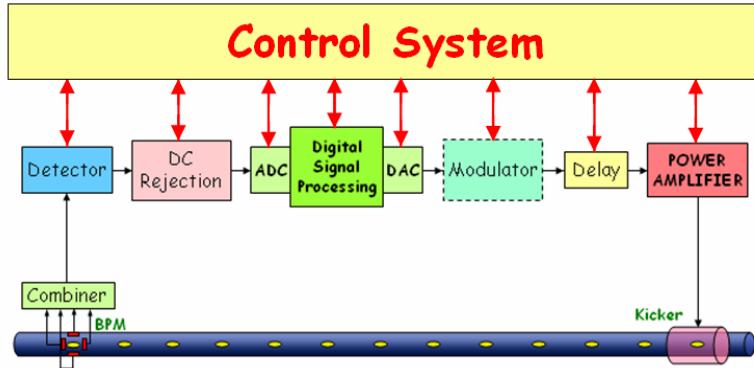


Fig. 62: Integration of the feedback system components into the accelerator control system

4 RF power requirements

In this section we determine the formulae to calculate the RF power required to damp coupled-bunch instabilities.

4.1 Transverse feedback

The transverse motion of a bunch of particles not subject to damping or excitation can be described as a pseudo-harmonic oscillator with amplitude proportional to the square root of the β -function:

$$x(s) = a \sqrt{\beta(s)} \cos \varphi(s), \quad \text{where} \quad \varphi(s) = \int_0^s \frac{d\bar{s}}{\beta(\bar{s})} \quad (12)$$

s being the longitudinal coordinate.

The derivative of the position, namely the angle of the trajectory, is

$$x' = -\frac{a}{\sqrt{\beta}} \sin \varphi + \frac{a \beta'}{2 \sqrt{\beta}} \cos \varphi, \quad \text{with} \quad \varphi' = \frac{1}{\beta} \quad . \quad (13)$$

By introducing $\alpha = -\frac{\beta'}{2}$ we can write

$$x' = \frac{a}{\sqrt{\beta}} \sqrt{1 + \alpha^2} \sin(\varphi + \arctan \alpha) \quad . \quad (14)$$

Let us consider now a kicker placed at coordinate s_k ; the kicker electromagnetic field deflects the bunch trajectory which varies its angle by k . As a consequence the bunch starts another oscillation

$$x_i = a_i \sqrt{\beta} \cos \varphi_i \quad (15)$$

which must satisfy the following constraints

$$\begin{cases} x(s_k) = x_i(s_k) \\ x'(s_k) = x'_i(s_k) + k \end{cases} . \quad (16)$$

By introducing

$$A = a \sqrt{\beta}, \quad A_i = a_i \sqrt{\beta} \quad (17)$$

Eqs. (16) become a two-equation, two-unknown-variables system:

$$\begin{cases} A \cos \varphi = A_i \cos \varphi_i \\ A \frac{\sqrt{1+\alpha^2}}{\beta} \sin(\varphi + \arctg(\alpha)) = A_i \frac{\sqrt{1+\alpha^2}}{\beta} \sin(\varphi_i + \arctg(\alpha)) + k. \end{cases} . \quad (18)$$

The solution of the system gives amplitude and phase of the new oscillation

$$\begin{cases} A_i = \sqrt{(A \sin \varphi - k \beta)^2 + A^2 \cos^2 \varphi} \\ \varphi_i = \arccos\left(\frac{A}{A_i} \cos \varphi\right) \end{cases} . \quad (19)$$

From the first equation, if the kick is small ($k \ll \frac{A}{\beta}$) then

$$\frac{\Delta A}{A} = \frac{A - A_i}{A} \approx \frac{\beta}{A} k \sin \varphi . \quad (20)$$

From the same equation it can be seen that in the linear feedback case, that is when the turn-by-turn kick signal is a sampled sinusoid with amplitude proportional to the bunch oscillation amplitude, in order to maximize the damping rate the kick signal must be in-phase with $\sin \varphi$

$$k = g \frac{A}{\beta} \sin \varphi \quad \text{with } 0 < g < 1 \quad (21)$$

that is in quadrature with the bunch oscillation (Eq. (12)).

The optimal gain g_{opt} (which must in any case be < 1) is determined by the maximum kick value k_{max} that the kicker is able to generate. If we want the feedback to work in a linear regime, i.e., not in

saturation, the feedback gain must be set so that k_{max} is generated when the oscillation amplitude A at the kicker location is maximum: $g_{opt} = \frac{k_{max}}{A_{max}} \beta$. Therefore

$$k = \frac{k_{max}}{A_{max}} A \sin \varphi. \quad (22)$$

For small kicks the relative amplitude decrease is

$$\frac{\Delta A}{A} \approx \frac{k_{max}}{A_{max}} \beta \sin^2 \varphi \quad (23)$$

and its average is

$$\left\langle \frac{\Delta A}{A} \right\rangle \approx \frac{\beta k_{max}}{2 A_{max}}. \quad (24)$$

The average relative decrease is therefore constant, which means that, on average, the amplitude decrease is exponential with time constant τ (damping time) given by

$$\frac{1}{\tau} = \left\langle \frac{\Delta A}{A} \right\rangle \frac{1}{T_0} = \frac{\beta k_{max}}{2 A_{max} T_0}, \quad (25)$$

where T_0 is the revolution period. By referring to the oscillation amplitude at the BPM location,

$$\frac{1}{\tau} = \frac{k_{max}}{2 T_0 A_{Bmax}} \sqrt{\beta_k \beta_B} \quad (26)$$

where A_{Bmax} is the maximum oscillation amplitude at the BPM location.

In the case of relativistic particles, the change of the transverse momentum p of the bunch passing through the kicker can be expressed by

$$\Delta p = \frac{e}{c} V_{\perp}, \quad \text{where} \quad V_{\perp} = \int_0^L (\bar{E} + c \times \bar{B})_{\perp} dz \quad \text{is the kick voltage,}$$

e = electron charge, c = light speed, \bar{E} and \bar{B} = fields in the kicker, L = length of the kicker.

Here p can be written as $\frac{E_0}{c}$, where E_0 is the beam energy, and V_{\perp} can be derived from the

definition of kicker shunt impedance: $R_k = \frac{V_{\perp}^2}{2 P_k}$.

The maximum deflection angle in the kicker is given by

$$k_{\max} = \frac{\Delta p}{p} = e \frac{V_{\perp}}{E_0} = \left(\frac{e}{E_0} \right) \sqrt{2 P_k R_k} . \quad (27)$$

From the previous equations we obtain

$$P_k = \frac{2}{R_k \beta_k} \left(\frac{E_0}{e} \right)^2 \left(\frac{T_0}{\tau} \right)^2 \left(\frac{A_{B\max}}{\sqrt{\beta_B}} \right)^2 . \quad (28)$$

Given a free bunch oscillation with amplitude at the BPM location $A_{B\max}$, Eq. (28) allows us to calculate the power necessary to damp this oscillation with time constant τ . In the case of excited oscillations, in order to damp the instability, the feedback damping time-constant must be less than the instability growth time-constant (see Section 1.3.3).

It has to be noted that this equation is only valid if all of the system components are ideal and the feedback is perfectly tuned.

The maximum power is supplied when the oscillation amplitude is maximum, namely at the time the feedback is switched on. During the damping process, the power decreases with the square of the oscillation amplitude. This leads us to an important conclusion: in order to determine the required power of the RF amplifier, we have to know not only the strength of the multibunch instability we want to damp, but also its maximum amplitude. For the same reason it is preferable to switch on the feedback when the oscillation is small: if we keep the feedback gain high (a small oscillation corresponds to the entire dynamic range of DAC/amplifier) it will be easier to catch and keep it damped.

The following is an example of calculation of the required power for the Elettra transverse feedback system. The parameters are

$R_k = 15 \text{ k}\Omega$ (average value), $E_B/e = 2 \text{ GeV}$, $T_0 = 864 \text{ ns}$, $\tau = 120 \mu\text{s}$, $\beta_{B\text{ H,V}} = 5.2, 8.9 \text{ m}$, $\beta_{K\text{ H,V}} = 6.5, 7.5 \text{ m}$.

The plots in Fig. 63 show the required power in the horizontal and vertical plane as a function of the initial oscillation amplitude.

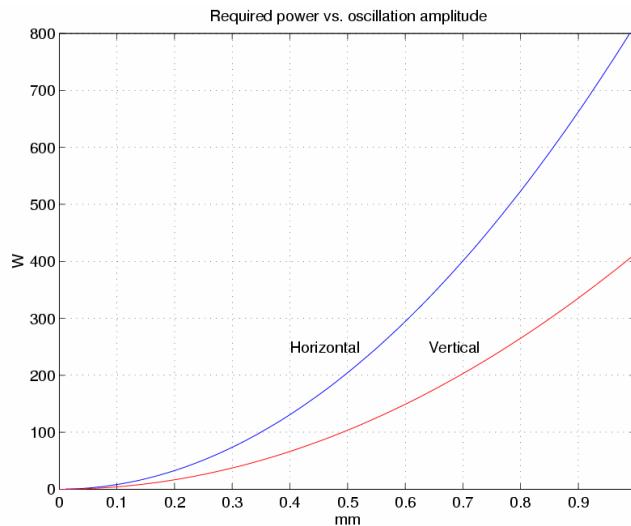


Fig. 63: Power required to damp a transverse instability as a function of the initial oscillation amplitude calculated for the Elettra storage ring

4.2 Longitudinal feedback

A similar procedure can be adopted to calculate the power required for a longitudinal feedback.

The relationship between the energy oscillation amplitude ε and the phase oscillation amplitude φ of a longitudinal synchrotron oscillation is given by

$$\varepsilon = \frac{E_0 \omega_s}{2 \pi f_{rf} \alpha} \varphi , \quad (29)$$

where ω_s is the synchrotron frequency, α the momentum compaction factor, and f_{rf} the RF frequency.

As in the transverse case, the damping time constant τ is given by the equation

$$\frac{1}{\tau} = \frac{eV_{\max}}{2 \varepsilon_{\max} T_0} . \quad (30)$$

where V_{\max} is the maximum kick voltage and ε_{\max} is the maximum energy oscillation amplitude.

By introducing the longitudinal kicker shunt impedance R_k

$$R_k = \frac{V^2}{2 P_k} , \quad (31)$$

the power required to damp a longitudinal oscillation is given by

$$P_k = \frac{2}{R_k} \left(\frac{\omega_s}{\omega_0} \frac{E_0}{e} \frac{\varphi_{\max}}{\alpha f_{rf} \tau} \right)^2 . \quad (32)$$

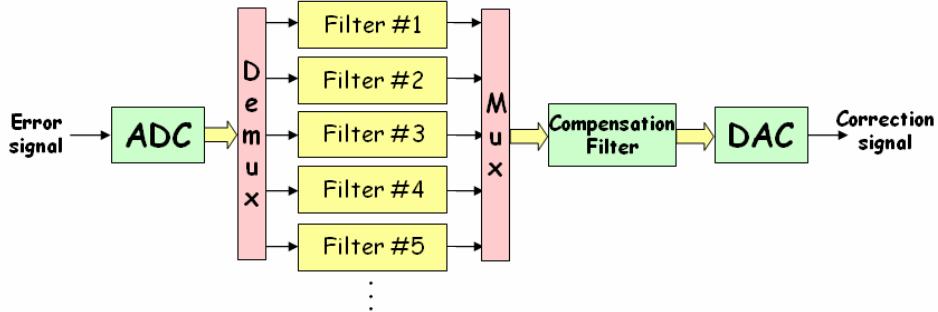
5 Digital signal processing

As mentioned in Section 3.4, the digital processing in a bunch-by-bunch feedback is performed by M separated channels, each dedicated to one bunch (Fig. 64).

In order to damp the bunch oscillations, the turn-by-turn kick signal must be the derivative of the bunch position at the kicker, namely, for a given oscillation frequency, a $\pi/2$ phase-shifted signal must be generated. In determining the real phase shift to perform in each channel, the phase advance between BPM and kicker must be taken into account as well as any additional delay due to the feedback latency (multiple of one machine revolution period). Moreover, any residual constant offset (stable beam component) must be rejected from the bunch signal to avoid DAC saturation.

These basic tasks can be accomplished by digitally filtering the streams of turn-by-turn samples in each channel, namely calculating the present correction sample on the basis of the past position samples.

Additionally, in case the not-ideal characteristics of amplifier and kicker can be somehow compensated, a digital filter can be put in the full-rate data path before the DAC.

**Fig. 64:** Block diagram of the digital processor

5.1 Digital filter design

Digital filters can be implemented with FIR (Finite Impulse Response) or IIR (Infinite Impulse Response) structures. In general, IIR filters offer better performance than FIR filters. The disadvantage is a more difficult design and an increased complexity of the filter implementation.

Various techniques are used to design digital filters, including frequency-domain and model-based design. In the following sections we show some examples of digital filters actually adopted in bunch-by-bunch feedback systems.

5.1.1 3-tap FIR filter

The minimum requirements for each of the M digital filters are

- DC rejection (for a FIR filter this means that the sum of the coefficients must be zero);
- given amplitude response at the tune frequency;
- given phase response at the tune frequency.

A simple 3-tap FIR filter can fulfil these requirements. Since it is a very simple filter, the coefficients can be calculated analytically. Let us consider the Z-transform of the filter response, namely the filter transfer function $H(z)$; in order to have zero amplitude response at DC, $H(z)$ must have a ‘zero’ in $z = 1$. Another zero in $z = c$ is needed in order to have the required phase response α at the tune frequency ω :

$$H(z) = k(1 - z^{-1})(1 - cz^{-1}) \quad . \quad (33)$$

The parameter c can be calculated analytically

$$\begin{aligned}
 H(z) &= k(1 - (1+c)z^{-1} + cz^{-2}) \quad z = e^{j\omega} \\
 H(\omega) &= k(1 - (1+c)e^{-j\omega} + ce^{-2j\omega}) \\
 e^{-j\omega} &= \cos \omega - j \sin \omega, \quad \alpha = \text{ang}(H(\omega)) \\
 \text{tg}(\alpha) &= \frac{c(\sin(\omega) - \sin(2\omega)) + \sin(\omega)}{c(\cos(2\omega) - \cos(\omega)) + 1 - \cos(\omega)} \\
 c &= \frac{\text{tg}(\alpha)(1 - \cos(\omega)) - \sin(\omega)}{(\sin(\omega) - \sin(2\omega)) - \text{tg}(\alpha)(\cos(2\omega) - \cos(\omega))} \quad (34)
 \end{aligned}$$

Here k is determined given the required amplitude response at tune $|H(\omega)|$:

$$k = \frac{|H(\omega)|}{\sqrt{(1-(1+c)\cos(\omega)+c\cos(2\omega))^2 + ((1+c)\sin(\omega)-c\sin(2\omega))^2}} . \quad (35)$$

Example:

$$\omega/2\pi = 0.2, \quad |H(\omega)| = 0.8, \quad \alpha = 222^\circ.$$

From the equations we can calculate c and k : $c = -0.63$, $k = 0.14$. The transfer function of the filter is therefore $H(z) = -0.63 + 0.49 z^{-1} + 0.14 z^{-2}$. The filter response is depicted in Fig. 65.

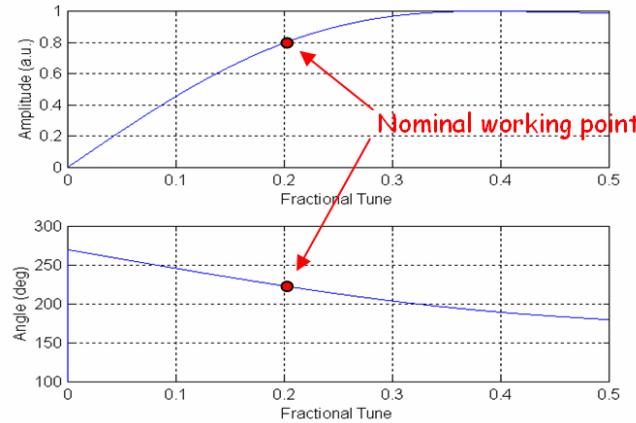


Fig. 65: Amplitude and phase response of a 3-tap FIR filter

5.1.2 5-tap FIR filter

With a longer FIR filter, more degrees of freedom are available in the design and additional features can be added to the filter. The following is an example of a 5-tap FIR filter currently used at Elettra for transverse feedback [28].

The tune of an accelerator can significantly change during machine operations; the filter can be designed in order to guarantee the same feedback efficiency in a given frequency range. In this case the requirement is to have an optimal phase response.

In this example the feedback delay is four machine turns, which must be taken into account when calculating the filter phase response. When the tune frequency increases, the phase of the filter must increase as well; this means that the phase response must have a positive slope around the working point.

The filter design can be made using the Matlab function *invfreqz()*. This function calculates the FIR filter coefficients that best fit the required frequency response using the least-squares method. The desired response is specified by defining amplitude and phase at three different frequencies: the origin and two frequencies, f_1 and f_2 , chosen in the proximity of the nominal tune (Fig. 66, lower plot). As we can see in the upper plot, this is achieved at the expense of a bad amplitude response, which features a minimum at the tune while the maximum is at a different frequency.

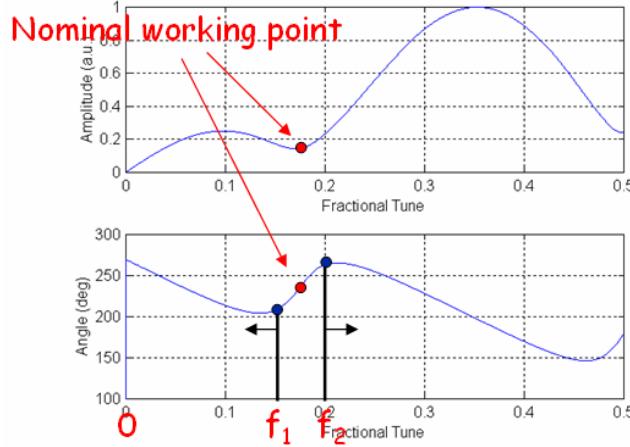


Fig. 66: Amplitude and phase response of a 5-tap FIR filter

5.1.3 Selective FIR filter

A filter often employed in longitudinal feedback systems is the selective FIR filter whose impulse response (the filter coefficients) is a sampled sinusoid with frequency equal to the synchrotron tune (see Fig. 67) [45]. As we can see in Fig. 68, the filter amplitude response has a maximum at the tune frequency and linear phase. The more filter coefficients we use the more selective the filter.

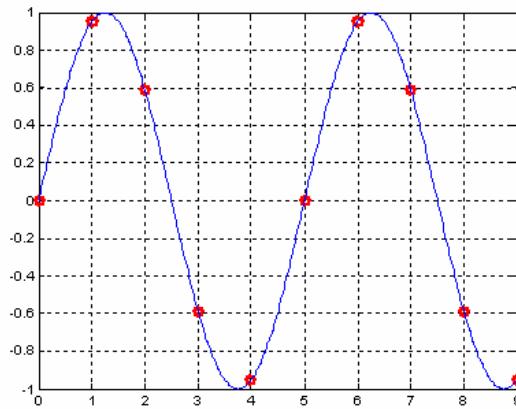


Fig. 67: Impulse response of a selective FIR filter (red dots)

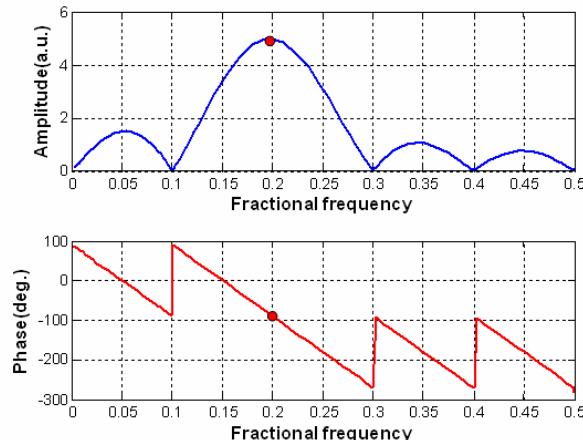


Fig. 68: Amplitude and phase response of a selective FIR filter

5.1.4 Advanced filter design

More sophisticated techniques using longer FIR or IIR filters enable a variety of additional features to exploit the potentiality of digital signal processing.

It is possible, for instance, to enlarge the working frequency range (as seen in Section 5.1.2) with no degradation of the amplitude response. Moreover, the filter selectivity can be enhanced to better reject unwanted frequency components (noise) or to minimize the amplitude response at frequencies that must not be fed back. An example of this design technique is the Time Domain Least Square Fitting (TDLSF) described in Ref. [33].

Another interesting possibility is to stabilize different tune frequencies simultaneously by designing a filter with two separate working points. This is required, for example, when horizontal and vertical as well as dipole and quadrupole instabilities have to be addressed by the same feedback system [35].

Advanced design techniques can also be employed to improve the robustness of the feedback under parametric changes of accelerator or feedback components (e.g. optimal control, robust control, etc.) [46].

5.2 Down sampling

The synchrotron frequency is usually much lower than the revolution frequency, which means that one complete synchrotron oscillation is accomplished in many machine turns. In longitudinal feedback systems, in order to be able to properly filter the bunch signal, down sampling is usually carried out [47]; one out of D bunch samples is used, where D is the down sampling factor (Fig. 69).

Filtering is performed over the down sampled digital signal and the filter design is done in the down sampled frequency domain (the original one enlarged by the down sampling factor D). After processing, the turn-by-turn correction signal is reconstructed by means of a hold buffer that keeps each calculated correction value for D turns, as shown in Fig. 69.

The reduced data rate allows for more time to be available to perform the filter calculations and more complex filters can therefore be implemented.

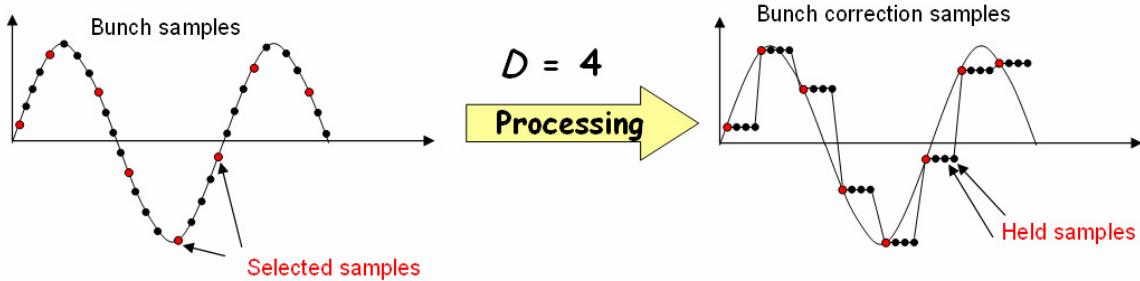


Fig. 69: Down sampling of the bunch position signal and reconstruction by the hold buffer

6 Integrated diagnostic tools

A feedback system can implement a number of powerful diagnostic tools that are useful for both commissioning and tuning of the feedback system as well as for machine physics studies. The following are some of the tools normally implemented in digital feedback systems:

- **ADC data recording** (see (1) in Fig. 70): acquisition and recording, in parallel with the feedback operation, of a large number of samples for off-line data analysis;

- **modification of filter parameters ‘on the fly’** with the required timing and even individually for each bunch (see (2) in Fig. 70): useful for switching on/off the feedback, generation of grow/damp transients, optimization of feedback performance, etc.;
- **injection of externally generated digital samples** (see (3) in Fig. 70): for the excitation of single/multibunches.

These tools are usually managed by an additional controller with a fast interface to the accelerator control system.

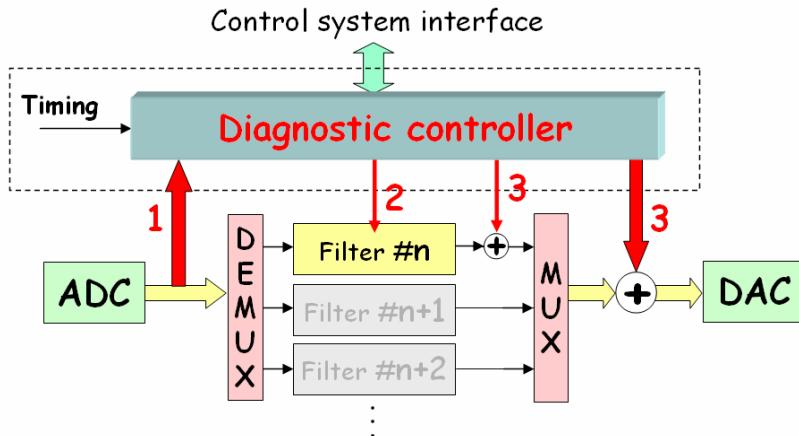


Fig. 70: Integrated diagnostics tools of a digital feedback system

In the next sections we give some examples of measurements carried out using the diagnostics tools.

6.1 Acquisition and recording of ADC data

Figure 71 depicts the amplitude spectrum of a vertical bunch-by-bunch signal sampled at $f_{rf} = 500$ MHz. The spectrum is calculated with the FFT algorithm. At the moment of acquisition the beam was vertically unstable with some multibunch modes excited around the base-band frequency of 5 MHz.

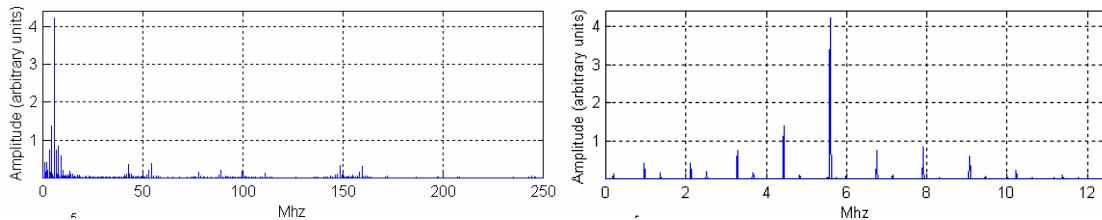


Fig. 71: Amplitude spectrum of bunch-by-bunch data of an unstable beam (left) and a zoom showing the unstable sidebands (right)

6.2 Excitation of individual bunches

To produce an excitation, the feedback loop is switched off for one or more selected bunches (for example, by setting to zero the filter coefficients) and an excitation signal is injected in place of the bunch correction signal (Fig. 72). Possible excitation signals are white or pink noise and sinusoids.

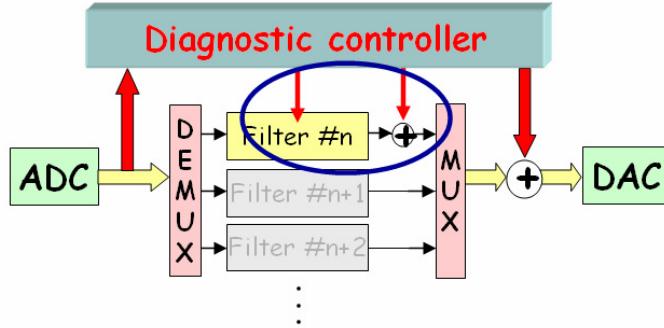


Fig. 72: Excitation of individual bunches through injection of appropriate signals

In the example of Fig. 73, two selected bunches are vertically excited with pink noise in a range of frequencies around the tune, while the feedback is applied on the other bunches. The upper plot shows the excitation of the two bunches. The spectrum of one of the excited bunches reveals a peak at the tune frequency (lower plot). This technique can be used to measure the betatron tune of a storage ring or of individual bunches with almost no deterioration of the beam quality.

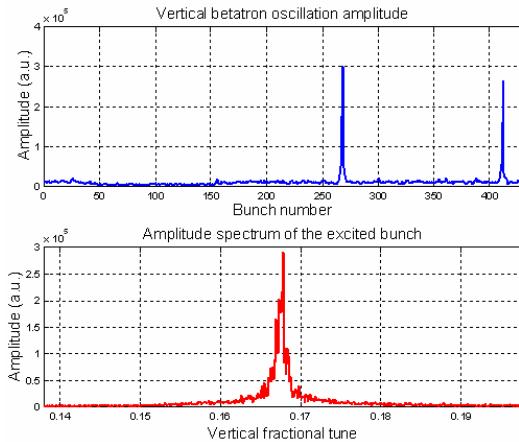


Fig. 73: Excitation of individual bunches by injection of appropriate digital signals

Excitation of individual bunches can also be done with feedback on by setting filter coefficients with inverted sign. This produces a positive feedback and anti-damping of the oscillations.

6.3 Bunch-by-bunch excitation

Interesting measurements can be done by injecting pre-defined signals in the output of the digital processor (Fig. 74). Depending on the particular excitation signal and on the filter settings, several experiments can be carried out.

By injecting a sinusoid at a given frequency, for example, the corresponding beam multibunch mode can be excited in order to test the performance of the feedback in damping that mode. If we inject an appropriate signal and record the ADC data when filter coefficients are set to zero, the beam transfer function can be calculated. By doing the same but with filter coefficients set to the nominal values, the closed loop transfer function can be determined.

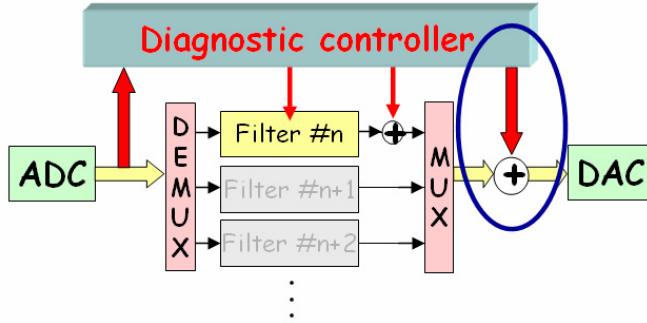


Fig. 74: Bunch-by-bunch excitation through injection of appropriate digital signals

6.4 Transient generation

A powerful diagnostic application is the generation of transients. Transients can be generated by changing the filter coefficients according to a predefined timing and by concurrently recording the oscillations of the bunches (Fig. 75).

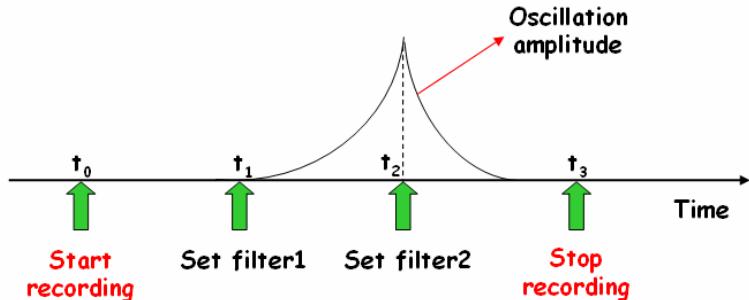


Fig. 75: Transient generation by changing the filter coefficients and recording the oscillations

The following are some examples of transients that can be used to measure damping times and growth rates by exponential fitting of the oscillation amplitude.

Let us suppose we have an excited multibunch oscillation which has reached equilibrium, i.e. constant amplitude: we can switch on the feedback and record the damping transient (Fig. 76(1)).

If the feedback is already on, we can switch it off and on again after a while and generate a grow/damp transient (Fig. 76(2)).

Transients can also be generated by artificially exciting a stable beam with a proper setting of the filter coefficients in order to realize a positive feedback (anti-damping). The feedback is then switched off letting the oscillation decay by natural damping (Fig. 76(3)).

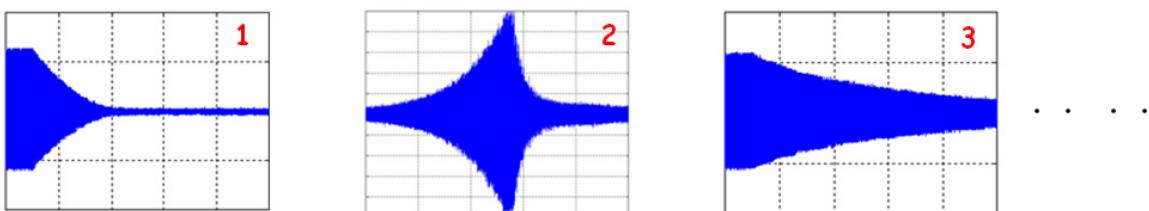


Fig. 76: Examples of transient generation

Grow/damp transients can be analysed by means of 3-D graphs. Figure 77 is an example where the amplitude of the oscillation vs. time is plotted for every bunch. The absence of oscillation for some of them is because the corresponding buckets are not filled with particles.

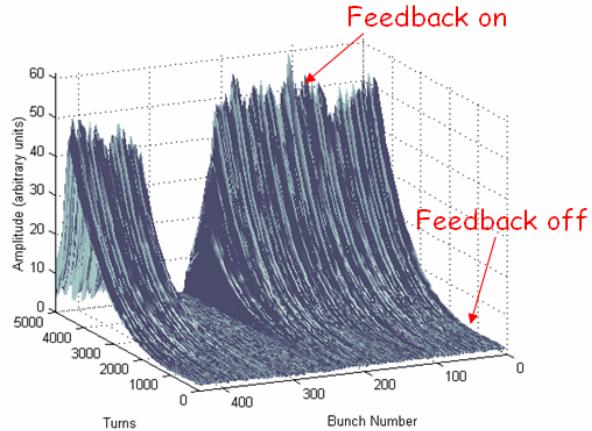


Fig. 77: 3-D graph showing the evolution of the oscillation amplitude during a grow/damp transient

Transients can also be analysed by displaying the bunch-by-bunch spectrum vs. time to study the evolution of individual coupled-bunch modes (Fig. 78).

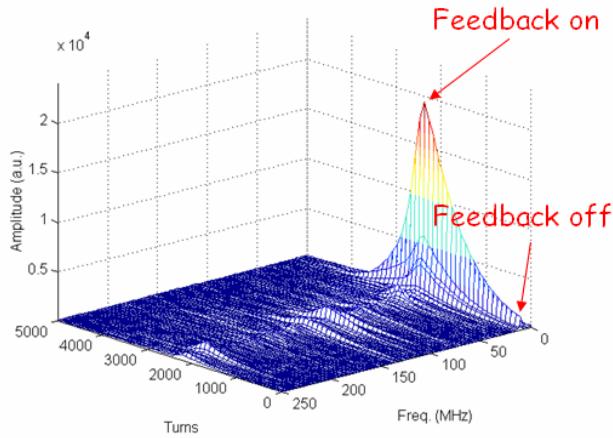


Fig. 78: Evolution of coupled-bunch unstable modes during a grow/damp transient

Transient generation is a helpful tool to tune a feedback system as well as to study coupled-bunch modes and beam dynamics [48]–[53]. The following list mentions some examples of applications using diagnostic tools and, in particular, transient analysis:

- **measurement of the feedback damping time:** can be used to characterize and optimize the feedback performance;
- **measurement of the feedback resistive and reactive response:** a feedback not perfectly tuned can have a reactive behaviour, namely produces a tune shift when switched on, that has to be minimized for optimum performance;
- **modal analysis:** measurement and analysis of the complex eigenvalue of the modes, namely the growth rate (real part of the eigenvalue) and the oscillation frequency (imaginary part of the eigenvalue);
- **impedance measurement:** the analysis of complex eigenvalues and bunch synchronous phases can be used to evaluate the machine impedance;
- **stable modes analysis:** coupled-bunch modes below the instability threshold, thus stable, can be studied to predict their behaviour at higher beam currents;

- **bunch train studies**: the analysis of different bunches along the bunch train can be used to determine the sources of the coupled-bunch instabilities;
- **phase space analysis**: the analysis of the phase evolution of unstable coupled-bunch modes is used for beam dynamics studies.

7 Feedback system tuning

In this section, some practical issues regarding the adjustments normally carried out during the commissioning of a multi-bunch feedback system are presented. They all fully exploit the diagnostic tools discussed in Section 6. We shall then see the main observable effects of the feedback action on the beam characteristics.

7.1 Detector phase

The phase of the ‘local oscillator’ signal, derived from the machine RF, has to be adjusted in order to be in phase (amplitude demodulation) or in quadrature (phase demodulation) with the bunch signal (see Section 3.2). The amplitude of the detected bunch oscillation is maximum when the phase is optimized. One way to find the optimal phase is by analysing the turn-by-turn data of an unstable bunch and by maximizing the measured oscillation amplitude.

7.2 ADC clock phase

The phase of the ADC clock must be adjusted in order to sample right on the top the ‘pulse’ generated by every bunch (see Section 3.2.4). This task is usually carried out with the ring filled with one single bunch: the optimum clock phase is achieved when the amplitude of the ADC samples relative to the filled bucket is maximum and the one of the adjacent empty buckets is minimum (Fig. 79).

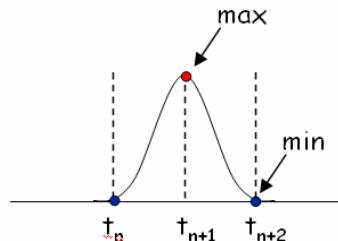


Fig. 79: Optimization of the ADC clock using a single bunch

7.3 Delay line

The delay line must be properly set in order to kick each bunch with the correction signal calculated for the same bunch. The adjustment can be done in two phases by using the feedback system:

- **Coarse adjustment** (resolution $\approx T_{rf}$): this can be done by trying to excite a stable bunch (e.g. with pink noise or with a sinusoid at the tune frequency) and adjusting the delay until the bunch we would like to excite is really seen excited by analysing the ADC samples;
- **Fine adjustment** (resolution $\approx T_{rf}/100$): this can be accomplished with the minimization of either the residual amplitude of damped coupled-bunch modes (natural or artificially excited by the use of the feedback system) or the damping time measured using transients.

7.4 Digital filter

The digital filter is designed on the basis of given requirements (e.g. phase and amplitude response). A further fine optimization of the filter parameters can be done either by using the beam and trying to maximize the feedback performance (e.g. maximum reduction of coupled-bunch modes), or with a careful analysis of grow/damp transients (damping time, reactive response, phase space analysis, etc.).

7.5 Observation of feedback effects on beam characteristics

We shall now see, with the aid of some pictures, some examples of the main observable effects of the feedback action measured using beam instrumentation.

7.5.1 Beam spectrum

When the feedback is conveniently tuned the sidebands corresponding to an unstable coupled-bunch mode are completely suppressed. Figure 80 shows an example of beam spectrum with feedback off and on.

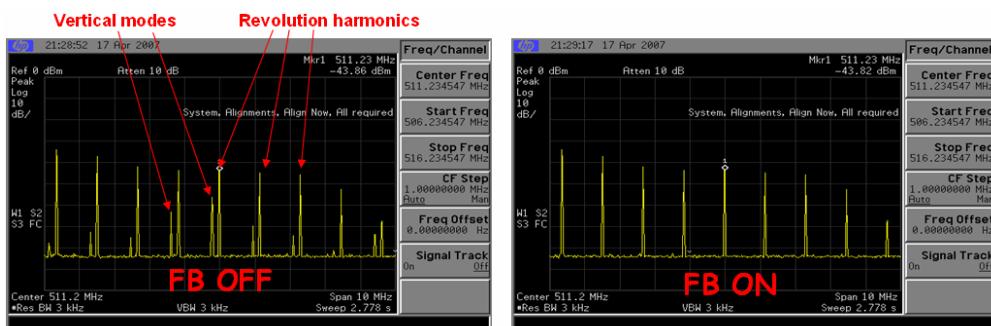


Fig. 80: Screenshots of a spectrum analyser connected to a stripline pickup. The sidebands corresponding to vertical coupled-bunch modes disappear as soon as the transverse feedback is activated.

7.5.2 Beam transverse profile

Images of an electron beam transverse profile can be produced using the synchrotron radiation generated by a bending magnet. As mentioned in Section 1.4.5, the transverse profile blows up in the presence of a transverse instability (Fig. 81, left picture). When activated, the feedback squeezes the beam bringing it to its original size (right picture).

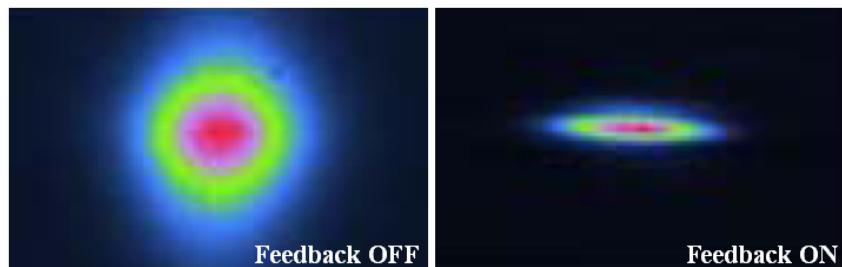


Fig. 81: Synchrotron radiation monitor images taken at TLS [34] showing the blow-up of the beam size due to a vertical instability (left) and its reduction when the feedback is switched on (right)

7.5.3 Streak camera images

Interesting measurements can be carried out by analysing the synchrotron radiation from a bending magnet using a streak camera. In the left-hand picture of Fig. 82 a longitudinal instability is clearly visible. The instability disappears when the feedback is activated (right picture).

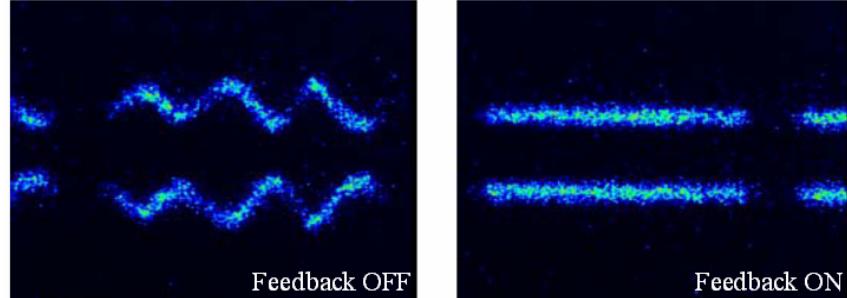


Fig. 82: Images of one machine turn taken with a streak camera in ‘dual scan mode’ at TLS [34].
The horizontal and vertical time spans are 500 and 1.4 ns, respectively. The longitudinal instabilities observable in the left picture are suppressed by the feedback in the right picture.

7.5.4 Photon beam spectra

In a synchrotron light source, the ultimate goal of a feedback system is the improvement of the emitted photon beam quality. This can be evaluated by measuring the energy spectrum of the photons produced by an undulator. As we can observe in the example reported in Fig. 83, the amplitude and shape of the generated harmonics is noticeably improved when vertical instabilities are damped by the feedback.

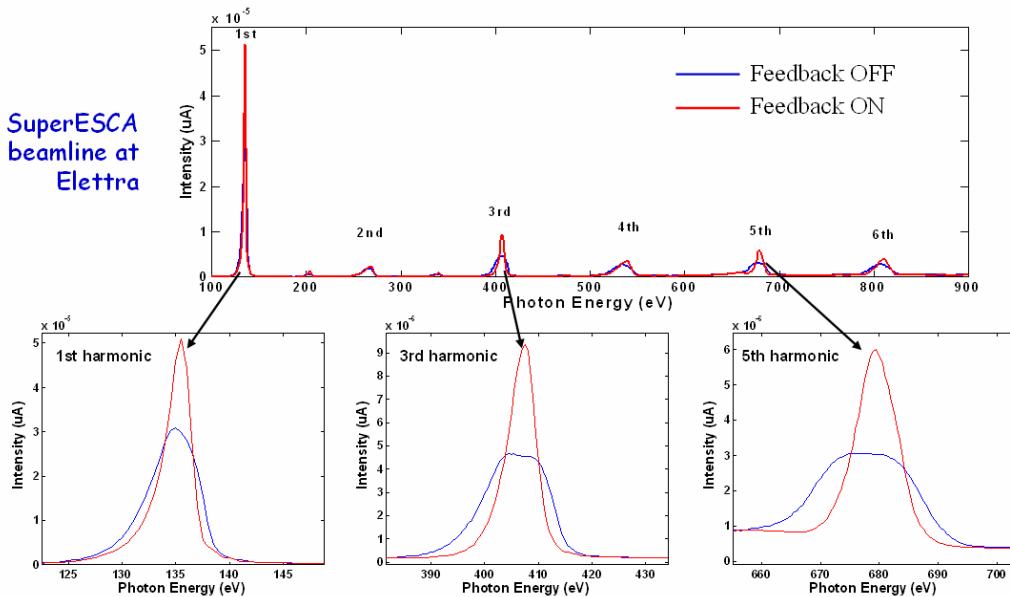


Fig. 83: Energy spectra of the photon beam measured on the SuperESCA beamline at Elettra

8 Conclusions

Feedback systems are indispensable tools for curing coupled-bunch instabilities in storage rings. Technology advances in digital electronics allow the implementation of digital feedback systems where the control algorithms are executed using programmable devices. The theory of digital signal processing is widely used to design and implement digital filters as well as to analyse data acquired by the feedback. In fact, thanks to their capability of exciting the beam in different ways and measuring the resulting beam motion, feedback systems are used not only for closed loop control but also as powerful diagnostic tools. The possibilities offered by these tools are manifold both for the optimization of the feedback performance and for beam dynamics studies.

Acknowledgements

I would like to thank Dmitry Teytelman, member of the SLAC feedback team, for many enlightening discussions and useful suggestions on these topics.

References

- [1] M. Svandrlik *et al.*, *The Cure of Multibunch Instabilities in Elettra*, Proc. PAC'95, Dallas, May 1995.
- [2] E. Courant *et al.*, *Transverse Coherent Resistive Instabilities of Azimuthally Bunched Beams in Particle Accelerators*, Rev. of Sci. Instrum., 37 (1966) 1579–1588,
- [3] M. Zobov *et al.*, *Measures to Reduce the Impedance of Parasitic Resonant Modes in the DAΦNE Vacuum Chamber*, Proc. 14th Advanced IFCA Beam Dynamics Workshop, Frascati, October 1997, Frascati Physics Series Vol. X (1998), pp. 371–378.
- [4] F. Zimmermann *et al.*, *First Observations of a ‘Fast Beam Ion Instability’ at the ALS*, Proc. PAC'97, Vancouver, May 1997.
- [5] R.A. Bosch *et al.*, *Suppression of Longitudinal Coupled Bunch Instabilities by a Passive Higher Harmonic Cavity*, Proc. PAC'93, Washington, May 1993.
- [6] E. Vogel *et al.*, *RF Amplitude Modulation to Suppress Longitudinal Coupled Bunch Instabilities in the CERN Super Proton Synchrotron*, Phys. Rev. Spec. Top. – Accel. Beams, 8 (2005) 102801.
- [7] G. Lambertson, *Control of Coupled-Bunch Instabilities in High-Current Storage Rings*, Proc. PAC'91, San Francisco, May 1991.
- [8] J. Fox *et al.*, *Feedback Control of Coupled-Bunch Instabilities*, Proc. PAC'93, Washington, May 1993.
- [9] M. Serio *et al.*, *Multibunch Instabilities and Cures*, Proc. EPAC'96, Sitges, June 1996.
- [10] F. Pedersen, *Feedback Systems*, CERN PS/90-49 (AR), 1990.
- [11] D. Teytelman, *Survey of Digital Feedback Systems in High Current Storage Rings*, Proc. PAC2003, Portland, May 2003.
- [12] K. Balewski, *Rewiew of Feedback Systems*, Proc. EPAC'98, Stockholm, June 1998.
- [13] F. Pedersen, *Multibunch Instabilities*, Proc. “Frontiers of Particle Beams: Factories with e⁺ e⁻ Rings”, Joint US-CERN School on Particle Accelerators, Benalmadena, November 1992, pp. 269–292.

- [14] F. Pedersen *et al.*, *Theory and Performance of the Longitudinal Active Damping System for the CERN PS Booster*, IEEE Trans. Nucl. Sci., Vol. NS-24, No. 3 (1977), p. 1396.
- [15] W. Barry *et al.*, *Commissioning of the ALS Transverse Coupled-Bunch Feedback System*, Proc. PAC'95, Dallas, May 1995.
- [16] S. Khan *et al.*, *BESSY II Feedback Systems*, Proc. PAC'99, New York, 1999.
- [17] H. S. Kang *et al.*, *Longitudinal and Transverse Feedback Systems for PLS Storage Ring*, Proc. PAC2001, Chicago, June 2001.
- [18] P. Wesolowsky *et al.*, *A Vertical Multi-Bunch Feedback System for ANKA*, Proc. PAC2005, Knoxville, 2005.
- [19] A. Young *et al.*, *RF and Baseband Signal Processing in the PEP-II/ALS/DAΦNE Longitudinal Feedback System*, Proc. DIPAC'97, Frascati, October 1997.
- [20] E. Plouviez *et al.*, *Broadband Bunch by Bunch Feedback for the ESRF Using a Single High Resolution and Fast Sampling FPGA DSP*, Proc. EPAC2006, Edinburgh, June 2006.
- [21] M. Enert *et al.*, *Transverse and Longitudinal Multibunch Feedback Systems for PETRA*, DESY 91-036, 1991.
- [22] D. Teytelman *et al.*, *Architecture and Technology of 500 Msample/s Feedback Systems for Control of Coupled-Bunch Instabilities*, Proc. ICALEPCS'99, Trieste, October 1999.
- [23] J. Byrd *et al.*, *Design of the PEP-II Transverse Coupled-Bunch Feedback System*, Proc. PAC'95, Dallas, May 1995.
- [24] J. Weber *et al.*, *PEP-II Transverse Feedback Electronics Upgrade*, Proc. PAC2005, Knoxville, 2005.
- [25] M. Tobiyama *et al.*, *Development of a High-Speed Digital Signal Process System for Bunch-by-Bunch Feedback Systems*, Phys. Rev. Spec. Top. – Accel. Beams, 3 (2000) 012801.
- [26] E. Kikutani *et al.*, *Present Status of the KEKB Bunch Feedback Systems*, Proc. EPAC2002, Paris, June 2002.
- [27] M. Lonza *et al.*, *Digital Processing Electronics for the Elettra Transverse Multibunch Feedback System*, Proc. ICALEPCS'99, Trieste, October 1999.
- [28] D. Bulfone *et al.*, *Bunch-by-Bunch Control of Instabilities with the ELETTRA/SLS Digital Feedback Systems*, Proc. ICALEPCS 2003, Gyeongju, October 2003.
- [29] M. Dehler *et al.*, *State of the SLS Multibunch Feedbacks*, Proc. APAC2007, Indore, January 2007.
- [30] J. Sikora *et al.*, *Performance of the Beam Stabilizing Feedback Systems at CESR*, Proc. PAC2001, Chicago, June 2001.
- [31] J. Randhahn *et al.*, *Performance of the New Coupled Bunch Feedback System at HERA-p*, Proc. PAC2007, Albuquerque, June 2007.
- [32] Z. R. Zhou *et al.*, *Development of Digital Transverse Bunch-by-Bunch Feedback System of HLS*, Proc. PAC2007, Albuquerque, June 2007.
- [33] T. Nakamura *et al.*, *Transverse Bunch-by-Bunch Feedback System for the Spring-8 Storage Ring*, Proc. EPAC2004, Lucerne, July 2004.
- [34] C.H. Kuo *et al.*, *Control of the Multi-Bunch Instabilities at TLS*, Proc. APAC2007, Indore, January 2007.

- [35] W.X. Cheng *et al.*, *Single-Loop Two Dimensional Transverse Feedback for Photon Factory*, Proc. EPAC2006, Edinburgh, June 2006.
- [36] R. Nagaoka *et al.*, *Transverse Feedback Development at SOLEIL*, Proc. PAC2007, Albuquerque, June 2007.
- [37] G. Rehm *et al.*, *First Tests of the Transverse Multibunch Feedback at Diamond*, Proc. DIPAC2007, Venice, May 2007.
- [38] <http://www.i-tech.si/>
- [39] T. Obina *et al.*, *Longitudinal Feedback System for the Photon Factory*, Proc. PAC2007, Albuquerque, June 2007.
- [40] <http://www.dimtel.com/>
- [41] M. Dehler *et al.*, *Current Status of the Elettra/SLS Transverse Multi-Bunch Feedback*, Proc. EPAC2000, Vienna, June 2000.
- [42] A. Ghigo *et al.*, *Kickers and Power Amplifiers for the DAΦNE Bunch-by-Bunch Longitudinal Feedback System*, Proc. EPAC'96, Sitges, June 1996.
- [43] M. Dehler, *Kicker Design for the ELETTRA/SLS Longitudinal Multi-bunch Feedback*, Proc. EPAC2002, Paris, June 2002.
- [44] <http://www.mathworks.com/>
- [45] H. Hindi *et al.*, *Analysis of DSP-Based Longitudinal Feedback System: Trials at SPEAR and ALS*, Proc. PAC'93, Washington, May 1993.
- [46] H. Hindi *et al.*, *A Formal Approach to the Design of Multibunch Feedback Systems: LQG Controllers*, Proc. EPAC'94, London, June 1994.
- [47] J. Fox *et al.*, *Down Sampled Signal Processing for a B Factory Bunch-by-Bunch Feedback System*, Proc. EPAC'92, Berlin, March 1992.
- [48] S. Prabhakar *et al.*, *Calculation of Impedance from Multibunch Synchronous Phases: Theory and Experimental Results*, SLAC-PUB-7979, October 1998.
- [49] D. Teytelmen *et al.*, *Frequency Resolved Measurement of Longitudinal Impedances Using Transient Beam Diagnostics*, Proc. PAC2001, Chicago, June 2001.
- [50] J. Fox *et al.*, *Multi-Bunch Instability Diagnostics via Digital Feedback Systems at PEP-II, DAΦNE, ALS and SPEAR*, Proc. PAC'99, New York, 1999.
- [51] D. Teytelman *et al.*, *Observation, Control and Modal Analysis of Longitudinal Coupled-bunch Instabilities in the ALS via a Digital Feedback System*, SLAC-PUB-7292. LBNL-39329, September 1996.
- [52] S. Khan, *Diagnostics of Instabilities at BESSY II Using the Digital Longitudinal Feedback System*, Proc. EPAC2002, Paris, June 2002.
- [53] A. Drago, *Horizontal Instability and Feedback Performance in DAΦNE E+ Ring*, Proc. EPAC2004, Lucerne, July 2004.

Bibliography

- H. Winick, *Synchrotron Radiation Sources - A Primer* (World Scientific, Singapore, 1994).
- A.W. Chao, *Physics of Collective Beam Instabilities in High Energy Accelerators* (Wiley, New York, 1993).
- A.V. Oppenheim and R.W. Schafer, *Digital Signal Processing* (Prentice-Hall, Englewood Cliffs, NJ, 1975).
- G.F. Franklin, J.D. Powell, A. Emami-Naeini, *Feedback Control of Dynamic Systems* (Addison-Wesley, Reading, MA, 1994).
- S. Prabhakar, *New Diagnostics and Cures for Coupled-Bunch Instabilities* (PhD thesis, Stanford University, 2000, SLAC-Report-554).
- D. Teytelman, *Architectures and Algorithms for Control and Diagnostics of Coupled-Bunch Instabilities in Circular Accelerators* (PhD thesis, Stanford University, SLAC-Report-633).

Real-time control of beam parameters

M. Dehler

Paul Scherrer Institut, Villigen, Switzerland

Abstract

This article gives an overview of the theory and application of real-time control of accelerator beams. The design and structure of orbit feedbacks are described, going from basic local feedbacks to modern state-of-the art global systems. The time domain behaviour is analysed for the building blocks of the systems as well as from the spectrum of random sources driving the orbit perturbations. The use of predictive filtering is shown for the design of the control algorithm. A second important class is the control of tunes and chromaticities. Advanced tune measurements are performed using a digital phase-locked loop. The feedback systems are typically hybrid, simultaneously working on tune and coupling and chromaticity. Adaptive feed-forward algorithms are shown to be a suitable approach for use in energy ramping. For application in a high-speed bunch-by-bunch feedback system, efficient low-noise data processing is presented for a digital filter. Also here, predictive filtering is shown to give well-adapted high-order filters.

1 Introduction

One of the countless corollaries to Murphy's Law states that everything able to go out of adjustment will do so with the utmost enthusiasm. As a consequence, today's accelerator facilities employ a myriad of controllers, from minor ones stabilizing individual components and subsystems, to the ones described here and directly affect the beam behaviour.

These systems are best classified according to the parameter in the phase-space distribution which they are supposed to control. This phase space describes the particle motion in the accelerator in six dimensions: the three spatial coordinates and the corresponding momenta. The most important are mean values (position, energy and flight angle) and standard deviations (size and energy/momentum spread) as a function of time and position inside the machine.

The longitudinal position and momentum (as well as the frequency of the longitudinal synchrotron oscillation in a ring machine) are mainly influenced by the amplitude and phase of the main RF. These are controlled by the regulation loop of the RF system, which is not covered here. Exceptions are fast instabilities caused, for example, by higher order modes in the system, which are typically controlled by bunch-by-bunch feedbacks.

Transversally, we observe drifts in the beam orbit due to ground motion and mechanical drifts. Other than the longitudinal case, sources of perturbations as well as required corrections are distributed over the whole machine, so that feedbacks and feed-forwards need multiple sensors and multiple corrections. As in the longitudinal case, there are fast transverse oscillations due to wake fields etc., which are controlled via dedicated bunch-by-bunch feedback systems.

Then there are the higher order properties of the beam such as the transverse tunes relating to the focusing strength of the magnet optics in the respective planes and the chromaticity of the optics, which describes the dependency between focusing strength and particle energy. The tunes, the coupling between the horizontal and vertical tunes, and the chromaticity are intrinsically related, so that typically hybrid stabilization schemes are employed.

1.1 Feedback or feed-forward

Implicitly, all controllers make use of pre-knowledge about the system and the excitations. The way this is done defines two general classes, feedbacks and feed-forward systems.

The main task of a feedback is to control random, non-predictable fluctuations in the accelerator parameters, which may be due to internal or external noise. (Notable exceptions are bunch-by-bunch feedbacks. Here the system is inherently unstable and the goal of the feedback is to change an unstable behaviour into a stable one.)

But there are also systematic changes like ramping the energy of an accelerator. While ramping the beam energy, the magnets controlling orbit and focusing have to be adjusted correspondingly, something which could in principle be done by a feedback system. But, since the parameters for the ramping curve are perfectly known in advance, a feed-forward system is the more suitable approach, where the corresponding set of magnet currents are precomputed and set directly.

A pure feed-forward system blindly sets values derived from a machine model and will not look at the quality of the correction. So the usual approach is to use an adaptive feed-forward, which combines the classical feed-forward with a feedback. The correction values from the feedback are used to update and optimize those used by the feed-forward system.

2 Controlling the orbit

One of the fundamental beam properties to control is deviations from the reference orbit in the accelerator. The measurement is done using beam-position monitors (BPMs), in which the passing beam induces electromagnetic fields into sets of electromagnetic electrodes in the shape of buttons or striplines. The beam position is proportional to the difference in signal strength coming from the electrodes. As an alternative, the synchrotron light radiated by the beam as it is deflected by the magnets can also serve for the position measurement. As the light strikes the electrodes of X-ray BPMs, it stimulates the emission of photoelectrons. Comparing the photocurrent from different electrodes again results in a position measurement. Corrector magnets—typically dipole magnets of low strength—give a transverse kick to the beam as it passes the corrector and thus adjust the orbit.

2.1 Local control

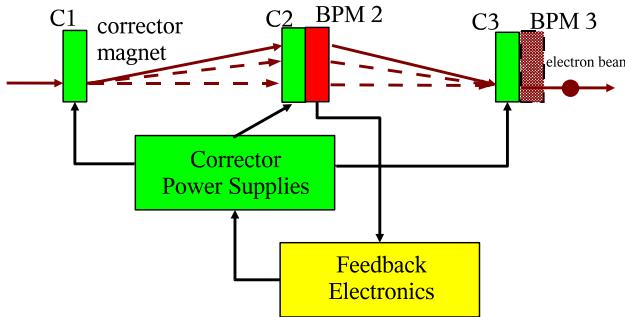


Fig. 1: Simple local orbit feedback

A model of the most basic local orbit feedback is shown in Fig. 1. It contains three corrector magnets C1 to C3 spaced at a certain distance and a BPM positioned directly at corrector C2. If we need to change the position at the BPM, we apply a kick to the beam at the corrector C1. Corrector C2 will be driven so that the overall change in beam offset at C2 is zero and corrector C3 will compensate for any change in the flight angle. The magnet triplet C1 to C3 is called a magnet bump; it will generate a local excursion in the beam trajectory, which (in first order) is transparent outside the bump.

A second triplet centred around BPM3 using correctors C2, C3 and a following corrector would help control the position there. Interlacing this bump with the one centred around BPM2 gives full control of beam position and angle between BPMs 2 and 3. Whereas a local feedback using a triplet is a classical setup around a collision experiment, a four-magnet bump is fairly common in synchrotron light sources, since the beam angle defines the direction of the synchrotron light generated, an important parameter for the machine.

It should be clear though that even with perfect magnets, local orbit bumps and feedbacks are local only in a first-order sense. Setting up a bump, we change the path length of the trajectory. As the beam has a different time of flight along the orbit, the synchronization with respect to the accelerating RF changes results in a different beam energy, which via dispersion leads to global orbit changes.

2.2 Going global

In principle, a global orbit feedback can be set up by interlacing multiple local feedbacks, such that we cover the whole accelerator. In practice, we do not have total freedom in placing the components of the feedback. The BPMs may be in different locations than the correctors, and in addition, the number of BPMs may be higher or lower than that of the correctors.

To characterize the relationship between correctors and BPMs, a response measurement is used. The machine is set to the reference orbit, we drive one of the correctors c_i and measure the resultant orbit excursion at all BPMs x_j inside the ring. If we restrict ourselves to small amplitudes, magnet non-linearities and hysteresis effects will be negligible, so that we end up with a linear relationship. Measuring offsets at all k BPMs using all n correctors, gives us the $n \times k$ size response matrix M :

$$\vec{x} = M\vec{c}. \quad (1)$$

To correct the orbit, we need the inverse mapping, giving us corrector settings \vec{c} for a given orbit distortion \vec{x} . Trying to invert M will fail in most cases. The matrix may not be quadratic and, even if it is quadratic and invertible, it may be badly conditioned, meaning that even tiny errors in measuring beam offsets lead to huge errors in the computed corrector settings. Effectively, we need a suitable approximation of M that we can invert without encountering the problems discussed above.

The appropriate way out of this dilemma is given by the singular value decomposition (SVD) algorithm [1]. The $n \times p$ matrix M is decomposed into a product of three matrices

$$M = USV^T, \quad (2)$$

where U and V are unitary $n \times n$ and $p \times p$ matrices:

$$U^T U = I, \quad (3)$$

$$V^T V = I. \quad (4)$$

The vectors u_i and v_i constituting U and V are called the left and right singular vectors and, together with the singular values σ_i , define a relationship similar to eigenvalues and eigenvectors:

$$\begin{aligned} M v_i &= \sigma_i u_i \\ M^T u_i &= \sigma_i v_i. \end{aligned}$$

As can be shown from these relations, the singular values define the diagonal matrix S :

$$s_{ii} = \sigma_i, i = 1 \dots n \quad (5)$$

$$s_{ij} = 0, \text{else}. \quad (6)$$

The most simple approach computes the decomposition as follows. The left singular vectors v_i are given by the eigenvectors with non-zero eigenvalue of the matrix product MM^T . In the same way, the right

singular u_i are eigenvectors with non-zero eigenvalue of the matrix product $M^T M$. The singular values themselves are given by the square roots of the non-zero eigenvalues of either MM^T or $M^T M$ —they are equal. (For a tutorial on SVD see Ref. [2].)

Given the components of the decomposition, we can define a pseudoinverse mapping as

$$\tilde{M}^{-1} = \sum v_i \sigma_i^{-1} u_i, i = 1 \dots n, \quad (7)$$

which will be a least-square approximation of the inverse of M . The smallest singular values σ_i correspond to states which are very sensitive to noise and errors in the input. Therefore one typically restricts the calculation of the pseudoinverse even further by using only the first $r < n$ largest singular values, so that one ends up with a well-conditioned mapping.

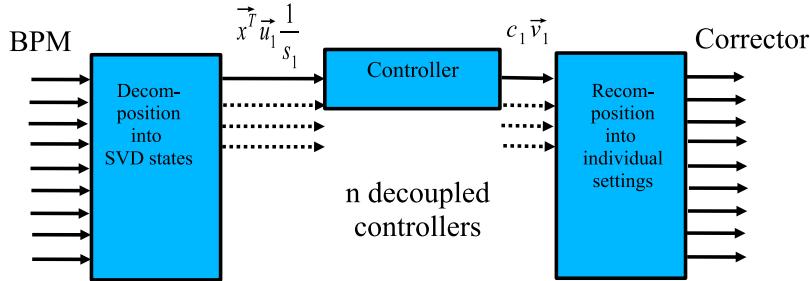


Fig. 2: Conversion of a MIMO feedback system into multiple SISO feedbacks using the singular value decomposition

Apart from computing a pseudoinverse, the approach has a second extremely useful feature. Let us assume we change the corrector magnet settings to values corresponding to one of the right singular vectors v_i . By definition, we see a change in BPM readings, which is proportional to the corresponding left singular vector u_i and nothing else. Thus we obtain a set of decoupled states of the machine, for each of which we can define a single input single output (SISO) controller, as shown in Fig. 2. The data coming from the BPMs is decomposed into decoupled states using the matrix U ; we multiply the state variables with the scalar SVD inverse σ_i^{-1} and feed the result into the scalar controller to compute correction factors. Multiplying these with the right singular vectors gives us the actual machine settings for the correctors. Instead of having a large-dimension multiple-input multiple-output (MIMO) system, we deal with a (large) number of scalar SISO systems, which are far easier to manage in terms of analysis and design.

In doing the decomposition, we implicitly assumed purely linear components in the system. For the corrector magnets, this assumption may not hold. For large amplitudes, a magnet with a yoke may show hysteresis effects corresponding to a variable gain. Superconducting magnets exhibit decay and snap back effects due to persistent currents in the magnet. If these effects are encountered, additional local control loops measuring and stabilizing the field strength should be used to compensate, so that the global orbit feedback sees more or less linear devices. For a discussion of the problem for the Large Hadron Collider superconducting correctors, see, for example, Ref. [3].

Finally, there is one important thing to keep in mind in applying this decomposition. Especially for a MIMO system, it is extremely important to avoid saturations in selected parts of the system such as ADCs, DACs, amplifiers or corrector power supplies. In a system with a single input and output, a saturating element will simply change the effective gain of the whole system. In a MIMO system, however, only parts of the response matrix saturate, so the singular value decomposition used by the controller is no longer valid and system behaviour changes in an unpredictable manner.

2.3 System topology

Given the large number of input and output parameters, data flux and processing seem to be rather complex. All the BPM data in the accelerator has to be collected to do the decomposition into the various machine states. The same thing looks true for the corrections. At first glance, this seems to favour a centralized processing architecture where one big central node is responsible for receiving, processing, and sending out data.

But looking at the local feedback using a three-corrector bump as described before, we can observe something interesting. If we forget about higher order coupling due to path length changes, this feedback already represents one of the decoupled states of the machine. Further, as has been mentioned already, we could set up a global feedback by interleaving bumps using the local feedbacks on the bump. The local feedbacks would correspond to the SISO controllers working on the decoupled SVD states of the machine, the corresponding hardware architecture would be that of a distributed system requiring only a localized data exchange.

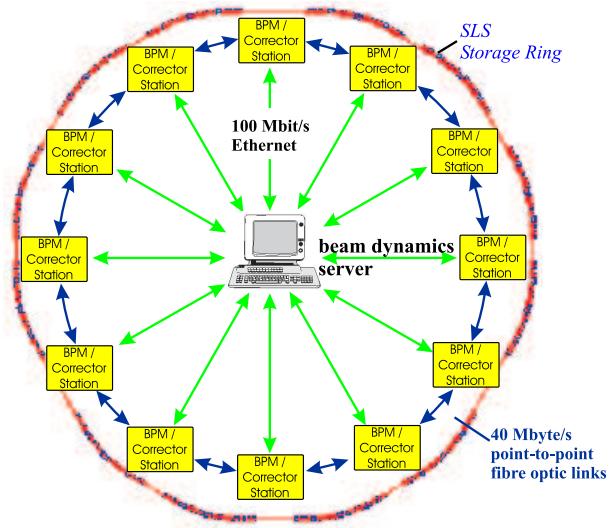


Fig. 3: Topology of orbit feedback system (SLS)

In general, the distribution of BPMs and correctors will differ from the idealized case discussed here, but we will still have a certain localization of right and left singular vectors relating to the fact that to correct an orbit deviation in a selected location of the ring, we have to adjust only correctors in the vicinity of that location and not over the whole machine. An appropriate architecture for a global system will most likely look like that in Fig. 3, showing the layout of the SLS feedback [4]. We have twelve BPM/corrector stations distributed along the ring, each being connected to six BPMs and correctors. Fast fibre optic links connect adjacent stations allowing the real-time exchange of data. For the computation of each corrector, the data from up to 18 BPMs in the vicinity can be processed. A central server connected to the stations via Ethernet is responsible for the distribution of the precomputed SVD coefficients and manages the data transfer to and from the control system.

2.4 Dynamics

What is the time and frequency domain behaviour of the overall system? The first and most important component, the beam, sees changes in the corrector fields instantaneously and propagates the information about corrector kicks as it passes through the accelerator.

The next component to consider is the position measurement and the controller itself. The hardware layout of a BPM/corrector station is shown in Fig. 4. An analog RF front-end converts the 500 MHz

raw signals coming from button BPMs down to an intermediate frequency, which is sampled digitally at a high sample rate of 33 MS/s. The digital down converter filters and down-samples the signal to a 2 kHz, 4 kS/s data stream, from which the first DSP calculates the position data. The second DSP performs the SVD calculations and contains the controller algorithms. The computed kicks are sent via fibre optic links to the digital power supply controllers steering the corrector magnets.

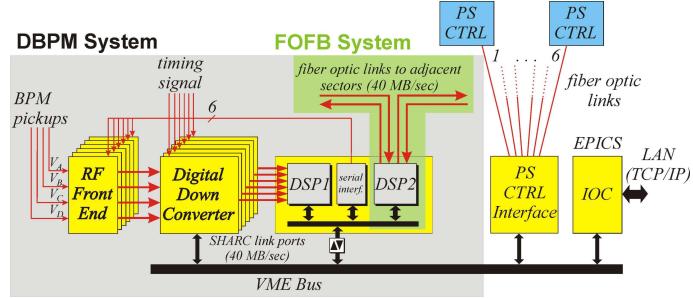


Fig. 4: Hardware layout of a SLS BPM/corrector station

The influence of these components on the frequency domain behaviour of the control loop is relatively simple. The principal effects are latencies due to data transfer, which in the case of the SLS amount to 1.5 milliseconds. Both digital down converters and corrector power supplies each have a 2 kHz bandwidth, which is large compared to the low pass behaviour due to eddy currents in the magnet core and the vacuum chamber wall, described in the following.

The magnetic flux in the corrector magnet builds up in two steps. First the field induced by the coils propagates into the spaces between the metal sheets of the lamination, a process which happens within picoseconds. In the second step the magnetic field enters the sheet material itself. Here the bulk of attenuation and delay takes place due to eddy currents.

The effect can be calculated analytically with good accuracy. The frequency dependency of the flux comes out to be

$$\frac{\psi(\omega)}{\psi_{DC}} = \frac{2}{kd} \tan(kd/2) \quad (8)$$

$$k^2 = \omega^2 \epsilon \mu - j \omega \mu \kappa. \quad (9)$$

An example [5] of the resultant frequency behaviour is shown in Fig. 5.

A second, similar effect are eddy current losses in the wall of the vacuum chamber. Other than the effects in the lamination of the corrector, the eddy currents also have an effect on the distribution of the magnetic field, so this needs to be computed numerically. As an example, Fig. 6 shows the loss density in the wall together with the resulting frequency dependency. These two effects, eddy current losses in the corrector magnets and the chamber wall, are the main determinants of the feedback performance.

2.5 Sources of orbit drift

Before talking about suitable controllers, we first need to know what effects the system is supposed to follow or correct.

There are plenty of sources of orbit drift and jitter such as, for example, machinery or human traffic, but the fundamental and unavoidable contribution is due to random drift of the ground. Tectonic drifts, settling effects, and other natural effects lead to a stochastic ground motion equivalent to a random walk or Brownian motion. If we look at two given points on the ground and try to predict the evolution of their distance $L(t)$ over time, we can make only statistical statements. There is no preferred direction, so the expected distance $E(L(t))$ will stay constant. But as time goes by, we can say less and less what

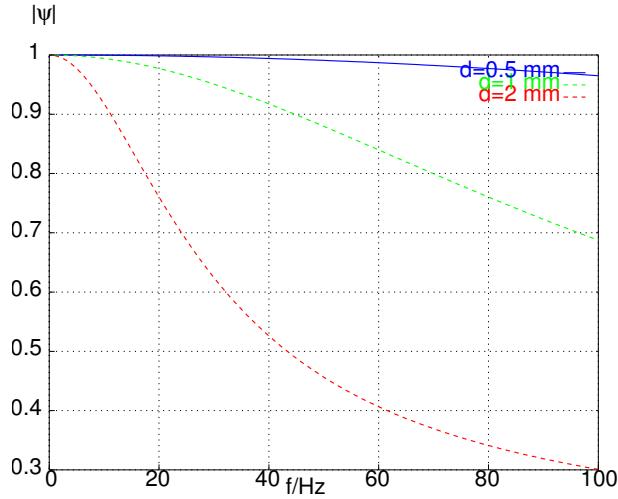


Fig. 5: Frequency dependency of the magnetic flux generated by a corrector magnet with laminated yoke. Thickness of lamination $d = 0.5, 1, 2$ mm. Steel type EBG 1200-100A.

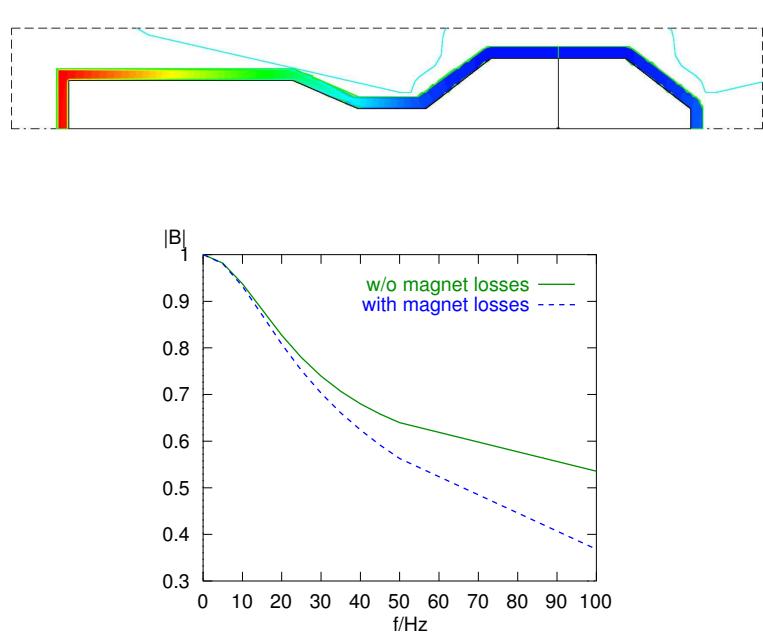


Fig. 6: Upper part: Eddy current loss density on the vacuum chamber walls at 50 Hz. Lower: Resulting frequency dependency with and without losses in the corrector magnets.

exactly it is going to be—the variance $E((L(t) - L(0))^2)$ will increase, in this case linear with time. This is described by the ATL law [6]:

$$E(L(t)) = L(0) . \quad (10)$$

$$E((L(t) - L(0))^2) = AtL . \quad (11)$$

The drift constant A has been measured in various parts of the Earth; a relatively constant value of approximately

$$A = 10^{-5} \frac{\mu m^2}{sm} \quad (12)$$

has been obtained. The corresponding noise power density comes out to be

$$S(j\omega) = \frac{A}{j\omega} . \quad (13)$$

The beam itself sees these shifts only indirectly via changes in the magnetic fields, as the magnets drift. As the field of quadrupole magnets varies linearly with position, it is the quadrupole motion which has the strongest effect on the beam. Thus the beam trajectory will fluctuate proportionally to the ground noise modulated by the mechanical transmission from ground to the magnets. The corresponding transmission function may add frequency peaks due to mechanical resonances in the girder and will change the spectrum considerably, especially for the higher frequencies. A measurement of the ground spectrum together with the corresponding quadrupole spectrum is shown in Fig. 7.

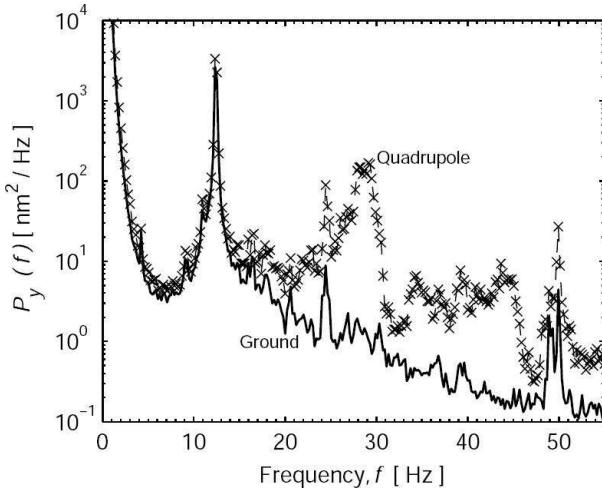


Fig. 7: Power spectrum of mechanical motion at the SLS measured on tunnel floor and quadrupole magnets [7]

A typical power spectrum of the beam motion is shown in Fig. 8, along with the corresponding curve for a running feedback using a PID controller. These have been computed from two-second samples of the beam motion, so the DC peak due to the random walk drift is not very visible. Most prominent is the line at 50 Hz, which shows the beam motion not from mechanical drifts but due to stray fields of the 50 Hz power mains. Next we see a few peaks between 10 and 30 Hz, which correspond to mechanical resonances of the girder structures. At 3 Hz there is still a tiny peak in the spectrum, which is also due to stray fields. In this case, these originate from magnets in the SLS booster, which is ramping in a 3 Hz cycle.

2.6 Proceeding further

The classic textbook example of a controller is a PID controller containing a proportional part, an integrator, and a differentiator. We determine only three parameters and, as the results in Fig. 8 show, good performance can already be obtained.

Nonetheless, it is interesting to consider what a truly optimum controller design could look like for a given machine, a given noise spectrum exciting the beam and—not to be forgotten—the internal noise sources present in the measurement, controller and corrector hardware.

This looks like a nice task for a specialized filter. The problem is that the filter is part of a feedback system, so its characteristic acts back on the feedback loop. Taking this into account leads to an optimization process involving the solution of a differential equation of the Riccati type. For purely Gaussian, white, and mutually uncorrelated noise sources, the solution is given by a linear quadratic regulator LQR [8].

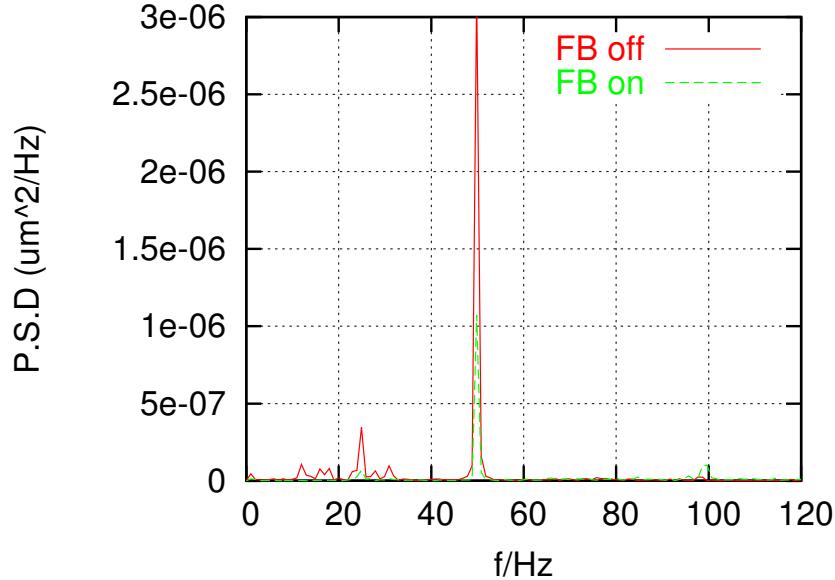


Fig. 8: Power spectrum of beam motion at the SLS with and without feedback using a PID controller

In the case of orbit correction schemes, these assumptions do not hold and we would have to find an adapted solution to the equivalent Riccati equation, and synthesize an equivalent FIR and/or IIR filter. Here, we restrict ourselves to a more approximative approach. We look for an appropriate stochastic filter fitting conditions which are explained in the following, neglecting the effects due to the feedback loop. Closing the loop and optimizing performance is then done using a classical (e.g. PID) controller. (For an in-depth overview on the topic see, for example, Ref. [9].)

2.6.1 A short introduction to predictive filtering

To keep things simple, we will assume linear, weak, stationary systems having time invariant means and variances.

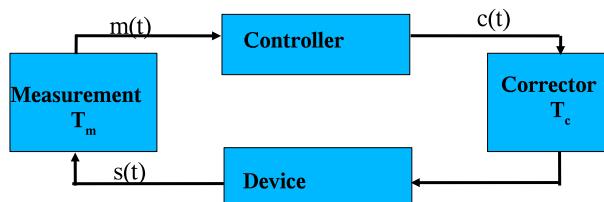


Fig. 9: Basic feedback loop

Given the basic feedback loop in Fig. 9, let us discuss what a perfect controller should look like. Any correction applied by the controller will affect the machine state only after a certain delay T_c . Furthermore, measuring the state of the device takes an additional delay (and causes bandwidth limitations and noise); the controller acts on old and only partially reliable information and needs to correct a future state of the machine. So we would like to have the best guess or estimate of the state of the machine at a future time $s(t + T_c)$, as computed from the currently available measurement values $m(t)$, $t < 0$:

$$\tilde{s}(t + T_c) = E\{s(t)|m(\tau), \tau < t\}. \quad (14)$$

This describes an estimator filter

$$\tilde{s}(t) = \int m(\tau)h(t - \tau)d\tau$$

with a yet to be determined pulse response $h(t)$. Requiring the estimation to minimize the mean square error

$$P = E((s(t) - \tilde{s}(t))^2)$$

leads to the defining equation of a Wiener filter [10]

$$R_{sm}(t, \zeta) = \int_{T_1}^{T_2} h(\tau)R_{mm}(\tau, \zeta)d\tau, \quad (15)$$

where R_{mm} and R_{sm} are the auto and cross correlation functions of $m(t)$ and $s(t)$. The integration limits $T_1 < T_2 < t$ choose the time span of data we are extrapolating from¹.

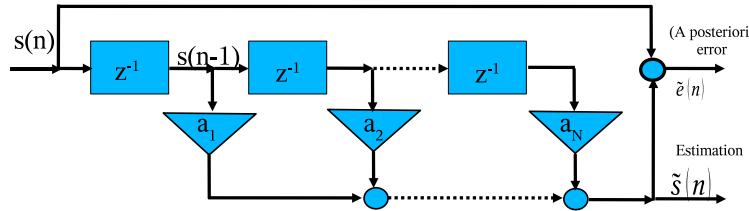


Fig. 10: A FIR predictor filter

In the discrete world, the estimator corresponds to a FIR filter as shown in Fig. 10. Minimizing the estimation error in terms of the unknown filter coefficients a_k leads to N equations of the form

$$E\left\{\left(s(n) - \sum_{k=1}^N a_k s(n-k)\right)s(n-m)\right\} = 0, 1 \leq m \leq N \quad (16)$$

or in terms of the auto correlation $R(n)$:

$$R(m) - \sum_{k=1}^N a_k R(m-k) = 0, 1 \leq m \leq N. \quad (17)$$

These are called the Yule–Walker equations. The corresponding matrix is a Toeplitz matrix, the system can be solved by Levinson’s recursion [11].

Given the coefficients a_k , we create the filter of Fig. 10 and, subtracting estimate and measurement, we can compute the a posteriori error $\tilde{e}(n)$. An ideal filter will extract only the predictable parts of the signal, so the error will not be zero but white noise with autocorrelation

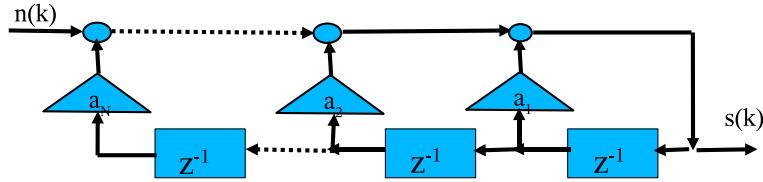
$$R_{\tilde{e}\tilde{e}}(k) = E\delta(k)$$

and power E . This property is also valid in reverse. If the signal to be estimated is white noise, then the corresponding Wiener filter will be zero in all coefficients. Signal levels for white noise can not be predicted by definition, so this certainly makes sense.

This argument can be extended a little bit further. If the signal can be modelled as an IIR filter as in Fig. 11, driven by white noise

$$s(m) = \sum_{k=1}^N a_k s(m-k) + n(m),$$

¹Using, for example, $T_1 < t < T_2$ will interpolate and $t < T_1 < T_2$ will give a best estimate of past values in terms of future ones.


Fig. 11: IIR filter

the inverse filter, which is an FIR type, will convert $s(m)$ back into white noise and so the corresponding Wiener filter gets the coefficients

$$h_k = a_k .$$

As we have seen, the autocorrelation of the *a posteriori* error $\tilde{e}(n)$ contains information about the quality of the estimator. From the estimation process itself, we can also compute an *a priori* error along with its autocorrelation. If we have a non-stationary process, which means the stochastic properties of the sources and devices involved vary with time, we can monitor both error signals and continuously update the filter coefficients. The resultant adaptive system is called a Kalman filter [12].

2.6.2 Application to typical spectra

The most simple model of orbit motion is given by the ATL law describing a random walk. In this case, the orbit deviation is simply an IIR integrator being driven by white noise:

$$s(k) = s(k - 1) + A n(k) .$$

Assuming a perfect measurement without additional noise, we can apply the result from the last section and directly write down the estimator as

$$\tilde{s}(k) = s(k - 1) .$$

Effectively, we take the value of the last measurement as the best guess of the current state.

In the next step, we take the same model for the orbit drift, but now we are more realistic in adding some uncorrelated white noise to the measurement $m(k)$.

$$\begin{aligned} s(k) &= s(k - 1) + A n_1(k) . \\ m(k) &= s(k) + B n_2(k) . \end{aligned}$$

In this case, measurement and signal differ. Minimizing the least-squares error between signal and estimates leads us to the following defining equations:

$$\begin{aligned} \sum_{l=1}^N a_l R_{mm}(l, k) &= R_s m(k); k = 1, N \\ R_{mm}(k) &= R_{ss}(k) + B^2 \delta(k) \\ R_{sm}(k) &= R_{ss}(k) \end{aligned}$$

Since the noise sources n_1, n_2 are uncorrelated, the correlation matrices R_{mm} and R_{sm} can be derived easily from those of $s(k)$. The result is shown in Fig. 12. Before, we had an all pass filter with a single tap pulse response, now we obtain a low pass filter giving us a high response at frequencies where the

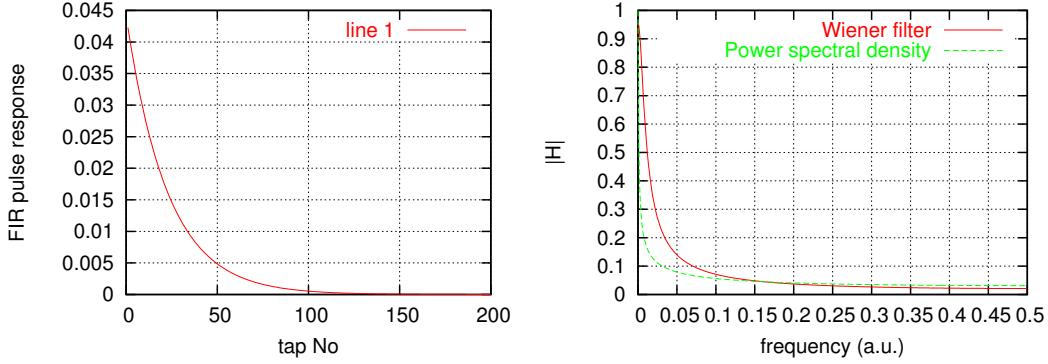


Fig. 12: Predictor filter assuming basic random walk orbit drift and white noise in measurement. Left: pulse response; right: frequency domain behaviour.

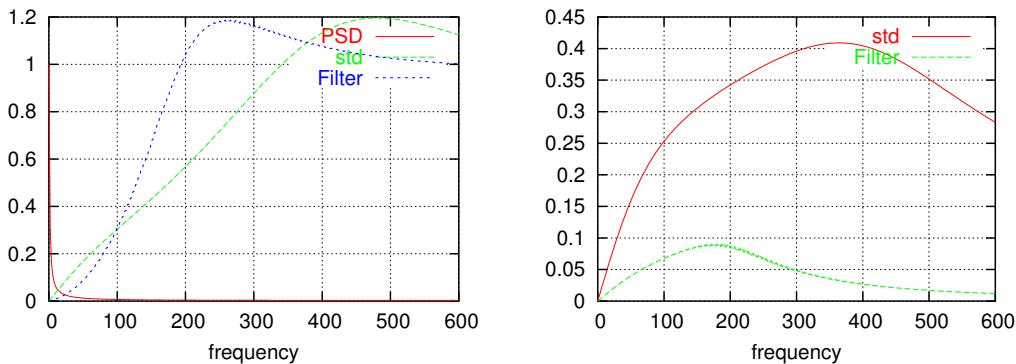


Fig. 13: Closed loop performance with and without basic predictor filters, as shown in Fig. 12. Left: suppression of orbit jitters caused by ground motion, right: additional orbit jitter excited by measurement noise.

signal is high compared to the noise. This does not come for free, as one sees in the spread-out pulse response and increased group delay.

What is the closed loop performance of a feedback using this predictor compared to just taking the measurement value? Figure 13 shows the relevant graphs assuming a PI controller. The prolonged latency of the filter will worsen the effectiveness of the controller in suppressing the effect of ground motion as can be seen in the left graph. The big advantage of the estimator is visible on the right graph which shows how noise in the measurement leads to additional orbit fluctuations in both cases. Orbit noise contains contributions from both sources; the estimator will provide a good compromise in limiting both.

As a second generic example, Fig. 14 shows a predictive filter assuming a random walk, where the ground spectrum is amplified by a mechanical resonance, for example in the girder. Varying the strength of the uncorrelated noise in the measurement, we see a similar effect as before. As we increase the measurement noise, the filter converges to a narrow band behaviour, limiting the influence of the noise, but at a high cost in terms of latency.

One observation should be made here with respect to a real world optimization of the closed loop performance. The objective is to minimize jitters in the estimation $\tilde{s}(k)$ and not in the measurement signal $m(k)$. This approach has some inherent danger. Depending on whether the resonance in Fig. 14 is really a girder resonance or, for example, a measurement artefact (and consequently, how we determine the filter coefficients), the predictor will enhance or suppress it, so that the result may be well off the

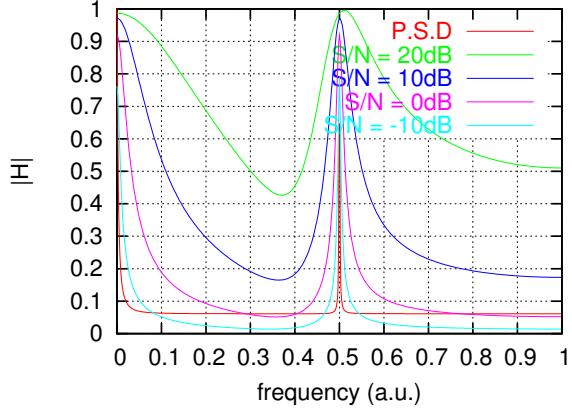


Fig. 14: Predictor filters assuming random walk and girder resonance for various noise levels in the measurement

optimum. A good and clean assessment of the system and its excitations are a prerequisite for the successful use of this technique.

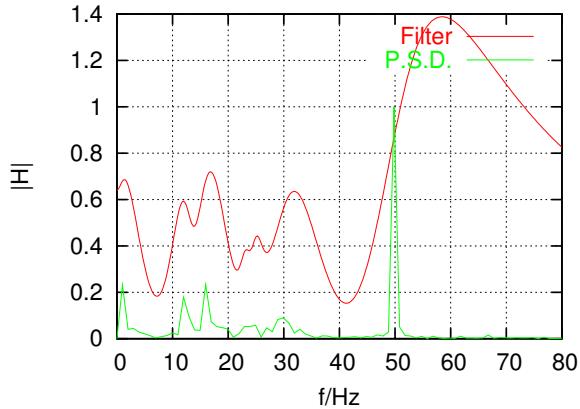


Fig. 15: Predictor generated from power spectral density of beam at SLS

As the last example, we look at a filter generated for real world data. Figure 15 show the power spectrum and the filter characteristics for beam jitter data seen at the Swiss Light Source. As seen before, the filter removes the peaks in the power spectral density; nonetheless the characteristic shown is not the optimum for the task. We have non-random components in the signal such as the 50 Hz line (which is due to magnetic stray fields of the power lines) or the 3 Hz line, which corresponds to the SLS booster cycle. These are completely predictable and should be handled by feed-forwards. Not visible in the spectrum are artificial lines in the spectrum showing up from other sources via cross-talk and interference, which also have to be handled separately. More on this topic is shown in the section on bunch-by-bunch feedbacks later.

3 Tune and chromaticity feedbacks

Quadrupole magnets show focusing effects only in one of the transverse planes; beam offsets in the other will be magnified. By setting a horizontally defocusing quadrupole into the focal point of a horizontally focusing one of equal strength, one obtains a net focusing action of the whole assembly. Applying this principle simultaneously for both planes, we get an arrangement of magnets called a FODO lattice. Other magnet types such as sextupoles or octupoles may be included in the design to correct for aberrations in

the optics. A detailed introduction to beam optics can be found in Ref. [13].

The focusing strength not only controls the transverse dimensions of the circulating bunches, but also affects the trajectories of bunches, which are off-axis. A bunch at an offset will oscillate transversally until damping effects due to synchrotron radiation cause it to settle down to the reference trajectory. The number of oscillations per machine turn is called the betatron tune Q and is proportional to the focusing strength of the machine.

The deflection of a particle due to magnetic fields is inversely proportional to its energy. Correspondingly the tune will vary with beam energy. Analogous to optics, this parameter is called the chromaticity Q' . Each ring optic has a natural chromaticity, which can be influenced by additional sextupole magnets. As with the tune values, we may have different chromaticities for each plane.

For a storage ring running at a constant energy and having a working orbit stabilization, we can expect only minor and very slow variations in tune and chromaticity. The situation is different if the ring is ramped in energy. Non-linearities of the magnets lead, even for a gentle ascent of the magnet currents, to fluctuations in the beam optics.

The control of the tunes and their associated parameters involves several aspects. The first is doing a clean measurement of the tunes, the second setting up a feedback. Third comes accounting for higher order effects due to tune coupling in the horizontal and vertical planes and the interaction with chromaticity measurement and feedback, which typically leads to a hybrid system design.

3.1 Tune measurement

In principle, all techniques of measuring the tune look at the oscillation frequency of particles around the design trajectory. The most direct way consists in kicking the bunches and doing a Fast Fourier Transform on the BPM data. This has several drawbacks. The beam quality (emittance) gets worse. The kick needs to be strong enough to give a visible signal, so the resultant orbit excursions cause particles in the beam halo to get lost and produce radiation—an effect which is of special concern for high-energy hadron colliders like RHIC or LHC. The latency of the measurement is mainly given by the rate by which the beam is kicked. A short latency, required for good feedback operation, means a high kick rate, but conflicts with the need for low emittance and particle loss rates.

An alternative relies on the fact that the charge distribution in a bunch is never 100% homogeneous—it consists of discrete, randomly distributed particles, so that we always have residual inhomogeneities exhibiting so-called Schottky or shot noise. These inhomogeneities oscillate with the tune frequency and can be measured directly in base band with Schottky monitors [14]. There is no external excitation to the beam, so this is the ideal measurement in terms of emittance dilution and the effect on beam loss rates. The headache is that we derive our measurements from noise spectra, so the signal-to-noise ratio may vary strongly depending on the initial condition of the injected beam and the machine settings. For low levels of Schottky noise, a relatively long measurement and measurement latency may be needed to obtain the required resolution, causing problems for the feedback.

The third and last option, which is nowadays used for feedback applications [15, 16] and which is described in more detail here, is to run an active phase-locked loop on the beam using fast kickers and pickups.

The typical structure is shown in Fig. 16. A numerically controlled oscillator (NCO) feeds a sinusoidal signal into a kicker deflecting the beam. The transverse beam oscillation has the maximum amplitude at the tune frequency. At this frequency, the motion is in phase with the excitation driving the kicker, whereas we have a phase difference approaching ± 90 degrees off crest. A phase detector compares the beam signal coming from a pickup with the reference from the NCO. The resultant phase difference signal is fed into a controller steering the frequency of the NCO.

The most basic phase detector would be a simple mixer followed by a low pass filter. With the

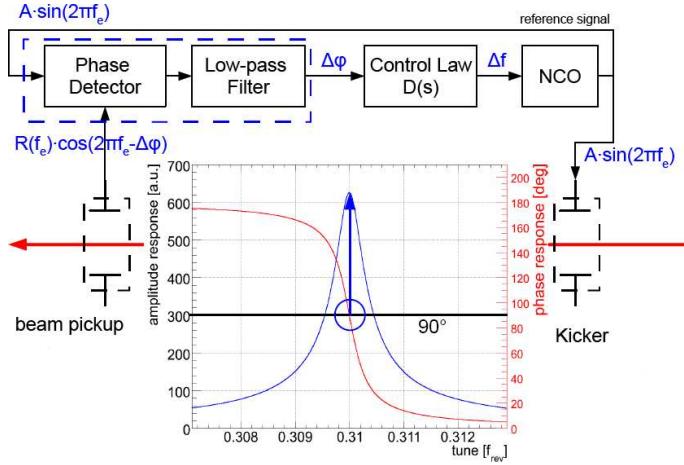


Fig. 16: Classical setup for tune measurement using a PLL loop

beam response

$$H(\omega) = A(\omega) * e^{-j\phi(\omega)},$$

we get

$$\begin{aligned} z_r &= A(\omega) \underbrace{\cos(\omega t)}_{\text{NCO signal}} \cdot \underbrace{A(\omega) \cos(\omega t + \phi(\omega))}_{\text{beam response}} \\ &= \frac{A(\omega)}{2} \left(\underbrace{\sin(\phi(\omega))}_{\approx \phi(\omega)} + \underbrace{\sin(2\omega t + \phi(\omega))}_{\text{removed by low pass}} \right). \end{aligned}$$

This detector can be implemented in a relatively simple and robust way using analog circuitry. The disadvantage are non-linearities arising from the linearization of the sine argument and the amplitude dependency of the beam response, which can lead to reduced performance and even chaotic behaviour of the PLL loop [17].

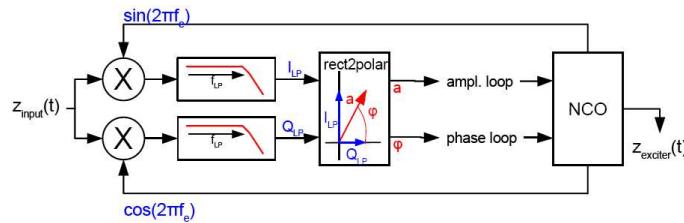


Fig. 17: Advanced PLL detector decoupling phase and amplitude as used at the LHC

An advanced digital detector design used in the LHC [18] which avoids these problems is shown in Fig. 17. The beam signal is split into two parts, which are mixed in phase and out of phase with the signal from the NCO followed by a low pass filter. A rectangular-to-polar converter generates pure amplitude and phase signals. In addition to the phase-locked loop controlling the frequency and obtaining the tune, a second controller keeps the amplitude constant. In the case of the LHC design, the controller used is a PI design, which was optimized with Youla's affine parametrization method [19], which is also described in the Appendix. The bandwidth of the measurement is of the order of 8 Hz.

If we compare the PLL measurement to the kicked excitation, a much smaller amplitude of the beam oscillation is required to keep the PLL locked. At prototype tests at the SPS ring at CERN, amplitudes well below $1 \mu\text{m}$ were required, while the emittance blow up and particle losses due to the measurement were negligible.

3.2 Full system and extensions

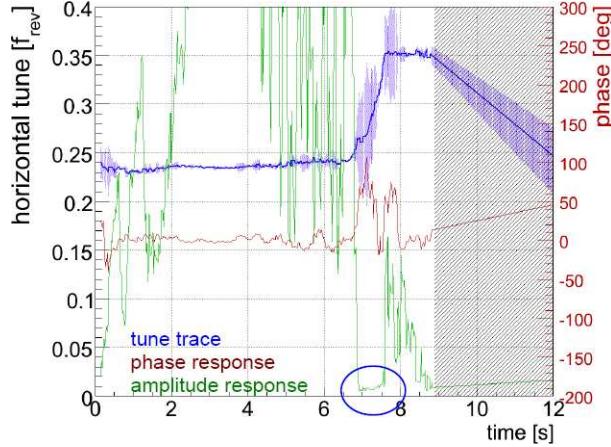


Fig. 18: Example of tune stabilization using PLL at the SPS/CERN

The closed loop performance of a tune feedback using the PLL is demonstrated in Fig. 18. Here a prototype was tested in the energy ramping cycle at the SPS, where within roughly seven seconds protons get accelerated from 26 GeV to 450 GeV. Apart from the tune, the output from the phase and amplitude control loop is also shown. Monitoring these value gives a good indication of whether the PLL is still locked. The region where the lock is lost (no more beam due to extraction) is marked with the blue circle.

The coupling of the beam motion between the horizontal and vertical planes can obscure the measurement, especially if, at certain times, both tunes are nearly equal, as happens for example during the ramp in RHIC [20]. In that case, exciting the bunches at the tune will result in a mixed vertical and horizontal motion. Also, the residual coupling leads to a perturbation of the tune values, decreasing the accuracy of the measurement. So one has to go one step further and use a hybrid feedback which simultaneously controls both tunes and coupling.

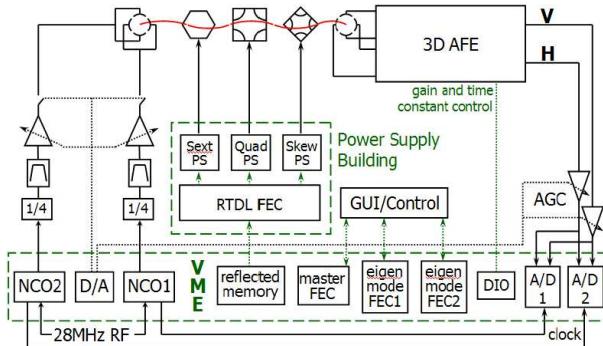


Fig. 19: Combined tune and coupling feedback at RHIC

The layout is shown in Fig. 19. We have phase-locked loops (analog front end AFE, A/D converter, numerically controlled oscillators NCO1 and 2) similar to that shown locking to the beam resonances,

which correspond to the perturbed tunes. Data processing for the individual planes is done by dedicated front end computers FEC1 and 2; a master FEC computes the required corrections, which are applied by a fourth computer to the sextupoles, quadrupoles, and skew quadrupole families.

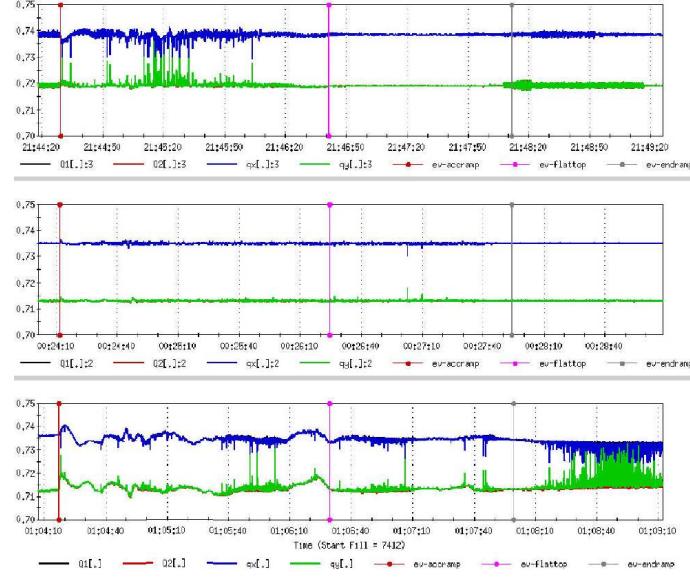


Fig. 20: Set and measured tunes of combined tune and coupling feedback at RHIC; uppermost trace: feedback on; middle trace: feedback on plus feed-forward on magnets; lower trace: feedback off

Figure 20 demonstrates the performance of the system for a 100 GeV development ramp done over five minutes. The lowest set of traces shows the tunes with the feedback switched on. In the upper-most set, the feedback is switched on showing a marked improvement.

As presented in the introduction, feedback should be used only to correct for non-predictable perturbations. Fluctuations which are predictable either due to their physics or because they are periodic like in an accelerator cycle are best handled by a feed-forward system. The middle set of traces are a nice demonstration of this. Magnet correction values from previous ramps were used to update an adaptive feed-forward system. The feedback itself has to perform only minor corrections, the result is an excellent stability of the system.

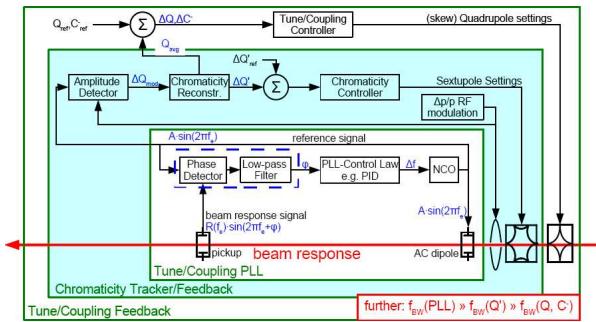


Fig. 21: Combined tune and chromaticity feedback at the LHC

For the LHC, a hybrid system is planned stabilizing all tune, coupling, and chromaticity. As shown in Fig. 21, it consists of a cascade of three different feedbacks. The innermost part is the PLL loop used to measure the tune. By modulating the RF, and measuring tune versus longitudinal momentum, the average

tune and the chromaticity are obtained and used for controlling the sextupole magnets. The measured tunes are used to compute the coupling and the unperturbed tunes and fed into a third controller steering the normal and skew quadrupole magnets.

It is clear that coupling between the nested feedbacks can be a problem, especially since the tune feedback would minimize the momentum driven modulation of the tune and render the chromaticity measurement useless. So the highest bandwidth of 8 Hz is assigned to the tune PLL, followed by one of circa 1 Hz for the chromaticity feedback. The tune feedback is the slowest one.

4 Fast data-processing algorithms for a longitudinal bunch-by-bunch feedback

Bunch-by-bunch feedback algorithms are covered in depth in Ref. [21], the idea for this section is to cover certain data processing features of these high-speed systems and keep the general description to a minimum.

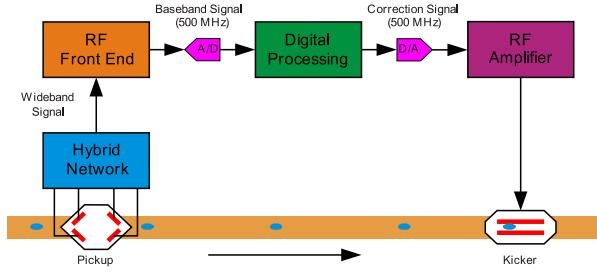


Fig. 22: Layout of bunch-by-bunch feedback system

The basic layout is shown in Fig. 22 (Ref. [22]). The effects to be corrected are unstable longitudinal oscillation driven by trapped resonances in the vacuum chamber, for example. For this application, the beam can be seen as resonant at the synchronous sidebands of the revolution harmonics of the accelerator.

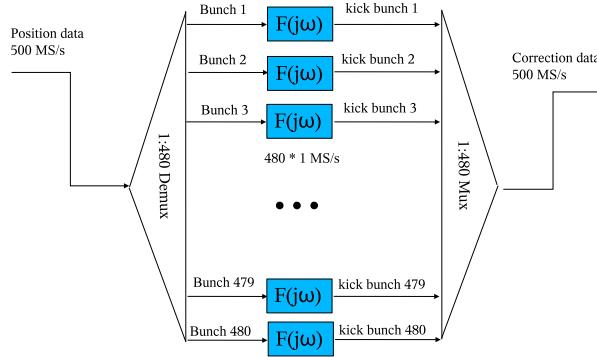


Fig. 23: Multiplexing and demultiplexing of data streams

The feedback is a single-input single-output (SISO) system with the special challenge that there are a large number of potentially instable resonances (e.g., 480 for the Swiss Light Source, 3564 for the LHC). By multiplexing the incoming signal into multiple channels (Fig. 23), each containing the data for one individual bunch, the problem can be simplified considerably. Owing to the inherent down sampling, all the resonant sidebands of the full signal get mapped to the same frequency for the single-bunch data stream, something which facilitates the controller design considerably. It may be tempting to see this approach as a decomposition of a decoupled state similar to using the singular-value decomposition

method for orbit stabilization. But the instabilities lead to a coupling of bunch motion, so this is not the case.

4.1 Signal and noise

Using the example of the SLS parameters, let us take a look at signal and noise levels. The detector works at a centre frequency of 1.5 GHz running at a band width of $B = 500$ MHz. Omitting for the moment cross-talk, beam harmonics, and other external noise sources, we are left with the thermal noise floor, which at a temperature of $T = 300$ K comes out as

$$N = k_B T B = -77 \text{ dBm} .$$

The RF front end may add another 6 dB, so that we end up with a noise power of -71 dBm.

As for the signal, we are not interested in static phase offsets, but in the instable oscillatory part. This is equivalent to a phase modulation of the beam signal and seen in the frequency domain as side bands of the revolution harmonics. The signal picked up by the RF front end is

$$V(t) = A e^{j(\omega t + \phi \cos \omega_s t)} , \quad (18)$$

where ω is one of the revolution harmonics and ω_s is the synchronous frequency. For small oscillations, we can approximate

$$A e^{j(\omega t + \phi \cos \omega_s t)} = A e^{j\omega t} e^{j\phi \cos \omega_s t} \approx A(1 + j\phi \cos \omega_s t) e^{j\omega t} .$$

Applying standard trigonometric identities, we obtain three frequencies, the beam harmonic at ω with amplitude A and the two side bands $\omega \pm \omega_s$ with amplitude $A\phi$. A beam of 400 mA creates a main peak of -1 dBm, the noise floor of approximately -71 dBm corresponds to a phase noise of

$$\Delta\phi = 10^{-70/20} \text{ dB rad} = 0.3 \text{ mrad} \sim 2 \text{ ps at } 1.5 \text{ GHz} .$$

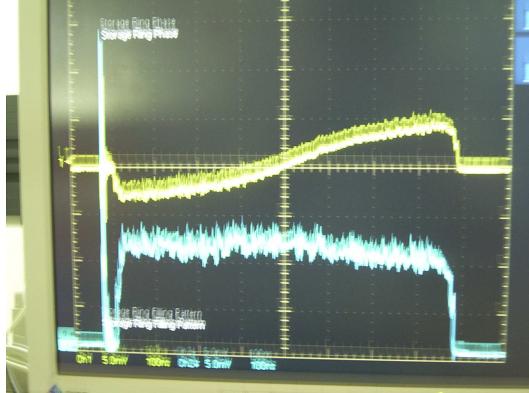


Fig. 24: Synchronous phase (yellow line) for fill pattern with gap (blue line), data from the SLS

On the analog side, everything looks fine for a successful conversion to digital and, for a homogeneous fill pattern with bunches in all RF buckets, this would hold true. But typically an inhomogeneous filling as in Fig. 24 is used, employing a gap to avoid accumulation of ions near the beam trajectory. As this inhomogeneous pattern circulates around the ring, the voltage in the cavity gets modulated by the fill pattern, so that the synchronous phase of the individual bunch will also vary from the reference phase. The result is that the ADC range has to be adjusted not only to cover the amplitudes of any dynamic

oscillations coming from instabilities, but also the static phase offsets. We end up with a pronounced decrease in resolution and a corresponding increase in quantization noise.

To make the picture complete, there are still other noise sources like truncation noise (due to the finite precision of the computations done in the controller) or white noise contributions (coming from the digital-to-analog converter and the power amplifiers). Nonetheless, the essential contribution is due to ADC quantization noise.

Is this critical for the system performance? In terms of a broadband excitation of beam jitter, the answer is essentially no, since the dynamic of the beam exhibits a narrow band-pass behaviour, so that it will react only to components in the noise spectrum close to the synchronous frequency. The big challenge lies in a possible saturation of digital controller, DAC, and the power amplifiers due to noise which can severely compromise the function and efficiency of the feedback. An example of how to set up the filter in terms of algorithm, data flux, and format in a manner that avoids these problems is shown in the next section.

4.2 Low-level processing

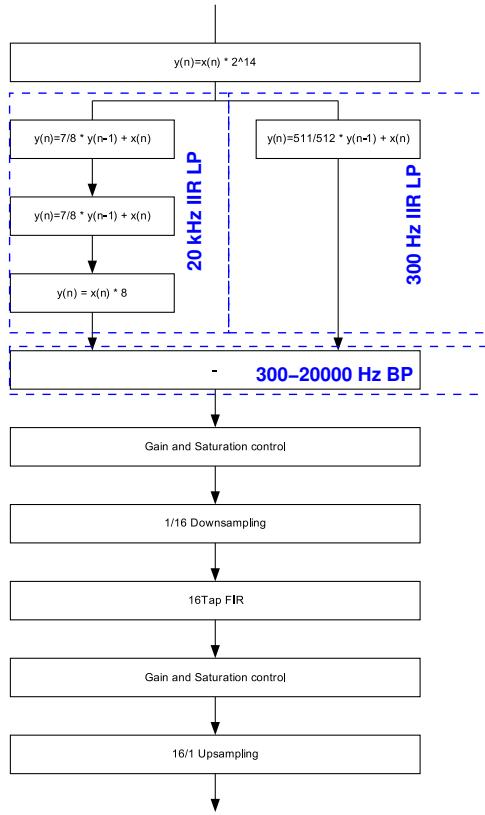


Fig. 25: Basic blocks of the total filter

After the analog-to-digital conversion, we are left with predominantly white noise, spread over the full spectrum. The only way of reducing it is to cut the bandwidth while leaving the signal intact. Given that the signal occupies frequencies in the near vicinity of the synchronous resonance², a band-pass design centred around the synchronous frequency is the most suitable approach. This implementation uses two filter stages. The first is a fixed IIR band-pass implemented so as to minimize internally generated noise,

²For the SLS, for example, we have a few hundred hertz around a synchronous frequency of 2–5 kHz compared with a full bandwidth of 500 kHz.

which cuts the noise and limits the bandwidth from DC –500 kHz to 0.3–20 kHz. Following that, the data rate is reduced by a 1:16 down sampling stage. Following that is the second, user configurable 16 tap filter. Figure 25 shows the layout.

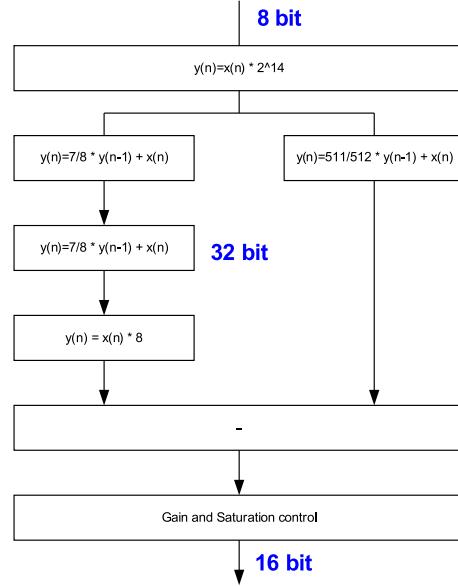


Fig. 26: Introductory bandpass IIR

Figure 26 gives a close look at the introductory IIR filter. The band pass is constituted as the difference of two IIR low-pass systems resulting in a bandwidth of 300–20 000 Hz. The internal computation is done in 32-bit fixed point. Finally a gain and saturation control stage reduces the data format to 16 bit.

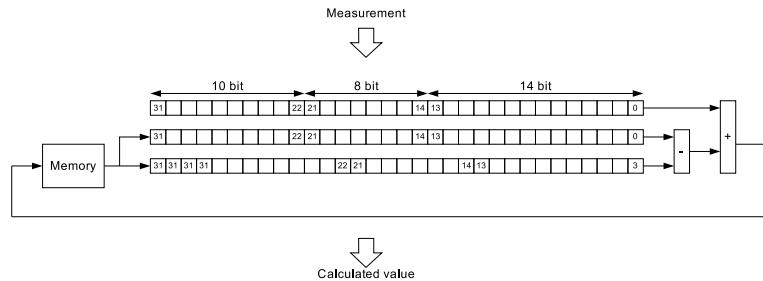


Fig. 27: Implementing the IIR low pass $y_n = \frac{7}{8}y_{n-1} + x_n$ with minimum noise

The low level implementation of one of the stages (shown in Fig. 27) computes the recursion

$$y_n = \frac{7}{8}y_{n-1} + x_n .$$

The input data is inserted into bits 14 to 21 of a 32-bit double word. The multiplication by 7/8 is done by writing the data y_{n-1} into two registers, into the first as is and into the second using a shift by three bits (or divided by eight). Subtracting both gives the required prefactor 7/8. Only the shift operation results in a truncation error in the least significant bit. The maximum gain of this iteration is eight, so the result has significant values in bits 0 to 24.

Putting the result into the second IIR stage fills its output 32-bit word from bits 0 to 27, the following multiplication by eight (realized again as a bit shift) makes use of bits from 0 to 30 without

incurring any noise effects. The IIR low pass in the second branch is done in an analog way. The gain and saturation stage at the end does a user controlled truncation to a 16-bit data format, choosing the right gain will give the optimum balance between saturation and noise.

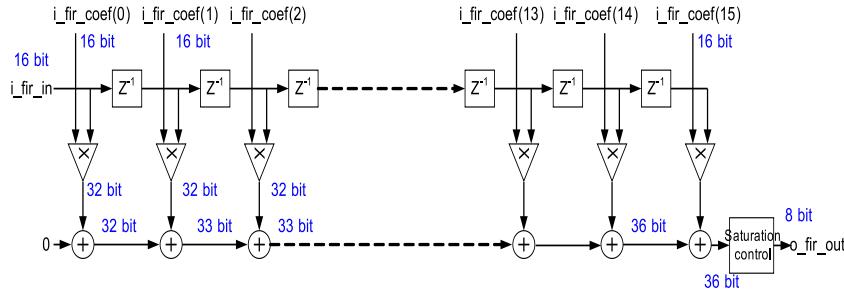


Fig. 28: Second stage: data flux and format of the 16-tap FIR filter

After the down sampling stage, the data rate is low enough to implement the user configurable FIR filter using 16-bit for coefficients and input (Fig. 28). Multiplying coefficients and input data without truncating leads to a 32-bit number. Adding two 32-bit numbers results in a 33-bit size, four 32-bit numbers give a 34-bit and so on until we end up with an untruncated data flow of 36-bits width. A second user configurable gain and saturation stage helps the user find a good balance between saturation and noise.

What remains is how to choose the sixteen coefficients. A quick way, suitable for a manual trial and error optimization, is shown in the next section.

4.3 Quick and easy controllers

A standard controller typically found in textbooks and used for universal application is a PID controller:

$$F(s) = P + \frac{b}{s} + cs .$$

Only three parameters have to be chosen to optimize the control loop—sometimes even the differentiator is omitted reducing these to two. For a bunch-by-bunch feedback, this approach cannot work, the main reason being that the PID controller is designed for operation around DC, but the perturbation is centred at a non-zero frequency. So it is interesting to ask what a corresponding filter looks like for a non-zero frequency ω_0 .

For the proportional part, things are easy. A constant will stay that way also for non-zero frequency, so there are no changes here:

$$F_{DC} = F_\omega = P . \quad (19)$$

Next we have the integrator. In DC, we have a pole at $s = 0$, so the shifted integrator needs to have two at $s = j\omega_0$ and $s = -j\omega_0$, consequently, it should look like

$$F(s) = \frac{1}{s^2 + \omega_0^2} .$$

To be realizable, the corresponding time domain function should be real. If we look in a table of Laplace transforms, we find two candidates

$$\begin{aligned} \frac{s}{s^2 + \omega_0^2} &\iff \cos \omega_0 t \\ \frac{\omega_0}{s^2 + \omega_0^2} &\iff \sin \omega_0 t \end{aligned}$$

which correspond to in-phase and out-of-phase time domain responses and which we can identify with a real and imaginary prefactor. The integrator comes out to be

$$F_\omega = \left(b_r + b_i \frac{\omega_0}{s} \right) \frac{s}{s^2 + \omega_0^2}. \quad (20)$$

The design of the differentiator essentially follows the same pattern. Instead of a DC zero $s = 0$, we need two zeros at $s = \pm j\omega_0$. Again, there is an in-phase and an out-of-phase candidate, so that we have a real and imaginary prefactor giving us

$$F_\omega(s) = \left(c_r + c_i \frac{s}{\omega_0} \right) \frac{s^2 + \omega_0^2}{s}. \quad (21)$$

The corresponding time domain response is a pair of opposing pulses combined with a step function.

Table 1: Comparison of filter functions for PID controller centre on DC and offset at ω_0

Type	PID at DC	PID at ω_0
P	$F(s) = A$	$F(s) = a$
I	$F(s) = B \frac{1}{s}$	$F(s) = \left(b_r + b_i \frac{\omega_0}{s} \right) \frac{s}{s^2 + \omega_0^2}$
D	$F(s) = Cs$	$F(s) = \left(c_r + c_i \frac{s}{\omega_0} \right) \frac{s^2 + \omega_0^2}{s}$

An overview of the resulting functions is shown in Table 1. Where a standard PID controller has three parameters, we now specify five. To do the digital implementation, the values of the pulse response at discrete times are used to define the coefficients of an equivalent FIR filter. As an alternative, a combined FIR/IIR design is possible. A description for a similar controller used for low-level RF control can be found in Ref. [23].

4.4 Applying predictive filters

The purpose of the feedback is not to suppress wide band noise, but to shift the instable resonances back into a region with positive damping. We are only interested in the response of the predictor in the vicinity of the resonance—multiplied with an appropriate scalar factor, the predictor can be used directly as a controller.

What is the measurement and what is the signal or state we are looking for? The power spectral density of the longitudinal momentum is described by a resonance at the synchronous frequency with positive or negative damping coefficient. Since the bunch oscillates in the longitudinal phase space, measurement (the bunch phase) and state (the bunch momentum) are at 90 degree phase offset with respect to each other. In addition, the measurement m contains white noise, giving us the following power spectra (while neglecting the damping terms):

$$\begin{aligned} S_m(j\omega) &= \frac{A\omega^2}{(\omega^2 - \omega_s^2)^2} + N \\ S_s(j\omega) &= \frac{A}{(\omega^2 - \omega_s^2)^2}. \end{aligned}$$

The resulting 16 tap filters in Fig. 29 show the typical behaviour. As the measurement becomes noisy, the controller converges more and more to a band pass.

In a real machine, we are not only going to observe the synchrotron tune obscured by white noise. The signal also contains static offsets combined with low-frequency noise due to the regulation loops of

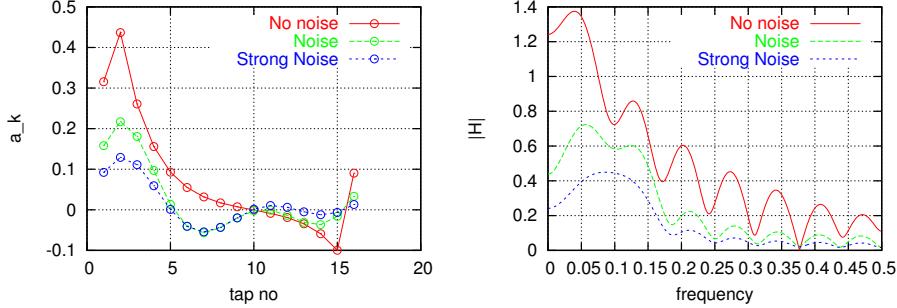


Fig. 29: Pulse and frequency response assuming only synchrotron resonance $f_s = 0.1$ for varying levels of measurement noise

the main RF, for example, something which we can approximate as a stochastic drift process. The naive approach would be to include this as a true signal in the description of the power spectrum of the state. The resulting filter would show a band-pass behaviour near the synchronous frequency as well as a peak near DC to bring out the drift signal. But the drift signal is precisely the one that the feedback should not react to! So, what we do is to define it as being a part of the measurement noise and let the predictor suppress it:

$$S_m(j\omega) = \underbrace{A \frac{1\omega^2}{(\omega^2 - \omega_s^2)^2}}_{\text{synchrotron oscillation}} + \underbrace{B \frac{1}{\omega^2} + N}_{\text{drift/measurement noise}}$$

Figure 30 shows the result. We get a minimum suppressing the drift, as one would expect.

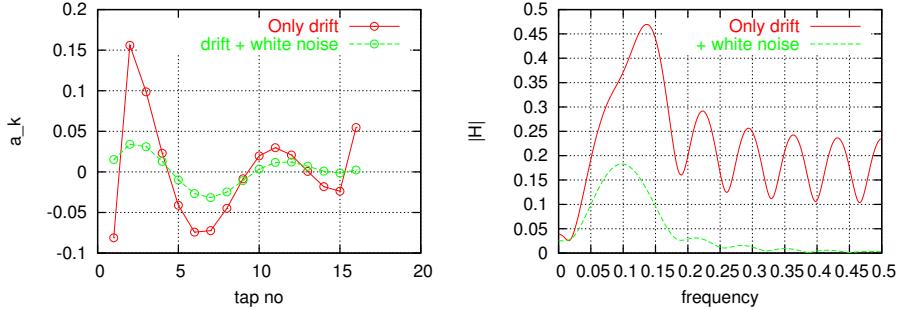


Fig. 30: Using predictor to suppress unwanted drift signals in the spectrum

4.4.1 Accommodating estimation and model errors

In the following discussion, let us take a look at how to handle tolerances in setting up the physical model (e.g., the value of the synchronous frequency) or non-stationary effects. The normal approach would be to use a Kalman filter, but incorporating an adaptive, time-varying filter into a feedback loop could easily lead to unstable closed loop behaviour. Also, if changes are rapid, the adaptive filter may not be able to follow within the required time.

In the context of bunch-by-bunch feedbacks, there is a good example for that. Often, synchrotron light sources have higher harmonic cavities (HHC) built into the ring. These are passive devices excited by the beam. By changing the slope of the RF voltage versus phase seen by the beam, they produce longer bunches with higher stability thresholds with respect to intra bunch oscillations. Since the synchronous

frequency depends on the slope, it also varies with the induced voltage in the HHC. In the case of the SLS, it varies from a standard value of approximately $\nu_s = 5 \cdot 10^{-3}$ down to values of $2 \cdot 10^{-3}$.

If we optimize the feedback only for the current operating value, the following vicious circle can appear: In the beginning, we may have the onset of a longitudinal instability. As the beam is oscillating, the lines in the beam spectrum widen up and drop in height, so the beam excites less voltage in the HHC. As a result, the synchronous frequency changes from the design value and so the feedback becomes less efficient in stabilizing the situation. The beam oscillation grows, the voltage in the HHC drops, the synchronous frequency deviates even more. We end up in a full blown instability. The process happens really quickly, too short for an adaptive filter algorithm to react.

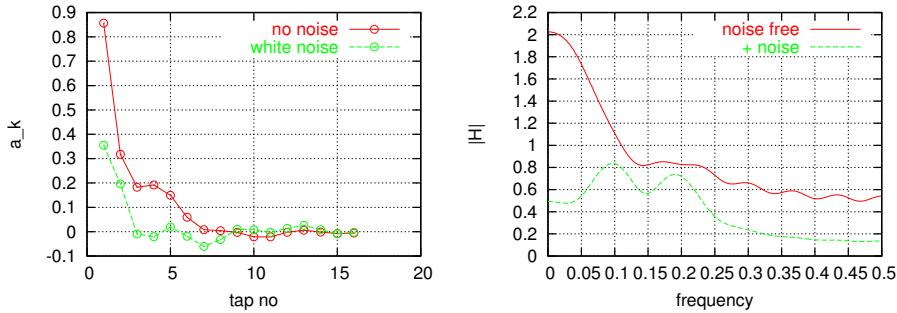


Fig. 31: Including variations in the synchrotron frequency into the predictor ($0.1 < f_s < 0.2$)

So what we could do is to include the expected range of the synchronous frequency (as well as other variables) as a stochastic density in the design process. Staying with our generic examples, we assume a variation of f_s between 0.1 and 0.2 and take a constant density distribution of

$$D_{fs}(f_s) = \begin{cases} C; 0.1 < f_s < 0.2 \\ 0; \text{else} \end{cases}$$

and create an adapted power density to give

$$S_s(j\omega) = \int S(\omega, f_s) D_{fs}(f_s) df_s = \int_{0.1}^{0.2} \frac{CA\omega^2}{(\omega^2 - \omega_s^2)^2} df_s.$$

The resultant filter shown in Fig. 31 takes account of this spread of frequencies. A similar filter to this is actually in use at the SLS bunch-by-bunch feedbacks.

5 Outlook

Accelerator-based feedbacks have come a long way from early analog systems used for basic local stabilization. Several trends are visible. Digital systems allowing for more flexibility have replaced more and more components in the analog signal path. The system architecture went from local systems to ones taking into account cross coupling effects, be it the global orbit or the relationships between tunes, tune coupling, and chromaticity. Digital systems always have the disadvantage of higher latencies compared to analog solutions, so lots of work is devoted to speeding up processing, data flux, and data rates.

Given that we will reach a point of vanishing returns, the next step is to replace the currently used standard controllers by optimized feedback controllers. The successful use of predictor filters or LQR controllers needs a profound knowledge of the accelerator and of the stochastic properties of the processes determining the perturbations to be expected.

Acknowledgements

I would like to thank many people for the information and data provided, especially Ralph Stein-hagen, who provided lots of material on LHC feedbacks, Peter Cameron, Thomas Schilcher and Goran Marinkovic. Special thanks go to Glenn Behrmann for valuable help in hunting down (I hope all) clumsy formulations in the text.

Appendix

A Youla–Kucera parametrization

Finding optimum controllers (with respect to internal noise, transient behaviour and so on) by performing an unrestricted optimization over a whole function space often runs into severe problems. The resultant system may be unstable or the controller not physically realizable. So it is quite attractive to first generate the complete set of controller functions with stable properties, before doing a restricted optimization on this set. One method, which will be shown in the following, is the Youla–Kucera parametrization.

In order to start, we first introduce some terms coming from pure mathematics. A ring R is a set of elements with the following properties:

- On all elements of R , there exists a commutative addition

$$A + B = B + A .$$

- On all elements of R , there exists a commutative multiplication operation

$$A \cdot B = B \cdot A .$$

- An element C having a multiplicative inverse D is called a unit element. Not every element of R needs to be a unit element.

Examples for rings are the following:

- The set of integer numbers \mathbf{Z} .
- The set of proper Hurwitz stable functions, where both nominator and denominator of the fraction are Hurwitz polynomials.

$$R_H(s) = \frac{\sum a_i s^i}{\sum b_i s^i} .$$

- The set of discrete FIR functions

$$R_f(z) = \sum a_k z^k .$$

If all elements in a ring are unit elements, that is, we can define a division for all elements, it is called a field. Examples for fields are the sets of real numbers or the sets of rational functions.

Equations on a ring are the so-called Diophantine equations [24]. An example for this is the search for solutions of

$$A^n + B^n = C^n \text{ for } A, B, C \in \mathbf{Z}, n = 2, 3, \dots$$

which for $n > 2$ is the subject of the famous last theorem of Fermat [25]. For our purposes, we are interested in solutions of the linear equation

$$AX + BY = C , \tag{A.1}$$

where $A, B, C, X, Y \in \mathbf{R}$, A, B, C are given and we are looking for solutions (X, Y) . This equation has the following interesting property. Assume, we have already found one pair X' and Y' solving Eq. A.1. Then we can create other pairs by setting

$$X = X' + BW \tag{A.2}$$

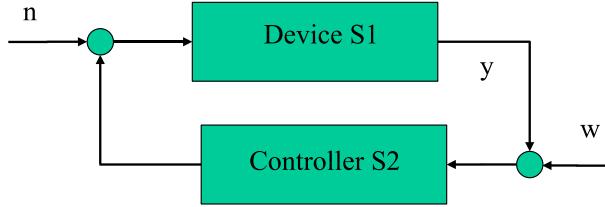


Fig. A.1: Simple control loop

$$Y = Y' + AW, W \in \mathbf{R} \quad (\text{A.3})$$

which will solve the equation. What is more, this method will create all solutions in \mathbf{R} .

How do we use that rather abstract result in the context of designing controllers? For that, let us take a look at the simple feedback schematic in Fig. A.1. We assume that we can write both device and controller transfer functions as fractions of Hurwitz stable functions $R_H(s)$:

$$\begin{aligned} S_1(s) &= \frac{B(s)}{A(s)}, \\ S_2(s) &= -\frac{Y(s)}{X(s)}. \end{aligned}$$

With this representation, the closed loop behaviour of the parameter $y(s)$ is given as

$$y(s) = \frac{BX}{AX + BY}n(s) + \frac{-BY}{AX + BY}w(s) \quad (\text{A.4})$$

and a condition for $y(s)$ to be stable is that the denominator $AX + BY$ be a Hurwitz stable function.

Now let us assume that we already know one stabilizing controller, which we again decompose into a fraction:

$$S'_2(s) = -\frac{Y'(s)}{X'(s)}. \quad (\text{A.5})$$

Using this particular solution and arbitrary functions $W(s) \in \mathbf{R}_H(s)$, we can generate a whole set of solutions

$$\begin{aligned} X(s) &= X'(s) + B(s)W(s) \\ Y(s) &= Y'(s) - A(s)W(s), \end{aligned}$$

since, according to our Diophantine equation above

$$AX + BY = AX' + BY' \in \mathbf{R}_H(s).$$

So the parametrization of all stabilizing controllers will take the form

$$S_2(s) = -\frac{Y'(s) - A(s)W(s)}{X'(s) + B(s)W(s)}, \forall W \in \mathbf{R}_H(s). \quad (\text{A.6})$$

A.1 Example

As a simple example, we assume the following characteristics for device and controller. The device is proportional with unity gain, so that we can write it simply as

$$S_1(s) = 1 = \frac{1}{1} = \frac{A(s)}{B(s)}.$$

As a particular case of a stabilizing controller, we assume an integrator

$$S_2(s) = -\frac{1}{s}.$$

When decomposing the transfer function into a fraction, we need both nominator and denominator to be Hurwitz stable, writing for example:

$$S_2(s) = -\frac{1}{s} = -\frac{\frac{1}{s+1}}{\frac{s}{s+1}} = -\frac{Y'(s)}{X'(s)}.$$

Now the parametrized controller can be written as

$$S_2(s, W(s)) = -\frac{\frac{1}{s+1} - W(s)}{\frac{s}{s+1} + W(s)}, \forall W(s) \in \mathbf{R}_H(\mathbf{s}),$$

using for example

$$W(s) = \frac{1}{s+a}; a > 0.$$

A possible disadvantage of this approach is, that the parametrization is in terms of a ring of functions and not functional space like a Hilbert space, for which we have lots of techniques for optimization and synthesis. Nonetheless, one is typically looking for controller designs of a low order and complexity, so that the effort stays manageable.

As an alternative, we can use the same approach with controller and device transfer functions exchanged. As a result, we obtain a parametrization of all devices, which can be stabilized by a given controller. Comparing these with the expected variations in the device behavior due to tolerances and drifts gives a valuable criterion for the robustness of the chosen controller.

References

- [1] G.H. Golub and C.F. Van Loan, *Matrix Computations* (Johns Hopkins University Baltimore Press, 1989).
- [2] http://web.mit.edu/be.400/www/SVD/Singular_Value_Decomposition.htm
- [3] T. Wijnands *et al.*, Requirements for real-time correction of decay and snapback in the LHC superconducting magnets, Proc. of the EPAC 2000, pp. 367–369, Vienna, Austria, 2000.
- [4] M. Boege *et al.*, Fast closed orbit control in the SLS storage ring, Proc. of the 1999 Particle Accelerator Conference, pp. 1129–1131, New York, USA, 1999.
- [5] M. Dehler, Eddy current calculations for the SLS corrector magnets, SLS note SLS-TME-TA-1998-0004, Villigen-PSI, Switzerland (1998).
- [6] Vladimir Shiltsev, Space-time ground diffusion: The ATL law for accelerators, Proc. of the 4th IWAA95, Nov. 14–17, 1995, KEK, Japan. IV 352.
- [7] S. Redaelli *et al.*, Vibration measurements at the Swiss Light Source (SLS), Proc. of the EPAC 2004, Lucerne, Switzerland, pp. 2278–2280, 2004.
- [8] B.D.O. Anderson and J.B. Moore, *Optimal Control: Linear Quadratic Methods* (Prentice Hall, Englewood Cliffs, NJ, 1990).
- [9] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*, 3rd ed. (McGraw-Hill, New York, 1991).
- [10] N. Wiener, *Extrapolation, Interpolation, and Smoothing of Stationary Time Series* (MIT Press, Cambridge, MA, 1950).
- [11] N. Levinson, The Wiener RMS error criterion in filter design and prediction, *J. Math. Phys.* 25 (1947) 261–278.

- [12] R.E. Kalman, A new approach to linear filtering and prediction problems, *ASME Trans.* vol. 82D (1960) 35–45.
- [13] A. Hofmann, Single particle dynamics 1 – Basic phase space, in *Beam Measurement*, Proc. Joint US – CERN – Japan – Russia School on Particle Accelerators, ISBN 981-02-3881-9, 1999.
- [14] F. Nolden, Instrumentation and diagnostics using Schottky signals, Proc. 5th European Workshop on Beam Diagnostics and Instrumentation for Particle Accelerators DIPAC 2001, pp. 6–10, Grenoble, France, 2001.
- [15] P. Cameron *et al.*, PLL tune measurement during RHIC 2001, Proc. European Particle Accelerator Conference EPAC 2002, pp. 1855–1857, Paris, France, 2002.
- [16] R. Steinhagen, Feedbacks on tune and chromaticity, Proc. 8th European Workshop on Beam Diagnostics and Instrumentation for Particle Accelerators DIPAC 2007, Venice, Italy (2007).
- [17] B. Harb *et al.*, Chaos and bifurcation in a third order phase locked loop, *Chaos, Solitons and Fractals* 19 (2004) 667–672, doi:10.1016/S0960-0779(03)00197-8.
- [18] R. Steinhagen, Real time feedback on beam parameters, Proc. Fourth Asian Particle Accelerator Conference APAC 2007, Indore, India (2007).
- [19] D.C. Youla *et al.*, Modern Wiener–Hopf Design of Optimal Controllers, *IEEE Trans. Automat. Control*, 1976, Vol. 21, No. 1, pp. 3–13 (part I) and pp. 319–338.
- [20] P. Cameron *et al.*, *PRST-AB* 9 (2006) 122801.
- [21] M. Lonza, Multibunch feedback systems, these proceedings.
- [22] M. Dehler *et al.*, State of the SLS multi bunch feedbacks, Proc. Fourth Asian Particle Accelerator Conference APAC 2007, Indore, India, 2007.
- [23] L. Doolittle *et al.*, Digital low-level RF control using non-IQ sampling, Proc. LINAC 2006, pp. 568–570, Knoxville TN, USA, 2006.
- [24] V. Kucera, Diophantine equations in control - a survey, *Automatica* 29 (1993) 1361–1375, UTIA AV CR. research report No. 1778, Academy of Sciences of the Czech Republic, 1993. Also to be found at <ftp://ftp.utia.cas.cz/pub/reports/utia1778.ps.Z>.
- [25] S. Singh, *Fermat's Enigma: The Epic Quest to Solve the World's Greatest Mathematical Problem* (Anchor Books, New York, 1998).

Beam diagnostics

U. Raich
CERN, Geneva, Switzerland

Abstract

Most beam measurements are based on the electro-magnetic interaction of fields induced by the beam with their environment. Beam current transformers as well as beam position monitors are based on this principle. The signals induced in the sensors must be amplified and shaped before they are converted into numerical values. These values are further treated numerically in order to extract meaningful machine parameter measurements. The lecture introduces the architecture of an instrument and shows where in the treatment chain digital signal analysis can be introduced. Then the use of digital signal processing is presented using tune measurements, orbit and trajectory measurements as well as beam loss detection and longitudinal phase space tomography as examples. The hardware as well as the treatment algorithms and their implementation on Digital Signal Processors (DSPs) or in Field Programmable Gate Arrays (FPGAs) are presented.

1 Introduction

1.1 Architecture of an instrument

Even though this CAS school is dedicated to *digital signal processing*, it is still a CERN Accelerator School and as such it should talk about techniques used in accelerators. Digital signal processing is a very technical field of electronic engineering and computer science and many people working in this field are not necessarily experts in accelerator physics. For this reason the lectures on beam diagnostics try to bridge the gap between accelerator physics and its measurements of beam parameters and the purely informatics and electronics aspects of digital signal processing.

Before looking at a few dedicated beam instruments, using digital signal processing principles, in some detail, let us first analyse the architecture of a beam measurement instrument in general. It consists of the following elements:

- The sensor (and maybe actuator) interacting with the beam
- The front-end analog electronics
- Cabling to get the signals from the accelerator tunnel to an equipment area
- Conversion of the sensor signals to digital values
- Data acquisition and control (readout of the digital values)
- Transformation of the acquired values to humanly understandable machine parameter values
- Transfer of the results to the control room
- Graphical representation and interpretation of the results

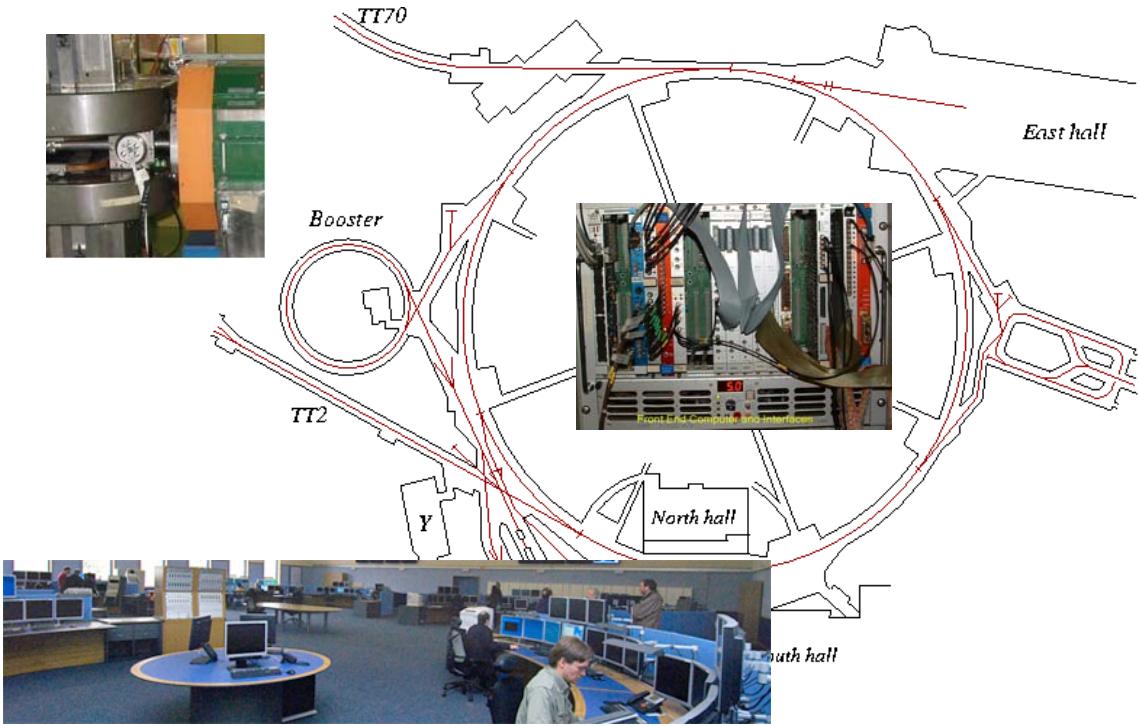


Fig. 1: The architecture of a measurement system

In order to avoid signal loss and capture of electro-magnetic noise, the first-stage electronics must be installed as close as possible to the sensor. On the other hand it may be subject to very high radiation doses (some kGrays). Therefore the front-end electronics is installed as near to the sensor as possible but as far away as necessary to protect the electronics from radiation damage. Typical cable length from the detector to the first stage electronics is a few (1–10) metres. The signal is amplified and shaped in this first stage of the electronics. The amplified analog signals are then transferred to the equipment gallery and digitized. In case of smaller machines like the PS, where the equipment gallery can be reached within some 100 m, the analog signals are transferred to the equipment rack and digitized there. In very big machines, like the LHC, where the distance from the underground tunnel to the equipment galleries at the surface may be 1 km or more, the analog signals are digitized in an equipment gallery in the tunnel und serial transmission is used to transfer the digital values to the surface. The digital results are stored in registers or memories and read out by front-end computers (called Device Stub Controllers (DSCs) at CERN). All front-end computers are connected through a local area network which also connects to operator consoles (PC-type computers with several screens attached to them) from which operators control the machine and from where they have access to measurement results in the form of numbers or graphical representation.

Digital signal processing may come into play at several levels, as soon as the sensor signals have been digitized. This may be the case in the equipment gallery in the machine tunnel for big machines or in the equipment room at the surface, but it may also take place only at the level of the operator consoles. Often the front-end computers are used to do the calculations, sometimes special FPGA or DSP based equipment take over this task, sometimes, for slow applications or very compute intensive ones, the operator consoles or special number-crunching machines execute the DSP algorithms.

Figure 1 shows the layout of a typical instrument in the PS. The sensor, here a beam position monitor, is connected to its analog front-end. The amplified signal is sent to the equipment room with

its VME base front-end computer ~ 150 m away. There the data are read out, pre-treated and then transferred to the main control room at a distance of some 5 km.

2 Tune measurements

As a first typical example, where digital signal processing techniques are used in beam diagnostics, we shall have a look the the machine tune Q .

2.1 What is the machine tune?

When a particle is not exactly on its design orbit it will not pass through the centre of the machine's focusing quadrupoles and therefore it will experience a focusing force, which will lead to so-called betatron oscillations. The number of oscillations around the closed orbit is called the betatron tune. This is true for horizontal as well as for vertical displacements. If the number of oscillations corresponds to an integer value, then each small imperfection in the machine will lead to a small kick, which will be repeated each time the particle passes that point. This leads to a (integer) resonance, which will result in the loss of the beam. It can easily be shown that the same is true for half integer tunes and in fact any particle for which $l \cdot Q_h + m \cdot Q_v = r$ holds where Q_h and Q_v are the horizontal and vertical tune values. The tune diagrams show these lines of instability, see Fig. 2.

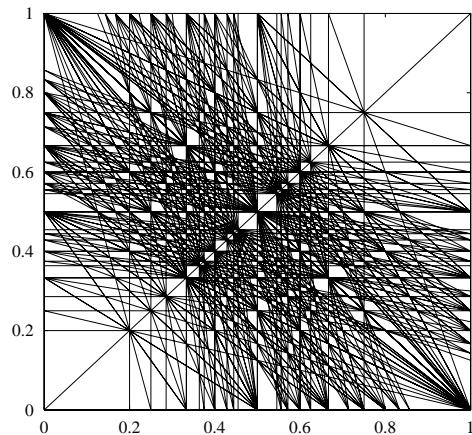


Fig. 2: The tune diagram

The machine must operate in a resonance-free zone of the tune diagram (the working point) Fig. 3, in order to avoid beam blow-up and finally loss of the beam. This shows the importance for a precise determination of the tune values.

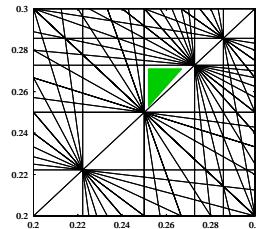


Fig. 3: The working point

2.2 Measuring the tune

In order to measure the tune values, the betatron oscillations must be observed. This can be done by placing beam-position monitors all around the accelerator ring and observing the beam trajectory. Since normally the beam circulates on its closed orbit with very small (unmeasurable deviations) the beam is usually coherently excited with a kicker magnet. As we have seen in the previous section we are actually only interested in the non-integer part of the tune and this can be measured with a single beam-position monitor. In Fig. 4 you can clearly see that it is not possible to distinguish q from $1 - q$ and that, in addition to the base frequency, higher harmonics can be fitted to the data measured by the pick-up.

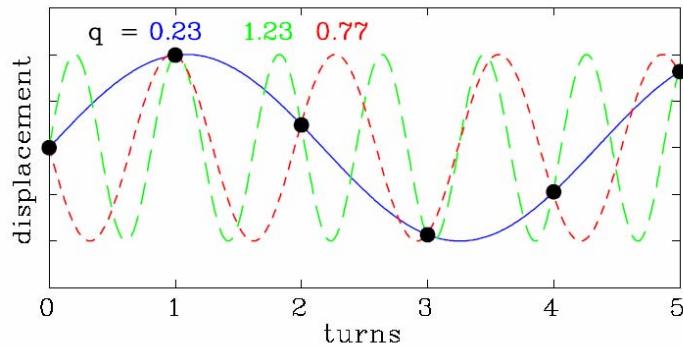


Fig. 4: Measuring tune with a single pick-up

The measurement at the CERN PS and PS Booster works as follows: The beam is excited with a kicker in short intervals (minimum 5 ms) (Fig. 5). The beam position is measured by a pick-up (PU) delivering horizontal and vertical difference signals which are proportional to the beam displacement. The PU is of shoebox type, consisting of a metal box with a diagonal cut. In fact, the PU used here (Fig. 6) combines horizontal and vertical cuts in a single device with four taps to extract the signals from the left/right and upper/lower plates.

A passive hybrid circuit, being virtually insensitive to radiation, is mounted directly on the pick-up (the denominations pick-up (PU) and beam position monitor (BPM) are used interchangeably). The hybrid combines the signals from the PU plates and converts them to difference (Δ) and sum (Σ) signals.



Fig. 5: The tune kicker

During acceleration the beam may undergo slow but rather large closed-orbit variations which are filtered out by a Beam Orbit Signal Suppressor (BOSS).

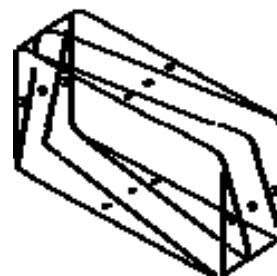


Fig. 6: Pick-up electrodes

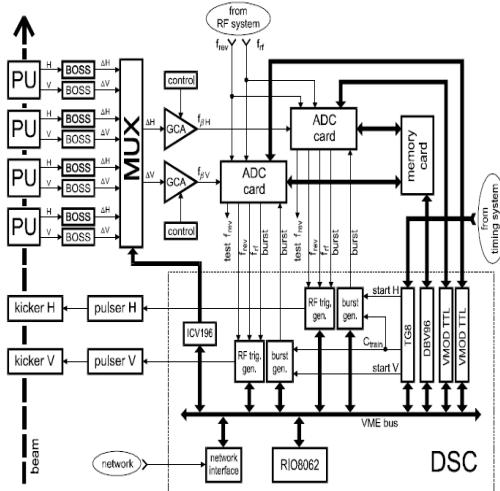
**Fig. 7:** Tune measurement electronics

Figure 7 shows the architecture of the readout electronics at the PS Booster. The Booster actually consists of four vertically stacked synchrotrons. The MUX is used to select the position signals from any of the four Booster rings. The difference signal is not only proportional to the displacement but also to the beam intensity, which may vary by several orders of magnitude. This is taken care of by a Gain Controlled Amplifier (GCA). The signal is converted to digital values with a 14-bit ADC and 2048 converted values are stored in a memory card. A digital signal processor card (DBV96) with a Motorola 96002 Digital Signal Processor accesses these data over a dedicated high-speed memory bus.

The results are stored in DSP memory and can be transferred to the VME processor through a communication mailbox implemented in the DSP. From there the results are transferred over the local area network to the operator consoles in the control room for visualization.

2.3 Calculating the tune

As can be seen from Fig. 4 the frequency of the measured signal is related to the revolution frequency and the oscillation mode through

$$f_\beta = (m \pm q) f_{\text{rev}} , \quad (1)$$

where m is the integer number of oscillations and q is the non-integer part of the tune. Since the integer part of the tune stays constant, only q needs to be measured. This reduces the equation to

$$\frac{f_\beta}{f_{\text{rev}}} = q . \quad (2)$$

As already mentioned, the closed orbit is not centred and stable during acceleration which results in an additional component of the revolution frequency in the signal. A convenient way to calculate q is to sample the PU signal at a rate proportional to the revolution frequency and to perform FFT analysis on N samples. Using a sampling rate of $k * f_{\text{rev}}$ (k is the over-sampling ratio) q can be determined by

$$q = k \frac{n_\beta}{N} , \quad (3)$$

where n_β is the bin number in the FFT spectrum.

2.4 Data handling

After the start of the measurement the beam is kicked (Figs. 8 and 9) and 2048 beam positions are measured and stored in a dedicated memory card. These data are transferred to the DSP and the memory is again freed to take new data. Through this double buffering, data can be taken while the previous batch is processed by the DSP.

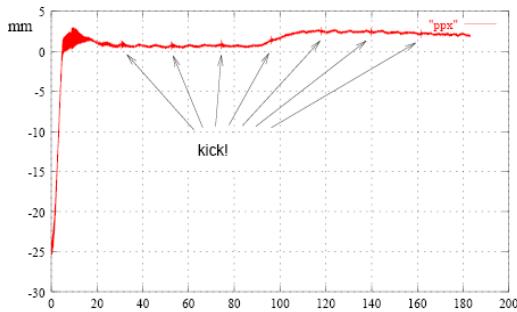


Fig. 8: Kicking the beam

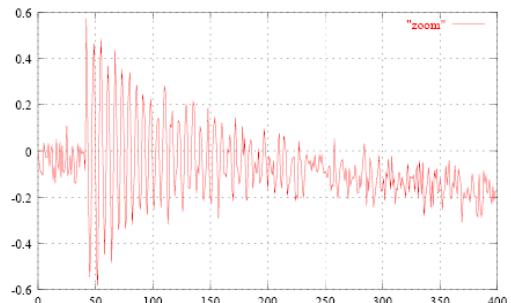


Fig. 9: Zoom of the kick

The Fourier transform expects a signal that constitutes one cycle of a periodic signal. Since our signal is cut out of the stream of oscillations in an arbitrary manner, discontinuities occur when extending it to a periodic wave form (Fig. 10).

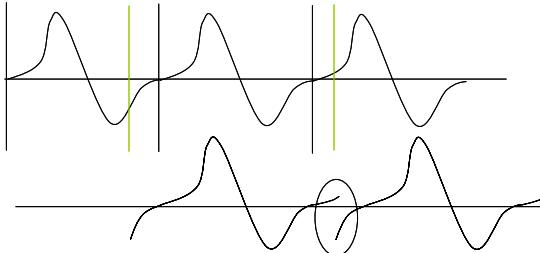


Fig. 10: Periodic extension of the position signal

For the calculation of the Fourier spectrum a 1024 point complex FFT is used, followed by an algorithm to extract the real signal spectrum from the complex spectrum data.

From the Fourier spectrum the q value must be found through a peak search routine. Since the over-sampling ratio in Eq. (3) amounts to 4, only a quarter of the power spectrum needs to be looked at. Q can only take values between 0.1 and 0.5 which means that the search is limited to a window of 50 and 256. The peak search algorithm first looks for the four bins for which

- the power value V^2 is bigger than the next value:

This phenomenon can be corrected by windowing. Each value is multiplied by a coefficient which approaches zero at the edges of the window. Typical windowing functions are shown in Fig. 11. For the tune measurement system the Blackman–Harris function is used. The values are pre-calculated and stored as coefficients in a table at initialization of the DSP.

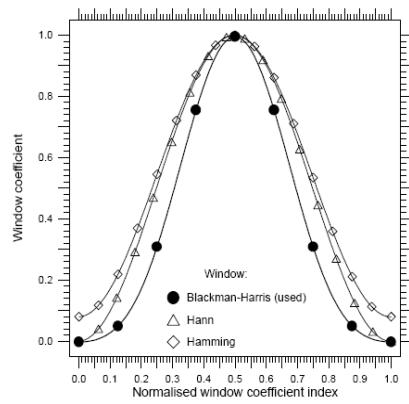


Fig. 11: Windowing functions

$$V^2(n-1) < V^2(n)$$

- the power value V^2 is bigger than the following value: $V^2(n+1) < V^2(n)$
- the power value satisfying the first two conditions is the biggest in the observed range
- the power value is at least three times as big as the arithmetic mean of all power bins. The value 3 has been determined experimentally.

If these conditions cannot be fulfilled then either the signal is too small or the spectrum is too noisy.

Most of the mathematics code for the DSP has been written in the C language except for small parts like the arithmetic mean, which has been implemented in assembler for reasons of speed.

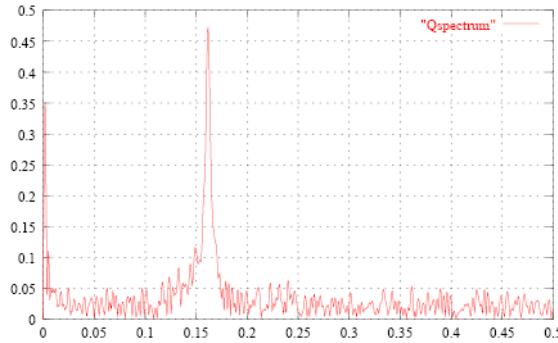


Fig. 12: The frequency spectrum

2.5 Spectrum interpolation

When just taking into account the bin in which the peak is found, the resolution for Q is very limited. In order to improve the resolution one can

- increase the number of points, but this would increase the computing time and therefore slow down the system;
- decrease the over-sampling, but this would lead to a degradation to the system input dynamics;
- interpolate between adjacent bins.

The betatron frequency spans over several bins, not only because of the discrete FFT but also because the incoming signal is not a pure harmonic but includes attenuated revolution frequency components, noise, etc. Interpolation allows one to find the peak position with a resolution better than a bin. The exact form of the peak is unknown but it could be shown that using a parabolic shape, which is easy to calculate, can improve the Q -value calculations significantly.

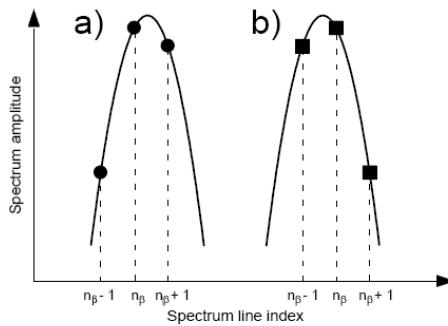


Fig. 13: Tune value interpolation

2.6 Measurement results

During the acceleration cycle the beam is kicked every 5 ms and the Q value is calculated for each of these kicks. A front-end computer reads the results from the DSP and passes them on to an application program in the central control room where the evolution of the tune during the cycle can be observed in graphical form (Fig. 14). The application program also allows the control of the measurement interval, amplifier gains, start of measurement in the cycle and other parameters.

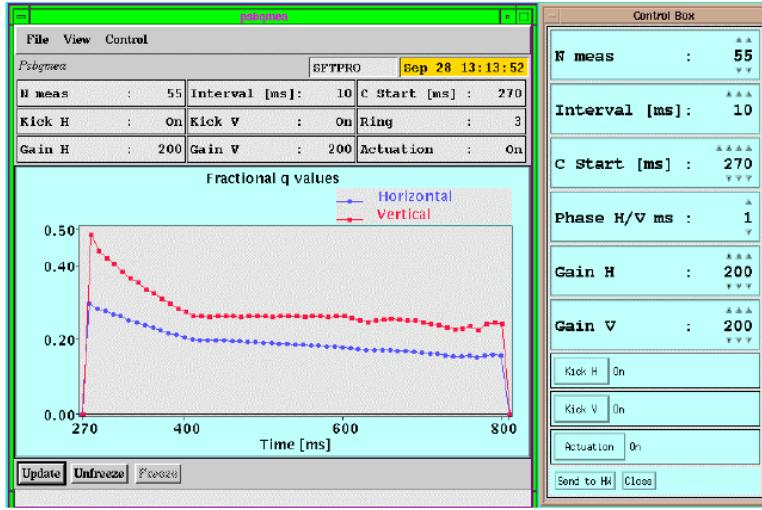


Fig. 14: Tune measurement results

2.7 Further improvements

As can be seen from Fig. 9 the position peaks are very sharp because the beam bunch is short with respect to the revolution frequency. As a result the signal power is distributed over many harmonics in the frequency spectrum. By rectifying the signal with a simple diode and capacitor circuit (see Fig. 15) the signal spectrum can be pushed to the base band and thus the signal power in the tune line to be measured can be significantly increased. This shows that a careful balancing between analog and digital methods is needed.

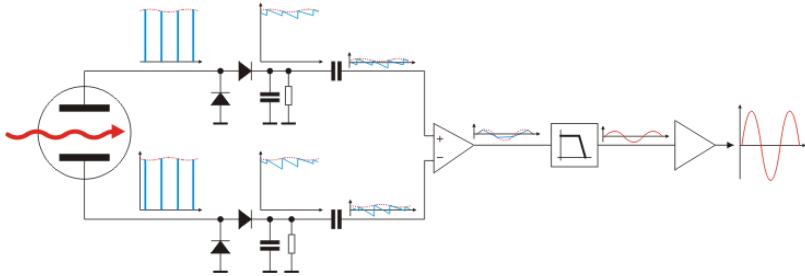
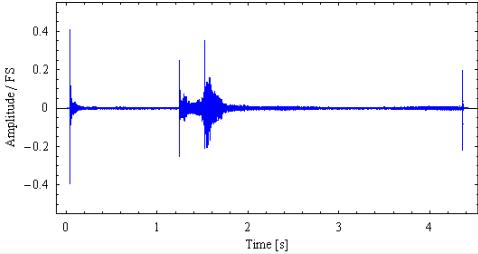
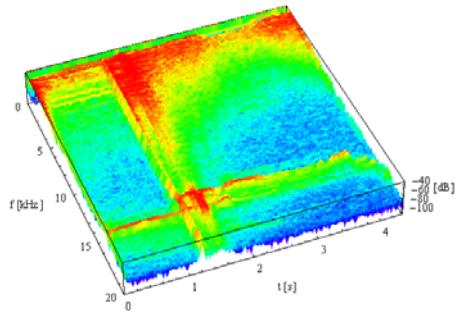


Fig. 15: Base band Q measurement system

Through this trick it was possible to measure tunes in the SPS without artificial coherent excitation of the beam.

Figure 16 shows beam oscillations as they arise during an acceleration cycle in the CERN SPS. The first strong peaks are due to injection, the last peak is created when the ejection kicker is activated. Figure 17 shows a waterfall model of the corresponding frequency spectra. It can be seen that the frequencies fall into the audio range which allows one to use very powerful and cheap audio range which allows one to use very powerful and cheap audio systems for data acquisition.

**Fig. 16:** Natural beam oscillations**Fig. 17:** Waterfall model of frequency spectra

more. This shows the common problem that digital hardware and its associated software tools are evolving much faster than the accelerators which have lifetimes of several decades (the CERN PS was inaugurated in 1955!).

3 Trajectory measurements

3.1 Trajectories and orbits

When particles are injected into a circular accelerator, injection errors may occur, resulting in coherent oscillations around the closed orbit. These oscillations may be caused by beam displacement and/or wrong injection angle and may cause emittance blow-up through filamentation. The oscillations can be seen when beam positions are measured turn by turn. When traversing transition (the moment when due to acceleration the orbit radius starts shrinking instead of increasing) the beam may undergo trajectory changes. This also occurs at ejection. Other effects during acceleration may also influence the trajectories. For this reason it is desirable to be able to measure beam trajectories at any time during the acceleration cycle.

In contrast to the beam trajectory, the beam orbit is the average beam position over several turns in the machine. During acceleration the orbit may move considerably and beam losses may occur due to aperture limitations. It is therefore important to be able to measure the orbit all along the acceleration cycle.

3.2 Synchronization

In order to measure beam trajectories it is necessary to have a large number of BPMs in the machine evenly distributed along the accelerator ring. In the CERN PS there are 40 PUs while in the LHC several thousands will be installed. Since the accelerator can treat several particle-bunches (depending

on the RF frequency) at the same time, each bunch must be measured turn by turn and the position data saved for the whole accelerator cycle. Another, but less flexible method, is to define beforehand the time interval during the acceleration cycle we are interested in and save the data for this interval only.

For low-energy accelerators, where the particle has not yet reached the speed of light, the acceleration will result in a speed increase and therefore in an increase of the revolution frequency. This means that the integration gate, needed for integration of the BPM's Σ and Δ signals must be continuously adapted to the revolution frequency. The current trajectory measurement system installed in the PS can measure one bunch during two turns every 5 ms and has a very complex synchronization system associated to it, following the revolution frequency in order to generate the integration gate supplied to the ADC.

In order to complicate things further, the RF harmonic number (the number of RF cycles per revolution, which is equal to the maximum number of bunches that can be accelerated) can be modified during the acceleration cycle. This allows the splitting of bunches into several bunchlets or the recombination of several bunchlets into one big bunch. In such a case, the number of integration gates must be changed on the fly.

The CERN control system allows several operators at different operator consoles to access data from the same measurement system. For example, one operator wants to check the orbit at injection, while another one wants to see the orbit at transition and this should be possible for the same accelerator cycle.

As can be seen from Fig. 18, the PS can take particles from Linac2 or Linac3 and it can eject particles to experimental areas (East Hall or nTOF), it can create antiprotons for the Antiproton Decelerator (AD), or it can transfer protons to the SPS for fixed-target physics or for injection into the LHC.

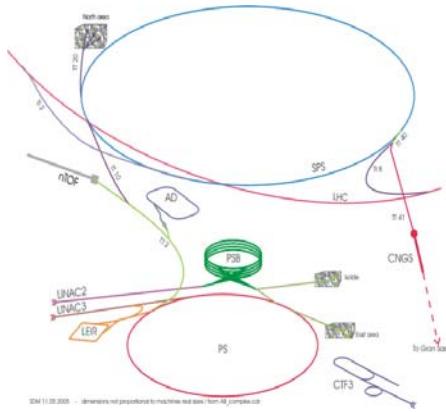
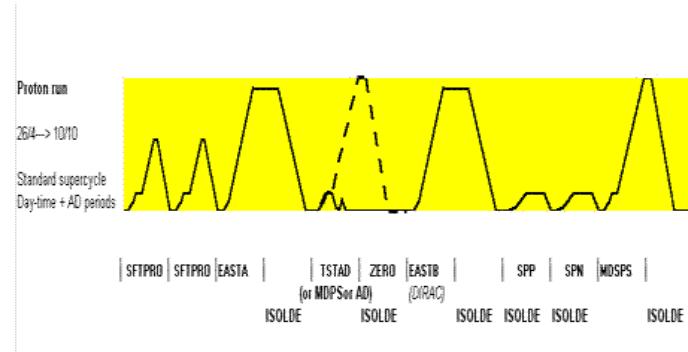


Fig. 18: Different beams in the PS

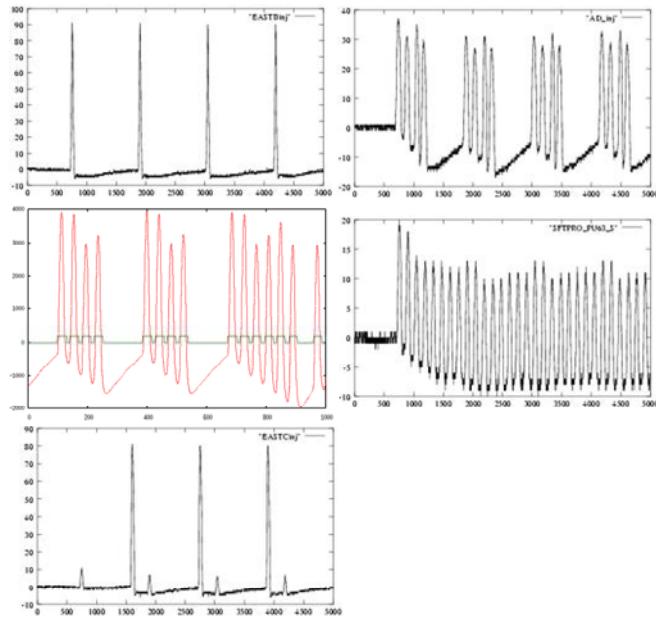
For maximum flexibility the PS implements a concept called *pulse-to-pulse modulation* (ppm) which allows assembling a series of different acceleration cycles into a so-called super-cycle which is repeated. At any moment it is possible to add or remove any of the individual acceleration cycles using a super-cycle editor. In this way several *users* can use the machine at the same time, everybody getting a few time slices of the global beam time.

Figure 19 shows a typical super-cycle. For low proton energy the cycle time in the PS is 1.2 s while for high energy the cycle time is doubled. For these cycles the Booster may accelerate two cycles where the one accelerated during the time the PS is busy is extracted from the Booster directly to ISOLDE. The first text line under the cycle diagram denotes the cycles in the PS (SFPRO, EASTA, etc.) while the second line shows additional cycles in the PS Booster.

**Fig. 19:** The PS super-cycle

The consequence of this flexibility is that the trajectory measurement system must deal with a great variety of different types of beams (Fig. 20):

- EASTB: a single bunch on $h = 8$
- AD: four bunches on $h = 8$
- LHC: four bunches on $h = 7$, two additional bunches are injected on a consecutive Booster cycle
- SFTPRO: eight bunches fill all buckets on $h = 8$
- EASTC: A small bunch is injected into the first RF bucket, a second, much more intense bunch is inject into bucket no. 7. The intense bunch is ejected at 14 GeV. The small one is further accelerated and extracted to a different area later in the cycle.

**Fig. 20:** Beam types in the PS

3.3 Readout requirements

In order to provide trajectory data at any time within the acceleration cycle, the bunch positions must be calculated turn by turn, from injection into the machine until ejection. This can be done by simply sampling the Σ and Δ signals coming from each BPM in the machine, fast enough to numerically integrate them and subsequently calculate the position from the integration results. It is by far easiest to use a fast sampling clock, asynchronous to the revolution frequency. With a bandwidth of the pre-amplifiers located near the BPMs of 30 MHz and expecting at least four samples for a 30 ns bunch, we need a sampling frequency of 120 MHz. Since an acceleration cycle in the PS can take 2 s, we need at least 300 Msamples per signal and therefore a total of $8 * 3 * 2 * 300$ Mbytes (eight bunches, three signals per pickup, 2 bytes per sample).

From these quick calculations it can easily be seen that it will be necessary to treat the data on the fly, such that only position-data for each bunch and not individual ADC samples are stored. The idea is therefore to use an FPGA to read out the ADC samples, create integration gates synchronous to the bunch frequency on the fly, and perform online baseline correction and integration. Figure 20 clearly shows how the baseline moves depending on the number of bunches and their intensity in the machine.

3.4 Readout electronics

As we have seen in the previous section, the electronics must be capable of

- Reading ADC samples at a rate of ~ 120 Msamples/s
- Synchronizing to the bunch frequency
- Generating the integration gate and numerically integrating the signal
- Finding the baseline and correcting for baseline movement
- Storing the integrated and baseline-corrected Σ and Δ signals in a large memory being capable of keeping all values acquired during a 2 s acceleration cycle
- Giving access to the acquired data to the external world

The only way this can be done with today's technology is by use of a fast FPGA associated with big memories.

Figure 21 shows the basic electronics layout. The data coming from the ADC (light blue) block is treated by the FPGA (violet) and stored in memory (yellow). The data can be read out by a single-board computer with Ethernet interface.

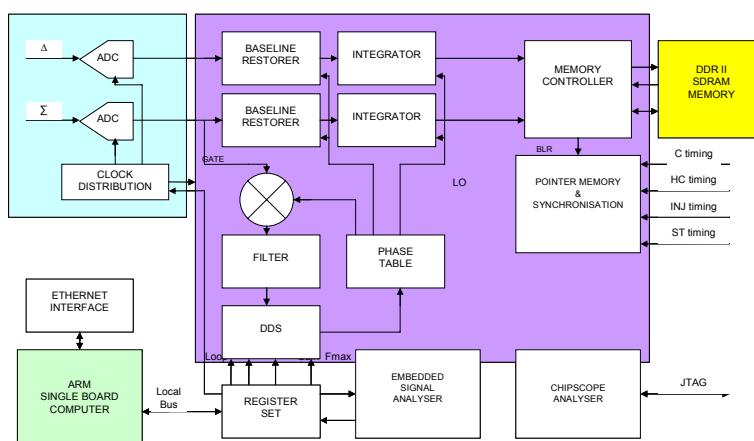


Fig. 21: Trajectory measurement electronics

3.5 The algorithms

In order to develop the algorithms for synchronization, baseline correction, and numeric integration for the different types of beams shown in Fig. 20, the FPGA is first programmed as a chart recorder, sampling—passing every sample from the ADC through to the memory. Like this only parts of the acceleration cycle can be recorded but by changing the acquisition trigger any time-slice in the cycle can be selected. The acquired data are read out from the memory and stored on files. The algorithms can be developed in MathLab or as offline C programs and can be tested on real data.

3.5.1 The PLL

The basic idea for synchronization is a numerical phase-locked loop. The local oscillator (LO) is implemented as a direct digital synthesizer (DDS). The phase of this DDS is compared to the BPM signal and the phase error is filtered and fed back to keep the DDS synchronized. The initial guess of the LO frequency is determined from the measured magnetic field in the accelerator from which the particle energy and thus its speed and revolution frequency can be calculated.

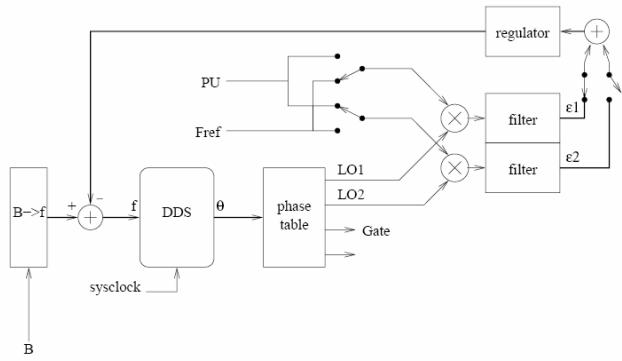


Fig. 22: Phase-locked loop

must be switched to new values on the fly.

The problem is similar at injection: Before having beam in the machine there are no signals coming from the PUs. Instead a reference signal coming from the RF system is used for synchronization. Once the beam is circulating in the machine an external timing pulse switches to the new reference frequency.

These algorithms can be easily tested offline on the data taken with the chart recorder.

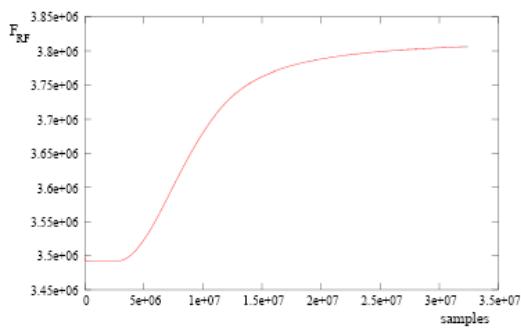


Fig. 23: Measured frequency swing during acceleration

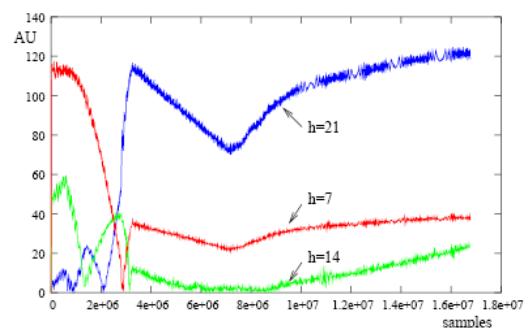


Fig. 24: Frequency contents during bunch splitting

Each incoming PU sample is compared with the phase table to decide if it is part of the signal or part of the baseline. The phase table is simply a circular buffer addressed by the highest significant bits of the DDS phase accumulator. The content of the phase table depends on the harmonic number and is pre-loaded before the start of the acceleration cycle. In case of RF gymnastics (see Section 3.7) the phase table

3.5.2 Baseline correction

The PU together with its load resistance delivers a high-pass filtered, and therefore differentiated version of the beam signal. The baseline restorer uses a low-pass filter and therefore an integrator, with the same cut-off frequency as the PU thus compensating the differentiation effect. When integrating, however, the integration constant is not known and a DC level may be added to the signal. Since we know that this DC level should be zero, a second accumulator is added, which is only active when the ADC sample is part of the baseline, which can be deduced from the synchronization PLL.

3.6 Algorithm implementation in the FPGA

In the first, offline approach, the PLL in Fig. 22 was implemented in a C program. The filter was a second-order Butterworth low-pass filter whose coefficients were determined using a Matlab program.

$$H_F = 9.8 \cdot 10^{-6} \frac{1 + 2z^{-1} + z^{-2}}{1 - 1.9911z^{-1} + 0.9911z^{-2}}$$

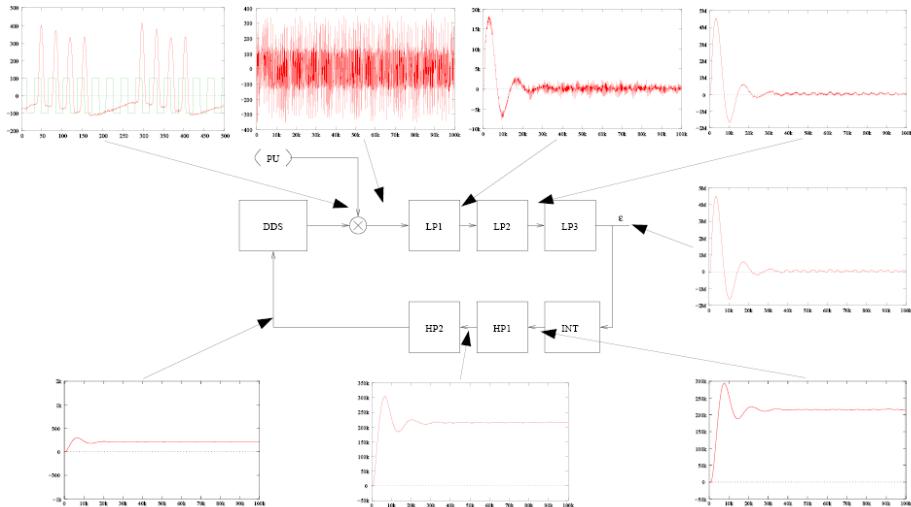


Fig. 25: Splitting the loop filter

Once implemented in the FPGA the algorithms must be capable of swallowing the data coming from the PU, which means a rate of ~ 120 MHz. Calculating this formula in floating point is simply not feasible. Therefore the algorithm was reviewed and replaced by a series of low-pass filters followed by an integrator and high-pass filters using exclusively integer calculations in such a way that the coefficients can be calculated with a few shifts and additions. By subdividing the filter into several stages, pipelining the algorithm in the FPGA becomes possible. Because of the large ratio between sample rate and the dominant system frequencies, a few clock cycles of pipeline delay do not affect the loop dynamics. Once the baseline is corrected and the signal is synchronized, the phase table is used to generate the integration gate. All Σ and Δ samples within the integration gate are added and the results stored in memory. The positions can then be calculated through

$$x = S_x \frac{\Delta_x}{\Sigma} .$$

3.7 Harmonic number changes

Synchronization is largely complicated through harmonic number changes. By changing the RF frequency the bunches can be split into several sub-bunches (bunch-splitting) or the distribution of the bunches on the ring circumference may be modified (batch compression).

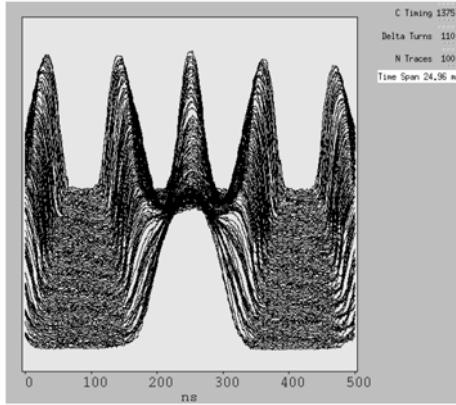


Fig. 26: Bunch Splitting

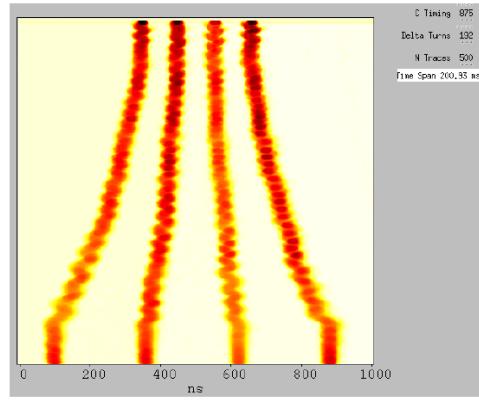


Fig. 27: Batch compression

The frequency contents of the PU signals during triple splitting (a bunch is split into three as seen in Fig. 26) is plotted in Fig. 24. At a certain moment, determined by an external timing signal one must decide to switch from measuring the position of a single bunch to measuring the positions of the three bunchlets. This moment corresponds to the crossing of the blue ($h = 21$) and the red ($h = 7$) frequency lines in Fig. 24.

In order to find back the position in the table of measured Σ and Δ values where the switch has happened, address pointers, linked to these external timing events, must be kept in a separate table.

3.8 Integration

Last not least, the baseline corrected sum and difference signals must be integrated, which is done by simple addition of the samples within the integration gate calculated by the synchronization algorithm.

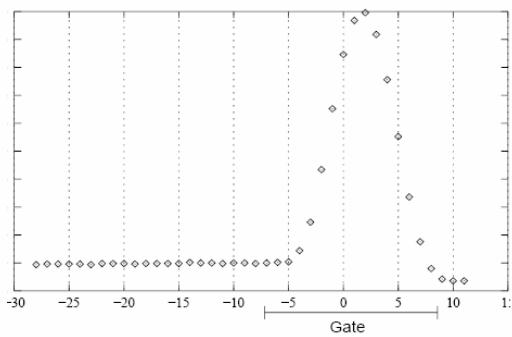


Fig. 28: Integration



Fig. 29: Energy of a LHC bunch at 7 TeV

4 Machine protection systems

While in low-intensity and low-energy accelerators the beam cannot do much harm even if the whole beam is lost in the vacuum chamber, this is clearly not the case for high-energy and high-intensity machines. Even at the CERN Linac, a machine with a top energy of 50 MeV, 160 mA during a 200 μ s beam pulse and a repetition rate of 1.2 s it was possible to burn a hole in one of the vacuum joints. How much more critical is the problem in the LHC, where we shall have 2808 bunches at 7 TeV? People amuse themselves calculating the equivalent kinetic energy of known everyday objects. The energy of single bunch in the LHC at top energy corresponds to a 5 kg bullet at 800 km/h¹ and don't forget that there are 2808 of them in the machine. On the other hand, these bunches circulate in a magnetic field which is generated by supra-conducting magnets. The loss of a very tiny fraction of these particles will result in a magnet quench, and the loss of a big amount, concentrated in a small volume, will result in destruction of the equipment.

At the Tevatron, even though its beam power is 200 times less than the power in an LHC beam, a hole was drilled into the primary collimator (Fig. 31), when the beam was displaced by 3 mm due to insertion of a movable device, when it was supposed to be out of the vacuum chamber. The secondary collimator was largely

damaged and
16 magnets
quenched.



Fig. 30: Ionization chamber



Fig. 31: Beam damage

4.1 Machine protection using beam loss monitors

The strategy for machine protection at the LHC is based on the measurement of beam loss with dedicated Beam Loss Monitors (BLMs). When a high-energy particle is lost, it will produce a particle shower whose energy is partially absorbed in the surrounding magnet coil but part of which can be detected by the BLMs. As long as the calibration factor of energy deposition in the magnet coils with respect to the energy deposited in the BLM is known, the BLM signals can be used to trigger beam dumps as soon as the energy deposition in the magnets gets to a level where a magnet quench must be feared. By dumping the beam and thus avoiding a quench, long beam downtime can be avoided. For even bigger beam losses, where the magnets or other equipment are at risk of destruction, this is even more true.

¹ Rüdiger Schmidt, <http://rudi.home.cern.ch/rudi/docs/VisitLHCWuppertal2006.ppt>

On the other hand, the levels for triggering beam dumps must not be set too low because this may prevent running the machine altogether.

In order to set the trigger levels correctly, the allowable losses must be known. These depend on the energy of the primary particle and on the loss duration as can be seen from Fig. 32. It is therefore important to integrate the measured losses over several time periods and set corresponding thresholds for each of these integration periods. Only like this can short and strong losses be treated as well as smaller but prolonged losses.

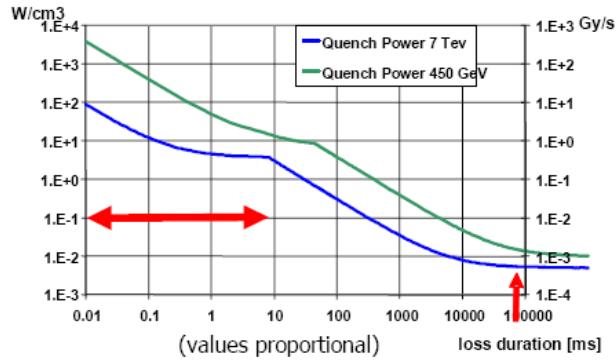


Fig. 32: Quench levels

For safety systems, in addition to the standard technical specifications like dynamic range, resolution, response time, etc., the *Mean Time Between Failure* (MTBF) is an important parameter. It defines how secure the system actually is. When the security system itself fails it must go into a failsafe state, which, as a consequence, makes the protected system unavailable.

Most of today's machine protection systems use ionization chambers as their BLMs (the LHC BLM can be seen in Fig. 30). These are gas-filled detectors (N_2 in case of the LHC BLMs) with the following properties:

- High dynamic range (10^8)
- Very good resistance to radiation (several MGray/year)
- High reliability and availability

The BLM signal is integrated using a charge-balanced integrator and converted to a proportional frequency with a current-to-frequency converter (CFC). This method allows the handling of a very high dynamic range. The CFC frequency is counted over a period of 40 μ s. For very low currents and to allow faster response, an ADC was added. The converted signal is transported from the machine, some 80 m underground to the surface through optical fibres. Eight detectors are multiplexed onto an optical link and the links are doubled. The digital signal treatment is performed by a radiation-resistant FPGA which

- reads out the converted signal,
- encodes the values in order to prepare them for transmission over the redundant optical serial links,
- multiplexes values from eight detectors onto a signal transmission channel,
- performs CRC calculations.

The electronics on the surface receives the values from the optical links, checks the CRC, de-multiplexes the signals from the eight BLMs and gives access to the beam loss values through the

VME bus. It also connects to the Beam Interlock System (BIC) generating beam dumps, should this be

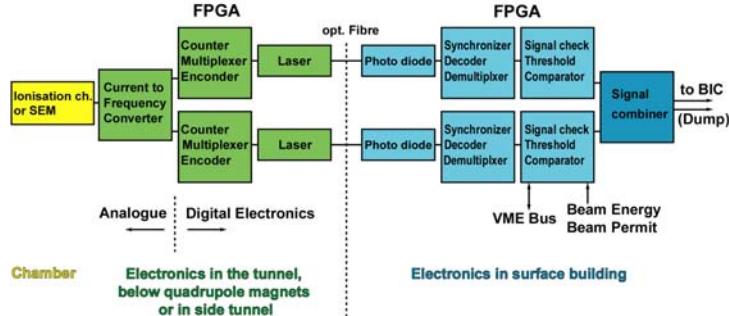


Fig. 33: Layout of the BLM electronics

necessary.

4.2 The data acquisition board

We have already seen an electronics layout consisting of ADCs, an FPGA reading out the converted signals and treating them through fast digital signal processing algorithms implemented as VHDL code in the FPGA, followed by memory, used to store the final result. Some sort of access to this memory is needed in order to further treat the result and make it available on the operator consoles in the control room in the form of easily readable numerical values, tables, or graphs.

The requirement of FPGAs connected to some sort of external signal in conjunction with access to the final stored data is general enough to allow the design of a generic card which can be used for a variety of applications.

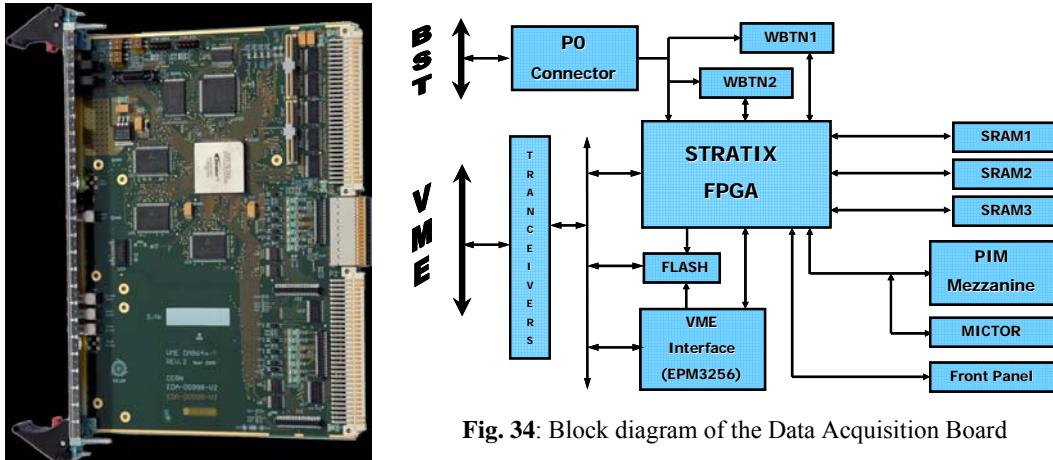


Fig. 35: Photo of the Data Acquisition Board

The Data Acquisition Board (DAB) is implemented as a VME board built around an Altera Stratix FPGA. The FPGA can be boot-strapped through a flash memory containing the FPGA code. This memory is also accessible via the VME bus. In addition, external timing signals (Beam Synchronous Timing (BST)) can be accessed by the FPGA and used within the FPGA algorithms. In order to be as flexible as possible the FPGA has access to a local mezzanine bus. Mezzanine boards implement the application-specific hardware. Like this the DAB can be used to treat signals from the orbit system, based on several thousands of BPMS, it can be used by intensity measurements systems

where signals from beam current transformers are treated, or it can be used to handle the communication protocol in order to read out signals from the beam loss system.

4.3 Data treatment for the beam loss system

4.3.1 Data treatment in the tunnel

The signal coming from the BLM is converted into frequency for coarse conversion, where a counter is used to get the final coarse value. The voltage measured on the ADC is the remainder between the last count and the first count from the next acquisition. Before sending these data to the optical communication channels, the ADC and counter data are combined to a 20 bit loss figure. Eight such values are encoded, multiplexed, and sent to the surface electronics after a CRC value has been added.

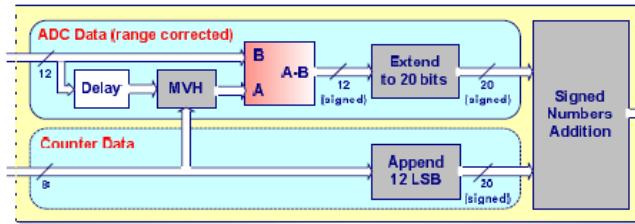


Fig. 36: Beam-loss dynamic range

4.3.2 Data treatment on the surface

The data treatment algorithms in the DAB, installed at the surface, must perform the following actions:

- Receive the values from the electronics in the tunnel via the optical fibres.
- De-multiplex the data coming from different BLMs.
- Check the CRC and compare the data coming from the redundant communication channels. If the data from the two channels differ: decide which one is right.
- Calculate successive sums in order to see fast big losses as well as slow small losses.
- Compare the successive sums to threshold values in order to trigger beam dumps should the losses be too high.
- Give access to beam-loss data for inspection in the control room together with status information.
- Keep measured data in a circular buffer for post mortem analysis.
- Provide error reporting.

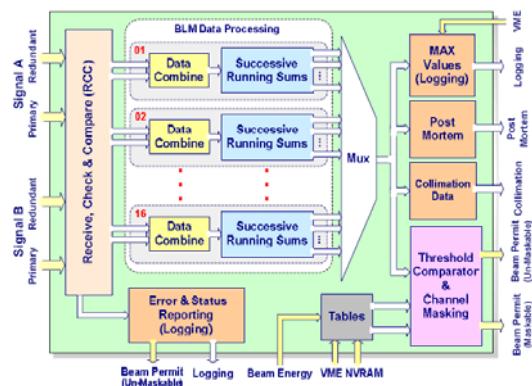


Fig. 37: Algorithms treating the BLM signals

As can be seen from Fig. 32, the threshold levels above which a beam dump must be triggered, depend on the beam energy and the loss duration. For this reason the loss values are integrated using running sums. Twelve integration periods, spanning from 40 μ s to 84 s are made available.

The calculation of running sums is rather simple. Each incoming value is added to a register, while the oldest value of the interval is subtracted again. Of course all values making up the sum must be kept.

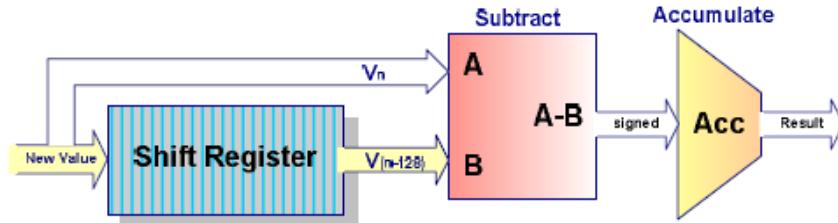


Fig. 38: Calculating running sums

Another way to handle the calculations would be to use a long shift register and always add the difference between the first and the last values.

The longer the integration interval, the longer the shift register must become. This problem can be overcome by using partial sums instead of keeping all previous values. This results in a cascade of shift registers and adders as shown in Fig. 39. In addition, a multi-point shift register is used calculating two running sums at once.

Of course, the latency of the sum output for each running sum depends on the time the previous sum needs for its calculation and therefore increases for longer integration periods. The running sums are compared with threshold values, dependent on the particle energy and a dump trigger is issued if the losses exceed those threshold values.

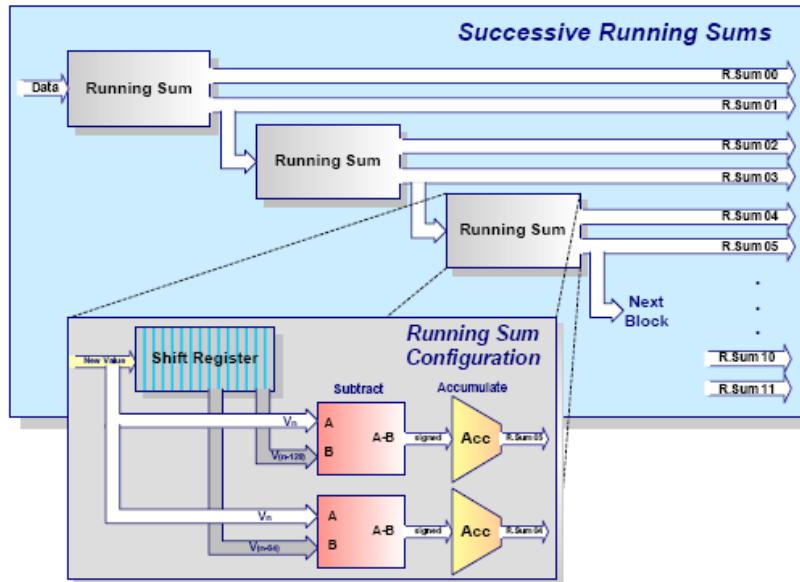


Fig. 39: Cascade of running sums

Apart from triggering beam dumps, the loss values are used for online viewing and logging as well as for post mortem analysis. The values for the last 20 000 turns (40 μ s samples) as well as the 82 ms summed values of acquired data during the last 45 minutes are available. In addition status information about the functioning of each individual BLM station is transferred from the tunnel to the surface electronics allowing online supervision of the whole system.

Part of the recorded data will be used

- to drive an on-line event display in the control room and
- write an extensive logging database both at a refresh rate of 1 Hz.

Other parts of the same processing units, initiated by external triggers, will provide fast updates of the loss pattern seen. For example:

- For the automated collimator adjustments, it will record and provide the last 20 ms by 640 μ s integrals.
- At every beam injection and scheduled dump, 100 ms worth of data will be pushed to the relevant systems to be used to verify the correctness of those procedures.
- A detailed post mortem analysis study will be possible, in the event of an unforeseen dump, by analysing the last 1.7 s by 40 μ s integrals stored in the electronics for each channel.

5 Phase-space tomography

Up to now we have only looked at digital signal treatment where the calculations were done on the fly and which had to be very fast in order to keep pace with the incoming data flow. However, sometimes it may be enough to collect the raw data and treat them using digital signal processing algorithms only after the acceleration cycle. A typical example for such a system is the measurement of longitudinal phase space distributions using tomography.

5.1 Longitudinal phase space

Consider a circular machine with a single RF cavity for acceleration, powered with a sinusoidal accelerating field. Synchronous particles having exactly the correct phase with respect to the RF field will get the pre-determined energy increase while particles coming early will be accelerated less and particles coming late will be accelerated more. This results in a movement in phase space shown in Fig. 40. The longitudinal profile, measured with a wall current monitor, for example, corresponds to a projection of the phase-space distribution onto the Φ axis.

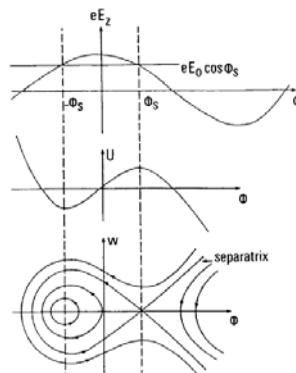


Fig. 40: Longitudinal phase space

Observing the bunch profile for many turns in the machine corresponds in fact to making ns Φ axis ‘rotating’ around the phase-space distribution.

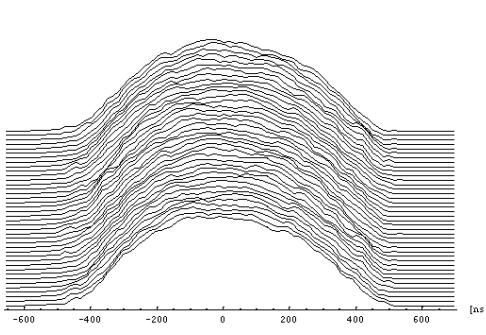


Fig. 41: Bunch profiles

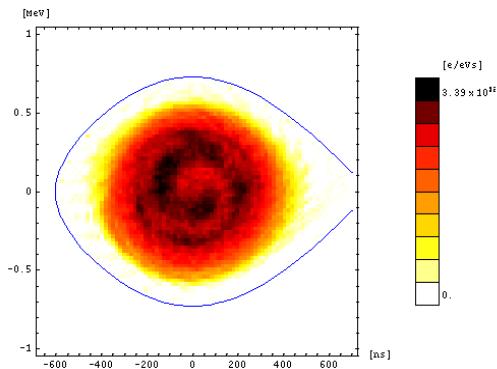


Fig. 42: Reconstructed phase space

Medicine uses similar measurements to visualize three-dimensional structures in the human body. Many X-ray images are produced with an X-ray camera rotating around the patient’s body. The 3-dimension object is re-constructed using Algebraic Reconstruction Techniques (ARTs).

The principle is rather simple (the implementation, however, can be rather complex and requires a lot of computing power): The X-ray image, which is in fact a projection of the three-dimensional object onto the camera axis, is back-projected. This means that the content of a one-dimensional bin is distributed over a two-dimensional array of cells that could have contributed to that bin. A first reconstructed version of the original object is obtained. Projecting this reconstructed object and comparing the result to the projection of the original object yields differences. These differences can again be back-projected yielding an improved reconstruction. Doing this for many images allows the



Fig. 43: CT scanner

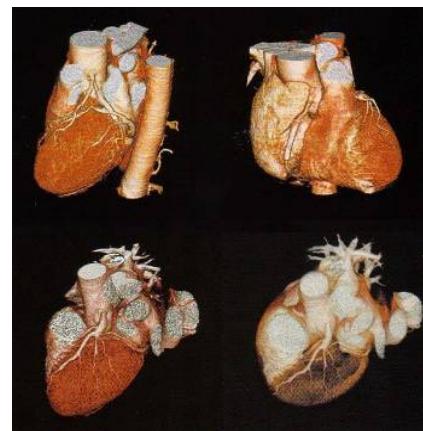
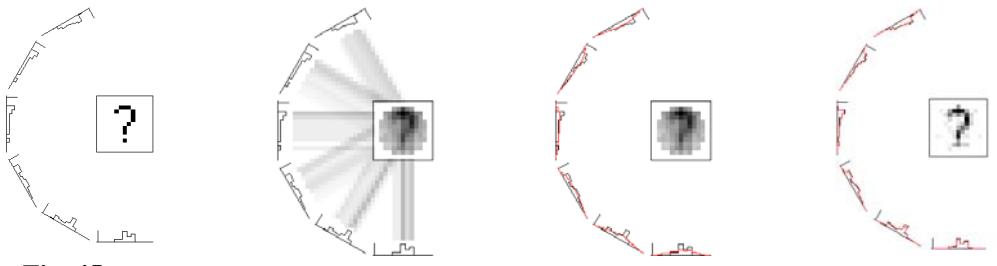


Fig. 44: Results from CT reconstruction

re-construction of the original object.

The method of ART can be applied to the profiles obtained from wall current monitors measuring bunch profiles over many turns. While the individual measurements do not exhibit any detailed phase-space information, the reconstructed phase-space plot displays a wealth of interesting details which cannot be observed otherwise.

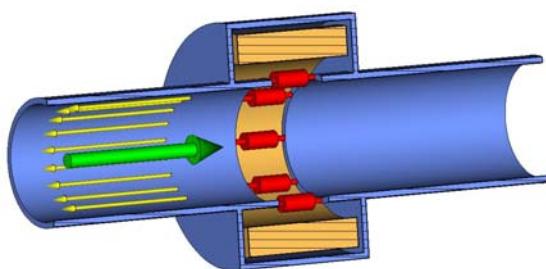
**Fig. 45:** Projection**Fig. 46:** Back projection**Fig. 47** Projection of
back-projected image**Fig. 49:** After 50
iterations

5.2 Particle tracking

The reconstruction technique explained in the previous section only works as long as the projections are taken with a constant rotation of the viewpoint: the object is rotated in front of the camera always at the same rotation angle, which would correspond to a circular movement in longitudinal phase space. Unfortunately, however, the synchrotron motion of particles represents large non-linearities. These non-linearities are taken into account by tracking particles in a simulated machine in order to build maps describing the evolution in phase space. These maps are taken into account for the reconstruction.

5.3 The hardware

When a beam travels through the vacuum pipe, an image current, equal in intensity with the primary beam but travelling in opposite direction, is created. By creating a gap in the vacuum chamber and bridging it with resistors, this image current can be measured. Such a wall current monitor (WCM) can have a very large bandwidth (typically in the GHz range) and the longitudinal shape of the particle bunches can be measured. The WCM signal is amplified and measured with a fast oscilloscope.

**Fig. 49:** The Wall Current Monitor (WCM)

Of course the signal must be measured on a turn-by-turn and bunch-by-bunch basis which requires an oscilloscope trigger synchronous to the revolution frequency. The problem is similar to the one described in the section on trajectory measurements.

A front-end computer is dedicated to the trigger timing while the scope trace data are directly read out by the application program via a dedicated (GPIB) Ethernet link thus improving data transfer speed.

The bunch profiles (see Fig. 41) are used as input to the ART algorithm. In a first version the ART code was implemented in Mathematica™ because of its large mathematical libraries. Later, however, it was implemented in High Performance Fortran-90 which allows parallelization and execution on several processors, further reducing execution time. Many tricks were played like the use of integer arithmetic instead of floating point when this was possible as well as extensive use of tables,

e.g., for the calculation of trigonometric functions. Like this the execution time could be reduced from hours to some 15 to 20 seconds.

6 Conclusions

Beam diagnostics is a field spanning a great number of disciplines: Machine physics, analog and digital electronics, and computer science. Quite naturally digital signal processing methods are employed to extract physically meaningful and easily understandable results from the raw signals created with the beam instrumentation sensors. Using a few typical measurement applications as examples, these lectures try to bridge the gap between the methods employed in digital electronics and the beam physics using the power of these methods.

References

- [1] M. Gasior and J. Gonzalez, DSP Software of the Tune measurement System for the Proton Synchrotron Booster, CERN PS-BD Note 99-10.
- [2] M. Gasior and J. Gonzalez, New Hardware of the Tune Measurements System for the Proton Synchrotron Booster Accelerator.
- [3] J. Belleman, Using a Libera Signal Processor for acquiring position data for the PS Orbit Pick-ups, CERN AB-Note-2004-059.
- [4] J. Belleman, A New Trajectory Measurement System for the CERN Proton Synchrotron, Proceedings of DIPAC 2005, Lyon, CTTM01, pp. 137-139.
- [5] B. Dehning, Beam Loss Monitor System for Machine Protection, Proceedings of DIPAC 2005, Lyon, TIMA01, pp. 117-121.
- [6] C. Zamantzas et al., The LHC Beam Loss Monitoring System's Surface Building Installation CERN-AB-2007-009 BI, presented at LECC 2006, Valencia, 25–29 Sep 2006, SP.
- [7] C. Zamantzas et al., An FPGA based Implementation for Real-Time Processing of the LHC Beam Loss Monitor System's Data, presented at IEEE Nuclear Science Symposium, 2006, San Diego, USA, Oct. 29/Nov. 4, 2006.

Control system integration

T.J. Shea

Oak Ridge National Laboratory, Oak Ridge, USA

Abstract

This lecture begins with a definition of an accelerator control system, and then reviews the control system architectures that have been deployed at the larger accelerator facilities. This discussion naturally leads to identification of the major subsystems and their interfaces. We shall explore general strategies for integrating intelligent devices and signal processing subsystems based on gate arrays and programmable DSPs. The following topics will also be covered: physical packaging; timing and synchronization; local and global communication technologies; interfacing to machine protection systems; remote debugging; configuration management and source code control; and integration of commercial software tools. Several practical realizations will be presented.

1 Accelerator control system definition

A modern accelerator control system conveys all monitor, control, model-based, and computed data from all accelerator, facility, safety, and operations subsystems to accomplish supervisory control, automation, and operational analysis. Its scope extends from the interface of the equipment being controlled through to the designers and operators of the accelerator facility. It includes all hardware and software between these bounds, incorporating computer systems, networking, hardware interfaces, and programmable logic controllers [1]. It is not restricted to closed-loop control systems (a primary focus in this school). In fact, at some accelerator facilities, these closed-loop systems are treated as black boxes developed by the technical groups and interfaced to the control system.

2 History and overview

A global view can be presented in terms of responsiveness, and Fig. 1 depicts the broad range occupied by accelerator control systems. Physics-based applications and IT infrastructure typically reside on the left side, while the embedded DSP applications of beam instrumentation and RF systems reside on the right.

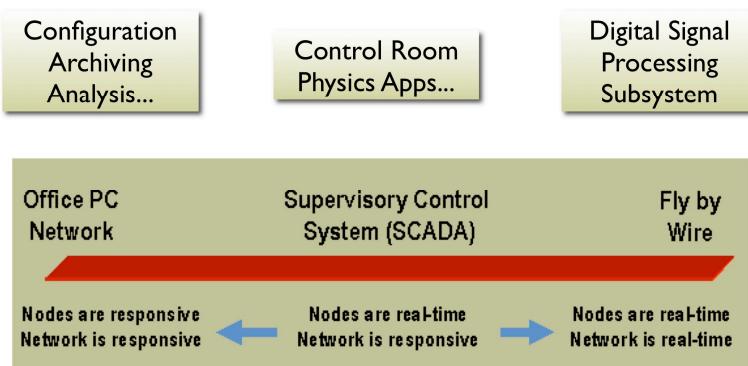


Fig. 1: Range of responsiveness (*lower graphic courtesy National Instruments*)

The centre of activity for accelerator facilities is the control room. The photographs in Fig. 2 present a pictorial history of this important room and hint at the evolution of the underlying architecture. In the first three photographs, we see the AGS control room at Brookhaven National Laboratory evolve from a) a terminus for analog signals to b) an upgraded room incorporating computing terminals and finally, after a complete remodelling, to c) a home for networked UNIX workstations (with some analog signals still routed to scopes and RF instruments). In the last photograph, Spallation Neutron Source (SNS) team members celebrate the final commissioning run in the SNS control room at Oak Ridge National Laboratory. Like virtually all contemporary control rooms, this one is a ‘glass cockpit’ consisting of flat panel displays and networked commodity PCs. This was almost not to be. The original plans for the SNS included one million US dollars for analog multiplexers and associated cabling. Ultimately, the project decided to go all digital and rely on waveform acquisition and digital signal processing capability built into the LLRF and beam diagnostic systems.



Fig. 2: Control rooms over the years. The AGS at BNL is shown in the upper left, top right, and lower left (*courtesy K. Zeno*). The SNS control room is in the lower right (*courtesy ORNL*).

To support these visible changes in the control room, the architecture and equipment in the field has also been upgraded over the years. Movement of digitizers and computing hardware down the signal processing chain from central facilities to areas closer to the signal sources can increase performance significantly. Performing these changes in an existing facility can also require major rework. The pile of discarded equipment and analog cabling shown in the left photograph of Fig. 3 was made obsolete when, in 1982, the Fermilab Linac control system received an upgrade from Scientific Data Systems (SDS) Sigma 2 computers and SDS I/O gear to the networked microprocessor-based systems shown on the right.



Fig. 3: The Fermilab Linac control system upgrade in 1982 (courtesy M. F. Shea)

During the 1980s, laboratories around the world performed similar upgrades, finally migrating to a nearly common architecture, dubbed the standard model during discussions at the 1987 Villars controls conference [2]. Shown during discussions at the 1993 conference in Berlin, Fig. 4 documents the standard model with the technologies in play at that time. Several layers are evident in this diagram, starting with the accelerator hardware itself, serviced by either simple or processor-based input/output (I/O) subsystems. On the next layer up, local stations interface to the I/O via a real-time fieldbus. The local stations are also known as Input/Output Controllers (IOCs) or Front-End Computers (FECs). The network is a central feature of the standard model and it connects the local stations with control room consoles and the servers that provide additional functionality at the highest layer.

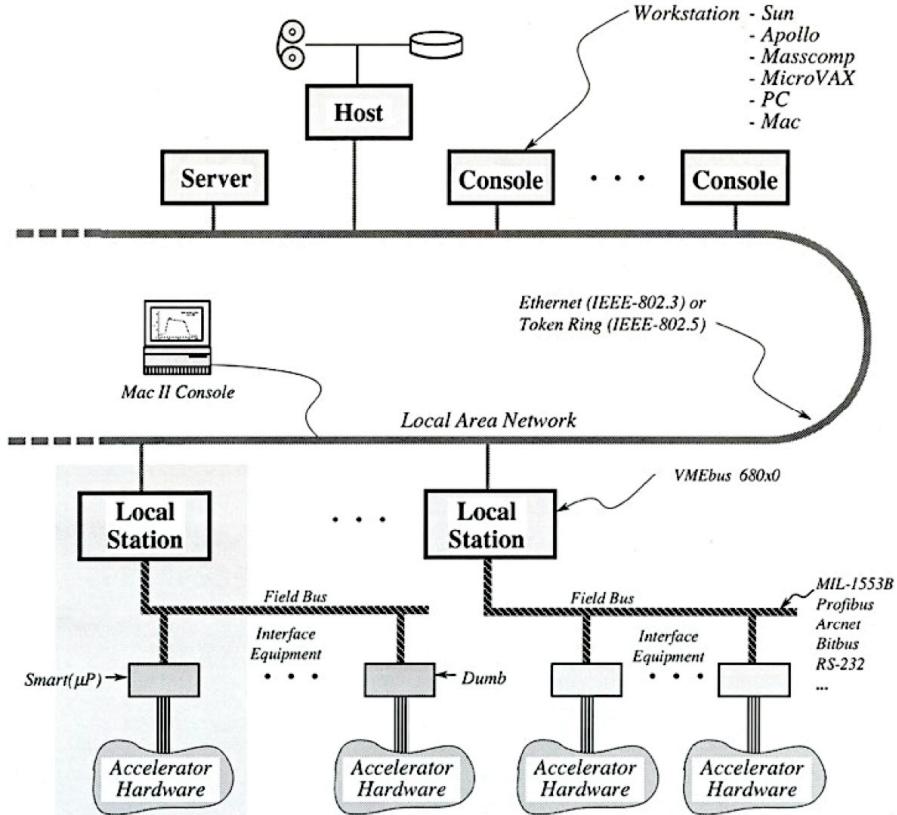


Fig. 4: The standard model of control system as presented in 1993 (courtesy M. F. Shea)

One popular control system toolkit is the collaboratively developed Experimental Physics and Industrial Control System (EPICS) [3]. A current, EPICS-based control system architecture is shown in Fig. 5, and even with some updates, it retains virtually all of the features of the early standard model. In this diagram, the layers 0 through 3 are explicitly annotated along with the functions provided at each layer. Commercial programmable logic controllers (PLCs) at Layer 0 serve as the I/O subsystems and communicate with the Layer 1 IOCs via proprietary protocols. In an EPICS system, the network protocol is Channel Access and the control console and higher layer servers incorporate Channel Access Client libraries to communicate with the IOCs [4]. Persistent storage of configuration information and other data is typically supported by a relational database (RDB).

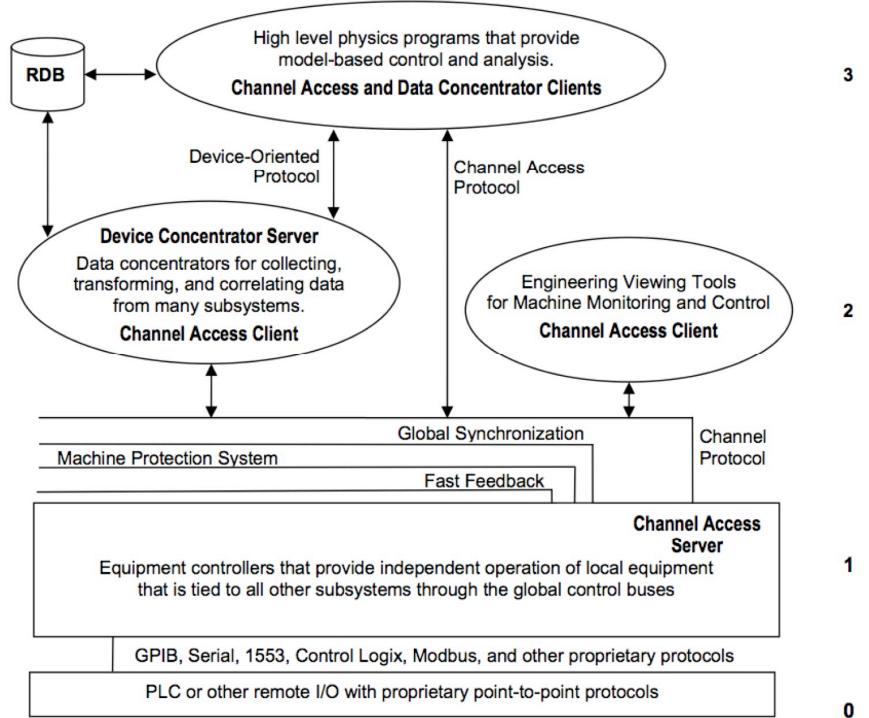


Fig. 5: Architecture of a modern EPICS-based control system (*courtesy L. R. Dalesio*)

3 Interfaces at the lower layers

When designing a high-functionality device such as a digital signal processing subsystem, a valuable exercise is to determine the system boundary and then define the interfaces at that boundary. Good, well-documented choices will ease system development, maintenance, and upgrades. Regarding obsolescence, interface standards typically outlast a device's internal technology. The following functional interfaces should be considered for a typical accelerator device:

- Backplane (Internal)
- Network
- Real-time data
- Event
- Reference clock
- Machine protect
- Utility

- Power
- Environmental
- Beamline device I/O

The first six involve communication technologies, some of which will be covered in an upcoming section. Utility functions include remote reset as well as health monitoring (supply currents and voltages, fan speed, temperature, and heartbeat). Environmental considerations include heat, ionizing radiation, and electromagnetic interference. Some of the listed functions can be supported by a single physical interface. For example, real-time data might be broadcast over the event link, or the device might be powered via the network interface.

Architectural choices at Layer 1 (the IOC) will address many of these interfaces. Currently, a popular approach is to deploy signal processing subsystems on modules within a crate. A typical control system crate is depicted in Fig. 6. The crate's backplane, along with other cards, supports the interfaces listed above. The system is partitioned by function (timing, multiplexed digitizer, etc.), with interfaces and resources potentially shared across device types (position monitor, vacuum, etc.) and channels. This architecture has served distributed systems for decades, through the migration from the I/O-oriented CAMAC [5] to the more computer-oriented buses like Multibus. Currently, VME [6] is the most widely specified crate standard for accelerator control systems.

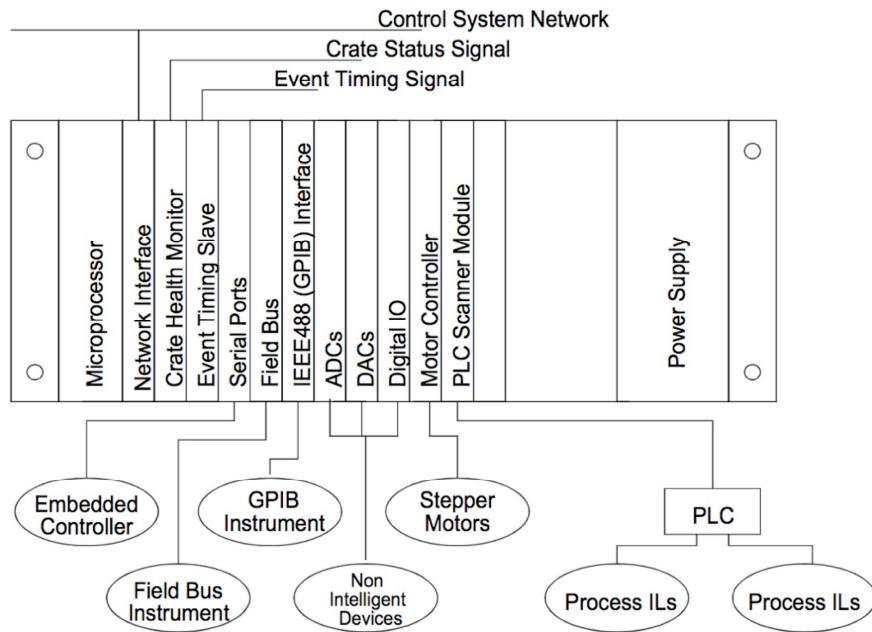


Fig. 6: Typical crate-based IOC (*Courtesy L. R. Dalesio*)

A variation on the crate-based approach is the Network Attached Device (NAD) concept introduced in the 1990s. Figure 7 shows a rough sketch comparing the two architectures, casually presented by the author in a 1999 VLHC Workshop. The intent is to avoid the brittleness that can be introduced by unnecessarily coupling devices via shared resources. In the sketch, single points of failure are indicated in red. The NAD approach can provide a high availability solution for instrumentation systems if the facility can withstand temporary loss of a channel. However, it is not as helpful for applications that are inherently coupled or cannot withstand loss of an individual device.

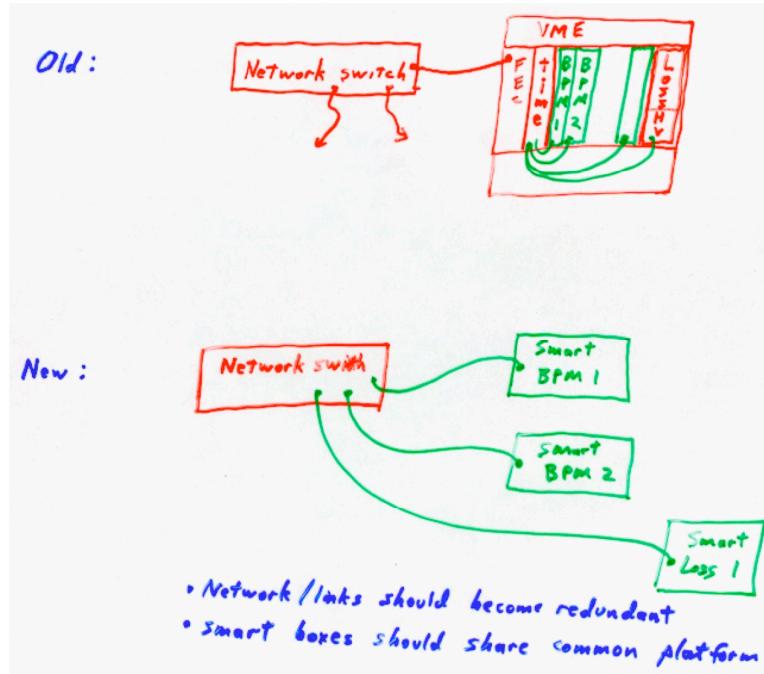


Fig. 7: Migration to Network Attached Device architecture

As an example, several SNS beam diagnostic systems were deployed as NADs [7]. Most were based on a rack-mounted PC platform, although if desired, modern digital technology would now allow implementation on a single FPGA supported by a few additional components. Each beamline device — a position monitor for example — was supported by a single PC-based IOC that executed the signal processing software and interfaced directly to the network, the real-time data link, and the event link. Prior to commissioning, this allowed single-channel vertical integration from the electromagnetic pickup up to the EPICS channel access client. Channels were tested individually as they became available, and this testing was relatively immune to activities associated with other systems.

4 Middleware

Turning to the upper layers of the control system, we see increasing use of middleware. As shown in the module titled ‘Device Concentrator Server’ in Fig. 5, middleware lives at Layer 2 and enables integration of disparate protocols and technologies. This allows uniform access to multiple information sources such as front-end computers, on-line simulators, and databases. Additional functionality particular to accelerator facilities can be provided. For example, a view of the data based on named components and channels, although very useful during integration and debugging, can be transformed to a physics view that maps nicely onto the machine model. This layer can also aggregate requests from multiple clients and responses from Layer 1 servers. Complete middleware solutions are used heavily in the financial sector, but less so in accelerators. Typically, only selected accelerator applications employ middleware.

Two middleware usage examples are shown in Figs. 8 and 9. With 728 distributed channels, the RHIC Beam Position Monitor (BPM) system [8] in Fig. 8 utilized services called managers to collect, correlate, log and serve the turn-by-turn waveforms and averaged orbit arrays. This simplifies development of the physics applications at Layer 3 [9].

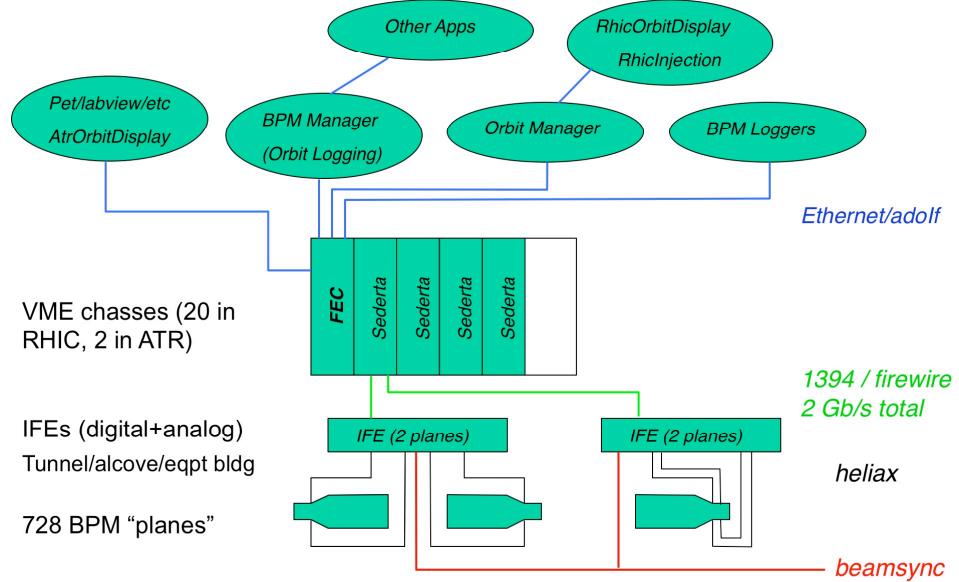


Fig. 8: RHIC position monitor system overview showing the middleware called managers (courtesy T. Satogata)

As introduced elsewhere in this course, Matlab [10] provides a rich environment for simulating and developing signal processing systems. At many light sources, it is also a popular toolkit for online accelerator modelling and application development. Like the RHIC managers, the Matlab Middle Layer shown in Fig. 9 eases the development of high-level applications that must communicate with many device and channels [11].

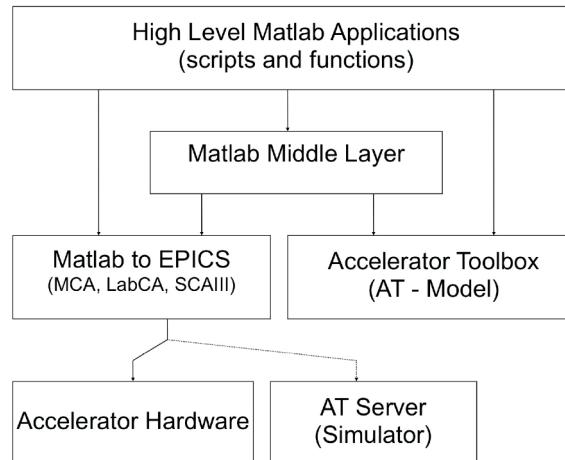


Fig. 9: The Matlab middle layer commonly used in light sources (courtesy G. Portmann)

5 Communications

Communications is a broad topic that spans multiple functional interfaces (including backplane, network, real-time data, event, reference clock, and machine protection) and impacts physical packaging. In accelerators, signal processing devices typically move more data than SCADA-style devices found in other parts of the control system. Therefore, it makes sense to look at trends in other data-intensive systems. For example, the data acquisition systems for the large physics detectors found at colliders have seen the following changes in data processing and communications technology [12]:

- *Intra-chassis standards*: The end of the traditional parallel backplane bus paradigm has been announced annually since about 1989. However, VME and PCI are still in heavy use. New designs may include PCI Express, RapidIO, and ATCA.
- *Commercial networking products*: During the DAQ '94 Conference, ATM, DS-Link, FibreChannel, and SCI were well represented. Today, Gigabit Ethernet (1, 10, 30 GB/s) is universally assumed.
- *The ideal processing/memory/IO data-handling device*: In the past, transputers and DSPs were discussed. Today: FPGAs integrate links, PPC cores, DSPs, and memory interfaces.
- *Point-to-point link technology*: The old style was parallel copper and serial optical. The modern style is serial copper and parallel optics.

For signal processing systems that function as closed-loop controllers, latency is a determining factor of stable loop bandwidth. Standards-based communication links can provide reasonably high throughput, but may contribute significantly to overall latency. Figure 10 presents a rough comparison of some popular standards.

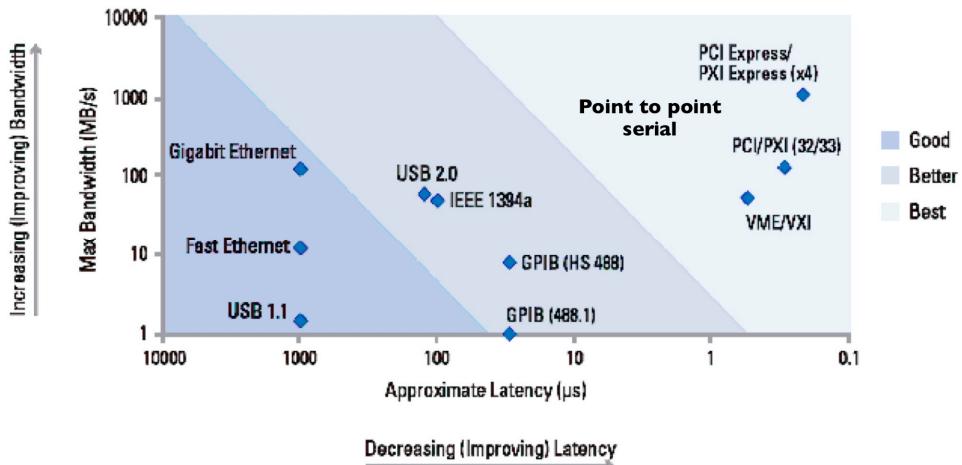


Fig. 10: Throughput versus latency for various communication standards (*courtesy National Instruments*)

5.1 Communication within a chassis

A signal processing subsystem may reside within a control system chassis at Level 1. In this case, the primary interface to the control system is through the backplane standard defined for that chassis type. Alternatively, these same standards may be employed within the subsystem itself when it is deployed as a network attached device.

CAMAC — the grandaddy of modular architectures for physics experiments — defines a simple I/O bus for modules within a crate. Today, CAMAC is rarely specified for new designs, but it was commonly used in accelerator control systems and signal processing applications up until the 1990s. As mentioned previously, VME is still a popular standard for Level 1 equipment. With its roots in commodity computers, PCI [13] enjoys broad commercial support and its cousin CompactPCI [14] has been used recently in a few control systems. For instrumentation and RF applications amenable to a modular solution, VXI and PXI are viable standards. VXI adds shielding, additional card sizes, trigger lines and other features to the base VME standard. PXI adds a smaller set of instrumentation-specific features to CompactPCI.

Recently, serial links have begun to displace these parallel busses. Commodity PCs now include PCI Express slots [15], and the Advanced Telecom Computing Architecture (ATCA) [16] specifies

only serial links for communication. Under consideration for future machines, ATCA provides features required to achieve an availability of ‘five nines’ (5 minutes of downtime per year). Modules within an ATCA chassis are essentially network attached devices that communicate via redundant, switched serial links. Table 1 presents a quantitative feature comparison of VME, PCI, and ATCA.

Table 1: A comparison of VME, PCI, and ATCA (*adapted from P. Le Dû*)

Parameter	ATCA	PCI	VME
Board area (square cm)	995	316	373
Max. power (watts)	200	25	30
Max. I/O bandwidth (Gb/s)	20	4.3	2.4
Panel size, height, width (cm)	30, 2	8, 1.2	21.5, 2
Component height (mm)	21.33	14.48	13.72

Two serial links from the PC platform have also found their way into accelerator devices both within the chassis and for short runs between chassis. These are IEEE1394 and more recently, USB. In the 1990s, switched Ethernet was still too expensive to support the RHIC BPM application, so IEEE1394 was selected. Today, a similar system would probably employ Ethernet. Industrial cameras are making a similar transition, from IEEE1394 to Gigabit Ethernet (although the highest-performance imaging applications still specify Cameralink). USB has become popular as a local communication interface in some signal processing devices, complementing the ubiquitous JTAG port. In particular, the higher speed of USB2 makes it suitable for transferring larger waveforms and configuration files. Inexpensive commercial USB-based devices are increasingly used for slow monitoring and control within a chassis. In an important development for integration, modern gate arrays can now directly support PCI, PCIe and most of these serial standards.

5.2 Networking

Virtually all networked control systems are now based on Ethernet. When communication over a distance is required in a signal processing system, the designer should first consider using the network and protocols native to the control system. Switches allow isolation of this traffic on a virtual LAN if necessary. This communication could occur directly between Layer 1 devices, or it could use a Layer 2 service as a message broker. If Ethernet cannot provide the required throughput (unlikely) or guaranteed latency (more likely), then other dedicated links might prove applicable.

Network protocols defined by control systems typically rely on the User Datagram Protocol (UDP) [17] and/or the Transmission Control Protocol (TCP) [18]. For example, EPICS Channel Access utilizes TCP connections to send process data between clients and servers. Some more recent real-time streaming protocols provide features that could be useful for large-scale, distributed signal processing, but beyond a few video applications, these protocols are not yet widely used in control systems.

TCP assures reliable data delivery, but gives up deterministic low-latency capability. Although this is an acceptable tradeoff for many applications, any distributed signal processing system that offers closed-loop control must treat late data (even if reliably delivered) as bad data. On a tightly managed network, UDP can provide adequate reliability and latency for many of these systems. Therefore, it is worth while to explore it in more detail.

The simple structure of a UDP packet is depicted in Fig. 11. Most of the header is dedicated to hardware addressing (the globally unique source and destination MAC addresses) and the IP header, which contains the source and destination IP addresses. These IP addresses can be directly converted

to the common numerical ‘dot’ notation (192.168.0.55 for example). Network switches use this address information to route the packet from the sender to the correct recipient. Options are also available to support broadcasts and multicasts. The UDP header also includes port numbers that allow network nodes to keep track of packets after reception.

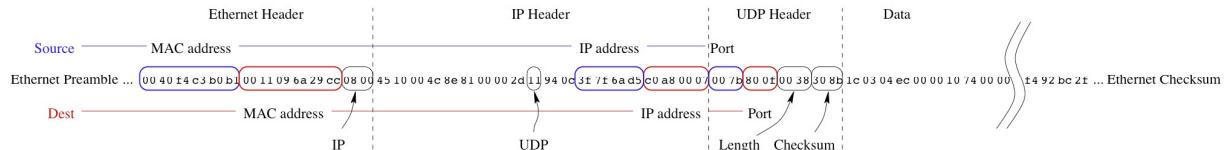


Fig. 11: Structure of a UDP packet (*courtesy L. Doolittle*)

To measure real-world latency, the SNS Data Acquisition Group performed tests using commercial hardware. The test setup is shown in Fig. 12. Round trip latency was measured between user mode processes running on two Windows PCs. For the initial test, the PCs were 2002 vintage machines running Windows 2000 and were connected directly with no network switch. Round trip latency from UDP Send to Response received was 300 microseconds as measured by a high-resolution timer. For another test, 2005 vintage PCs running Windows XP were connected with a Gigabit Ethernet switch. Results are shown in Figs. 13 and 14.

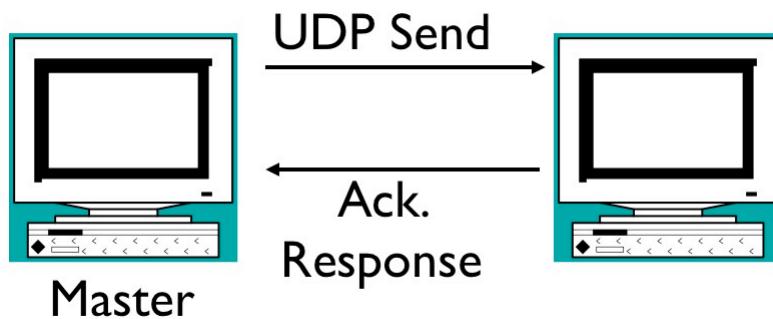


Fig. 12: Test setup for measuring round trip latency of UDP based communications (*courtesy R. Riedel*)

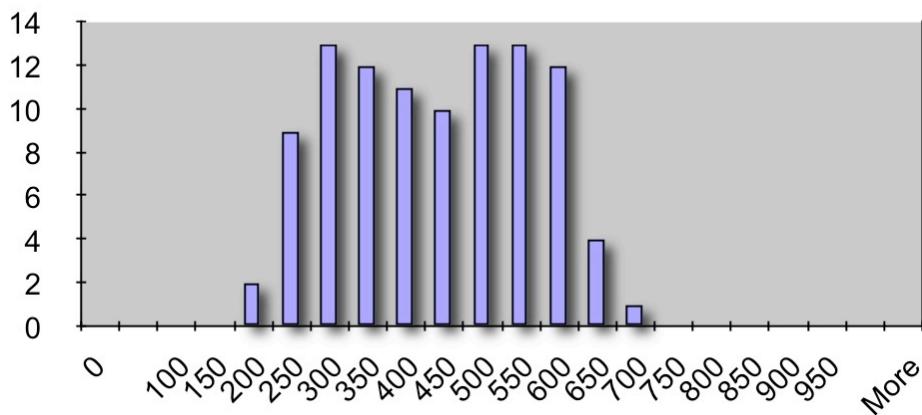


Fig. 13: Distribution of round trip latency measurements, showing jitter added (in microseconds) by the operating system and the network switch (*courtesy R. Riedel*)

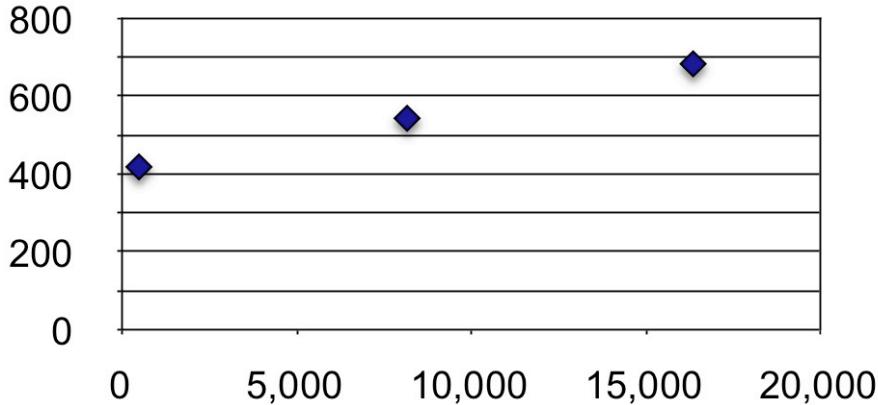


Fig. 14: Plot of round trip latency (microseconds) versus packet size (bytes) (*courtesy R. Riedel*)

Based on these results, the Data Acquisition Group decided to receive data from the accelerator's event link and the real-time data link and rebroadcast it as UDP packets. These packets are broadcast from a master and received by all neutron instruments between each 60 Hz beam pulse. At many facilities, the SNS accelerator included, the real-time data link is implemented with custom network hardware and protocols. For systems with moderate requirements, UDP transmissions on a well-managed Ethernet network can also work.

For another application example, consider the ALS orbit feedback system [19]. The design avoids the use of custom hardware and protocols by relying on a commercial RTOS, 100baseT Ethernet switches, and messages carried in UDP transmissions. Currently, the update rate is 1.1 kHz, but it is expected that a 5 kHz rate could be achieved with 1000baseT network hardware. At the current rate, latency and jitter have not impacted closed-loop performance.

All of the latency and jitter caused by the processor and operating system can be avoided by implementing the UDP protocol directly within an FPGA. This approach might be considered when low latency is required while retaining standard internet protocols.

In very high performance systems, particularly when very low latency must be maintained, standard Ethernet hardware may not be suitable and a custom or proprietary solution may be helpful. One option is reflective memory. This is also known as replicated memory and is available for all popular parallel busses. Each participant in the reflective memory network has a reflective memory module that can be written and read as simple memory. Anything written to a particular address in one reflective memory module is written over the network to the same address in the other modules. The APS orbit feedback system [20] utilized reflective memory in the topology illustrated in Fig. 15. For 22 nodes and 1200 metres of fibre, the calculated loop settling time is 21.4 microseconds and the transfer rate is 29.5 Mbytes/s. Within the nodes, using dedicated DSP boards minimizes additional latency.

Commercial reflective memory is now giving way to a more integrated approach that takes advantage of serial I/O available on FPGAs. Higher end FPGAs now come with several multi-gigabit/s serial interfaces. While offering high performance, these links also offer flexibility in two ways. First, the protocol is determined at configuration time and can be changed after the hardware is constructed and installed. Second, the physical interface external to the chassis can also remain reconfigurable by use of the Small Form Factor (SFP) or the newer XFP modules. Commercial modules are available to support various optical or copper links up to 10 Gb/s and can be installed in the field. The orbit feedback system for the Diamond light source employs this technology.

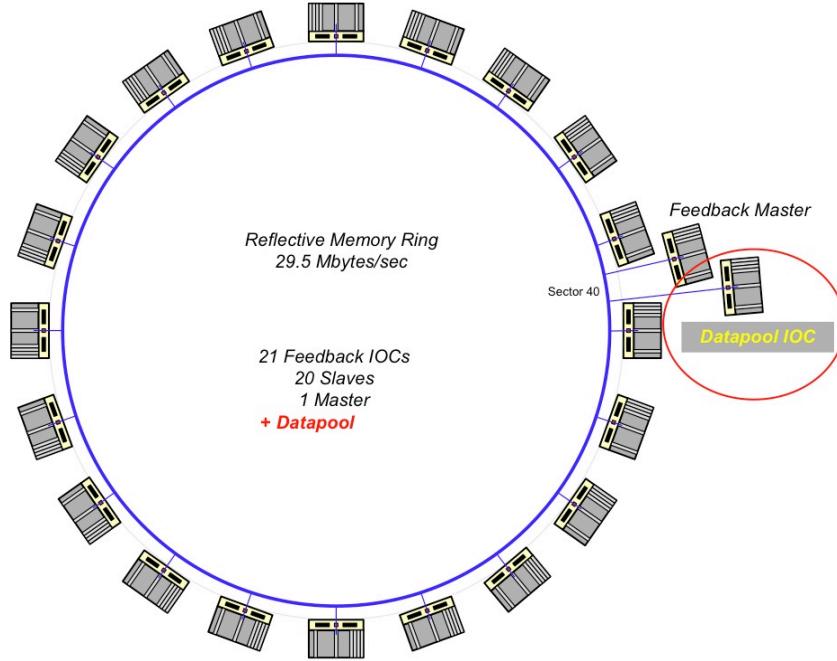


Fig. 15: Reflective memory for the APS orbit feedback system (*courtesy F. Lenkszus*)

5.3 Timing systems

In the context of an accelerator control system, a timing system synchronizes actions of distributed devices and supports acquisition of correlated data from distributed measurement systems. It is typically implemented as an event link that broadcasts event codes from a central master to receivers located in or near the devices. Optionally, the term ‘timing system’ might also refer to real-time data links and RF references. This section will primarily focus on event links.

Referring to Fig. 16, the event generator may send a particular event code in response to a hardware trigger, an entry in a clocked buffer, or a software sequencer’s write to a register. For large facilities, a hierarchy of fanouts may be required to broadcast the events to all receivers. Upon receipt of an event code, a receiver typically generates a bus interrupt or an output pulse of predetermined delay and width. The received event code can be buffered in a register so that software can respond to a particular code.

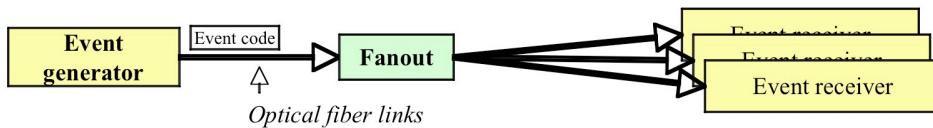


Fig. 16: Event system diagram (*courtesy T. Korhonen*)

The SNS timeline in Fig. 17 illustrates the flavour of the event types that are broadcast on a pulsed machine’s timing system. Each event is a unique 8-bit code that is serialized and broadcast to all receivers by the timing master. Cycle Start signifies the start of a new event sequence for a particular beam pulse. After Cycle Start, a predetermined sequence of events (annotated in yellow) triggers the behaviour of all equipment during that pulse, determining, for example, whether beam is on or off, and what beam measurements are stored. This pattern repeats at the beam pulse repetition rate of 60 Hz. The response to these events is determined by configuration of the individual receivers and their associated systems. It is important to note that there are very few surprises and this

configuration can be essentially static. Events-related Machine Protect System (MPS) trips are virtually the only events not predetermined prior to the pulse. For intelligent subsystems such as signal processing systems, calculations and general behaviour may require additional data associated with each pulse and this is sent as 24-bit words on a separate link called the Real-Time Data Link (RTDL). The most important piece of data carried by this link is the global timestamp that is used to correlate data from widely dispersed systems.

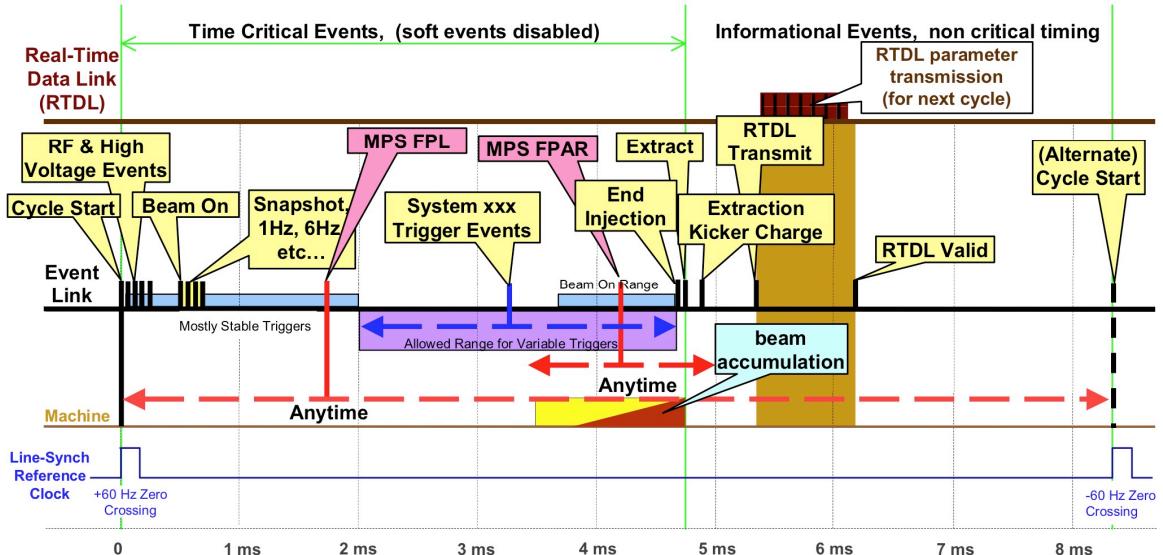


Fig. 17: SNS timeline for one pulse (*courtesy C. Sibley*)

Traditionally, timing system receivers have been packaged as general-purpose modules that include multiple pulse generators, clock outputs, and a bus interface. The designer of a signal processing system would integrate this module with external cabling, in some instances without tightly integrating the timing system support software. As an example of the logic required for a receiver, Fig. 18 shows a design that has been used for over two decades at FNAL and BNL. Clearly, devices that utilize a gate array can easily incorporate a full receiver directly into each device. This approach was taken in the RHIC BPM system and is particularly helpful when following the network attached device architecture. More recently, event systems have been deployed using standard physical layers such as gigabit Ethernet, therefore allowing the link to be supported by an FPGA's serial ports and an SFP module.

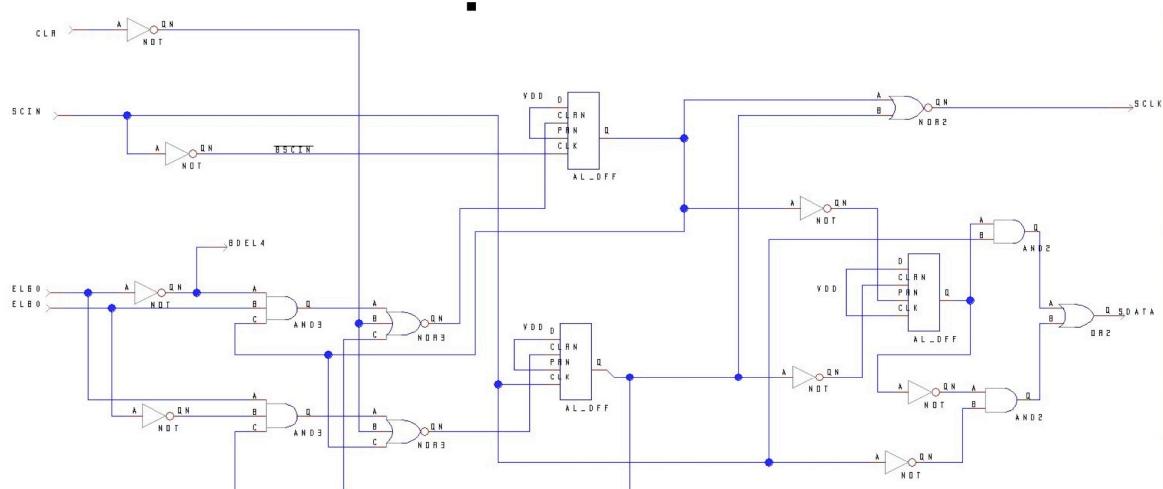


Fig. 18: Schematic of simple event receiver (*courtesy J. Mead*)

Event systems are critical for facilities that support multimode operation. This is also known as multi-user operation, Pulse-to-Pulse Modulation (PPM) or multi-flavoured operation. Based on receipt of a particular event, a beam-generating device may change a certain beam parameter, a LLRF device may use a certain feedforward table, or an instrument may update certain measurement results. Complex hadron facilities at Fermilab, CERN, BNL, and others make extensive use of this functionality.

5.4 Machine protection

A machine protection system is a global facility that protects the accelerator's equipment from damage [21]. It includes a global link that carries an indication of an off-normal condition to critical devices that shut down beam generators or other equipment. It does not directly protect personnel from harm. Therefore, while the machine protection system itself and the equipment that interfaces to it should meet high quality standards, it typically does not require the formal qualifications of a Personnel Protection System (PPS). Intelligent subsystems may interact with the machine protection system in two general ways:

1. A subsystem may drive an MPS input. For example, a DSP in a closed-loop system may detect loss of control, or a differential current measurement system may detect high beam loss.
2. A subsystem may respond to an MPS trip. If it is part of a critical device, the subsystem would be connected directly to an MPS receiver and in response to the trip, would shut down beam or its associated equipment. Otherwise, the subsystem would receive an indication of the trip via the event or real-time data link and respond by, for example, freezing circular buffers containing post mortem data [22].

Systems that drive MPS inputs incorporate circuitry similar to that shown in Fig. 19. This copper interface includes a loopback to determine cable integrity. Although simple and robust, interfaces like this are not standardized and are not available on commercial equipment. Alternatively, optical interfaces could be employed to increase noise immunity and even take advantage of standard modules like SFP that are showing up on high-performance commercial devices.

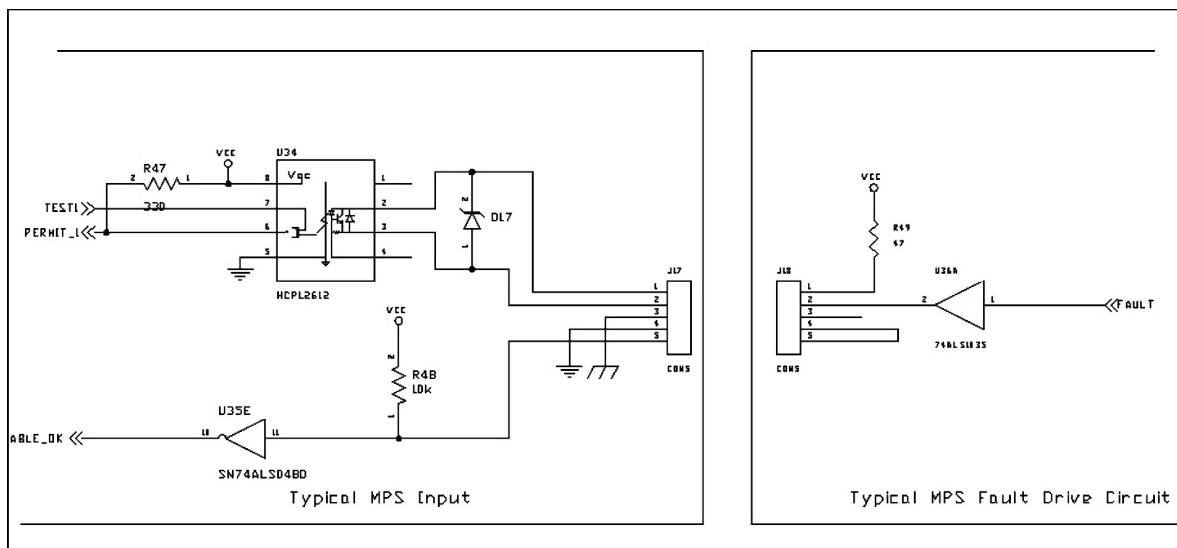


Fig. 19: Typical MPS interface (*courtesy A. Jones*)

6 Utility functions

6.1 Housekeeping

Several housekeeping functions are useful for remote devices. These include monitoring functions (fan speed, power supply voltages and current, temperature, and heartbeat), and remote control (power and reset). The temperature monitor could be used for two purposes: for fault monitoring (i.e. clogged air filter), or for correction of thermal effects (i.e. preamp offset drift).

A heartbeat is useful for assuring that an intelligent system is still alive. This could be as simple as a process variable that is updated periodically. The control system's alarm handler may monitor the heartbeat and inform operators when a system needs attention. Alternatively, a local watchdog timer may be used to automatically reset the system upon heartbeat loss.

Remote power control and remote reset are useful in larger facilities or when equipment is located in areas with personnel access restrictions. The extreme case is equipment in the accelerator enclosure. These devices may receive radiation exposure that causes single-event upsets without permanent damage, but still require a reset to return to a known state.

6.2 Remote reconfiguration

When remote reconfiguration is desired, the following techniques are commonly implemented to avoid sending the ‘turn antenna away from earth’ command.

- Protected boot memory containing communication code
- Permanent communication subsystem (tiny IOC on a card, hard core on FPGA, etc.)
- Out of band communication as an option — but may add interconnect or cable in some cases

To illustrate one safe reconfiguration technique, consider the simple diagram in Fig. 20, depicting various states of the RHIC BPM software. On a power cycle, the system (a Layer 0 device) enters boot mode, in which a fixed-point DSP boots from a special partition in the local flash memory, and then — from another special partition — loads the gate array configuration file which enables communication with the IOC (a Layer 1 device). The DSP then loads the debug code. In this state (Debug Mode in the figure), applications at Layer 2 and above can now communicate with the BPM electronics and command it to enter any of several measurement and calibration modes. The Debug Mode also supports a download of new firmware followed by reprogramming of the flash memory. During this process, the special partitions that support boot and debug modes are left unchanged. This assures that in the case of problems with the new firmware, higher layers of the control system can always re-establish communications with the device by commanding the remote power system to cycle power. After the device resets, boots, and again enters the debug mode, a previous, known-working version of the firmware can be written to flash. Communication with the remote power system occurs over a physically distinct link that does not rely on the BPM device.

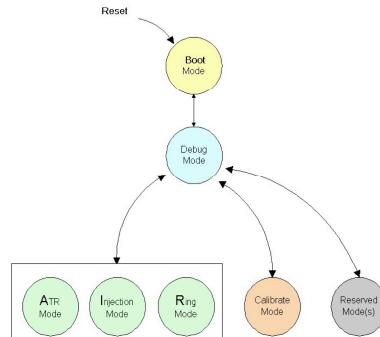


Fig. 20: Operating modes of the RHIC BPM electronics (*courtesy J. Mead*)

7 Physical packaging

Although physical packaging is not the most glamorous aspect of system development, it can directly impact system performance. Physical packaging addresses the following issues:

- Power
- Cooling
- Shielding
- Connections
- Partitioning
- Standardization
- Reliability
- Mean time to replace

Field replaceable units can be modular (like VME cards), self-contained (rackmount, wall mount), or integrated (i.e. embedded within power supply).

Examples of modular packaging standards include the previously mentioned bus standards: VME, PC chassis, CompactPCI, VXI, PXI and ATCA. The broad feature set of the ATCA standard is illustrated by Fig. 21 [23].

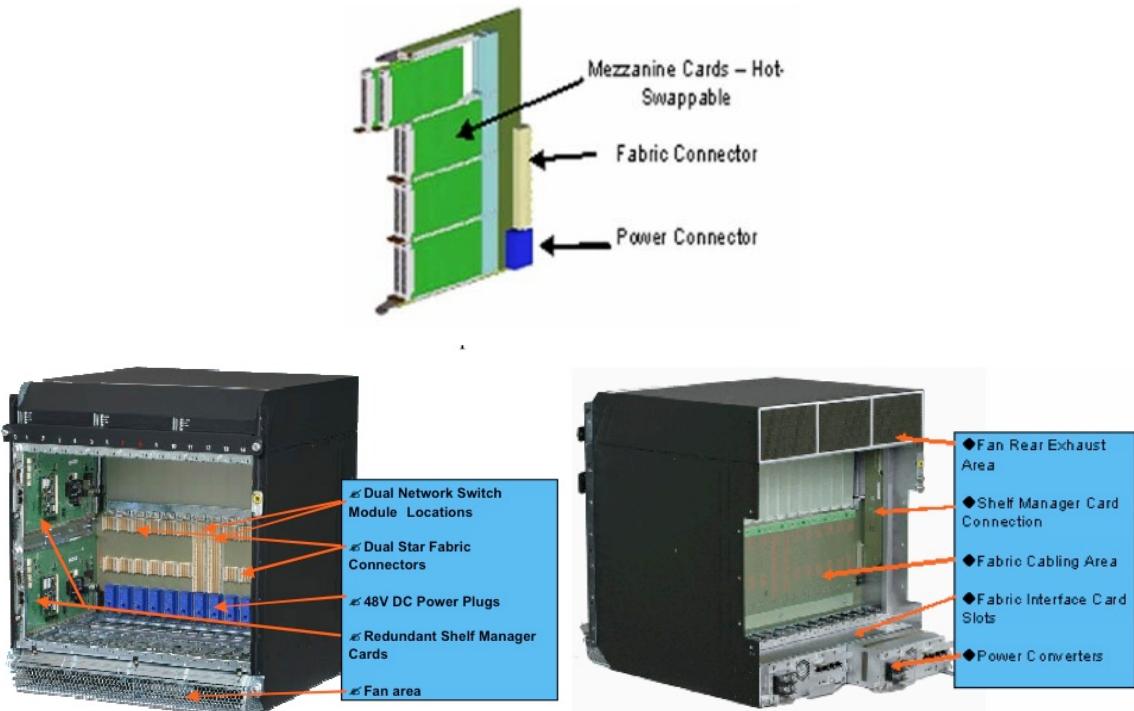


Fig. 21: ATCA standard showing: modules with mezzanine cards (upper), crate front (left), and crate back (right)

If a signal processing subsystem communicates with the rest of the control system via a serial link, self-contained packaging may make sense. One advantage of this approach is that the enclosure may be sized to directly receive long-haul cabling and avoid the separate patch panels that commonly accompany modular crates. However, the designer must contend with several packaging issues such as power and cooling that are engineered into the modular standards.

8 Source code control

A source code control system serves as a central repository, enabling version control, and allowing multiple developers to coordinate their efforts. For example, a bug fix written on a laptop may be merged into the copy on the main development machine, or after a bug fix produces even worse results (usually on Friday afternoon), a system's software can be rolled back to a previous known-working version.

Within the accelerator community, one of the most popular source code control systems is the Concurrent Versions System (CVS). It is open-source software available for virtually every operating system. At EPICS-based facilities like SNS, it is a repository for the following text file types:

- Plain ASCII or LaTeX documents
- EPICS Sequencer and Database sources
- C/C++
- Verilog/VHDL
- Front-end computer configuration files

For binary files, only date and comments are available, and no comparisons are possible within the files. At a facility like SNS, other techniques are used to manage binary source files like LabVIEW, but it has been convenient to store binary FPGA configuration files in the CVS repository.

Other systems are available, including subversion, another open-source tool that is becoming more popular. Several commercial systems are also available and in use at accelerator laboratories.

9 Configuration management

During system design and development, versioning can be managed with the source code control systems described in the previous section. As devices move to the field and become operational, a more global solution is indicated. A good configuration management strategy is to employ a single entry system with all configuration data stored in a relational database. Configuration files are then generated from that data so that all updates to files result from changes to records in the database. As an example, Fig. 22 shows the schema (tables and relationships between them) of the SNS device database. As shown in the figure, information in these tables can be extracted in a useful report format by triggering a stored procedure of SQL statements. IOC applications also have direct access to the database via standard web services. Beyond the data required to produce these files, the database can store a variety of information that is useful during operations. For example, a table could catalogue sources of power for all devices. When a power panel is taken down during scheduled outage, all device owners could be informed in advance.

In the SNS, one of the signal processing systems with a large channel count is the linac LLRF system. The linac includes almost 100 LLRF systems, handled by about 50 IOCs. Although the systems control several types of warm and super-conducting cavities, they rely upon a single source base, with the differences handled by configuration setting. Scripts use this configuration data to generate the start-up files and overview displays. An example overview display is shown in Fig. 23.

To be useful, the configuration control system must be supported by virtually all of the technical groups responsible for the accelerator. This is a difficult mandate, often made impossible by a lack of quality tools to interact with the database. A successful system relies on tools that are easily used by anyone in the organization who produces or maintains configuration data. The task of producing these tools and championing their use is arguably the most difficult aspect of deploying and maintaining a facility-wide configuration management system.

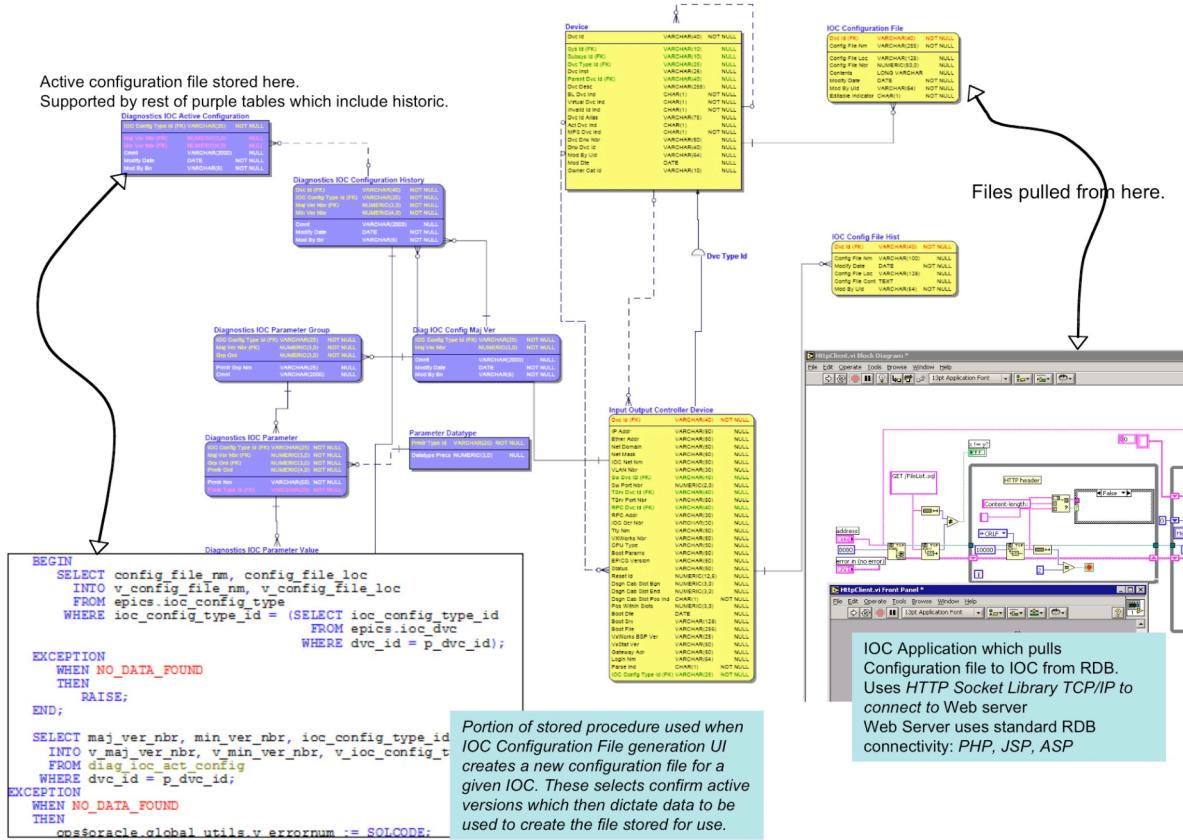


Fig. 22: Schema of the SNS device database (courtesy J. D. Purcell)



Fig. 23: Overview display of SNS linac RF status

Figure 24 shows one user interface to the SNS global database. This tool supports reviewing, editing, loading, and restoring configuration data and tracks changes by recording:

- Who made the change
- When the change was made
- Why the change was made

Configuration files consist of structure and data. Structure (collections of properties that describe the device) is typically common across devices of a specific type (beam current monitors for example). Data typically varies for each device and represents the values for a device's properties. By convention, structure is associated with a configuration's major version number and data is associated with the minor version number.

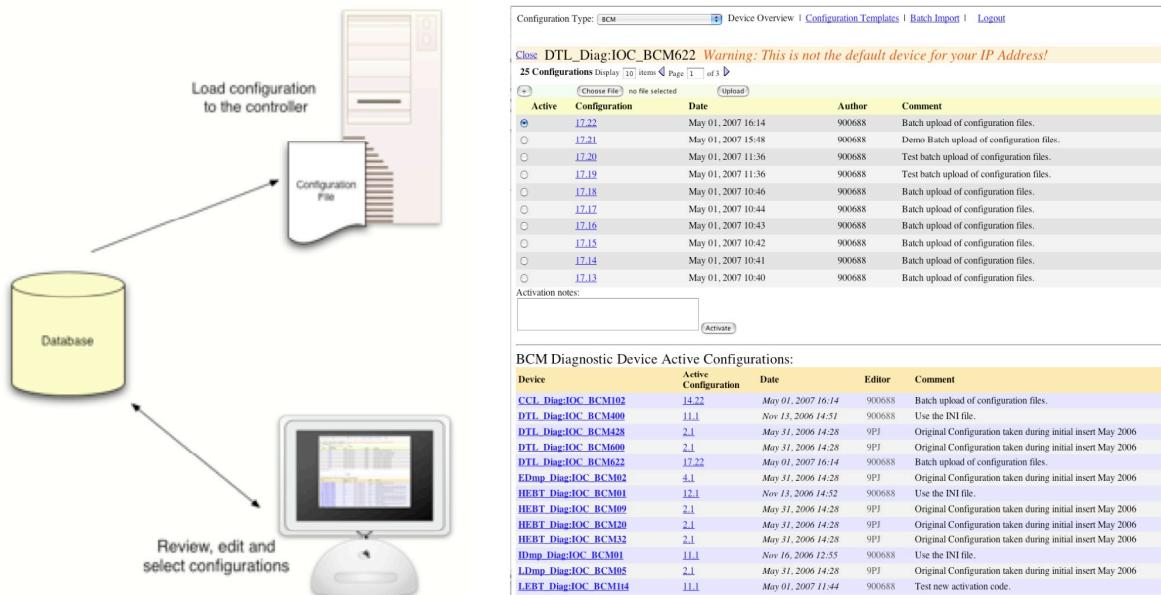


Fig. 24: User interface to a configuration control system (*courtesy T. Pelaia*)

10 System integration and remote debugging

When developing and testing a device on the bench, an engineer can employ all of the rich vendor-supplied tools. After the device is installed in the field, physical access may be limited and a revised toolkit must be employed for integration and in situ debugging. Normally, system experts begin this process by vertically integrating each device, from the beam-line hardware up through the layers of the control system. Horizontal integration follows and addresses coupling and dependencies between devices and systems.

Figure 25 shows the tools developed for vertical integration and testing of RHIC instrumentation. The left side of the diagram shows the standard control system from Layer 0 on the bottom to Layer 3 on the top. The right side shows the high-level application used by system engineers and its connections at various points along the vertical data-flow path. The engineering application — ideally based on the bench testing tools — communicates via the control system network so that it can be used anywhere in the facility, from a laptop adjacent to the device under test, to a console in the control room. In the RHIC case, these communications rely on Snap, a custom protocol that had one primary design constraint: it should be so simple that a programmer could implement and test it on virtually any platform in one weekend. This allows the engineer to use nearly any programmable tool for the automation of the engineering application. LabVIEW, Matlab, C++, Java, and various scripting languages can all be supported. The two upper layer protocol translators (Snap-cdev and Snap-ADO)

can run on the same computer that hosts the engineering application, or can be run on a separate server. Snap-memory runs as a VXWorks process in the Layer 1 front-end computers.

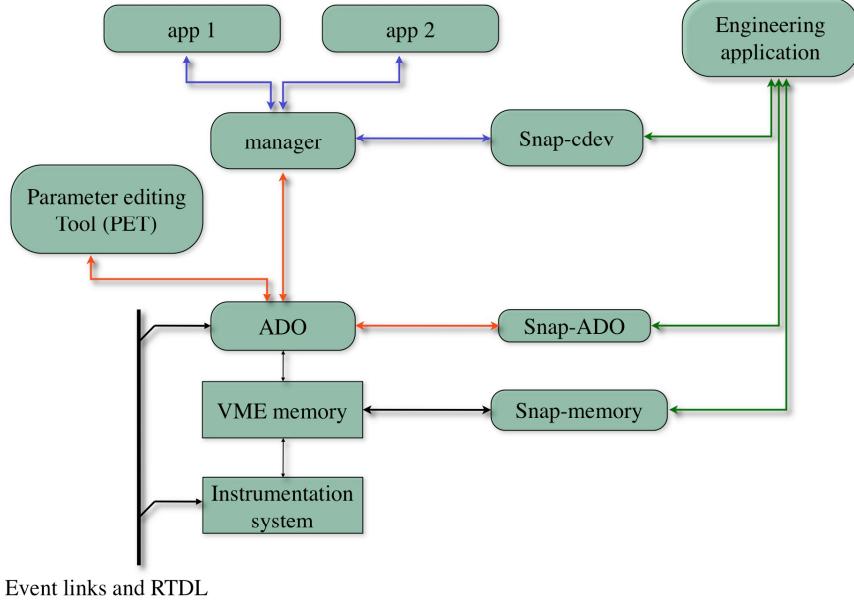


Fig. 25: Tools used for vertical integration and testing

Debugging at Layer 0 was accomplished at RHIC by reading and writing VME memory locations that mapped directly to the low-level device registers. In many devices, these registers reside in memory that is shared by the control system processor and the signal processing system. An example of device register definitions is shown in Fig. 26. This snapshot is taken from the SNS LLRF documentation. As an alternative, some systems make JTAG or other debugging ports accessible from the network via an IOC. The engineering application can set raw configuration data and acquire waveforms directly from a digitizer's buffers. The raw waveforms can be compared to filtered results to demonstrate proper behaviour of the signal processing system. Typically, the operating system at Layer 1 also provides similar access from the console. Figure 27 demonstrates access to a DDS frequency register, whose functionality and address (22 in hexadecimal) was defined in Fig. 26. In Fig. 28, an EPICS screen provides a graphic interface to the same register.

Register[†] LLRF.DDS.FREQ

The offset frequency for the on-board Direct Digital Synthesizer (DDS). This is a signed 16-bit number that adjusts the frequency of cavity operation, relative to the Master Oscillator, in the range of ± 625 kHz. One bit corresponds to 19.073486328125 Hz exactly (assuming the RF sampling clock is a perfect 40 MHz).

Register LLRF.STATUS

Read-only bit map of module status.

D15	D14	D13	D12	D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0
-	-	-	-	-	-	-	-	CLP	FLT	LRC2	LRC1	PLL	IL	RF.ON	KCM

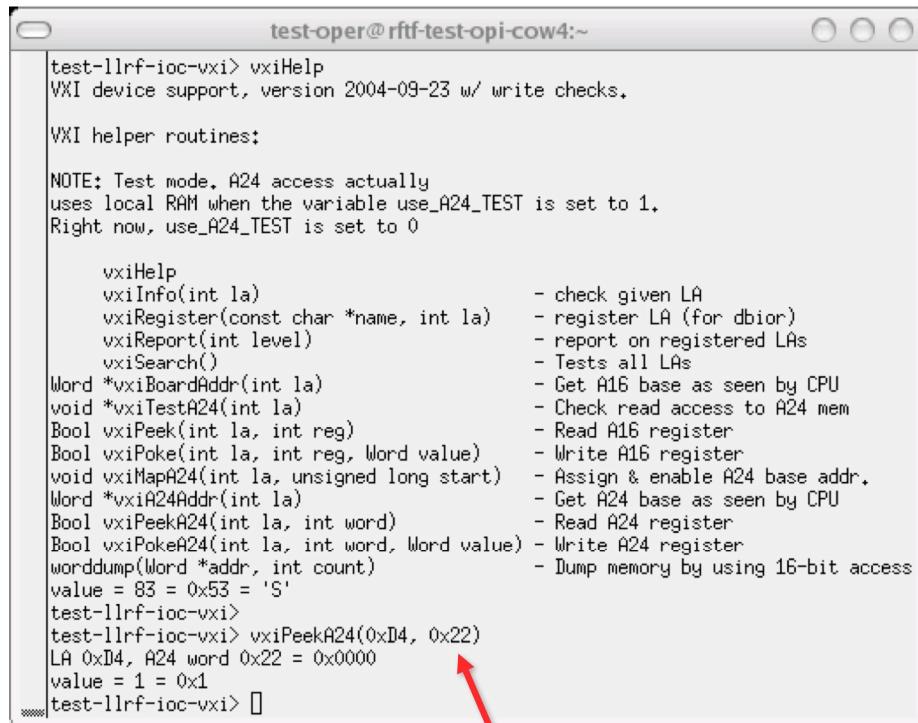
CLP: Latched indicator of occurrence of any premature termination of RF due to output magnitude clipping fault. Cleared at beginning of each pulse.

FLT: Latched indicator of occurrence of any premature termination of RF due to faults within the pulse. Cleared at beginning of each pulse.

LRC2,LRC1: FCM-specific readout of the configuration switches that define its position. 0=Left, 1=Center, 2=Right, 3=Invalid. Non-FCM systems read 0.

PLL: Instantaneous readout of the same signal described in the PLL bit of the

Fig. 26: Example of register definitions



```

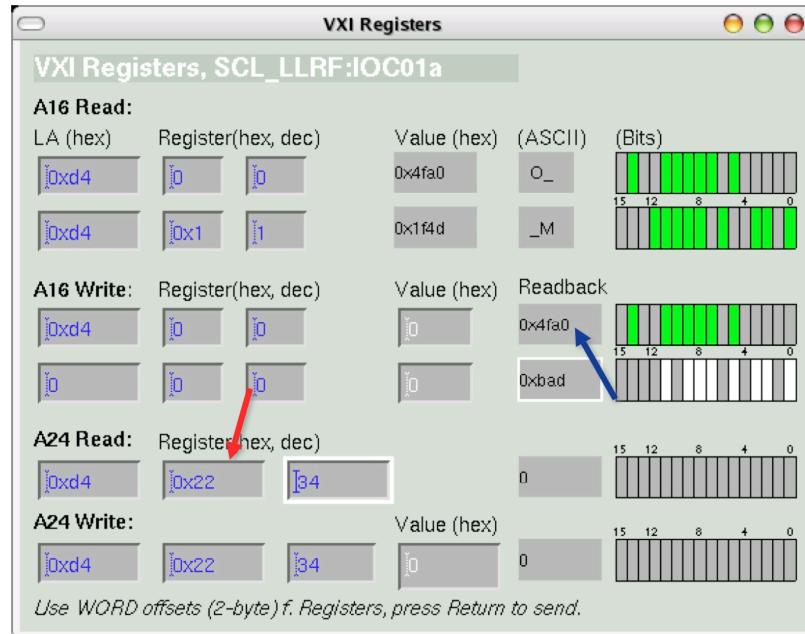
test-llrf-ioc-vxi> vxiHelp
VXI device support, version 2004-09-23 w/ write checks.

VXI helper routines:

NOTE: Test mode, A24 access actually
uses local RAM when the variable use_A24_TEST is set to 1.
Right now, use_A24_TEST is set to 0

vxiHelp
vxiInfo(int la)           - check given LA
vxiRegister(const char *name, int la) - register LA (for dbior)
vxiReport(int level)      - report on registered LAs
vxiSearch()                - Tests all LAs
Word *vxiBoardAddr(int la) - Get A16 base as seen by CPU
void *vxiTestA24(int la)   - Check read access to A24 mem
Bool vxiPeek(int la, int reg) - Read A16 register
Bool vxiPoke(int la, int reg, Word value) - Write A16 register
void vxiMapA24(int la, unsigned long start) - Assign & enable A24 base addr.
Word *vxiA24Addr(int la) - Get A24 base as seen by CPU
Bool vxiPeekA24(int la, int word) - Read A24 register
Bool vxiPokeA24(int la, int word, Word value) - Write A24 register
wordddump(Word *addr, int count) - Dump memory by using 16-bit access
value = 83 = 0x53 = 'S'
test-llrf-ioc-vxi>
test-llrf-ioc-vxi> vxiPeekA24(0xD4, 0x22)
LA 0xD4, A24 word 0x22 = 0x0000
value = 1 = 0x1
test-llrf-ioc-vxi> []

```

Fig. 27: SNS LLRF register access via the console (*courtesy K. Kasimer*)Fig. 28: Register access over the network via EPICS screen (*courtesy K. Kasimer*)

At RHIC, Layer 1 functionality is provided by Accelerator Device Objects (ADOs). At this layer, data might be time-stamped and undergo a conversion to engineering units. Debugging at this layer is accomplished by allowing the engineering software to communicate with the front end via the Snap-ADO protocol translator. As previously mentioned, EPICS-based systems use IOCs at Layer 1 and communicate via Channel Access. Channel access client libraries are now available on a wide variety of platforms and accessible by most engineering tools. Keeping with the SNS LLRF example, Fig. 29 shows the DDS register control (now in engineering units) on an EPICS screen.

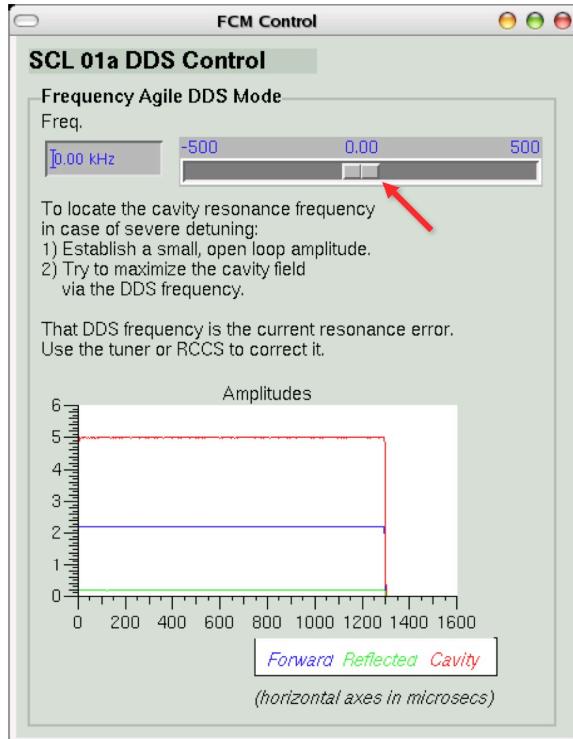


Fig. 29: Register interface in engineering units (*courtesy K. Kasimer*)

At the time of RHIC construction, some of the middleware at Layer 2 used the cdev protocol. This was also supported by a translator and allowed the engineering applications and physics applications access to the same managers. Arrays of values from multiple devices could be analysed in the engineering applications primarily to check for proper correlation.

11 Implementation choices

Although this course focused on the two most popular implementations of signal processing systems, they are among a wide variety of options. The following list is ordered from the highest-level implementations (those that trade performance for flexibility and rapid development) to low-level implementations (those that accept many tradeoffs to achieve higher performance).

- Human
- Commercial/scripted at high level
- High-level application (typically multi-user OS)
- Low-level application (typically RTOS target)
- Embedded (DSP, typically limited OS)
- FPGA
- ASIC
- Analog

The tradeoff between performance and rapid development is evident in the 1997 benchmark results shown in Fig. 30. Even 10 years ago, FPGA implementations based on distributed arithmetic were faster than general-purpose processors, which were in turn faster than high-level

implementations. However, the implementation time varied widely, from a few minutes in LabVIEW to several days for the FPGA [24].

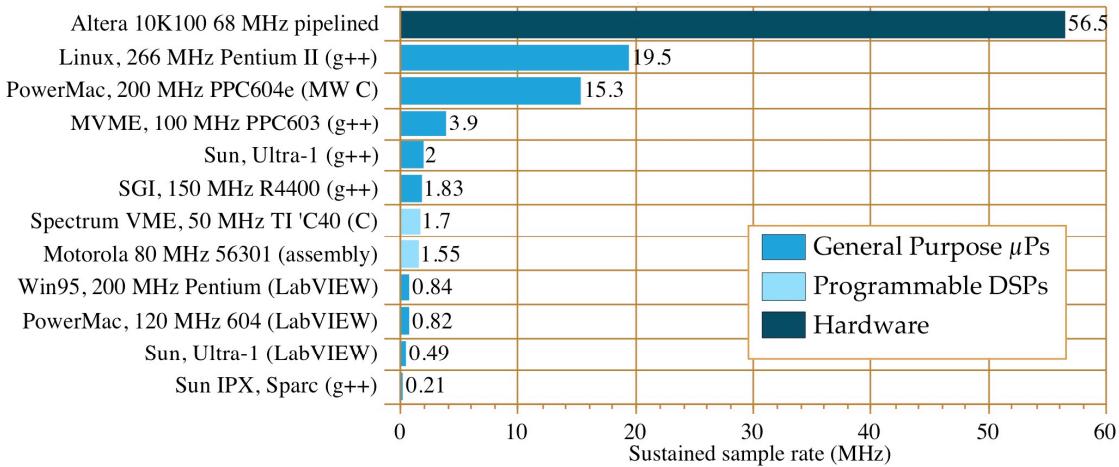


Fig. 30: Benchmark results for the signal processing kernel from the RHIC BPM system

For a recent example, consider the implementation choices made in the development of the SNS LLRF systems [25]. In order from the highest level to the lowest, this implementation included the following blend of technologies from the EPICS control system toolkit and the RF and digital engineers' toolkit:

- Matlab provided an environment for algorithm test and development. These algorithms were eventually deployed at lower levels.
- Extensible Display Manager (EDM) screens provided the user interface. The display manager includes an editing mode to place labels, numerical indicators, meters, graphs and other objects on a screen. These objects can be connected to process variables to allow live updates to and from the IOCs. Important features of this editing process are that it results in a text-based configuration file and does not require code development. An example EDM screen was shown in Fig. 29.
- The IOC uses the EPICS state machine tool for automation and the runtime database for data flow. The state machine tool, also called a sequencer, was used for tasks that required response times of 100 ms or longer. It can run on the host as well as on the IOC, and has the advantage that the sequences can be started, stopped, and updated without a reboot. Standard EPICS record processing achieved response times down to a few milliseconds.
- C/C++ code was used when higher performance was required, for example, in complex pulse-by-pulse algorithms. Code complexity as measured by lines of code (LOC) increases when algorithms are converted from higher to lower level implementations. In 2007, several SNS LLRF algorithms were converted from sequencer code and database records to C++. Figure 31 shows the resulting increase in lines of code.
- Local feedback loops with a 40 MHz sample rate were implemented in Verilog, VHDL, and AHDL.
- The analog front end was made as simple as possible by directly sampling the IF signal to produce an I/Q data stream.

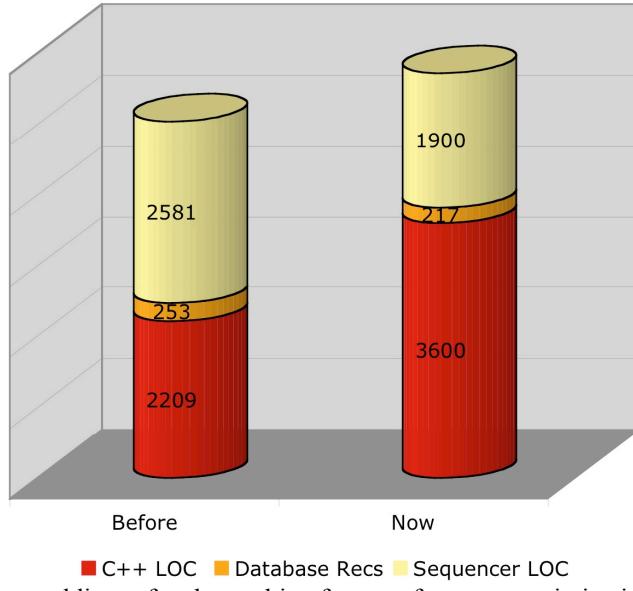


Fig. 31: Increased lines of code resulting from performance optimization in 2007

As is typical in accelerator applications, LLRF requirements evolve through commissioning and operations, so choices tended to be biased toward the options that allowed flexibility and rapid development. Algorithms were first developed in Matlab, and then implemented on the IOC using standard EPICS tools. Only if necessitated by performance requirements, were the functions implemented in C++ or ultimately, within the FPGA.

Commercial tools like Matlab and LabVIEW have long been used for algorithm development and bench testing. Vendors also supply integrated solutions that support FPGAs as a target. Alternatively, signal processing applications have been deployed by simply relying on the Matlab or LabVIEW runtime environments. Matlab is typically deployed at Level 2, with the ALS slow-orbit feedback system being a good example. The script for this is shown in Fig. 32 and demonstrates the simplification that results from utilizing Matlab's rich mathematical functions. LabVIEW has been deployed at Levels 0 and 1 with SNS beam diagnostics being the largest example. Integration with the EPICS IOC software is accomplished by using shared memory libraries as shown in Fig. 33.

```
% Get the vertical orbit
Y = getam('BPMy');

% Get the Vertical response matrix from the model
Ry = getrespmat('BPMy', 'VCM');           % 120x70 matrix

% Computes the SVD of the response matrix
Ivec = 1:48;
[U, S, V] = svd(Ry, 0);

% Find the corrector changes use 48 singular values
DeltaAmps = -V(:,Ivec) * S(Ivec,Ivec)^-1 * U(:,Ivec)' * Y;

% Changes the corrector strengths
stepsp('VCM', DeltaAmps);
Figure <matlab>. A Matlab script implementing closed orbit
feedback
File:Labview IOC shared memory diagram
```

Fig. 32: Matlab script for orbit feedback application (*courtesy G. Portmann*)

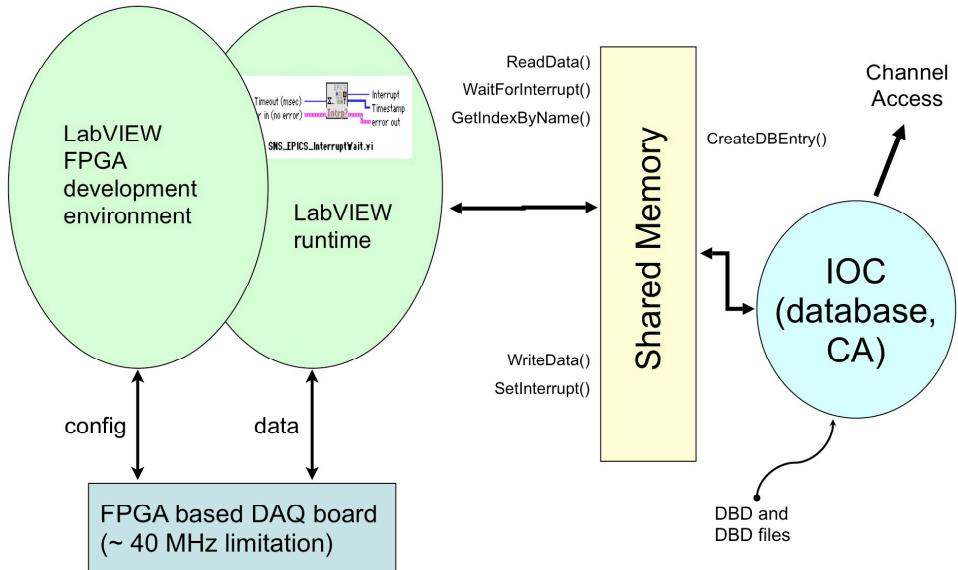


Fig. 33: Architecture of a LabVIEW-based signal processing system (*courtesy W. Blokland*)

Acknowledgements

The author thanks the following people for their contributions to the lecture materials and to this document: Alan Biocca, Wim Blokland, John Carwardine, Bob Dalesio, Larry Doolittle, Dave Gurd, Al Jones, Kay Kasimer, Timo Korhonen, Frank Lenkszus, Joe Mead, Tom Pelaia, Greg Portmann, Dave Purcell, Rick Reidel, Todd Satogata, Don Shea, Michael Shea, Patty Shea, Coles Sibley, and Dave Thompson.

References

- [1] Adapted from a description by L.R. Dalesio.
- [2] D. Gurd, private communication.
- [3] L. Dalesio et al., EPICS Directions to Accommodate Large Projects and Incorporate New Technology, ICAL-EPCS 99, Trieste (1999).
- [4] J.O. Hill, Channel Access: A Software Bus for the LAACS, ICAL-EPCS 89, Vancouver (1989).
- [5] IEEE-583-1982, Standard Modular Instrumentation and Digital Interface System.
- [6] IEEE-1014, Standard for a Versatile Backplane Bus: VME-bus
- [7] W. Blokland, et al., Network Attached Devices at SNS, DIPAC 2003, Mainz (2003).
- [8] M. Bai, P. Cameron, P. Cerniglia, R. Connolly, J. Cupolo, C. Degen, A. Drees, R. Fliller, D. Gassner, J. Mead, V. Ptitsyn, T. Satogata, T. Shea, R. Sikora, P. Thompson, and R. Witkover. Rhic Beam Instrumentation, Nucl. Instrum. Methods Phys. Res. A. **499** (2003), 372–387.
- [9] J. van Zeijts, CDEV Generic Servers for RHIC Commissioning and Operations, ICAL-EPCS 99, Trieste (1999).
- [10] The MathWorks, Inc., Natick, MA.
- [11] G. Portmann, et al., An Accelerator Control Middle Layer Using Matlab, PAC 2005, Knoxville (2005).

- [12] D. Calvet, A Review of Technologies for the Transport of Digital Data in Recent Physics Experiments, 14th IEEE-NPSS Real Time Conference, Stockholm, 4–10 June 2005.
- [13] Conventional PCI 2.3, PCI-SIG, <http://www.pcisig.com>.
- [14] CompactPCI Standard, 1995, PICMG, <http://www.picmg.org>.
- [15] PCI Express 1.1, PCI-SIG, <http://www.pcisig.com>.
- [16] PICMG 3.0 Revision 2.0 AdvancedTCA Base Specification, PICMG, <http://www.picmg.org>.
- [17] J. Postel, User Datagram Protocol, RFC 768, August 1980.
- [18] J. Postel, Transmission Control Protocol, RFC-793, September 1981.
- [19] C. Steier, et al., Orbit Feedback Development at the ALS, EPAC 2002, Paris (2002).
- [20] F. Lekszus, State-of-the-Art Developments in Accelerator Controls at the APS, PAC 1999, New York (1999).
- [21] C. Sibley, Machine Protection Strategies for High Power Accelerators, PAC 2003, Portland (2003).
- [22] J. S. Laster, et al., Post Mortem System – Playback of the RHIC Collider, ICALEPS 2001, San Jose (2001).
- [23] R. W. Downing and R. S. Larson, High Availability Instrumentation Packaging Standards for the ILC and Detectors, 2006 IEEE Nuclear Science Symposium, San Diego, November 2006.
- [24] T. J. Shea, DSP Implementations for Accelerator Instrumentation, ICALEPS 1997, Beijing (1997).
- [25] Adapted from presentations by K. Kasimer.

LHC technological challenges: use of digital signal processors in the power converters for the LHC particle accelerator

H. Schmickler

CERN, Geneva, Switzerland

Abstract

The Large Hadron Collider (LHC) is the next accelerator being constructed on the CERN site. It will be installed in a 27 km circumference tunnel, about 100 m underground. The LHC design is based on superconducting magnets (up to 9 T) which operate in a superfluid helium bath at 1.9 K. This machine is scheduled to come into operation in 2008. In all, there will be 1720 power converters having a total steady-state input power of 63 MW and a peak power of 86 MW. They will supply a total current of about 1850 kA and are, in general, characterized by having high current (up to 20 kA) and low voltage with very high precision. We describe the main components of the LHC powering and their challenges. The performance, design constraints, and topologies of the power converters will be presented. We discuss in detail the use of CERN-designed digital signal processor boards with the main emphasis being on the control loop design.

1 Introduction – The LHC project

The European Organization for Nuclear Research (CERN) is a European intergovernmental organization with 20 Member States. It has its seat in Geneva but straddles the Swiss-French border. Its objective is to provide necessary tools for physicists to explore what matter is made of and what forces hold it together. CERN designs, constructs, and runs the necessary particle accelerators and the associated experimental areas. At present more than 6000 physicists from research institutes worldwide use the CERN installations for their experiments. It is the world's largest particle physics centre.

The Large Hadron Collider (LHC), now under construction at CERN, will be the most advanced research instrument of the world's high-energy physics community for the next twenty years. It will allow exploration of the energy frontier above 1 TeV per elementary constituent, by providing proton-proton collisions at the unprecedented centre-of-mass energy of 14 TeV and luminosity of $10^{34} \text{ cm}^{-2}\text{s}^{-1}$ to two large multi-purpose detectors, ATLAS and CMS, and two more specialized experiments ALICE and LHCb. The LHC will also operate as a heavy-ion (Pb) collider. It is served by the CERN injector complex, upgraded to meet the new requirements of the LHC machine.

One outstanding challenge of the LHC is the safe operation with beam energy 7 times, and luminosity 100 times, higher than the previous particle accelerators. The 350 MJ in each beam of the LHC are sufficient to heat up and melt some 500 kg of copper. This requires careful and reliable handling for safe operation and dump of the complete beam within 88 μs in case of failure, as well as controlling beam losses to avoid a transition of a superconducting magnet to the resistive state. A fast, localized beam loss of 10^6 to 10^7 protons ($0.5 \cdot 10^{-6}$ of the nominal beam) could quench a superconducting magnet when operating at 7 TeV.

The high collision energy is achieved by bending and focusing two counter-rotating beams around the circumference of the collider through a system of twin-aperture, high-field superconducting

magnets operating at 8.3 T in superfluid helium at 1.9 K, and bringing them into collision at a small crossing angle in the four locations equipped with detectors (Fig. 1). Specific to the LHC is the large number of superconducting magnets: 1232 main dipoles and 392 main quadrupoles, but also many types of auxiliary magnets (dipole, quadrupole, sextupole, octupole, decapole), correcting multipoles, chromaticity and closed orbit and more generally to adjust beam optics. There are about 8000 magnets.

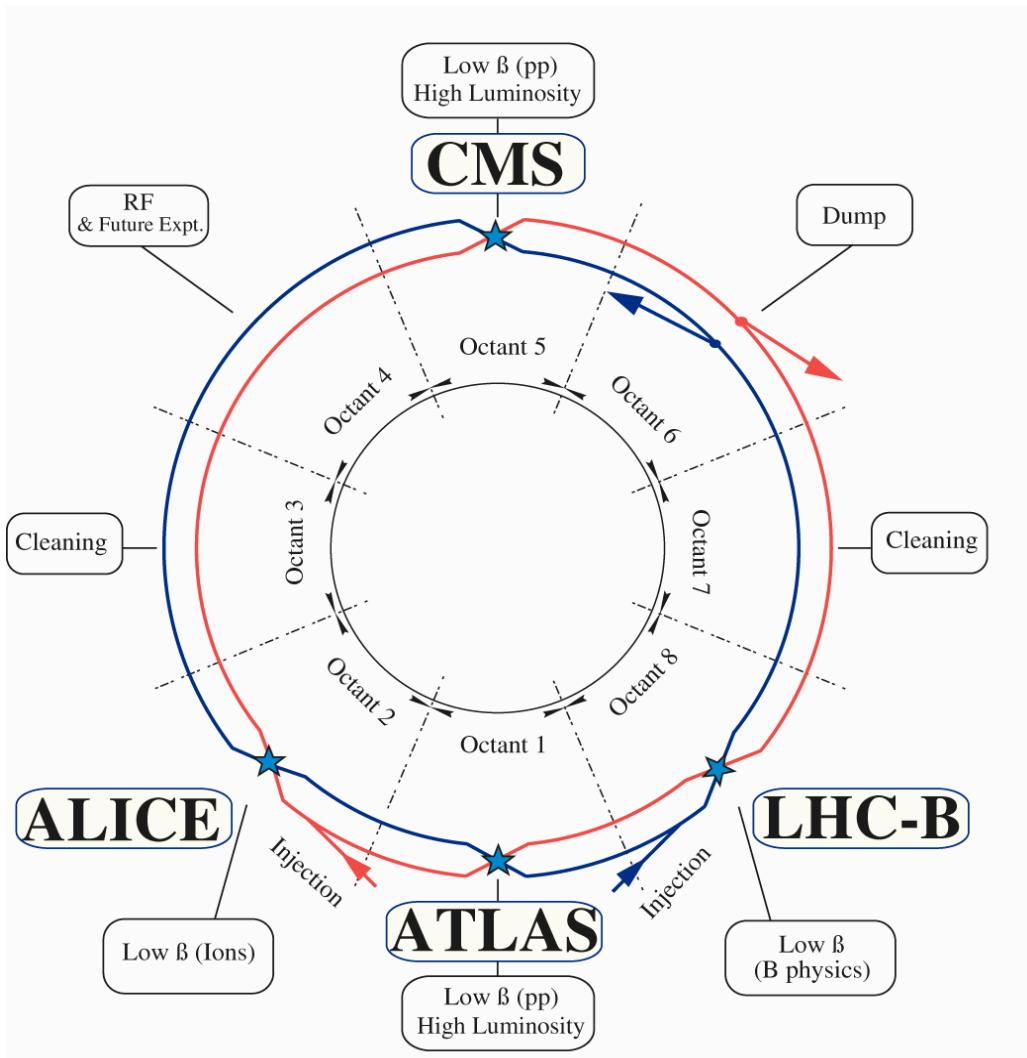


Fig. 1: General layout of the LHC

Another challenge is to handle the huge energy in the magnet system (up to 13 GJ). To illustrate this, 13 GJ corresponds to

- the energy of 3000 kg of TNT
- the energy for heating and melting 15 500 kg of copper
- dropping one LHC main dipole magnet (35 t) from a height of 28 km
- the energy produced by one nuclear power plant unit during about 10 s

Table 1: Main parameters of the LHC

Circumference	27 km
Beam energy at collision	7 TeV
Beam energy at injection	0.45 TeV
Luminosity	$10^{34} \text{ cm}^{-2} \cdot \text{s}^{-1}$
Luminosity lifetime	10 h
Beam current	0.56 A
Protons per bunch	1.1×10^{11}
Number of bunches	2808
Beam size at interaction point	15.9 μm
Typical beam size in arcs	300 μm
Nominal bunch spacing	25 ns 7.5 m
Stored energy per beam	350 MJ
Dipole field at 7 TeV	
Nominal	8.33 T
Ultimate	9.00 T
Operating temperature	1.9 K
Dipole current at 7 TeV	
Nominal	11.8 kA
Ultimate	13.0 kA
Dipole current at injection	760 A
Stored energy in magnets	
Nominal	11 GJ
Ultimate	13 GJ

2 LHC powering

Superconducting magnets have zero resistance and large inductance. As they require high currents, it is important to place the feeding converters close to the cryogenic connections. The consequences are large time constants (e.g. L/R of up to 6 hours for the main dipole circuits, 10 hours for the ATLAS toroid and 39 hours for the CMS solenoid). Management of the electrical stored energy (kJ to GJ range) needs great care.

Superconducting magnets for accelerators are excited very close to their critical current and a small deposition of additional energy (a few mJ) may cause a quench (transition to normal conduction). Such a change of state needs to be rapidly detected and precautions must be taken within the powering system so that the magnet is not destroyed.

A high precision and reproducibility of field is mandatory for LHC to ramp the beam and to get stable physics conditions for the experiments. The field quality in superconducting magnets is determined to a large extent by the current distribution in the coils. While great care is taken in the design and manufacture of the magnets, certain higher-order errors still remain which are of a non-linear and dynamic nature requiring relatively complex correction schemes. The powering system in terms of control, precision, and complexity must take these requirements into account

The LHC reuses the 27 km circumference underground tunnel and technical infrastructure of the former LEP collider. The LHC is divided into eight arcs of 2.9 km with eight straight sections of about 500 m.

The arcs have a continuous cryostat in which are installed the main dipole (bending) and quadrupole (focusing) magnets, as well as various corrector magnets, all operating at cryogenic temperatures.

These circuits, each consisting of up to 154 magnets in series, are powered from one end of the arc by superconducting busbars running through the arc cryostat. The electrical feedboxes, containing high-temperature superconducting (HTS) current leads, are located at the end of the arcs in the machine tunnel. Here, parallel underground galleries were available, or have been excavated, to install the power converters and other equipment needed for powering the machine.

The straight sections have in general individually powered magnets that prepare the beams for collisions or for other machine utilities (RF, collimation, injection, dump). (See Fig. 1.)

For all existing accelerators, all the main bending and main focusing and defocusing magnets are each connected in series giving three main circuits. For the LHC, it was decided to align the electrical segmentation of the machine on the natural segmentation into eight sectors. This choice was made for the following main reasons:

1. For the main dipole circuits, the stored energy would be dangerously high (up to 13 GJ at ultimate current) and large voltages would be needed to de-energize the magnet and especially to extract the huge energy during a quench.
2. Each sector of the machine is galvanically isolated and the possibility of an avalanche quench through the entire machine is avoided.
3. The warm interconnections across the 500 m straight sections would have been expensive, bulky, and would dissipate high power. Some investigations were made to have a superconducting link between the arcs.
4. Installation, testing, commissioning, and maintenance are easier.

The consequence of this segmentation is to have 24 main circuits instead of 3: eight main bending, focusing and defocusing circuits. A good tracking is vital for the LHC beam quality. The accuracy for these 13 kA circuits must be below 20 ppm (parts per million) and the reproducibility should be close to 5 ppm [1], [2]. The achievement of this high precision is certainly the most difficult challenge for the LHC power converters.

Figure 2 shows the powering and energy extraction of one sector of LHC main dipole magnets.

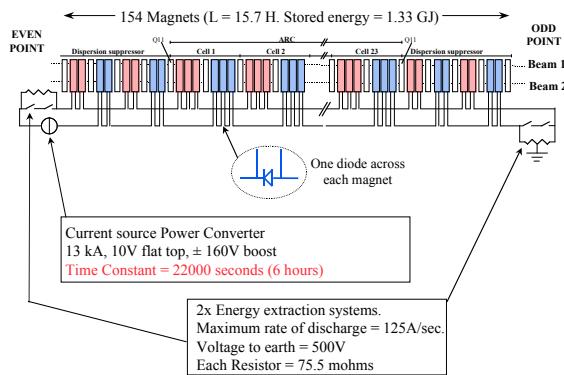


Fig. 2: Powering and energy extraction of one sector of main dipole magnets

In addition to the main circuits, individually powered magnets, mainly located in the straight sections, will require 4, 6 and 8 kA. The correction circuits are powered at 600 A, 120 A and 60 A

and the majority will need to operate in four quadrants. With the exception of the main bend circuits, all circuits require rather low voltages (between 8 V and 18 V). The LHC also uses a few warm resistive magnets that will require 100 kW to 1 MW per circuit. In all, the LHC will have about 1800 circuits requiring approximately 1.8 MA in total.

Furthermore, the LHC experiments require large converters for their magnets (dipoles, solenoids, or toroids):

- ATLAS: [20.5 kA, 18 V] for the superconducting toroid magnet (inductance $L = 7.5$ H and a time constant $\tau = 37\ 500$ s) and [8 kA, 8 V] for the superconducting solenoid magnet ($L = 1.4$ H)
- CMS: [20 kA, ± 26 V] for the superconducting solenoid ($L = 14$ H and $\tau = 140\ 000$ s)
- ALICE: [31 kA, 170 V] for the warm solenoid ($L = 0.33$ H, $\tau = 80$ s); and [6.5 kA, 950 V] for the warm dipole magnet ($L = 1$ H; $\tau = 10$ s)
- LHCb: [6.5 kA, 950 V] for the warm dipole magnet ($L = 1.3$ H; $\tau = 10$ s)

3 LHC power converters

3.1 Performance and design constraints

The performance of the powering system is dominated by the stability of the beam. This is made more complex by the segmentation of the machine and the fact that the field-to-current relationship in a superconducting magnet is a complex function of many parameters, both static and dynamic [3]. Superconducting magnets display dynamic effects even when their current is stable, with changes in field and field errors due to an effect known as the decay of persistent currents. These changes alone, if not corrected continuously, would be fatal to the beams. Further, these errors depend greatly on the past powering history of the magnets and vary very rapidly at the start of a current ramp when the errors due to persistent current decay ‘snap back’ to near their original value. It is particularly delicate at the low currents of injection where great care must be taken to precisely cycle and set the magnets to take into account the effects of DC and AC magnetization and snap-back of the persistent currents. Initial studies have shown that the overall performance of the main circuits, in order to bootstrap the machine, needs to attain about 1.5×10^{-5} of maximum. However, a resolution and short-term stability of the power converters of the order of a few 10^{-6} will be needed to allow precise cycling and fine adjustment.

The installation of the power converters on the surface and the cabling to their underground magnets would be prohibitive because of the high DC currents. Thus underground installation is obligatory and this leads to the necessity for reduced volume and high efficiency of the power converters. It should be noted that a lot of power converters will be installed back-to-back in a confined space. The difficult and restricted access to the underground zones imposed a modular approach for the converter design, allowing quick replacement of faulty modules and off-line repair in surface workshops. To minimize the ventilation installation, low air loss was an important requirement for the design of the power converters. All the power converters are water cooled, except the low-power converters (<1.5 kW).

Because of the high precision, the compact installation of the power converters and the close vicinity to all the other equipment (magnet protection, beam injection and beam extraction systems, experiments, etc.), the Electro Magnetic Compatibility (EMC) has been a severe design constraint for the power converters. Systematic measurements (immunity and emission) were done on the prototypes and also on all production converters.

To achieve these difficult goals, great development effort was needed in the following domains: soft-switching power converter topologies, analog-to-digital conversion techniques, high DC current measurements, and digital control technologies.

All the power converters are split into three independent parts:

- A power part acting as a voltage source.
- High-precision current transducers and analog-to-digital converters (ADCs). All converters will be equipped with two independent current transducers and ADCs.
- A digital electronics control module, which performs the current regulation and makes the link with the slow control network.

This logical and physical separation was created in order to subcontract the largest possible portion of the work to industry (see Section 4).

3.2 Main power converter topologies

The main requirements for the LHC power converters could be summarized as follows:

- Underground installation: low volume, low weight, only front access possible, no access during operation
- High reliability (MTBF \approx 80 000 h)
- Repairability and rapid exchange of different parts
- High efficiency ($> 80\%$ for the unipolar converters and $> 70\%$ for the bipolar converters) and reduced air losses
- EMC: low emission (conducted noise AC and DC side) and high immunity
- Wide output current range ($I_{max}/I_{min} \sim 100$)
- High precision: DC current, low voltage ripple, perturbation rejection, etc.

The above requirements for the LHC converters imply the use of high switching frequencies for very large quantities of converters [4]. Operation at higher frequencies results in a considerable size reduction (volume and weight) for transformers and filter and better dynamics. It gives a better rejection of the perturbations and a lower ripple of the output voltage. However, losses associated with high-frequency operation have to be kept as low as possible to achieve efficient power conversion. Switch-mode power conversion technologies have evolved from the basic PWM converters to the soft-switching converters. The PWM converters process power by interrupting the power flow with abrupt switching (hard switching). This operation results in high losses dissipated in the switching elements during the turn-on and turn-off intervals. It is necessary to include complex and lossy protection snubbers against the effects of the hard-switching, resulting from the presence of parasitic components in the converter. High voltage and current stresses are applied to semiconductor devices.

The attractive properties of soft-switching are

- the large reduction of switching losses
- the improved reliability due to reduced stress
- a limited frequency spectrum, which means an advantage with respect to EMI and losses in passive components
- a reduction of weight and volume of the components resulting from the higher switching frequency
- a higher bandwidth resulting from the high internal switching frequency
- integration of parasitic elements in the commutation mechanism (e.g. leakage inductance of the transformer in the resonant or quasi-resonant circuit).

3.2.1 One-quadrant switch-mode converters

To meet the requirement of a relatively large quantity of multi-kA power converters with similar output voltage and current ratings, the concept of parallelling several smaller current sources was adopted. This architecture provides many advantages:

- A modular approach to the converter design could be made. Much effort could initially be focused on optimizing the sub-converter design, both technically and for manufacture, as this would become the fundamental building block of the converter.
- The quantity of sub-converters could be varied to adapt most closely to the circuit requirements.
- Additional sub-converters could be used to enhance the fault tolerance of the converter, thus a fault in one sub-converter would not interrupt the supply of current to the magnet load. A study was made to optimize the number of sub-converters in relation to cost and MTBF.
- Once the complete converter is in operation, management of a series of related converters is greatly enhanced, both from the perspective of training of personnel and of the issue of maintenance and spare parts.
- In spite of the increased quantity of individual elements, a further advantage of the sub-converter architecture is the use of components with lower rating.

After a long optimization process, only two types of sub-converters are necessary for the LHC machine and experiments: [3.25 kA, 18 V] and [2 kA, 8 V]. Each sub-converter can be considered as a controllable, stabilized, unipolar, current source. Under normal conditions, all the sub-converters work in parallel.

The topology of each sub-converter is split into different stages (Fig. 3):

- An input stage with a circuit breaker, an AC contactor, a three-phase six-pulse diode rectifier, the necessary filtering on the AC or DC sides, and a soft-start circuit to limit the inrush current.
- An inverter stage with a Full-Bridge Zero-Voltage-Zero-Current Switching Phase-Shift inverter (FB-ZVZCS-PS) or Full-Bridge Zero-Voltage Switching Phase-Shift inverter (FB-ZVS-PS) with a switching frequency above 20 kHz.
- An output stage with high-frequency transformers for insulation and adaptation, a Schottky diode rectifier stage and an output filter.

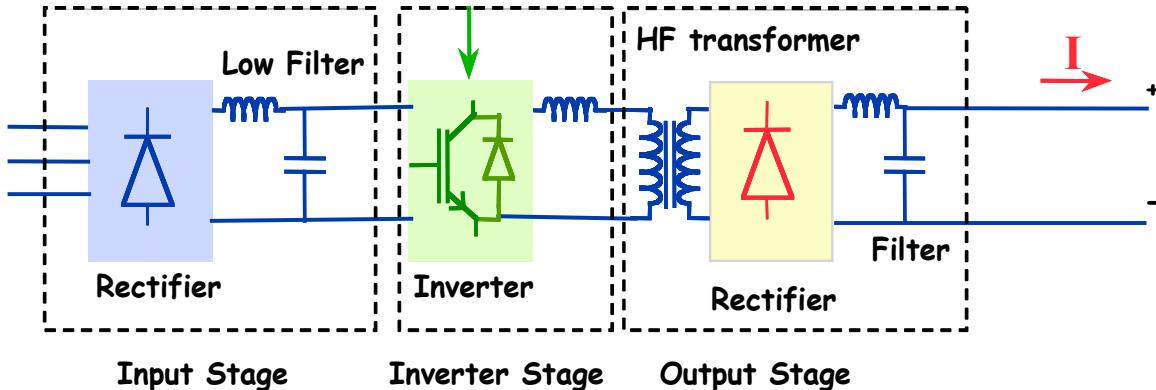


Fig. 3: Sub-converter topology

Thus LHC one-quadrant switch-mode converters were designed as modular water-cooled converters, each with a number of sub-converters, whose outputs are connected in parallel. To improve system availability, each converter uses one sub-converter more than required by the load current. Each sub-converter is identical for all power converter output current ratings. More technical details are given in Refs. [5] and [6].



Fig. 4: [20.5 kA, 18 V] ATLAS converter

Two large contracts have been placed for the design, according to a detailed CERN specification, and the production of these 1-quadrant converters:

- [13 kA, 18 V] converters (18 units made of (4+1) [3.25 kA, 18 V] sub-converters) and one [20.5 kA, 18 V] converter (made of (7+1) [3.25 kA, 18 V] sub-converters) were designed and are being produced by Transtechnik GmbH & Co. (Germany) (Fig. 4) [7].



Fig. 5: [6 kA, 8 V] power converter

- [8 kA, 8 V] (24 units made of (4+1) [2 kA, 8 V] sub-converters) and [6 kA, 8 V] (176 units made of (3+1) [2 kA, 8 V] sub-converters) were designed by Kempower Oy and are being built by Kemppi Oy (Finland) (Fig. 5) [8].

3.2.2 Four-quadrant switch-mode converters

The LHC machine will make extensive use of true bipolar power converters with soft zero-crossing (without discontinuity) of the current and the voltage. The industrial use of four-quadrant converters is very limited. From the approval of the project in 1994, special development programmes were launched at CERN in collaboration with universities and industries on different topologies [9], [10].

The topology of the four-quadrant switch-mode converters is similar for all four types of converter: $[\pm 600 \text{ A}, \pm 10 \text{ V}]$, $[\pm 600 \text{ A}, \pm 40 \text{ V}]$, $[\pm 120 \text{ A}, \pm 10 \text{ V}]$, and $[\pm 60 \text{ A}, \pm 8 \text{ V}]$ (Fig. 6).

It includes:

- A mains rectifier stage with a circuit breaker; an AC contactor; a three-phase, six-pulse diode rectifier; the necessary filtering on the AC or DC sides; and a soft-start circuit to limit the inrush current.
- An inverter stage using a soft-commutated bridge with IGBT switching at high frequencies from 25 kHz to 100 kHz.
- A high-frequency transformer, output filter and a bipolar output stage. A bipolar output stage provides reversal of the polarity. The magnet energy, during the ramp-down of the current, is dissipated by the bipolar output stage.
- An output protection circuit with a free-wheel safety and discharge circuit (also called crowbar).

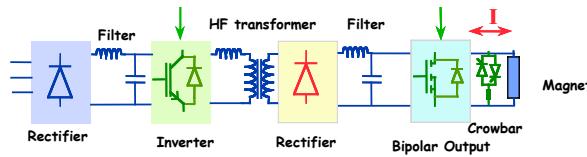


Fig. 6: Four-quadrant converter topology

The topologies of these converters are presented in Refs. [11] and [12].

For the medium water-cooled power converters, two contracts have been placed for the design, according to a detailed CERN specification, and the production of the 600 A power converters:

- $[\pm 600 \text{ A}, \pm 10 \text{ V}]$ (440 units) were designed and are being produced by Cirtem (France) [13] and EEI (Italy) (Fig. 7) [14].
- $[\pm 600 \text{ A}, \pm 40 \text{ V}]$ (50 units) were designed and are being produced by Transtechnik GmbH & Co. (Germany) (Fig. 8) [7].



Fig. 7 Two $[\pm 600 \text{ A}, \pm 10 \text{ V}]$ converters in a rack

For the low-power four-quadrant converters, the design was made at CERN and two production contracts have been placed:

- $[\pm 120 \text{ A}, \pm 10 \text{ V}]$ (330 units) are being produced by Efacec (Portugal) [15].
- $[\pm 60 \text{ A}, \pm 8 \text{ V}]$ (832 units) are being produced by CEL Groupe Cofidur (France) [16].



Fig. 8: One $[\pm 600 \text{ A}, \pm 40 \text{ V}]$ converter in a rack

The $[\pm 60 \text{ A}, \pm 8 \text{ V}]$ converters are used as dipole orbit correctors for the arc regions. They are located under the magnet and are in radiation areas ($\sim 10 \text{ Gy}$ in 10 years). Special developments, component selection, and radiation tests were necessary to withstand the radiation dose and Single Event Upset (SEU) phenomena.

Figure 9 illustrates the zero crossing of the voltage and Fig. 10 the zero crossing of the current for a $[\pm 600 \text{ A}, \pm 40 \text{ V}]$ converter.

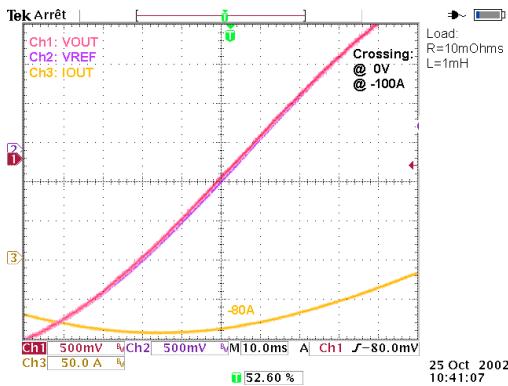


Fig. 9: Zero crossing of voltage for -100 A current (converter $[\pm 600 \text{ A}, \pm 40 \text{ V}]$ with a $10 \text{ m}\Omega$ and 1 mH load)

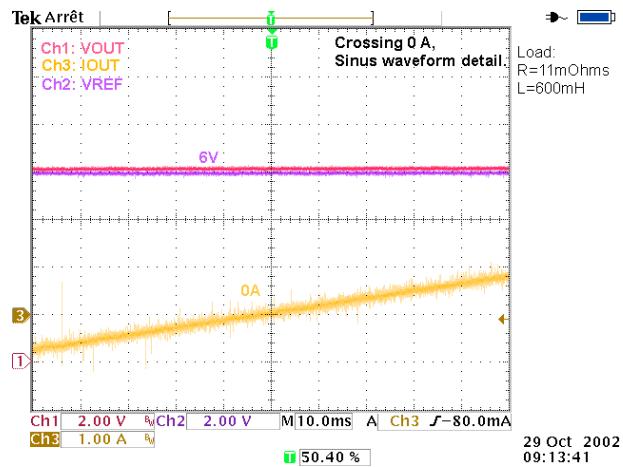


Fig. 10: Zero crossing of current for 6 V current (converter [± 600 A, ± 40 V] with a $11 \text{ m}\Omega$ and 600 mH load)

3.2.3 Output voltage ripple and EMC

The level of conducted noise emission was a very difficult constraint for the development of the LHC power converter. The curve C of the IEC 478-3 norm was applied on the AC side and on the DC side for all the switched-mode power converters (20.5 kA to 60 A). For the output ripple of the voltage source, the C curve was extended for all the frequencies up to 9 kHz according to the dashed curve of Fig. 11.

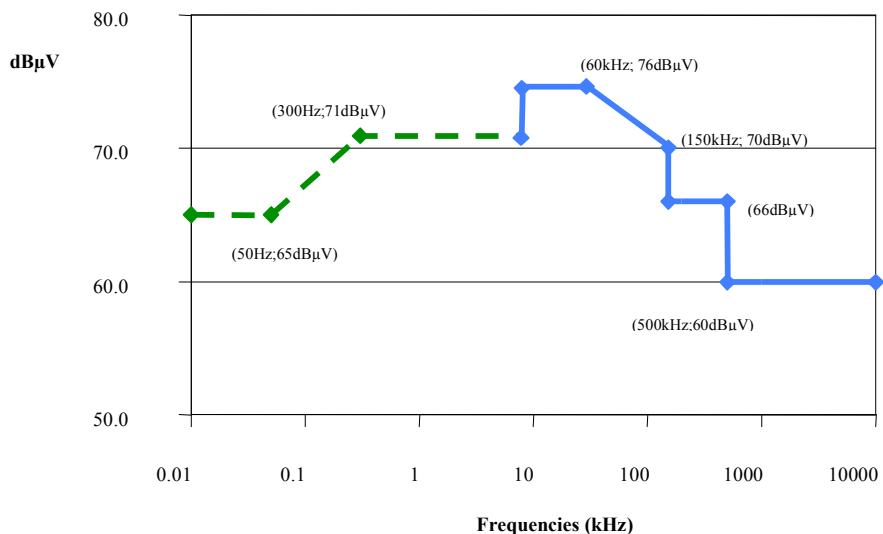


Fig. 11: Differential and common-mode noise levels for the LHC power converters on DC output

Figure 12 illustrates the common-mode measurement on the DC side on one converter type. Similar results have been obtained on all converters.

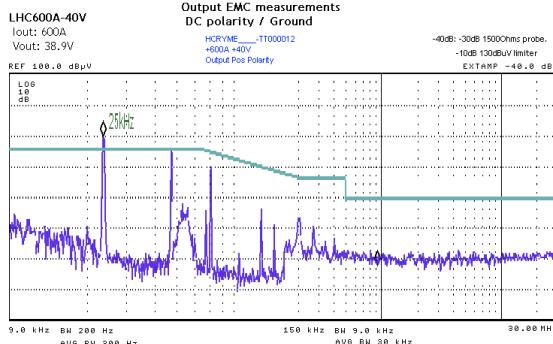


Fig. 12: Common-mode noise measurement on the DC output of [± 120 A, ± 10 V] converter (9 kHz to 30 MHz)

3.2.4 Main dipole power converter: [13 kA, ± 190 V]

Development studies were undertaken to design a suitable switch-mode topology for the main dipole converter [17]. Owing to the large energy and the necessity to have a two-quadrant converter (bipolar in voltage), a line-commutated, phase-controlled thyristor technology was still the most simple, economical, and reliable choice. The gain in volume with switch-mode topology was less than a factor 2.

The eight main dipole voltage sources consist of line-commutated, phase-controlled power converters to which a parallel injection active filter is added to improve mains rejection and ripple performance. To achieve the 13 kA 190 V output rating, a parallel topology (Fig. 13) is used consisting of two sub-converter, each containing an 18 kV-2 MVA cast resin transformer, a six-pulse thyristor rectifier, and a passive filter. The rectifiers are phase-shifted by 30° and connected in parallel. For installation purposes the power converters are made of seven modules: two transformer modules, two thyristor bridge modules, two filter choke modules, and one central module containing the filter capacitors and the output current measurement transducer (DCCT). The thyristor bridges and the filter chokes are water-cooled, while the rest of the equipment is air-cooled.

In order to handle the magnet current run-down under the worst fault condition, e.g., power cut and no cooling water flow, the power converter is equipped with free-wheel thyristors. This free-wheeling system is rated to handle the 13 kA up to 100 s without water-cooling.

In normal operation, the magnet current run-down is made under feedback control with the thyristor bridges used as reverse voltage source, or through the free-wheel thyristors.

The active filter, connected in the passive filter capacitor branch, has a working range of 4% of the total output voltage. This provides rejection of mains transients and gives a wide dynamic range for the control loops.

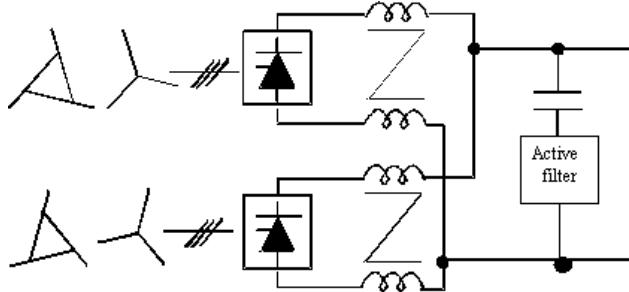


Fig. 13: Main dipole power converter topology

Production contracts have been placed:

- 18 kV/190 V cast resin transformer with Trasfor SA (18 units) (Switzerland) [18].
- Thyristor rectifier parts (9 units) with OCEM (Italy) [19].



Fig. 14: Photo of main dipole converter

(L = 11 m, w = 1.8 m, h = 2.4 m; total weight = 28 t)

3.3 High-precision DC current transducers

A DC current transducer is required to measure the converter output current in order to control it accurately. Currently there is only one technology that will fulfil the LHC accuracy requirements, the zero-flux DC current transducer. The primary current is passed through a toroidal transducer core of special high- μ magnetic material creating a one-turn transformer (Fig. 15).

In the first stage the current is divided with a fixed ratio to a low level. Its bandwidth is extended down to DC through a feedback loop, measuring the DC flux in the core and producing a compensation current, which will balance the flux to zero at all times. The ratio can be established to the ppm (part per million) level.

In the second stage the low-level compensation current is passed through a high-precision burden resistor. The voltage chosen is a compromise between power dissipation and producing enough voltage to overcome noise and thermal emf effects. This voltage is then amplified in a high-precision amplifier providing 10 V output at the nominal primary current.

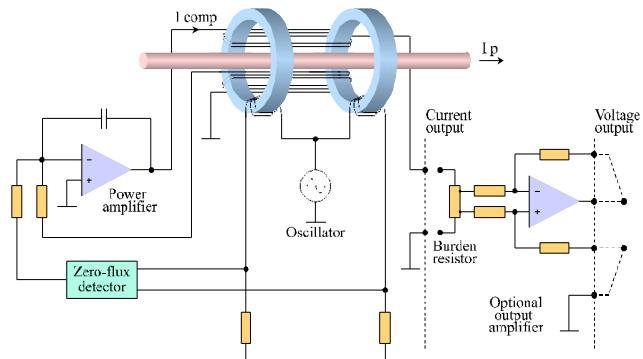


Fig. 15: DCCT principle

The long-term accuracy of a DCCT depends on several factors and needs periodic re-calibration if the highest performance is required. A complete system has been developed to perform calibrations on-site in a fully automated fashion [20]. The principle requires that the DCCTs in question be equipped with an integral calibration winding in the transducer head (Fig. 16).



Fig. 16: 13 kA DCCT (head and electronics)

Three contracts have been placed for the development and production of the LHC DCCTs:

- High-current DCCTs (4 to 13 kA, 473 units) with Hitec Power Protection BV (The Netherlands) [21].
- 600 A DCCTs (1060 units) with Hitec Power Protection BV (The Netherlands).
- 120 A DCCTs (2430 units) with KG Ritz Messwandler gmbh [22].

3.4 Digital control module and high-precision regulation loop

As already mentioned, the LHC demands an order of magnitude improvement in accuracy and reproducibility over previous accelerators. The power converters must follow a very precise acceleration curve with absolutely no over- or under-shoot. It was therefore necessary to use a digital loop for the high-precision current loop. Furthermore it would have been difficult to realize an analog current loop with the high time constant of the magnets.

New technology had to be developed for the ADCs to satisfy the need of precision and the Sigma-Delta conversion principle was judged to be the most promising to meet this aim. An entirely new circuit had to be developed for the highest precision ADCs [23].

The digital control module receives control vectors from the central accelerator control computers and converts them to an output current reference value every millisecond. A digital regulation loop also resides in this module comparing the reference value and the ADC output to calculate the appropriate control value for the voltage source periodically (10–100 ms). This digital value is converted by a digital/analog converter (DAC) to a 0–10 V analog control signal, which is sent as a reference voltage to the voltage source, thereby closing the loop. This current control loop is designed to make the complete system behave like a ‘perfect’ current source.

A special effort has been made towards standardization, using the same electronic control modules for all types of converters. The development of these modules was realized by CERN and a large production contract for 2000 electronics modules was placed with Glentronics Limited (Ireland) [24].

The basic structure of the digital loop is shown in Fig. 17. The general structure of the digital controller can be described by a tri-branched structure known as RST structure.

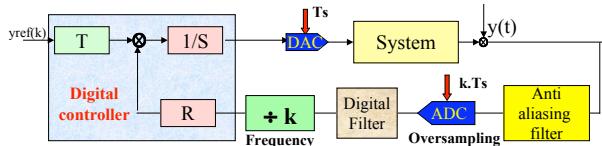


Fig. 17: RST digital control loop

The RST controller makes it possible to obtain the desired tracking behaviour (following the reference) independent of the desired regulation behaviour (rejection of a disturbance). The RST control can be evaluated by the ‘Tracking and Regulation with Independent Objectives’ method (R and S give the regulation behaviour and T gives the tracking behaviour) [25]. This method obtains good tracking of the reference: no lagging error and no overshoot.

Figure 18 shows as an example the round-off at the end of the LHC acceleration ramp. It can be noted that the overshoot is equal to zero.

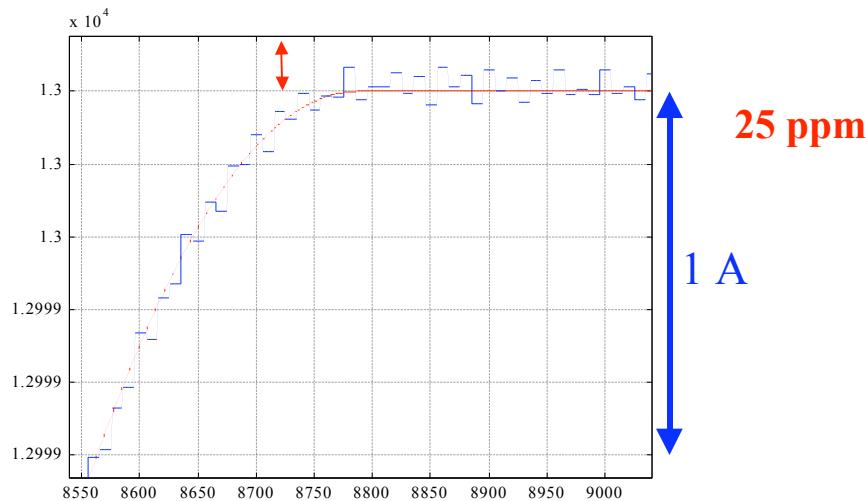


Fig. 18: End of the ramp: from 12 999 A to 13 000 A

4 Production of LHC power converters

It is CERN’s policy to broaden the involvement of industry in all its activities. Only specialized equipment, for which industry would have little or no interest, is designed by CERN.

In line with the previous statement, the strategy for the design, production, and test of the LHC power converters was based on the following considerations:

- Identify subsystems that are suitable for industrial design and production
- Place development and production contracts
- Design and build prototypes of remaining subsystems
- Place production contracts with industry
- Assume system integration responsibility
- Carry out integration and system test at CERN before installation.

With a cost to completion of around 40 million euros, the production of the LHC power converters represents a large industrial project in advanced technologies. Special developments on soft-switching topologies, high-precision ADC and DCCTs, digital control and calibration system and their validation on prototypes were spread over almost one decade. Validations of the performance of the power converters were realized on individual superconducting magnet but also on a full-sized model of a regular cell in the LHC (two quadrupoles and six dipole magnets in series) [26]. After this,

the placement of contracts for series production was launched through competitive tendering and adjudication of commercial contracts.

Acknowledgements

The work reported here is mainly that of the CERN staff participating in the LHC powering and in particular the members of the Power Converter Group, as well as of our partners in industry. The author wishes to thank the many CERN, academic, and industrial colleagues who participated in the projects described in this paper. In particular the author would like to thank Frederick Bordry (head of the Power Converter Group at CERN), who has helped enormously in preparing the seminar in Sweden with his slides and who has provided a good part of the manuscript for the write-up. Originally it was foreseen, that Frederick would give the seminar himself, but time constraints did not allow him to teach at the CAS in Sigtuna.

References

- [1] F. Bordry, "LHC Power Converters: Performance requirements", Proceedings of the LHC, Chamonix XI, France, Jan. 2001.
- [2] F. Bordry, "LHC Powering from String 2 to Sector Test", Proceedings of the LHC, Chamonix XI, France, Jan. 2001.
- [3] R. Bailey et al., "Dynamic Effects and their Control at the LHC", PAC, Vancouver, Canada, May 1997.
- [4] F. Bordry and A. Dupaquier, "High Current, Low Voltage Power Converters for LHC", EPAC'96, Sitges, Spain, June 1996.
- [5] F. Bordry, G. Kniegl, V. Montabonnet, R. Pauls, H. Thiesen and B. Wolfes, "Soft Switching (ZVZCS) High Current Low Voltage Modular Power Converter [13 KA, 16 V]", EPE 2001, Graz, Austria, August 2001.
- [6] F. Bordry, P. Korhonen, D. Nisbet, V. Montabonnet, R. Turunen, H. Volotinen, "Development, Test and Large Production of Soft Switching High Current Power Converters for Particle Accelerators", EPE 2005, Dresden, Germany, Sept. 2005.
- [7] www.transtechnik.com
- [8] www.kempower.fi, www.kemppi.com
- [9] J. Benavent, F. Bordry, J. Carrasco, E. Dede and A. Dupaquier, "On the Design of a High-Power Four Quadrant Power Converter for Superconducting Magnets", EPE'97, Trondheim, Norway, Sept. 1997.
- [10] F. Bordry, J.P. Burnet, A. Dupaquier, P. Coulibaly, F. Iturriiz and T. Meynard, "Novel Topology for Four-Quadrant Converter", EPE'99, Lausanne, Switzerland, Sept. 1999.
- [11] F. Bordry, A. Dupaquier, G. Kniegl and R. Weber, "Four-Quadrant Converter [± 600 A, ± 12 V]. Prototype for LHC", Particle Accelerator Conference PAC99, New-York, USA, March 1999.
- [12] F. Bordry, P. Cussac and A. Dupaquier, "High Precision and High Frequency Four-Quadrant Power Converter [± 600 A, ± 12 V]", EPE'99, Lausanne, Switzerland, Sept. 1999.
- [13] www.cirtem.com
- [14] www.eei.com
- [15] www.efacec.pt
- [16] www.groupe-cofidur.com/cofidur2004/holding/organigramme
- [17] T. Rogne and M. Hernes, "LHC Main Dipole Converter, Study of Power Circuit Topologies", Technical report, EFI – Sintef Group, Trondheim, Norway, Oct. 1997.
- [18] www.trasfor.com
- [19] www.ocem.com
- [20] G. Fernqvist, "The Measurement Challenge of the LHC Project", IEEE Trans. Instrum. Meas., vol. 48, April 1999.
- [21] www.hitecups.com/sms/index.html

- [22] www.ritz-international.de
- [23] J.G. Pett, "A High Accuracy 22-Bit Sigma-Delta Converter for Digital Regulation of Superconducting Magnet Currents", 3rd Int. Conference on Advanced A/D and D/A Conversion Techniques and their Applications, Glasgow, UK, Jul. 1999 , pp. 46–49.
- [24] www.glentronics.co.uk
- [25] F Bordry and H. Thiesen, "RST Digital Algorithm for Controlling the LHC Magnet Current", Electrical Power Technology in European Physics Research EP2, Grenoble, France, October 1998.
- [26] R. Saban et al., "First Results and Status of the LHC Test String 2", EPAC'2002, Paris, France, June 2004.

Participants

ALBERTONE, J.	CERN, Geneva, CH
ALVAREZ SANCHEZ, P.	CERN, Geneva, CH
ARAZ, A.	Institut für Kernphysik, Darmstadt, DE
BELIKOV, O.	Budker Institute of Nuclear Physics, Novosibirsk, RU
BELOHRAD, D.	CERN, Geneva, CH
BOCCARD, C.	CERN, Geneva, CH
BRIELMANN, A.	CERN, Geneva, CH
BURGER, S.	CERN, Geneva, CH
CAMERON, P.	BNL, Upton, NY, US
CATTIN, M.	CERN, Geneva, CH
CHARRONDIERE, C.	CERN, Geneva, CH
CHIEN, Y.C.	NSRRC, Hsinchu, TW
COBANOGLU, O.	INFN & University of Turin, IT
DALY, A.	STFC/ISIS, Didcot, UK
DAMBACHER, M.	Physikal. Institut, University of Freiburg, DE
DAMERAU, H.	CERN, Geneva, CH
DE GASPARI, M.	Physikalisches Institut, University of Heidelberg, DE
DE RUVO, G.	INFN (Sezione di Bari), IT
DENIAU, L.	CERN, Geneva, CH
DUBOUCHET, F.	CERN, Geneva, CH
EMHOFER, S.	Siemens Medical Solutions, Erlangen, DE
FELBER, M.	DESY, Hamburg, DE
FOCKER, G.J.	CERN, Geneva, CH
FONTANINI, S.	Sincrotrone Trieste SCpA, Trieste, IT
FRABOULET, P.	CERN, Geneva, CH
FUJITA, T.	JASRI, Hyogo, JP
GAJEWSKI, W.	CERN, Geneva, CH
GRECKI, M.	DESY, Hamburg, DE
HACKER, K.	DESY, Hamburg, DE
HAGMANN, G.	CERN, Geneva, CH
HAMDI, A.	CEA/Saclay, Gif-sur-Yvette, FR
HEVINGA, M.	KVI, Groningen, NL
HOFFMEISTER, P.	HISKP, University of Bonn, DE
HOMFRAY, D.	UKAEA Fusion, Culham, UK
INGLESE, V.	Università degli Studi di Napoli Federico II, Naples, IT
JALMUZNA, W.	DESY, Hamburg, DE
JEZYNSKI, T.	DESY, Hamburg, DE
KOPREK, W.	DESY, Hamburg, DE
KOTZIAN, G.	CERN, Geneva, CH
KRASNICKY, D.	CTU, Prague, CZ
KRUPCHENKOV, I.	DESY, Hamburg, DE
KUMM, M.	GSI, Darmstadt, DE
KUO, C.	NSRRC, Hsinchu, TW
LANG, K.	GSI, Darmstadt, DE
LENSCH, T.	DESY, Hamburg, DE
LUDWIG, F.	DESY, Hamburg, DE
MAGRANS DE ABRIL, M.	CERN, Geneva, CH
MAKOWSKI, D.	Tech. Univ. of Lodz, PL
MANARIN, A.	CERN, Geneva, CH

MASI, A.	CERN, Geneva, CH
MATLOCHA, T.	NPI, Rez, CZ
MENG-SHU, Y.	NSRRC, Hsinchu, TW
MOLARO, D.	Sincrotrone Trieste SCpA, Trieste, IT
MOLINA MARINAS, E.	CERN, Geneva, CH
MURRAY, S.	iThemba Labs, Cape Town, ZA
NEUMANN, R.	DESY, Hamburg, DE
NIELSEN, J.	ISA, Aarhus, DK
NOLLE, D.	DESY, Hamburg, DE
NOUCHI, P.	CERN, Geneva, CH
ODLUND, E.	CERN, Geneva, CH
OLIVIER, B.	Max Planck Institute for Physics, Munich, DE
ORRETT, J.	ASTeC, Daresbury, UK
OTTOBRETTI, M.	Sincrotrone Trieste SCpA, Trieste, IT
PELLEGRINI, D.	I.N.F.N.
PIQUET, O.	CEA/Saclay, Gif-sur-Yvette, FR
PLUCINSKI, P.	The Andrzej Soltan Institute for Nuclear Studies, Swierk-Otwock, PL
POMPILI, F.	INFN-LNF
POUCKI, V.	Instrumentation Technologies, Solkan, SI
PUCYK, P.	DESY, Hamburg, DE
REPIC, B.	Instrumentation Technologies, Solkan, SI
REYMOND, H.	CERN, Geneva, CH
RODRIQUEZ, J.	PSI, Villigen, CH
ROGERS, J.	CCLRC, Daresbury, UK
RUPPI, M.	INFN (Sezione di Bari), IT
SALOM SARASQUETACELLS, A.	CELLS Sincrotron, Bellaterra, ES
SCHMIDT, C.	DESY, Hamburg, DE
SCHMIDT-FOHRE, F.	DESY, Hamburg, DE
SCHOKKER, M.	CERN, Geneva, CH
SENKOV, D.	The Budker Institute of Nuclear Physics, Novosibirsk, RU
SLAVIN, H.	University of Lugano, CH
SODERSTROM, P.-A.	Department of Nuclear and Particle Physics, Uppsala, SE
SPAZIAN, D.	University of Ferrara, IT
SPIEZIA, G.	University of Naples Federico II, IT
SREEDHARAN, R.	Synchrotron SOLEIL, Gif-sur-Yvette, FR
STAFINIAK, A.	GSI, Darmstadt, DE
SUKHANOV, D.	Budker Institute of Nuclear Physics, Novosibirsk, RU
SUSEN, R.	DESY, Hamburg, DE
TODD, B.	CERN, Geneva, CH
TRABER, T.	DESY, Hamburg, CH
VOLLHARDT, A.	Physik Institut, Zürich, CH
VORENHOLT, H.	KVI, Groningen, NL
VOUMARD, N.	CERN, Geneva, CH
WERNER, M.	DESY, Hamburg, DE
WINNEBECK, A.	HISKP, University of Bonn, DE
WINTER, A.	DESY, Hamburg, DE
WORM, T.	ISA, Aarhus, DK
ZWERGER, A.	Physikal. Institut, University of Freiburg, DE