University Bremerhaven

Course Documentation

Medical Systems
[ES–MED]

- Part 1: Medical Devices and Applications
- Part 2: Embedded Medical Systems
- Part 3: Biopotential Acquisition
- Part 4: Digital Signal Processing of Medical Data
- Part 5: Networks for Embedded Medical Devices
- Part 6: Embedded Devices Labs

Revision: V1.1a

Release: Oktober 2020

Prof. Dr.-Ing. Kai Mueller
University of Applied Sciences Bremerhaven
Institute for Automation and Electrical Engineering
An der Karlstadt 8

D–27568 Bremerhaven / Germany

Phone: +49 471 48 23 – 415
FAX: +49 471 48 23 – 555

Email: kmueller@hs–bremerhaven.de

I Introduction

I.I Course Documentation

See < http://www1.hs – bremerhaven.de/kmueller/> for updates.

I.II Medical Systems

Medical devices are a leading force in the development of embedded systems. The advances in electronic components, DSPs and programmable logic lead to many useful products in consumer medical devices as well as diagnostic, patient monitoring and therapy. “Telehealth” monitoring with networked embedded devices are under development. With the availability of high performance FPGAs medical imaging and new imaging technology (nuclear, positron emission) is making progress.

The course covers the technical aspects of medical devices. Physiological basics are contained as long as they are essential for data acquisition and data processing. The scope if this course extends medical devices, since many methods are also useful for industrial or scientific applications.

Bremerhaven, September 2011

Kai Müller
<kmueller@hs – bremerhaven.de>
phone: +49 471 4823 – 415

II Contents

1 Introduction to Physiology

1.1 Medical Instrumentation for Human Beings

Technically speaking, the human being is a highly developed (over thousands of years in an evolutionary process) embedded system. It has sensors, actuators, a central processing unit, many control loops and energy supply systems. Medical devices take advantage of the fact that information is transmitted on nerve fibers by electrical signals. Therefore it becomes possible to track information by electrodes mounted dedicated positions on the skin. Since these signals are “weak” in the electrical sense, the data acquisition system requires a careful design to amplify small voltages (several mV) and filtering to reject unwanted noise from the environment (biopotential amplification). Nerves and body functions can also be stimulated by electrical signals (e.g. cardiac rhythm management – CRM).

With suitable mechanical and optical sensors other important body signals like temperature, blood oxygen saturation, ventilation and respiration can be processed in addition.

Medical imaging systems are good examples for advances in embedded systems. With the availability of inexpensive and high performance digital signal and image processing devices medical image systems offer precise information in real-time. This involved ultrasound, CT, MRI and X-ray systems. The radiation exposure to patients dropped by a factor of approximately 10 with the introduction of digital X-ray processing.

Modern medical devices are embedded systems. From simple digital thermometers or insulin pumps to high performance medical imaging systems these instruments share data acquisition, digital signal processing, a computerized user interface and with increasing importance, network capabilities.

1.1.1 Classification of Medical Instrumentation

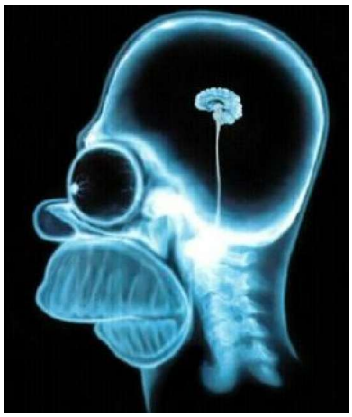
(A) Consumer medical Devices

- Digital thermometers
- Blood glucose monitors
- Blood pressure monitors
- Insulin pumps, general purpose infusion pumps
- Heart rate monitors
- Audiology (digital signal processing hearing aids)

(B) Diagnostics, patient monitoring, therapy

- ECG (cardiac monitor)
- EEG (brain)
- Blood oxygen analyzer (pulse oximetry)
- Blood pressure
- Temperature
- Ventilation, respiration monitors
- Defibrilator
- Implantable devices (cardiac, nerve stimulators etc.)

(C) Medical Imaging



- Ultrasound

- CT (X-ray)
- Digital X-ray
- MRI (magnetic resonance imaging)
- Nuclear imaging
- PET (positron emission tomography)

(D) Hospital Instruments

- Laboratory equipment
- Dialysis machines
- Analytical Instruments
- Surgical Instruments
- Dental Instruments

The devices in section (D) Hospital Instruments are beyond the scope of this course. They consist often of several embedded systems while this course has a focus on single embedded medical devices.

1.2 Body Geometry and Terminology

The standard notation of body regions is important to exactly locate specific parts of the body without confusion. A coarse division of the body is shown in fig.

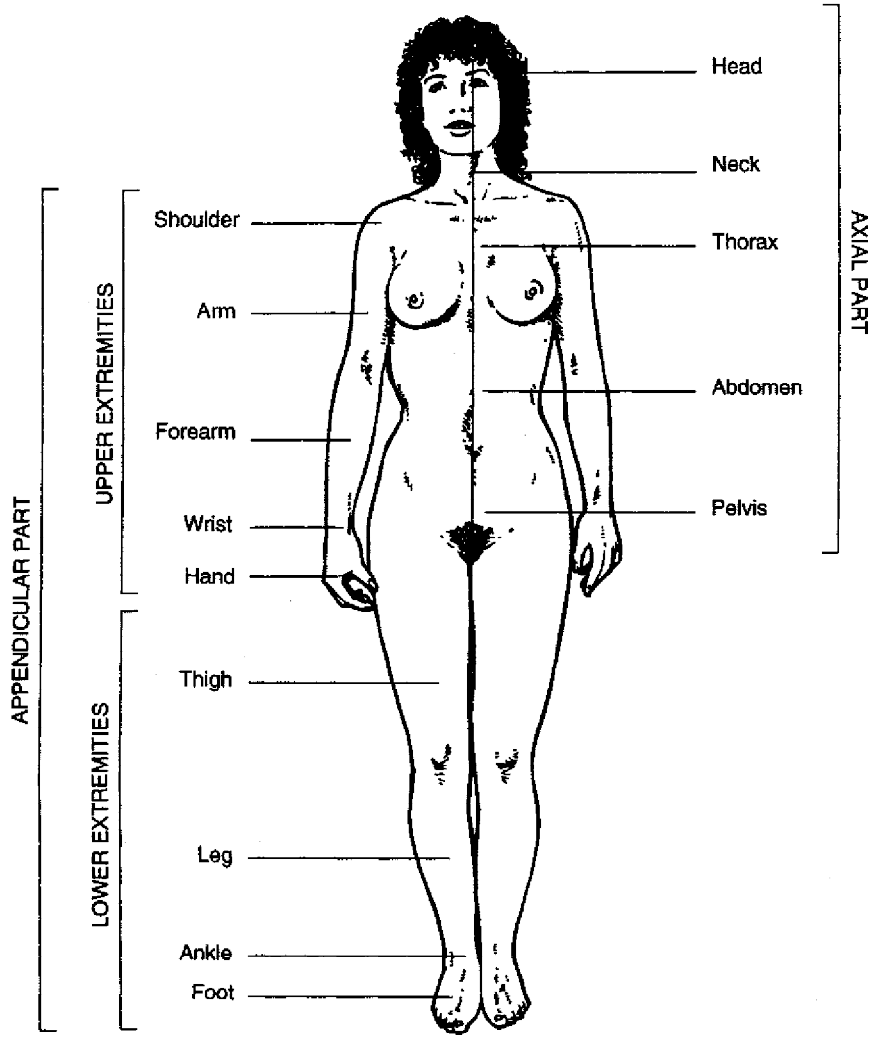


Figure 1.1: Human body regions (according to [4])

Note that there is no overlap between the axial part and the appendicular part as the figure might suggest. The axial part consists of all "central" body elements while the appendicular part consists of all extremities. The body components are shown on fig. 1.2.

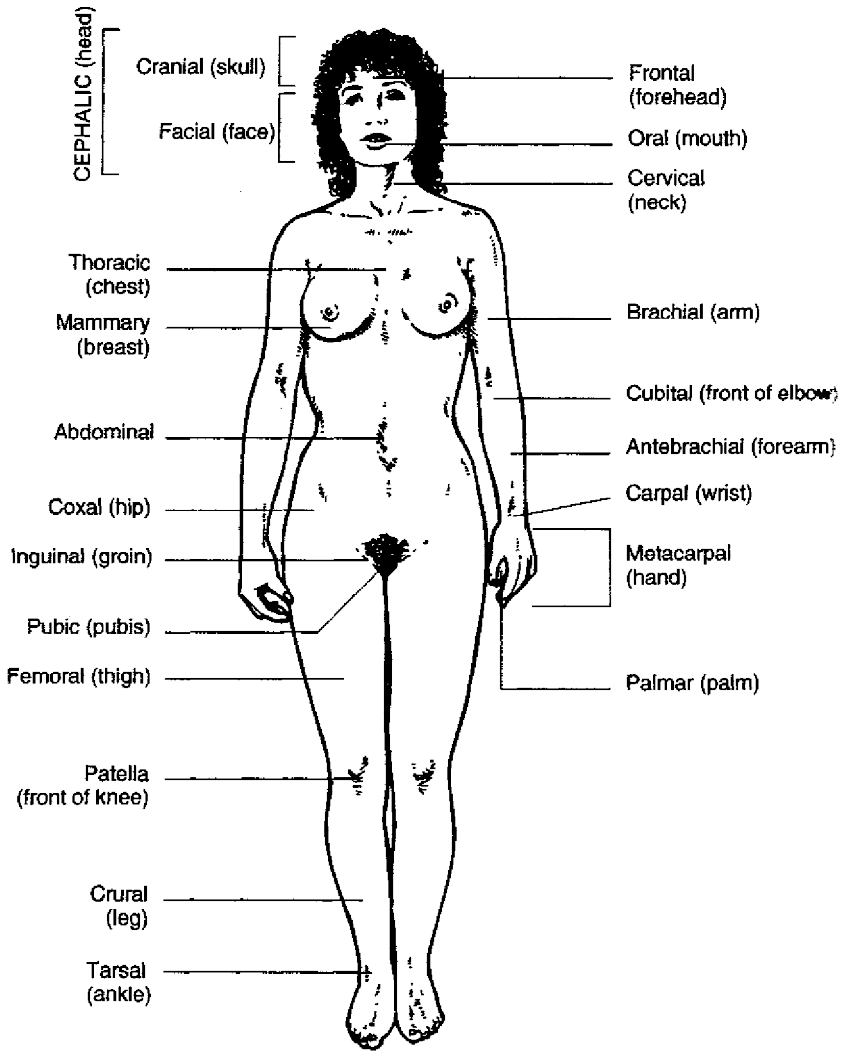


Figure 1.2: Human body components (according to [4])

The body can be geometrically divided in several *planes*. These planes belong to the normal vectors of a cartesian x–y–z coordinate system.

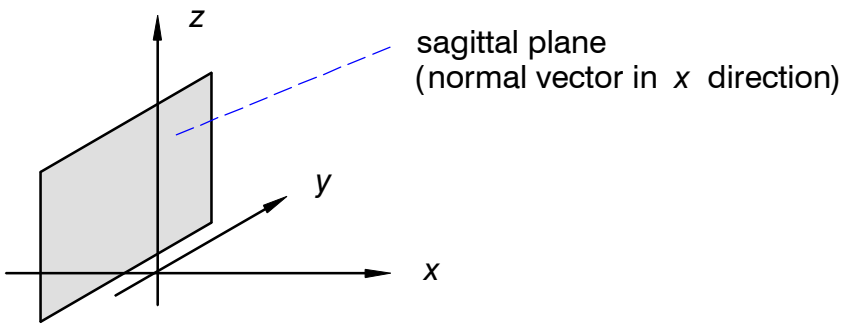


Figure 1.3: Cartesian coordinate system and planes

The planes are denoted as follows:

- sagittal (normal vector in x direction)
  Due to the symmetric body outline a "center" plane exists with the name midsagittal plane
- frontal or coronal (normal vector in y direction)
- transverse (normal vector in z direction)

The planes divide the body into asymmetrical sections. The *midsagittal* plane in fig. 1.4 seems do be a symmetric plane but – of course – the human body is asymmetrical with respect to all planes.

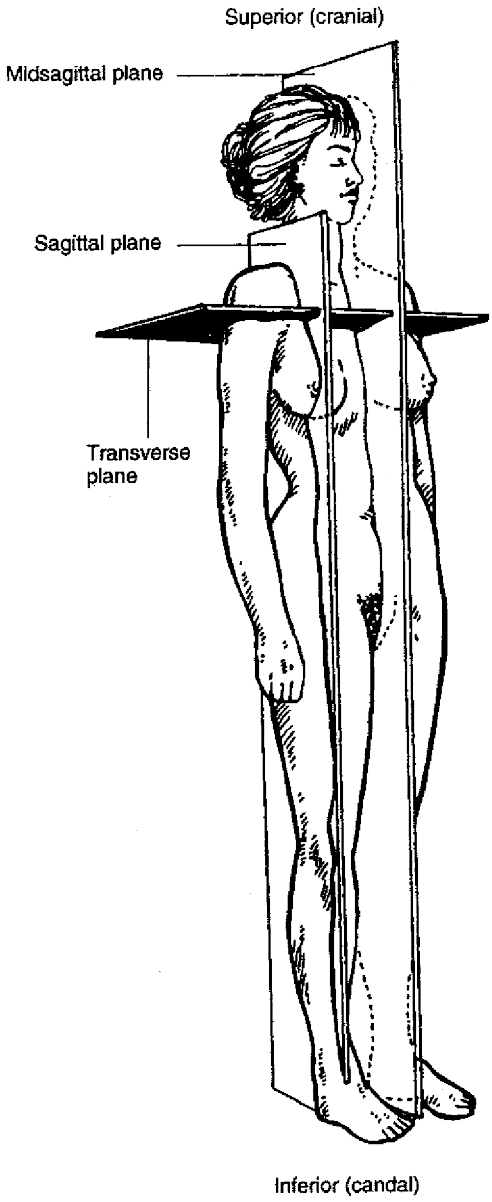


Figure 1.4: Body transverse and side planes – front (according to [4])

The planes are useful to locate specific body areas and to 2-D medical imaging. All planes can be specified according to their position along the coordinate axes.

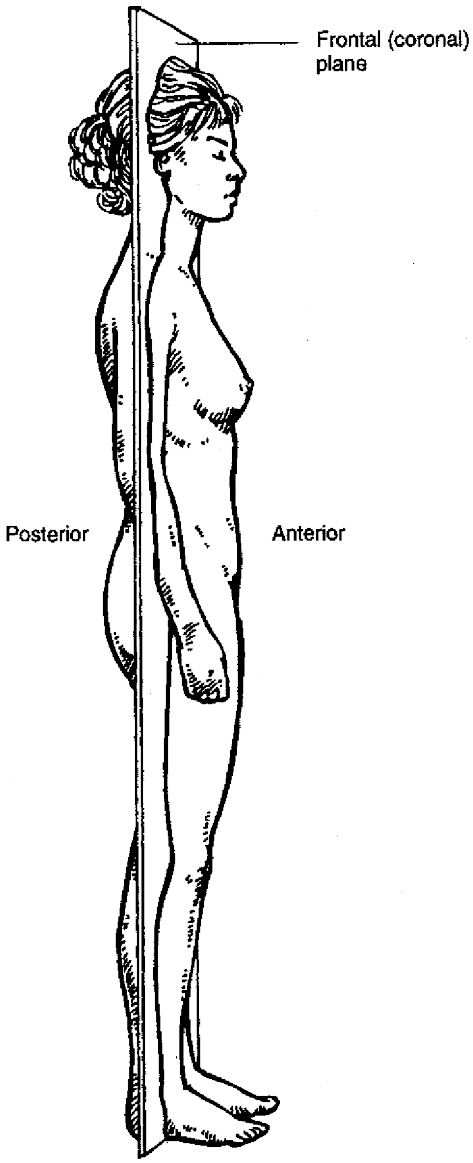


Figure 1.5: Body front (coronal) plane (according to [4])

1.3 Cells and Tissue

The human body is a “decentralized” system. It consists of interconnected (networked) more or less independent components with dedicated functions. The smallest unit relevant to medical devices is the cell. Common to all cells is it’s structure.

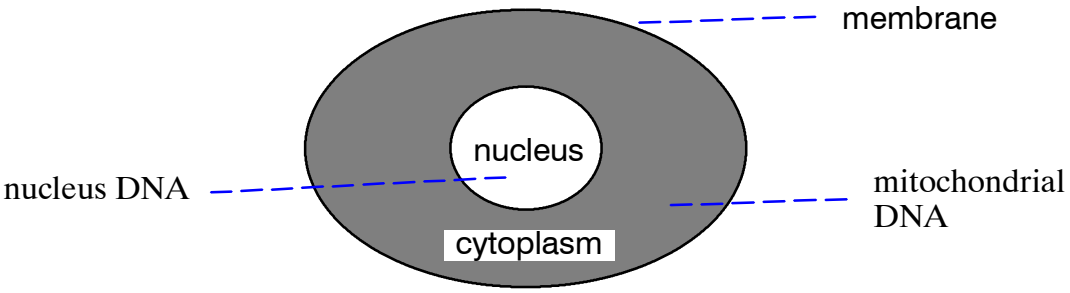


Figure 1.6: Simplified cell structure

The cell nucleus contains the cell specific information, i.e. the "program" of the cell. Multiple dedicated cells form a tissue. Basic types tissues are nerves, muscles, epithelial (skin) or connective (e.g. fiber, blood).

- **nervous tissue:** builds the network for fast information transport between brain, sensors and actuators of the body. Information is transferred by electric current impulses. This allows electronic sensors to track information.
- **muscle tissue:** form the actuator system, the next section treats this in more detail
- **epithelial tissue:** body surface, tubes. This type of tissue is responsible for protection and transport, and digestion.
- **connective tissue:** provides structural support for other tissue. In addition it is responsible for defending the body from harmful bacteria and for storing energy.

> The most important difference between a cell and technical units is that a cell is able to "read" information and to correct errors. This mechanism has not been fully understood. The genetic information in the nucleus contains apparently a lot of useless information which – in fact – is important for the proper function of the cell.

1.4 Body System

Similar to technical equipment, the body can be regarded as interconnected subsystems which provide dedicated functions. Usually these subsystems contain several types of tissues. It is therefore the next higher level with respect of tissue.

The body "system" has many thing in common with technical systems and it's comparison is very helpful to understand the mechanisms of life.

> The human body is – in contrast to technical systems – a *robust* and *self-healing* systems. Even under wrong diagnosis and a false therapy the body can heal itself. This is something an engineer could dream of. Physicians have sometimes an easier job.

The following table list the major body subsystems and compares them to technical devices. This comparison is not meant to be cynical and is done only to illustrate the function of these

subsystems. Life tissue operates often much different from technical devices. The body function classification is taken from [4]. Physicians have specialized for each subsystems.

body subsystem	major function	"technical" equivalence
cardiovascular (heart, blood, blood vessels)	blood pump, supply of energy and transport of waste	electrical, hydraulic, or pneumatic supply, waste removal
digestive (stomach intestines)	break of large molecules in usable form for transfer into blood, remove of solid waste	chemical cracking, waste removal and detoxication
endocrine (ductless glands)	management of hormones to regulate the body (chemical by hormones)	slow high level controller for almost all supersystems
integumentary (skin hair nails, sweat)	body and organ protection, temperature regulation actuator	machine enclosure, protection and temperature regulation
lymphatic (glands, lymph nodes lymphatic vessels oil glands)	immune protection, return of excessive fluid to blood	virus protection, liquid level control
muscular (skeletal, cardiac muscles)	body movement, heat production	main actuators, generation of process heat
nervous (brain, spinal cord, peripheral nerves, sensory organs)	regulates fast responses of the body to events, receives and interprets sensor information, initiation of muscle activities	electronic control unit, wires and network connections, data acquisition and processing, control and actuator processing
reproductive (ovaries, tests, reproductive cells, ducts)	reproduction of cells and beings	– little match to technical systems – limited repair capabilities by redundant components
respiratory (airways, lungs)	mechanism of breathing, exchange of gases between blood and air	gas exchange through membranes, gas filtering
skeletal (bones, cartilage)	protection, level mechanism for movement, production of red blood cells	chassis, wheels, manipulators (robots) – no equivalence to blood cells
urinary (kidney, ureters, bladder)	eliminates metabolic wastes, supports regulation of blood pressure, acid-waste and water-salt balance	waste removal and cleaning system

1.5 Muscular System

The muscles build the actuator system of the body. Their function is closely related to the nervous system, since movements require significant control effort. Muscles are important for body heat production. This section will focus on the actuator task of muscles. The mechanism of muscle movement cannot be explained without some knowledge of the nervous system which will be treated basically. The nervous system itself have an own section 1.6.

Three types of muscles exist:

- **skeletal:** the muscle tissue that is involved in body movement. The muscle ends are connected to the skeletal bones and are organized as long fibres. Their length varies from 1 mm to approx. 30 cm. The visible part of muscles is a *bundle* consisting of muscle *fibres*. The fibre itself consists of smaller fibres called *myofibrils*.

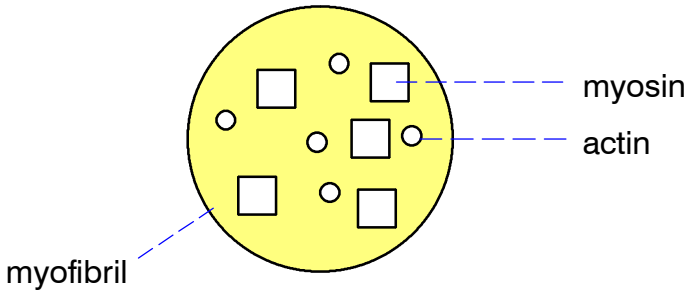


Figure 1.7: Myofibril structure

  The thicker *myofilament* is called **myosin**; the thin myofilament is called **actin**. Calcium bridges between the myofilaments connect myosin and actin. Like most of life functions the mechanism of muscle contraction has not been fully understood. A common theory assumes that the myosin rotates thereby contracting the muscle with the aid of the calcium bridges. Relaxation occurs by releasing the rotation. The muscle is mainly fueled by ATP (adenosine-5'-triphosphate, discovered in 1929 by Karl Lohmann).

- **smooth:** not connected to bones, see section 1.5.2.
- **cardiac:** heart muscles, see section 1.5.3.

1.5.1 Nerve Interaction

In technical systems an information is sent on a (metal) wire. Usually the voltage level carries the information (analog or digital). Along a nerve fiber information is also transferred by electrical signals, but in this case a current impulse carries the information. Since no metal conduction exists in the body the current flow occurs by ionic conduction. Since current flow always imply voltages, the signals can be tracked by suitable electrodes.

Nerve ends (*axon termal branch*) is connected to muscle fiber. This is called a *motor end plate*. The signal transmission occurs electro-chemical by ion charge emission. Fig. shows the mechanism for muscle contraction.

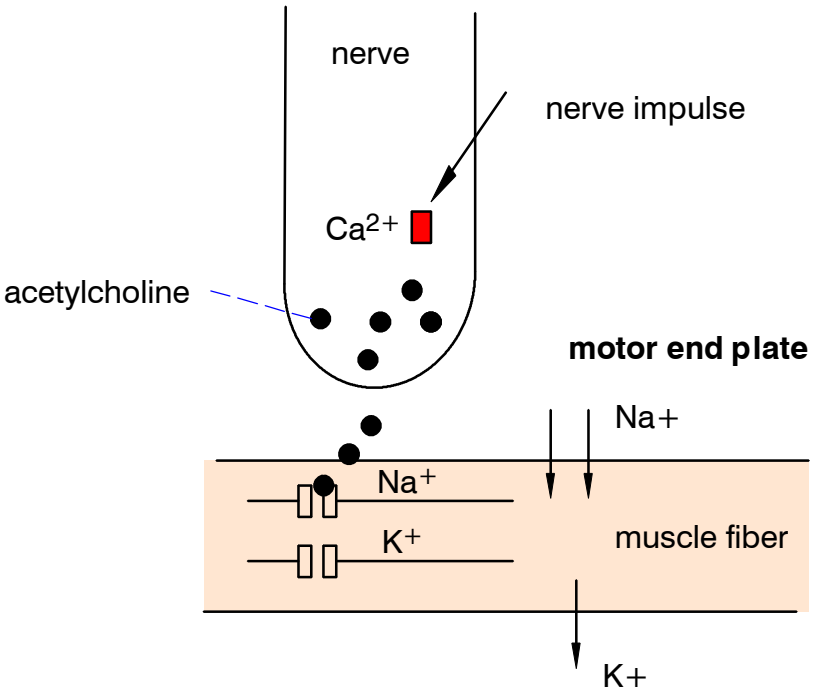


Figure 1.8: Muscle stimulation by ion injection

A nerve impulse traveling along the nerve fiber causes Calcium ions to diffuse to the end of the nerve (motor end plate). So called *neurotransmitters* (acetylcholine molecules) are released into the muscle fiber through it's membrane.

Once neurotransmitters are inside of the muscle fiber the permeability to Na^+ (sodium) and K^+ (potassium) increases. Since Na^+ enters the fiber at a faster rate the fiber will become positively charged. This causes contraction of the muscle due to interaction between myocin and action filaments. The charge of muscle fibers create electric fields or voltages which can be measured (*myogram*).

> If the stimulus extends a certain threshold value, the muscle fiber will contract. Any stronger stimulation will not result in a stronger contraction. In this way muscle fibers are digital systems.

1.5.2 Smooth Muscles

This type of muscle is not connected to bones and is often controlled by an autonomous nervous system. Examples are stomach (digestive) muscles or the respiratory system. Typical is a rhythmical action at a low frequency.

1.5.3 Cardiac Muscles

Cardiac muscles are special and – as the name implies – exist only in the heart. The main difference to other muscles is that they are able to contract rhythmically without external

stimulation. Special junctions allow synchronization with other muscles of the heart. External stimulation works as well allowing medical devices for cardiac rhythm management (e.g. cardiac pacemaker).

1.6 Nervous System

Perhaps the most astonishing development of the evolution is the human nervous system with the central processing unit – the brain. Due to the complexity of the brain until now the function of the brain has not been fully understood. However, medical devices can give many valuable information to the physician to detect possible malfunction of the nervous system.

In technical systems the central processing unit and network devices are separated. For the human body this is not possible or useful. The human nervous system is divided into *central* and the *peripheral* nervous system.

- **central:** brain and spinal cord (can be regarded as CPU of the body)
- **peripheral:** nerves from spinal cord to and from extremity muscles and organs (the network – although nerves are not “bus” organized, instead signal transmission occurs only in one direction)

1.6.1 Central Nervous System

The layout of the brain is shown in fig. 1.9. From the technical perspective it is not clearly design instead it is the result from million of year of evolution. This make it almost impossible to analyze and understand its functions completely. Although it is known that certain areas are responsible for special purposes all areas are highly interconnected.

The brain is well protected mechanically (skull and *cerebrospinal* fluid) as well as chemically. A blood-brain-barrier is highly impermeable and thus prevents toxic substances entering the brain. Otherwise the delicate neurons could be easily damaged.

The **brain stem** is the interconnection between the brain components and the spinal cord. (technically it acts as a relay/router). It also takes responsibility for heart beat rate, and blood pressure, respiratory rate and processes some sensory information (hearing, taste and some other senses).

The **cerebellum** has a definite function: it controls skeletal movement by stimulating muscles.

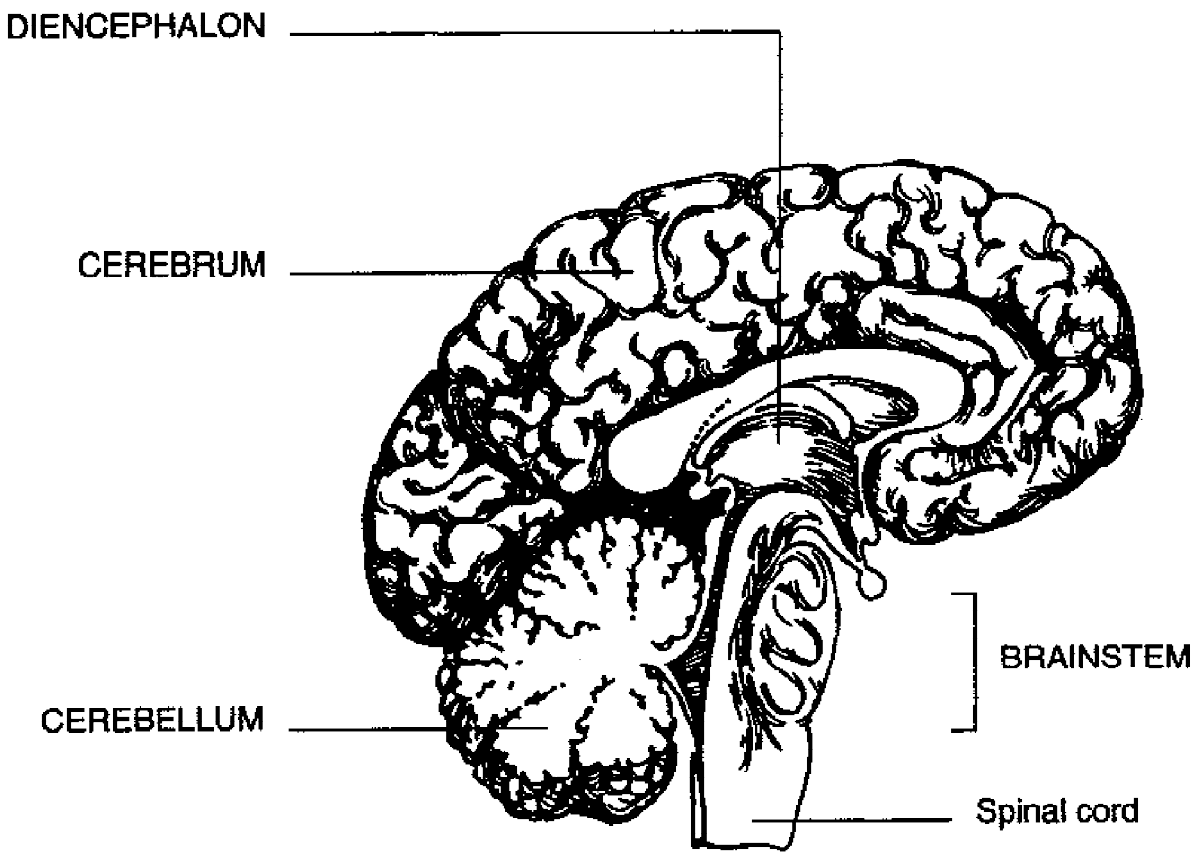


Figure 1.9: Human brain divisions (according to [4]), pituitary not shown for clarity

The biggest part of the brain is the **cerebrum**. It mainly processes conscious activities, e.g. mental activities and body movement. In addition it processes smell (olfaction) in a special part of the cerebrum. It is not surprising that his part is smaller as in some animals. The cerebrum is structured in four regions with more or less dedicated functions. In medical literature the cerebrum is treated as four *lobes* (frontal lobe, parietal lobe, occipital lobe, temporal lobe).

The main functions of the cerebrum are:

- Movement (primary motor cortex)
- Sensory processing (visual, auditory, olfactory): The sensor information is synthesized into our perception of the world around us.
- Language and Communication
- Learning and Memory

The central **diencephalon** (sometimes called interbrain) is responsible for important vital functions (autonomous nervous system, regulate body heat, water balance, sleep/awake, food intake, behavioral responses associated with emotions). It also connects the mid brain with the cerebral hemispheres. It is also involved in control of sensor information with the exception of smell which is performed in the cerebrum.

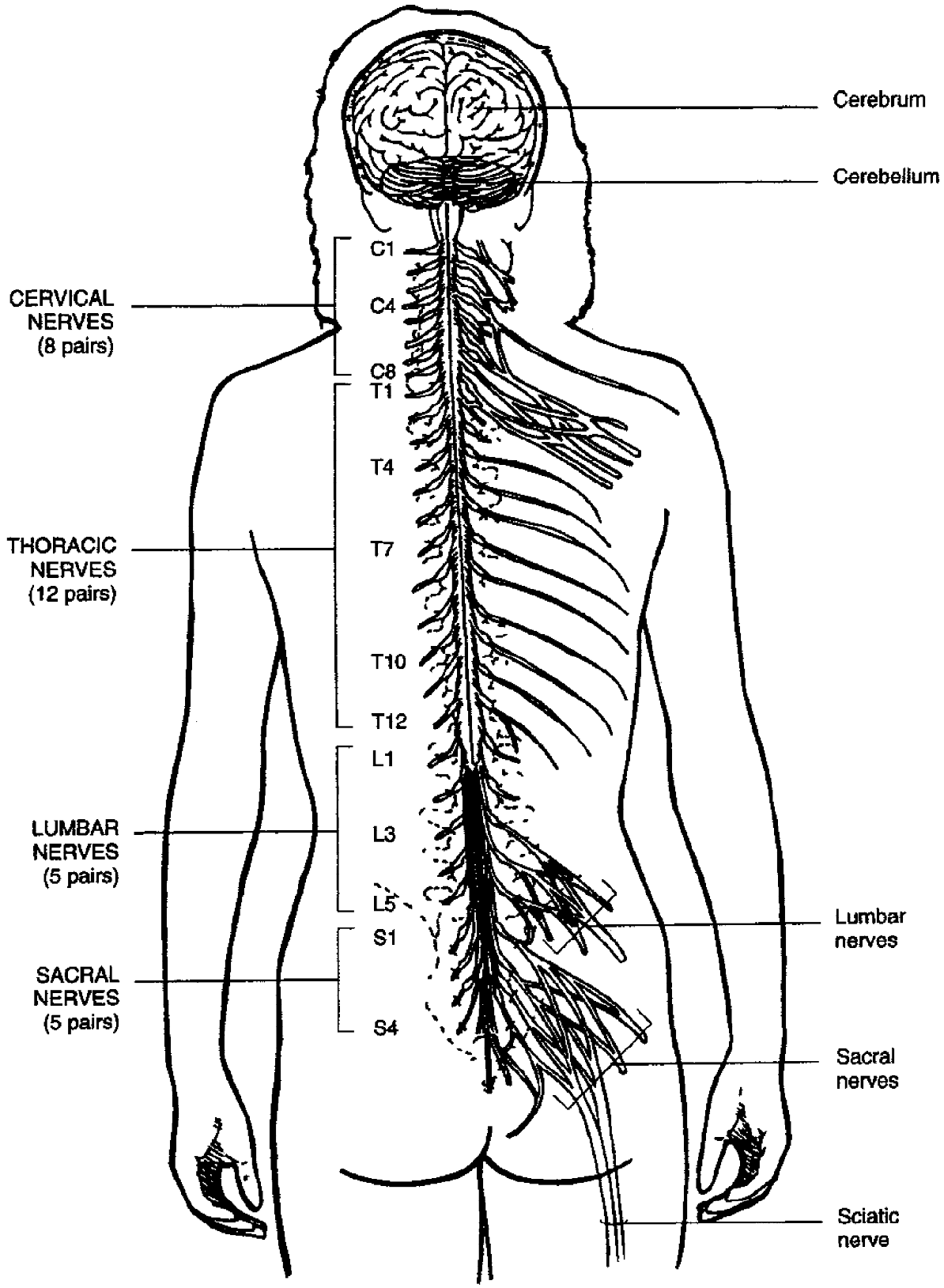


Figure 1.10: Peripheral nerves connecting the spinal cord (according to [4])

Connected via the brain stem the spinal cord distributes nerve fibers on the back of the body to all regions (see fig. 1.10). In the vertebras (spine discs) the central nervous system splits into to peripheral nervous system. Nerve fibers for incoming and outgoing signals leave vertebras on different locations.

The spinal chord is part of the central nervous system. This is because it can process nervous information from sensory organs without interaction of the brain. Figure 1.11 illustrates this with a sensor information from the foot.

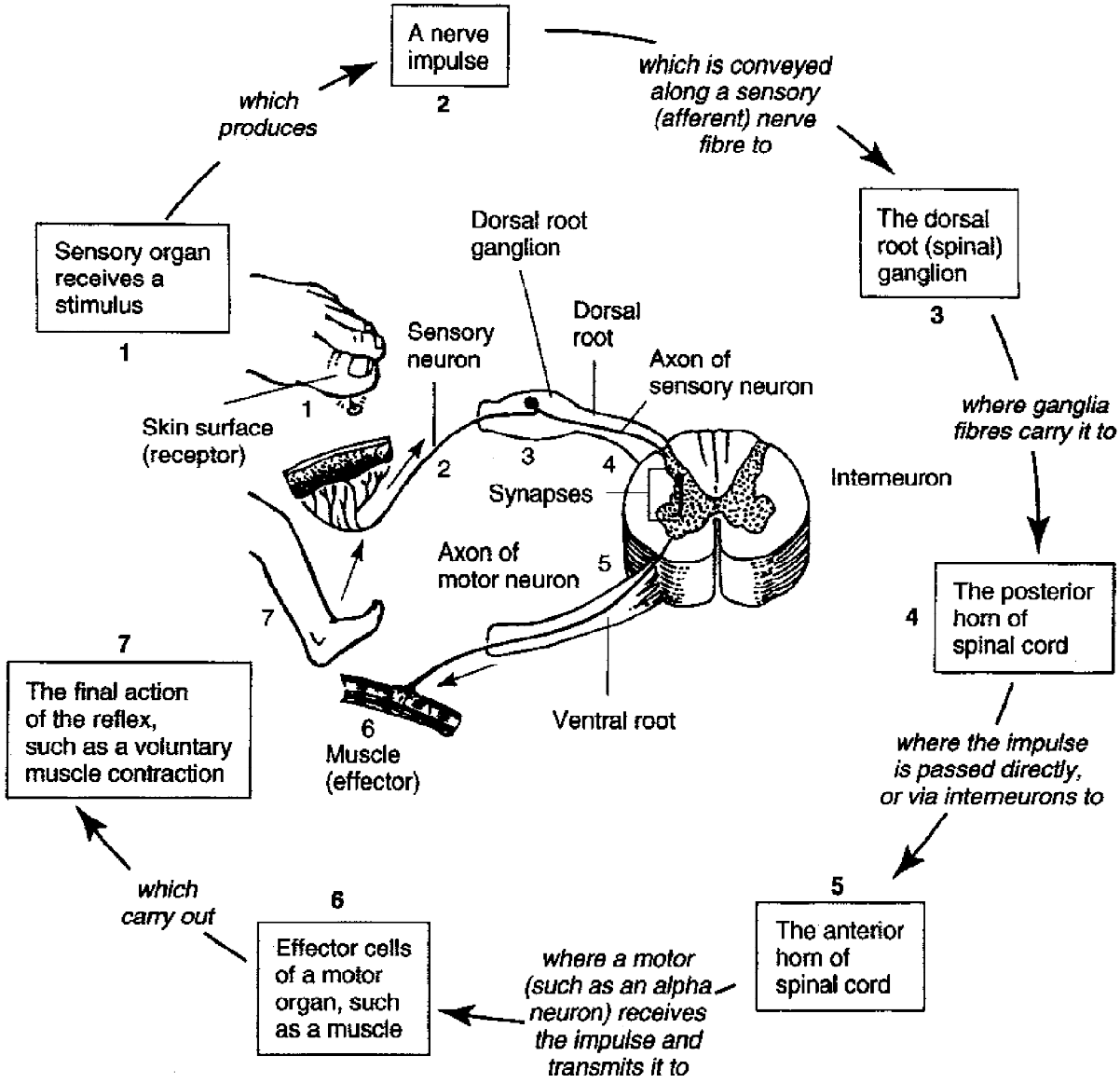


Figure 1.11: Closed loop control via nerves (according to [4] and Carola et. al, 1990)

1.6.2 Neurons

Neuron are nerve cells. These cells provide very simple and elementary signal processing and transmission. The performance from the the nervous system is the result of massive parallel processing of over one hundred billion neurons (> 100,000,000,000 cells).

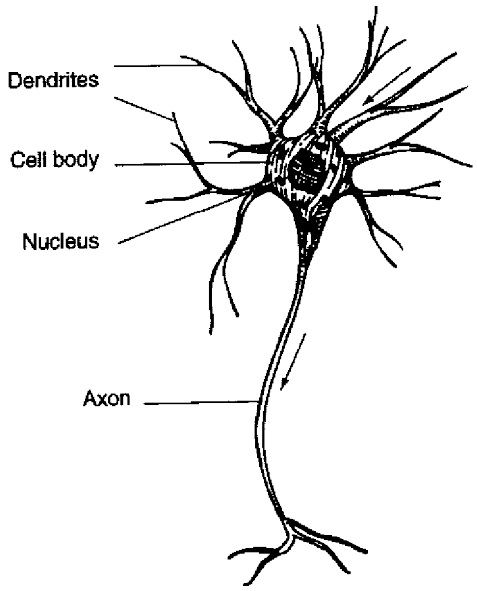


Figure 1.12: Neuron structure (according to [4])

Neurons can simply conduct signals or can respond to stimuli (excitation). The "inputs" of a neuron are the dendrites. The connection to other neurons occurs through *synapses* at the end of the dendrites. The total number of synapses ranges up to 10,000. The "output" is the *axon* with synapses acting a connector to adjacent cells.

1.6.3 Signal Transmission and Generation

Signal transmission occurs by electrical current impulses. A current flow requires different potentials between areas. Most nerves have an insulation "shield" called myolin. The appropriate fibres are thus called *myolinated* and *unmyolinated* fibres. Unmyolinated fibres are found in the peripheral nervous system. The transmission speed is much higher for myolinated nerves (120 m/s). Unmyolinated nerves can transmit at 0.7-2.3 m/s.

Nerves transmit information in a binary manner, i.e. there are no strong or weak signals. A nerve transmits a signal if it has been activated above a certain level. A stronger activation does not initiate a different signal. Nerves transmit therefore digital information. This matches exactly the way muscles operate.

In an inactive state the inside is negatively charged by a so-called *sodium-potassium-pump*. Na^+ ions (sodium) are pumped out of the cell and K^+ ions (potassium) enter the cell by

diffusion. This is possible because K^+ ions are smaller than the Na^+ ions. This mechanism requires energy and is fueled by ATP. For every two K^+ ions entering the cell three Na^+ ions are moved to the outside. This mechanism caused the *resting state*. Since there are more positive ions on the outside than inside the cell becomes negatively charged. The potential difference becomes $-70mV...-90mV$. With this potential difference the cell is ready to react to an electrical impulse.

Depolarization / Repolarization: By an electrical impulse the cell membrane becomes permeable for Na^+ . This process destabilizes itself by the fact that the potential in the cell increases above 0V. K^+ ions rush out since the cell membrane is also permeable for these ions but this happens 0.5ms later. Figure 1.13 show the permeability for Na^+ and K^+ ions over time.

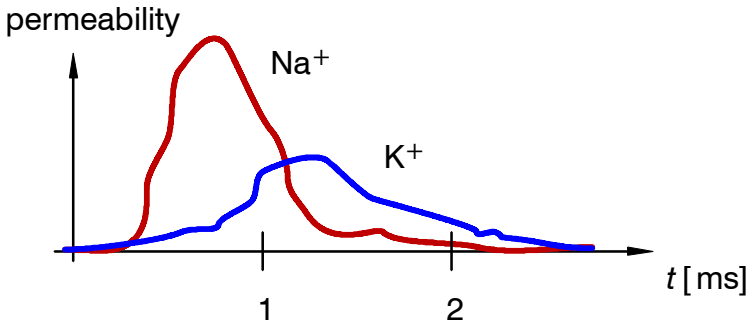


Figure 1.14: Permeability of cell membrane after activation

This process of positive charge is called *depolarization*. The return to the initial voltage level is called repolarization.

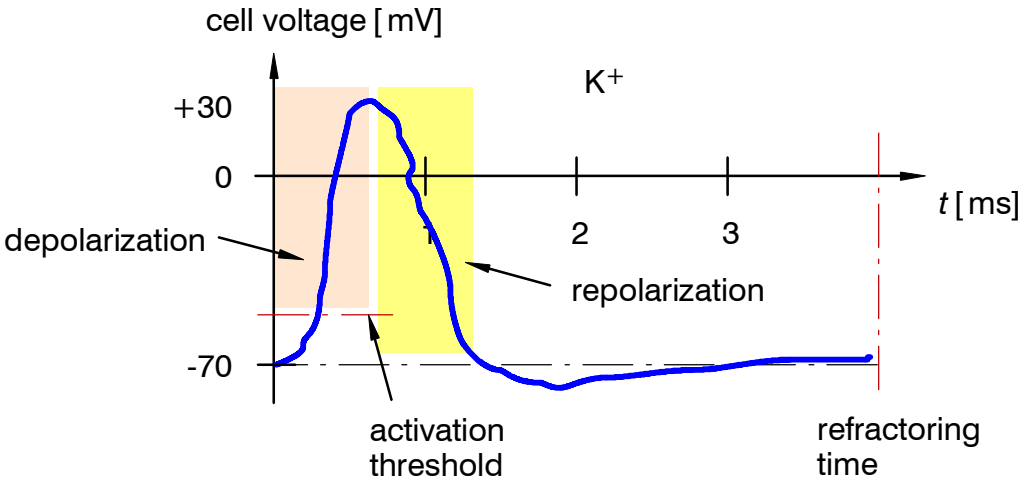


Figure 1.15: Internal cell voltage after activation

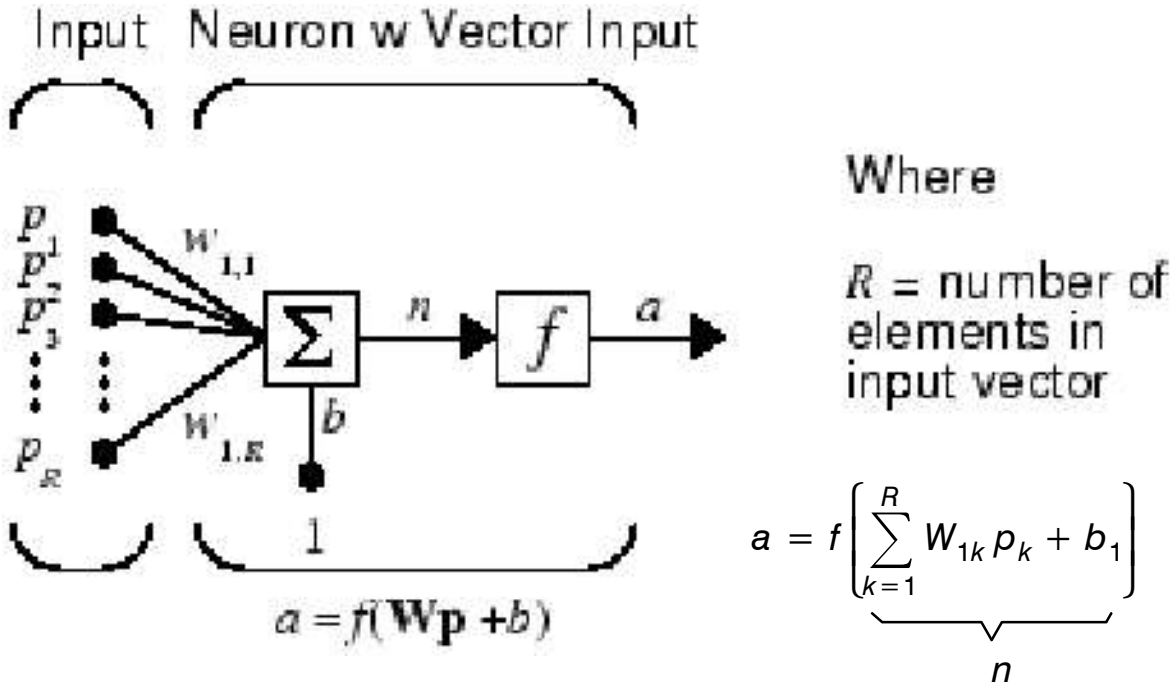
During the refactoring time the sodium-potassium-pump recovers the cell to it's resting state. During this period of time the cell cannot be activated again. This prevents safely the transmission in the wrong (backward) direction of the activating impulse. The adjacent sell activates and transmits the signal to the target.

1.6.4 Information Processing with Neurons

The incoming information on the dendrites of a neuron cell is weighted individually. The sum of these products determine if the output (the axon) is activated or not. This simple mechanism builds the basis for the human brain activity. The performance of the brain is the result of billions of parallel working neurons.

the following sections is about the mathematical model of information processing of a group of neurons. The realization of this model on a computer is called ANN (**A**rtificial **N**eural **N**etwork). This has been successfully used in some application to track problems that cannot be solved by other algorithms. However, the ANN often fails on many application mainly for three reasons:

- Complex tasks require thousands of neurons. No computer has the resources to perform this computations is real-time. Special neuron processor had no success so far.
- The programming of the neural net requires “learning”. No satisfying algorithm has been found up to now which carries out the learning process in an acceptable period of time and without too many errors. The programming is not unique, i.e. completely different programmings result in comparable performance.
- Like the human brain the neural network can produce erroneous outputs. This happens especially in situation where the net has been been trained on. This is probably the weakest spot on ANNs.



$$a = f\left(\underbrace{\sum_{k=1}^{R} W_{1k}\, p_k + b_1}_{n}\right)$$

Figure 1.16: Mathematical description of a neuron’s programming

Depending on the inputs p_i and the weights w_i and an optional bias value b the internal value n determines the output a. The value

$$n = \sum_{i=0}^{R} w_i p_i + b \tag{1.1}$$

is calculated according to fig. 1.16. Positive weights are called activation weights, negative weights are inhibitive weights. The *activation function* $f(n)$ determines the output. A real neuron's output will be activated or not (digital value). In an ANN the output often is continuous which reduces the number of required neurons (on the computer integer or real values are used for this purpose). In biological neurons the output is digital which can be described by the step function.

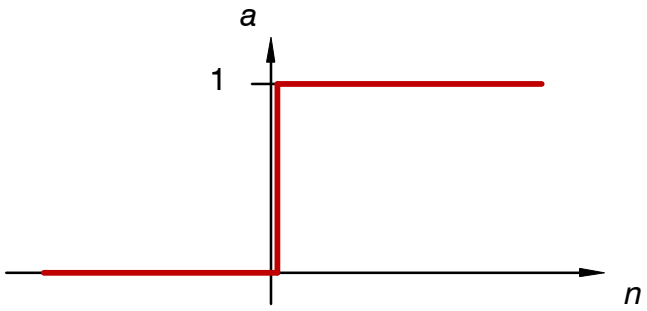


Figure 1.17: Step activation function $a = f(x)$

While the step function is used in ANNs it turns out that other functions give better (smoother) results. The simplest alternative is the linear function.

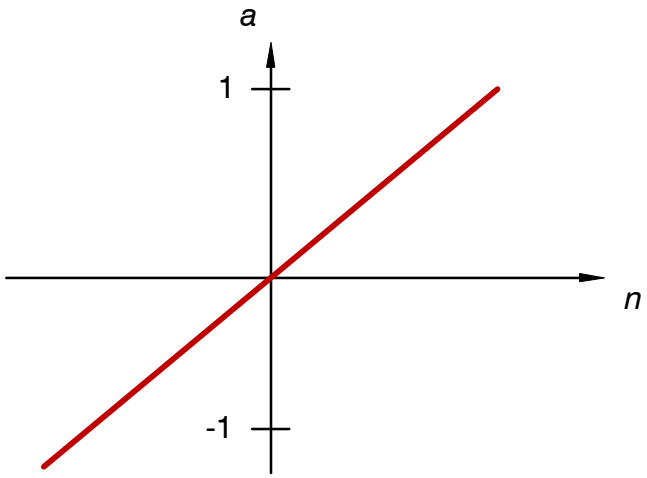


Figure 1.18: Linear activation function $a = f(x)$

The most widely used function is the sigmoid function

$$a = f(n) = \frac{1}{1 + e^{-kn}} . \tag{1.2}$$

The sigmoid resembles the step function but it is differentiable and crosses the a-axis always at 0.5.

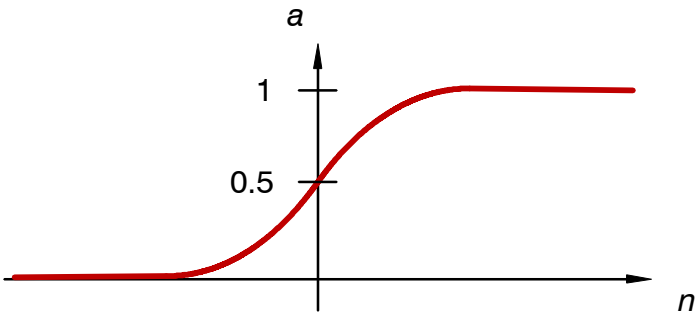


Figure 1.19: Sigmoid type activation function $a = f(x)$

For artificial neural networks the total number of neurons is restricted to several hundreds. Of course the results are way behind the human brain with billions of neuron cells.

Artificial Neural Network (ANN): see extra document

2 Cardio-Vascular System and Heart Monitoring

The cardio-vascular system (CVS) comprises the heart, and the blood vessels (arterial and venous system). The heart builds the center of the cardio-vascular system. It can be regarded as a four chamber pump which with a rate of approximately with 70 beats/minute. The CVS is responsible for the supply of all parts of the body with blood especially with oxygen.

The heart muscle is called cardiac and it is different from other muscle tissue in that it has a build-in pacemaker. This make the heart independent from the central nervous system, although the central nervous system may change the heart beat rate if required. The heart is not the only motor for blood transmission, the arterial systems acts as a “slave” pump for the heart.

The heart function can be explained in conjunction with the main circulation systems (head/arms, legs and left and right lungs, see fig. 1.20). We start with the de-oxygenated blood from the to main veins from superior and inferior regions. The heart operates by contracting chambers with one-valves which prevent blood from flowing backwards.

The blood from veins enters the right atrium. The right atrium contracts and the blood enters the right ventricle. The right ventricle (in the figure on left!) contracts, sending the blood to the pulmonary system (lungs). In the lungs the blood exchange gas with the air in the lungs and oxygenated blood returns to the left atrium. From this atrium the blood enters the left ventricle. If the left ventricle contracts the fresh blood is sent to all areas of the body through the arterial vessels. This completes the *cardiac cycle*. Since the mechanism of nervous stimulation has the same electrical properties as other nerves, it can be monitored well with electrodes attached to certain positions on the skin.

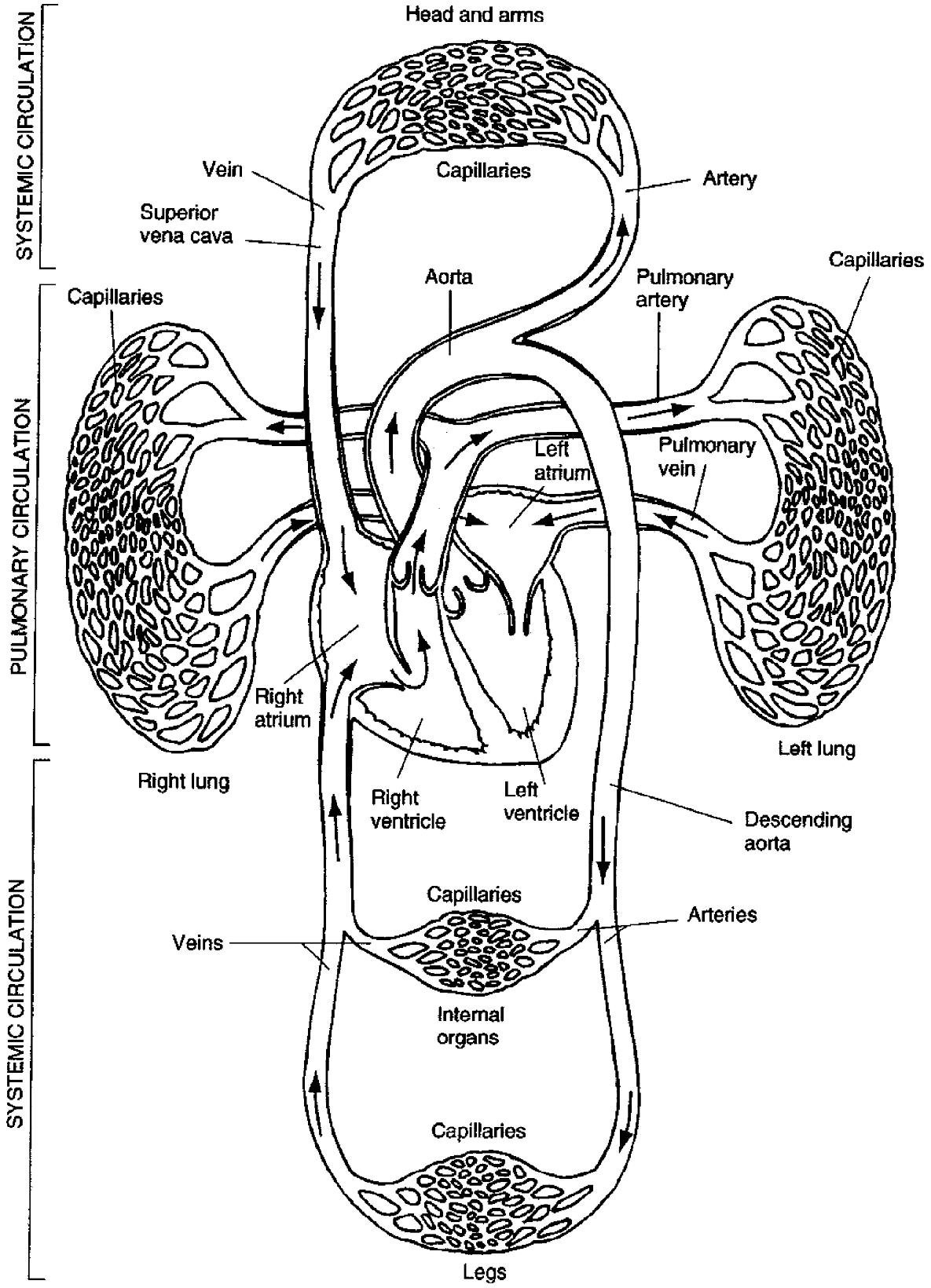


Figure 1.20: Heart and circulation systems (according to [4] and Carola et. al, 1990)

The timing of the autonomous heart nervous system is shown in figure 1.21. It starts with the right atrium contraction after a few milliseconds. The atrial contraction comes first followed by ventricular contractions.

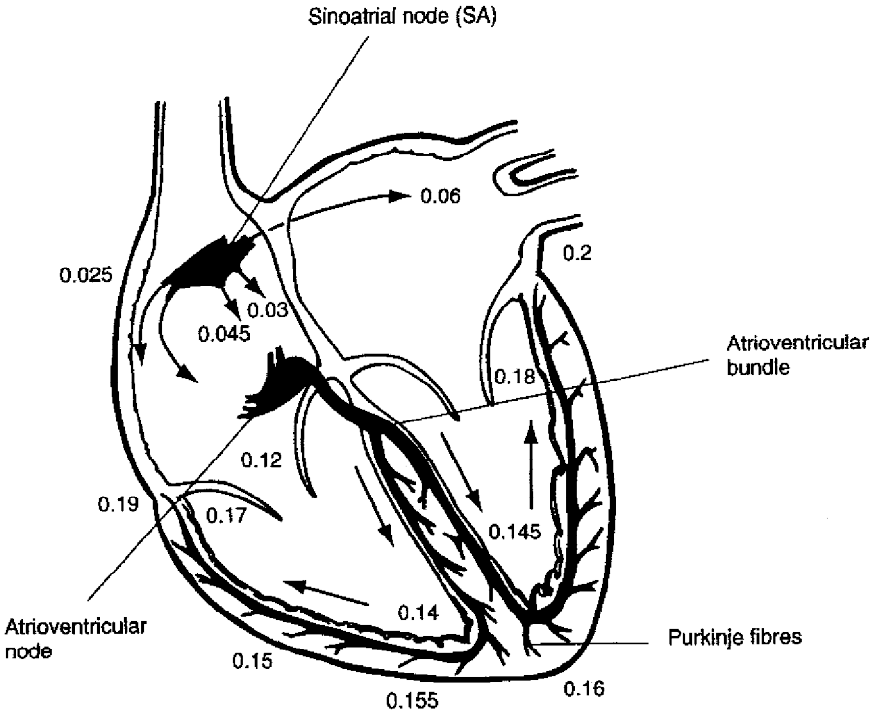


Figure 1.21: Autonomous nervous heart timing under no-load condition (according to [4] and Carola et. al, 1990)

2.1 Electro-Cardiograph (ECG)

The impulses of the heart nervous system on the pathways result in measurable voltages on the skin. However, signal condition and amplification is necessary to measure these small voltages in the range of approx. 1 mV. Additional conditions should be satisfied for a reliable measurement:

- warm environment, the patient mus not shiver (additional nervous signals),
- the patient must not move (except for an ECG under load),
- the electrodes need to positioned on well defined positions and the electrodes should have a silver chloride jelly attached to reach the *dermis* portion of the skin (the *epidermis* is isolating).

The ECG measurements depend on the placing of electrodes. Several connection schemes exist. Using multiple electrodes (standard today) the measurements can indicate a possible area for a malfunction of the cardiac system. An example is shown in fig. 1.22.

Figure 1.22: ECG example with multiple electrodes (PTB Berlin, 2004)

The basic ECG can be measured with electrodes on the right arm (reference or –) and the left leg (Anode +) according to fig. 1.23.

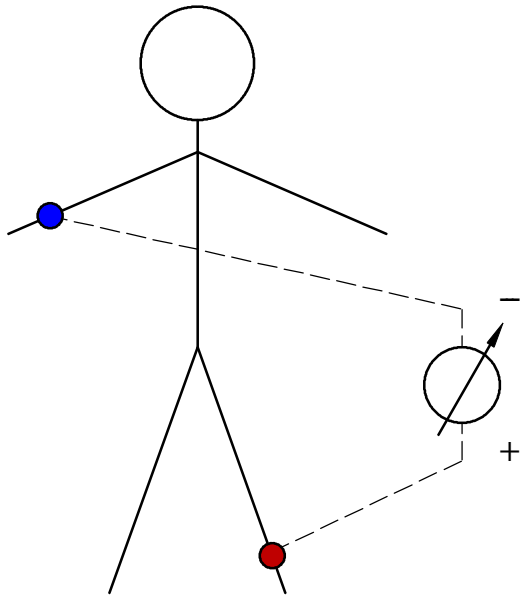


Figure 1.23: Position of electrodes for simple ECG measurement

The simplified diagram of voltage over time is shown in the next figure.

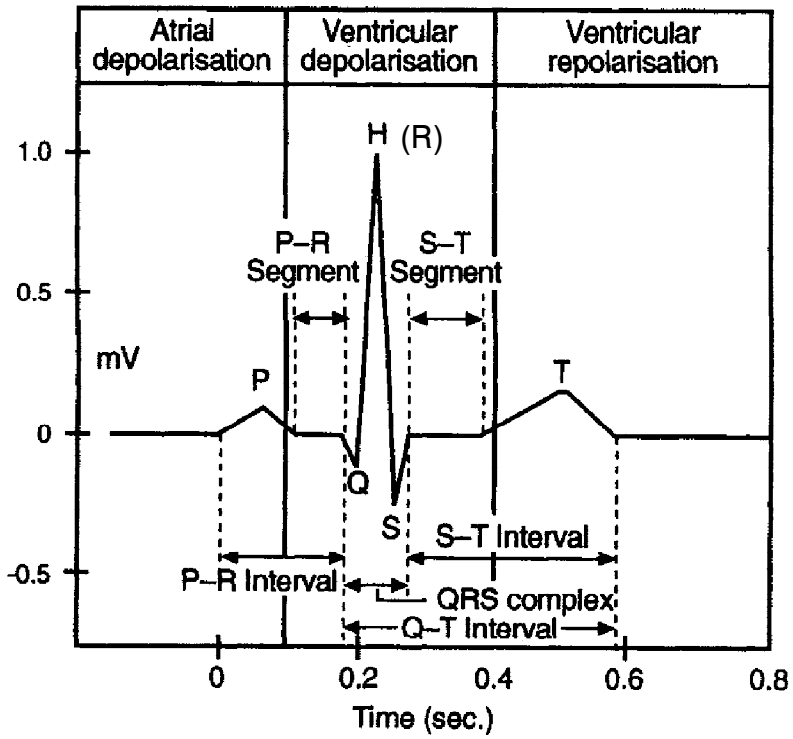


Figure 1.24: Simplified ECG structure (according to [4])

The segments and intervals and their typical times are shown below. For real measurements see fig. 1.22.

ECG Event	Range of duration (seconds)
P wave	0.06 – 0.11
P–R segment (wave)	0.06 – 0.10
P--R interval (onset of P wave to onset of QRS complex)	0.12 – 0.21
QRS complex (wave and interval)	0.03 – 0.10
S–T segment (wave) (end of QRS complex to onset of T wave)	0.10 –.0.15
T wave	Varies
S–T interval (end of QRS complex to end of T wave)	0.23 – 0.39
Q–T interval (onset of QRS complex to end of T wave)	0.26 – 0.49

Figure 1.25: ECG wave and segment timing under normal conditions (according to [4])

The standard for ECG acquisition are 12-lead connections. The most popular are the extremity leads according to Einthoven and Goldberger

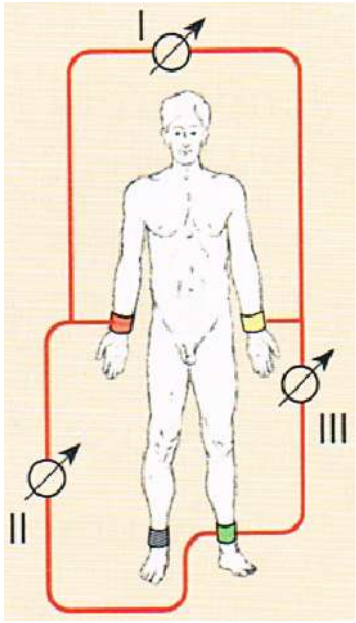


Figure 1.26: Extremity leads according to Einthoven (from [3])

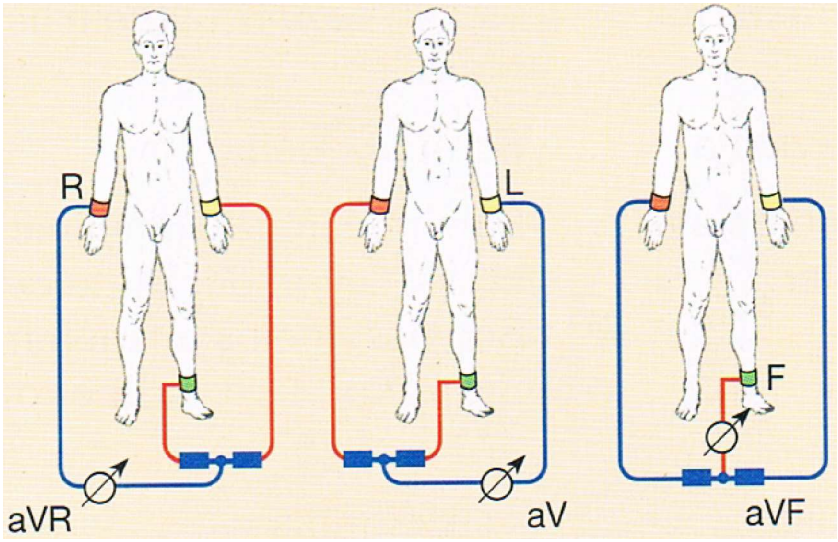


Figure 1.27: Extremity leads according to Goldberger (from [3])

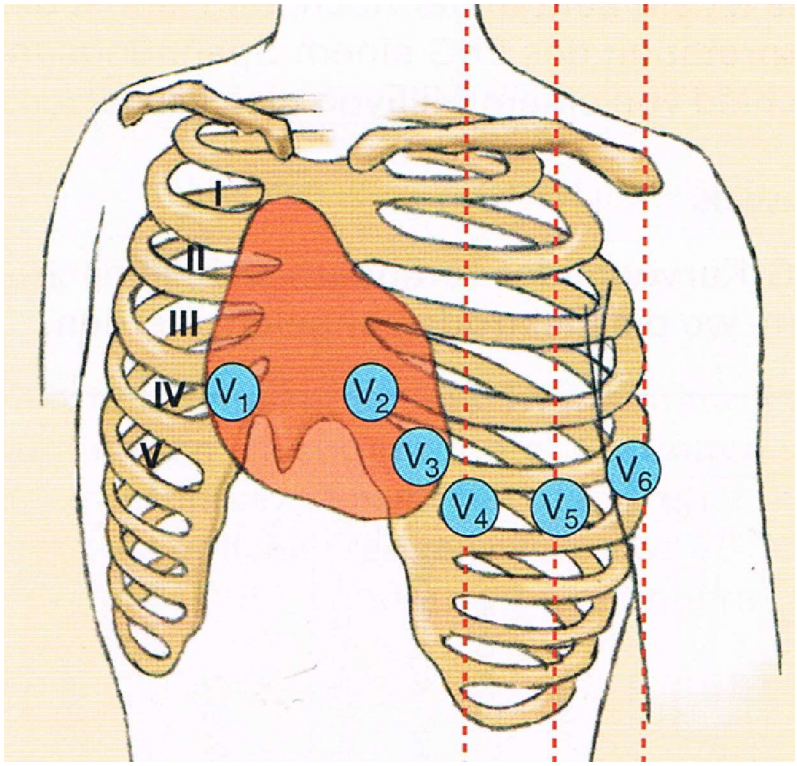


Figure 1.28: Wilson leads (V1...V6) (from [3])

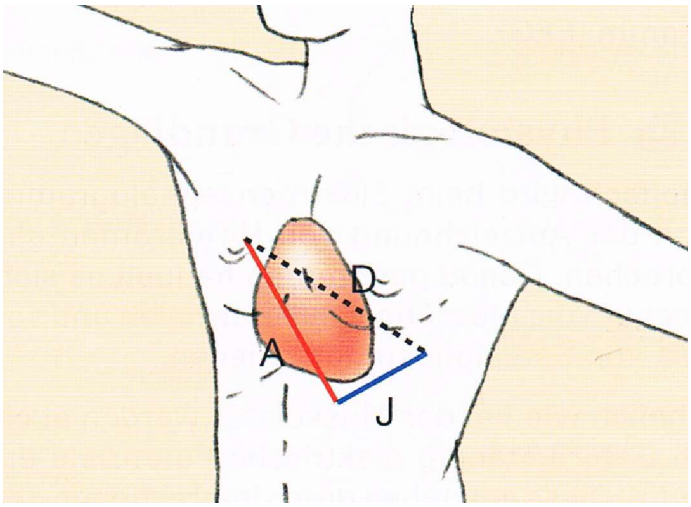


Figure 1.29: Nehb leads (ND, NA, and NI) (from [3])

In any practical ECG the Wilson leads are always included.

Each ECG starts with a 1mV square signal as reference.

***

LAB #01

3 Analyzing ECG Data

In this lab ECG raw data from ADCs (15 leads) should be analyzed using basic digital signal processing (DSP). The data is available from the 3 TB (Tera Byte) PhysioNet data base for complex physiologic signals. The data under investigation have been recorded by the Physikalisch-Technische Bundesanstalt (PTB) in Berlin/Brandenburg, Germany.

The raw (integer) data was sampled with 1 kHz and 16 bits resolution. This is considered to be sufficient for analyzing ECG data. Newer ECG devices sample up to 24 bits on 8 parallel channels. These channels, of course, can be extended if required. 24 bits offer a much better resolution for small signals.

The data is available in Matlab format with ASCII description files (complete header file ".hea" and compact info file ".info"). The data is kept in a large matrix variable "val".

▶ Locate the "data" directory and load the "s0020brem" data set using "load_datm.m". The header file ".hea" show the reason for recording of the ECG. The patient suffered from AMI (acute myocardial infarction) where blood supply of the heart was interrupted, which causes heart cells to die. Thus the ECG differs somewhat from the normal ECG.

```
Header file information:
# age: 69
# sex: male
# ECG date: 23/10/1990
# Diagnose:
# Reason for admission: Myocardial infarction
# Acute infarction (localization): antero-septal
# Former infarction (localization): no
# Additional diagnoses: Arterial hypertension, Obesity
# Smoker: no
# Number of coronary vessels involved: 1
# Infarction date (acute): 19-Oct-90
# Previous infarction (1) date: n/a
# Previous infarction (2) date: n/a
# Hemodynamics:
# Catheterization date: 29-Oct-90
# Ventriculography: Akinesia antero-lateral wall and apex
# Chest X-ray: normal
# Peripheral blood Pressure (syst/diast):  140/80 mmHg
```

```
# Pulmonary artery pressure (at rest) (syst/diast):   21/8 cmH2O
# Pulmonary artery pressure (at rest) (mean): 13 cmH2O
# Pulmonary capillary wedge pressure (at rest): 4 cmH2O
# Cardiac output (at rest): 4,57 l/min
# Cardiac index (at rest): 2,1 l/min/sqrmBSA
# Stroke volume index (at rest):  25.0 ml/beat
```

- Inspect the individual traces (leads) as integer data (variable "val") and the scaled data (variable "val_f").
- Investigate the same curves with a 1 minute data set "s0020brem60". Note that the loading procedure works different. Obviously digital signal processing can improve the data sets.

3.1 Frequency Domain Analysis

Some properties of signals can be shown in frequency domain. If the data is a discrete time series the discrete Fourier transform (DFT) allows frequency domain analysis. The DFT assumes that the whole time series is periodic. This means that if the time series $u[k]$ consists of N values from index $k = 0 .. N\text{-}1$ then

$$u[k + N] = u[k] . \tag{1.3}$$

Special windowing techniques are required if this assumption is not satisfied. This is an issue for time series where begin and end do not "match".

Since the data is discrete the frequencies from the DFT are also discrete. The frequencies range from 0 (DC) to

$$f_{N/2} = \frac{1}{2T_S} , \tag{1.4}$$

where T_S is the sampling period and $f_{N/2}$ is the Nyquist frequency.

In general the frequencies are

$$f_k = \frac{n}{NT_S} , \quad n = 0...\frac{N}{2} . \tag{1.5}$$

The following figures illustrates this for $N = 8$.

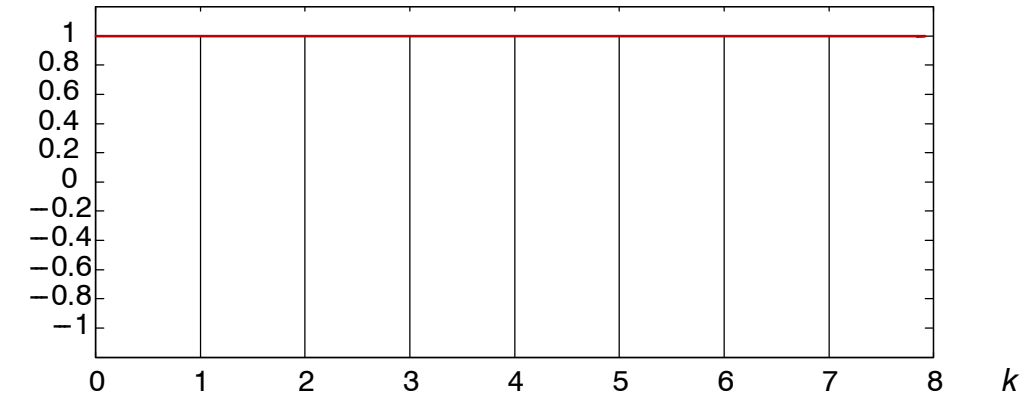


Figure 1.30: $N = 8,\ n = 0$

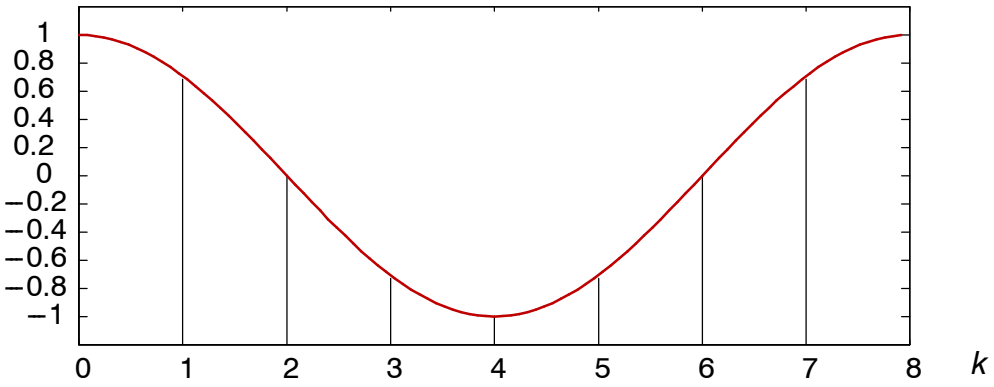


Figure 1.31: $N = 8,\ k = 1$

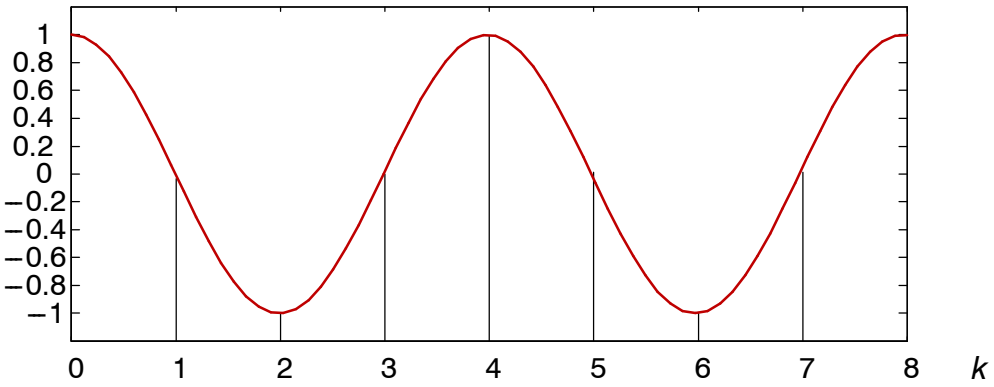


Figure 1.32: $N = 8,\ n = 2$

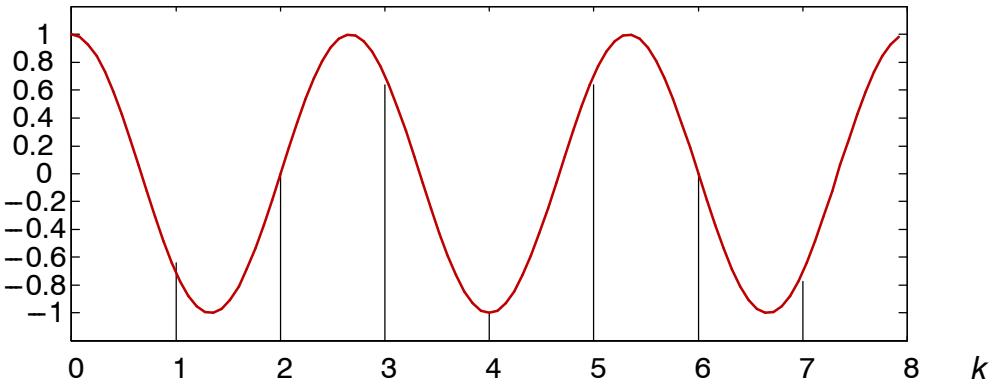


Figure 1.33: $N = 8,\ n = 3$

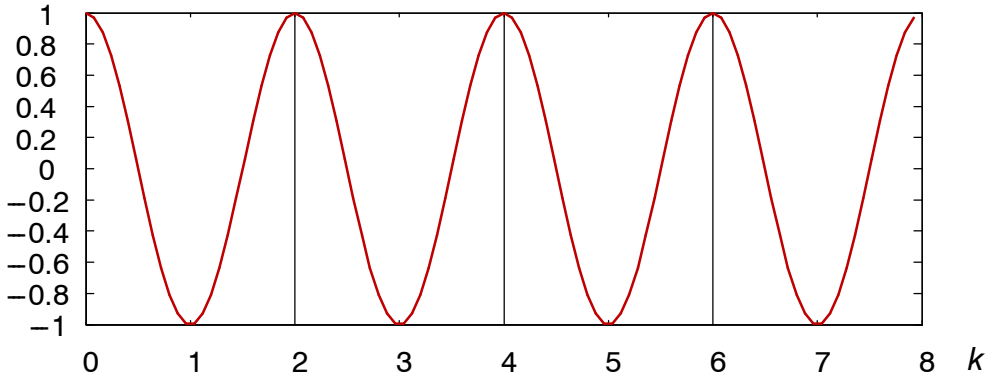


Figure 1.34: $N = 8,\ n = 4$

It is obvious that higher order harmonics cannot exist.

The DFT is calculated by *correlating* the time series by the appropriate complex frequency $e^{-j\omega}$. The continuous Fourier-transform

$$U(j\omega) = \int_{-\infty}^{\infty} u(t)\, e^{-j\omega t}\, dt := \mathcal{F}\{u(t)\}\,. \tag{1.6}$$

becomes the discrete sum

$$U(n) = \sum_{k=0}^{N-1} u[k]\, e^{-j\frac{k2\pi n}{N}}, \quad n = 0, 1, 2, ..., N-1\,. \tag{1.7}$$

Since the DFT is an equivalent signal description as the time domain the inverse discrete Fourier transform exists (not often required)

$$u[k] = \frac{1}{N}\sum_{n=0}^{N-1} X(n)\, e^{+j\frac{k2\pi n}{N}}, \quad k = 0, 1, 2, ..., N-1\,, \tag{1.8}$$

3.2 Frequency Analysis of ECG Data

- Write a function "my_dft" which implements the algorithm (1.7).
  Test this function with simple data vector and compare the result with Matlab's "fft".
- Select V6 lead (signal 11) from the data set. Analyze the signal with "my_fft". Make the x-axis equal to frequency in Hz according to (1.5). Plot absolute values (not real and imaginary parts).
- Discuss the frequency components in this signal.

3.3 DFT Details and Properties

The practical calculation of (1.7) is carried out in the *rectangular* form by using the equivalence for the complex exponential function

$$e^{jx} = \cos x + j\sin x\,. \tag{1.9}$$

The time-domain function is real since an imaginary port of functions of time have no physical meaning. The DFT is complex to express frequencies and phase dependencies. The inverse DFT should give a real function again (the imaginary parts should be zero with respect to rounding errors).

We will create an artificial example which will be used throughout the section. If a radix-2 number, i.e. a power of 2, is used the FFT (**F**ast **F**ourier **T**ransform) can be applied to this example as well

$$x[k] = x[kT_S] = 0.6\cos(2\pi f_1 kT_S) + \sin\left(2\pi f_2 kT_S - \frac{\pi}{6}\right). \tag{1.10}$$

The umber of samples is $N = 16$ with a sampling frequency of $f_S = 1\text{kHz}$ ($T_S = 1\text{ms}$). The discrete frequencies of the DFT are according to (1.5)

n	f_n [Hz]
0	0.0
1	62.5
2	125.0
3	187.5
4	250.0
5	312.5
6	375.0
7	437.5
8	500.0

In order to get results which can be easily verified we choose $f_1 = 125\text{Hz}$ and $f_2 = 270\text{Hz}$. So $x[k]$ becomes:

k	*x*[k]
0	0.100000
1	1.346127
2	0.268920
3	-1.413536
4	-0.620942
5	0.570258
6	-0.228351
7	-0.513018
8	1.063296
9	1.245413
10	-0.669131
11	-1.077685
12	0.232921
13	0.020371
14	-0.944376
15	0.216352

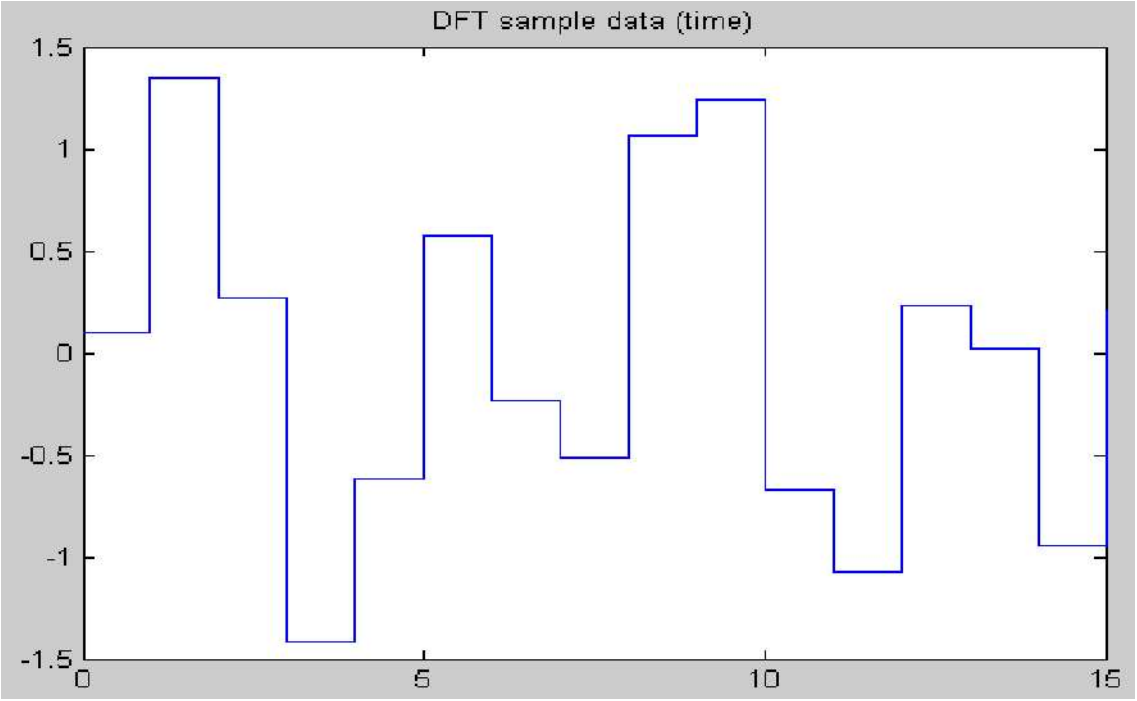


Figure 1.35: Sampled data in time domain

From the diagram in fig. 1.35 the frequencies contained in the signal cannot be seen easily. The DFT shows the frequency components. Since the frequency domain signal becomes complex, three diagrams can be drawn (real, imaginary, magnitude).

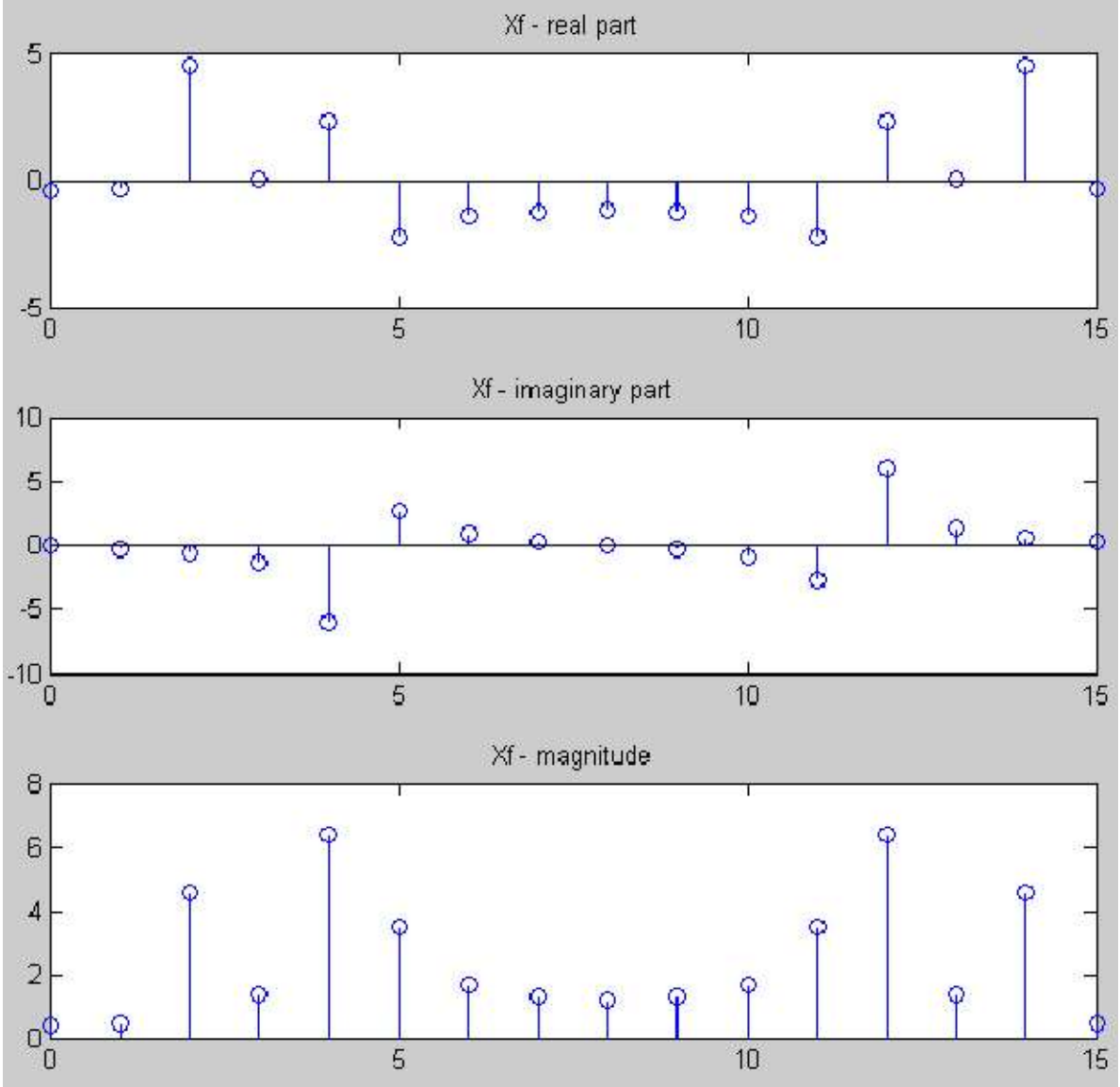


Figure 1.36: Discrete Fourier Transform (DFT) of $x[k]$

The real parts correspond to cosine harmonics while imaginary parts belong to sine oscillations. Any frequency which is not cosine nor sine will result in real *and* imaginary parts.

A fourth diagram shown the phase of the discrete frequencies

$$\phi(n) = \operatorname{atan}\left[\frac{\Im\{X(n)\}}{\Re\{X(n)\}}\right]. \tag{1.11}$$

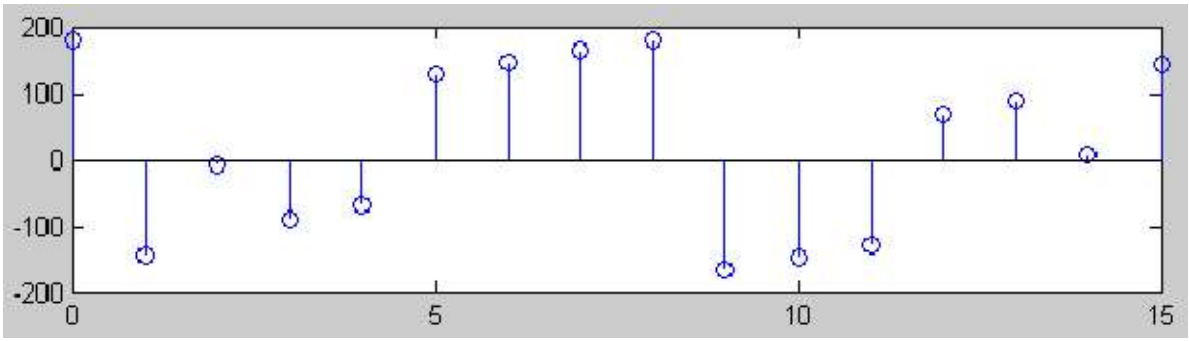


Figure 1.37: Phase of frequencies

The DFT is symmetric due to the real time signal

$$X(n) = X^{*}(N - n), \tag{1.12}$$

where * denotes a conjugate complex value. The following equations hold for instance

$$\Re\{X(5)\} = \Re\{X(11)\}, \quad \Im\{X(5)\} = -\Im\{X(11)\}. \tag{1.13}$$

In order to analyze the frequencies that a time signal contains a frequency axis can be used as in fig. 1.38. The amplitude has been normalized to the number of samples to measure amplitudes. The 0.6 amplitude of the 125Hz signal can be seen clearly. The 270Hz signal cannot be analyzed exactly since it is described by several frequencies (mainly 250Hz and 312.5Hz). Better results are obtained when more samples a higher sampling rate are taken.

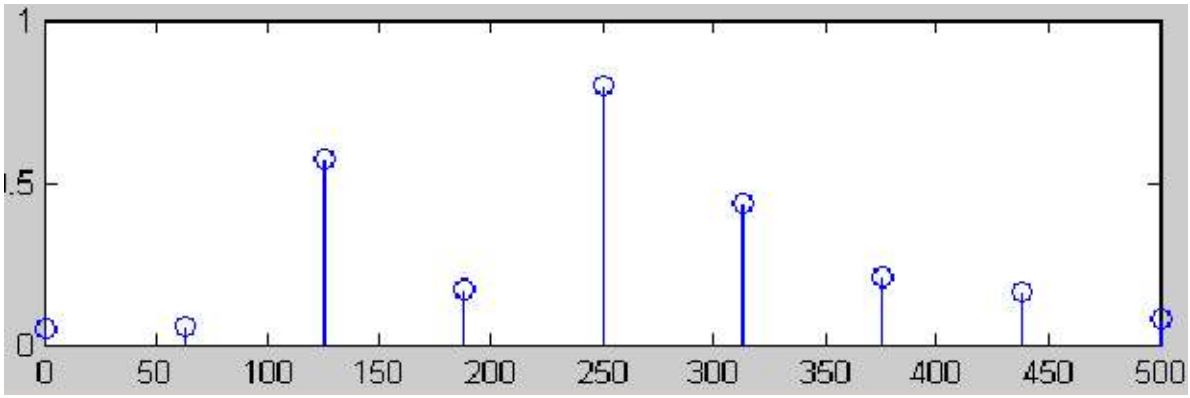


Figure 1.38: Frequency axis for magnitude plot

3.4 The Fast Fourier Transform (FFT)

The DFT requires N^2 complex multiplications. This number makes it impossible for a large numbers of samples to compute the DFT in real time. In 1965 a fast algorithm for DFT computation was developed by Cooley and Tukey. This algorithm is called FFT.

They take advantage of the fact the DFT has symmetric properties and that several operation are very similar to each other. Basically the DFT is reduced to a sequence of 2-samples DFTs. The number of multiplications drops to

$$\frac{N}{2}\log_2 N, \tag{1.14}$$

where $\log_2$ is the dual logarithm. However, the algorithm requires that the number of samples is a power of 2, thus FFT is denoted also as **radix-2 FFT**. An example from Lyons [6] demonstrates that a 2 million samples FFT takes 10 seconds to finish on a desktop PC, while the normal DFT algorithms takes about three weeks! With fewer number of multiplications and additions the numerical properties will improve also.

From now on we assume the N is a power of 2. The DFT algorithm is defined as (now with x as signal)

$$X(n) = \sum_{k=0}^{N-1} x[k] e^{-j\frac{2\pi kn}{N}}, \quad 0 \le n < N. \tag{1.15}$$

There is an equal number of even and odd indexed terms

$$X(n) = \sum_{k=0}^{\frac{N}{2}-1} x[2k] e^{-j\frac{2\pi(2k)n}{N}} + \sum_{k=0}^{\frac{N}{2}-1} x[2k+1] e^{-j\frac{2\pi(2k+1)n}{N}} \tag{1.16}$$

or

$$X(n) = \sum_{k=0}^{\frac{N}{2}-1} x[2k] e^{-j\frac{2\pi(2k)n}{N}} + e^{-j\frac{2\pi n}{N}} \sum_{k=0}^{\frac{N}{2}-1} x[2k+1] e^{-j\frac{2\pi(2k)n}{N}}. \tag{1.17}$$

With the constant factor (complex angle)

$$W_N \triangleq e^{-j\frac{2\pi}{N}} \tag{1.18}$$

the equation (1.17) can be simpler written as

$$X(n) = \sum_{k=0}^{\frac{N}{2}-1} x[2k] W_N^{2kn} + W_N^n \sum_{k=0}^{\frac{N}{2}-1} x[2k+1] W_N^{2kn}. \tag{1.19}$$

From the definition (1.18) we can derive also

$$W_{N/2} \triangleq e^{-j\frac{2\pi}{N/2}}. \tag{1.20}$$

This results in

$$X(n) = \sum_{k=0}^{\frac{N}{2}-1} x[2k] W_{N/2}^{kn} + W_N^n \sum_{k=0}^{\frac{N}{2}-1} x[2k+1] W_{N/2}^{kn}. \tag{1.21}$$

We can restrict n to be $0 \le n \le N/2\text{-}1$. This is because $n+N/2$ can be calculated easily

$$X(n + N/2) = \sum_{k=0}^{\frac{N}{2}-1} x[2k] W_{N/2}^{kn} - W_N^n \sum_{k=0}^{\frac{N}{2}-1} x[2k+1] W_{N/2}^{kn}, \tag{1.22}$$

almost identical to (1.21). The variable n appears as exponents to the W terms. Due to

$$W_{N/2}^{n+N/2} = W_{N/2}^{n} W^{N/2}_{N/2} = W_{N/2}^{n} e^{-j\frac{2\pi N/2}{N/2}} = W_{N/2}^{n} \tag{1.23}$$

and

$$W_{N}^{n+N/2} = W_{N}^{n} W^{N/2}_{N} = W_{N}^{n} e^{-j\frac{2\pi N/2}{N}} = W_{N}^{n} e^{-j\pi} = - W_{N}^{n} \tag{1.24}$$

the equation (1.22) is verified.

The equation (1.22) shows that any N-point DFT can be reduced to two $N/2$ point DFTs plus simple add and subtract operations. Moreover, sine and cosine operations disappeared (the W factors can be computed in advance).

With

$$A(n) = \sum_{k=0}^{\frac{N}{2}-1} x[2k] W_{N/2}^{kn} \tag{1.25}$$

and

$$B(n) = \sum_{k=0}^{\frac{N}{2}-1} x[2k+1] W_{N/2}^{kn} \tag{1.26}$$

the equations (1.21) and (1.22) simply become

$$X(n) = A(n) + W_{N}^{n} B(n)\,, \tag{1.27}$$

$$X(n+N/2) = A(n) - W_{N}^{n} B(n)\,. \tag{1.28}$$

The reduction to two N/2 DFTs for an 8-point DFT is shown in fig. 1.39.

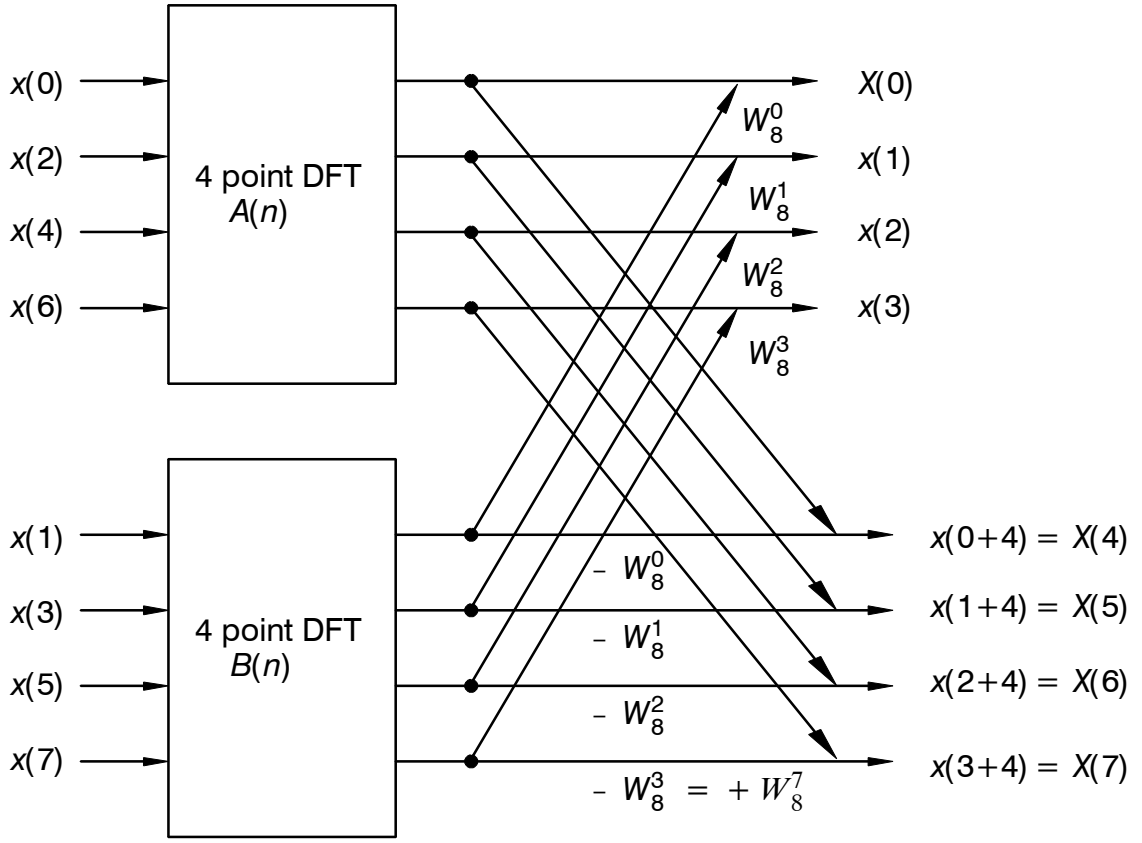


Figure 1.39: An 8 point DFT reduced to two four point DFTs

It is straight-forward to reduce the 4-point DFTs in fig. 1.39 to 2-point DFTs which are trivial. More general:

> Any N-point DFT where N is a power of 2 can be divided into a $\log_2$ stage DFT. The first stage is a trivial 2-point DFT.

Let us look at the first stage with $N = 2$. The sums disappear and we have

$$W_N^0 = W_{N/2}^0 = 1\,. \tag{1.29}$$

This results in

$$X(0) = x(0) + x(1) \qquad [\text{ DC }] \tag{1.30}$$

and

$$X(1) = x(0) - x(1)\,. \qquad [\text{ AC }] \tag{1.31}$$

Due to the graphical shape in figure 1.40 the basic operation is called “butterfly”.

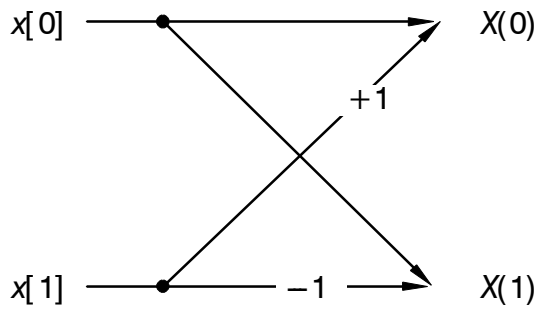


Figure 1.40: Butterfly algorithm for one stage (first stage)

The three stages of an 8-point FFT is shown in figure 1.41.

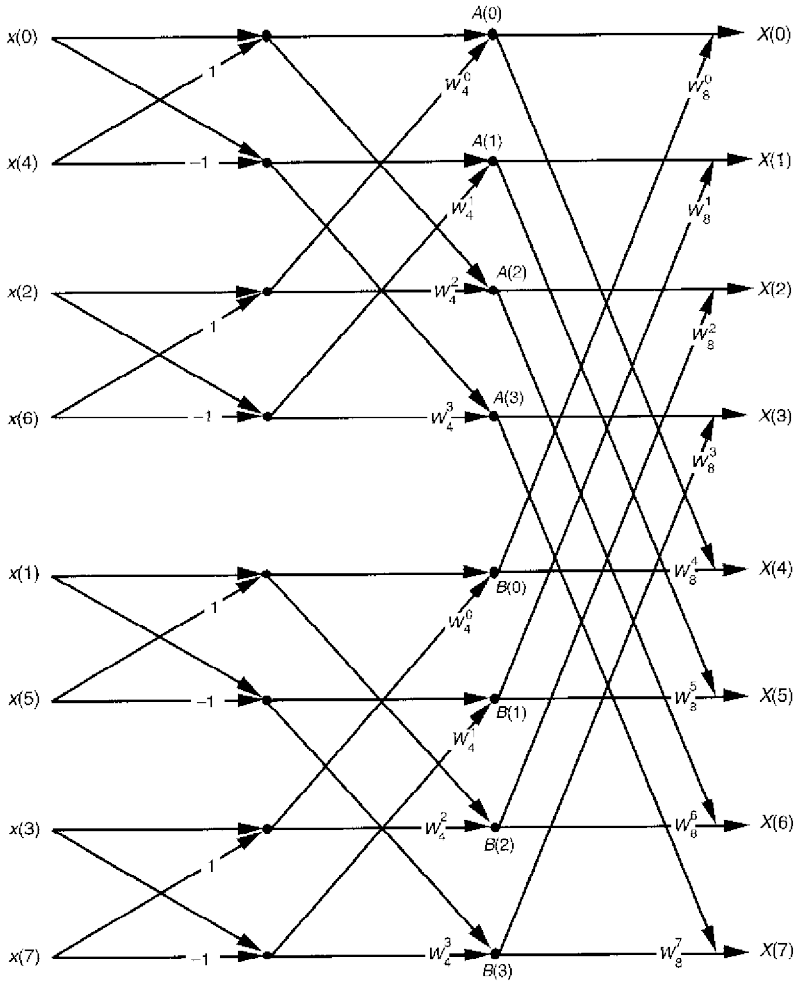


Figure 1.41: 8-point FFT structure (Lyons [6])

For every stage the inputs are divided into odd and even parts. This results in a "scrambled" order of time data for the first stage (see fig. 1.41). However, there is a simple rule to calculate the sequence of the input data index. This is called *bit reversal*. The following table illustrates this for the 8-point FFT.

normal index	binary	reversed bits	bit-reversed index
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

The last column is the input index to the first stage.

The algorithm is denoted as **decimation-in-time**. This name is derived from the fact that we start by dividing the time vector into odd and even parts. A similar algorithm exists to calculate the odd and even parts of the frequency output separately. This is called the **decimation-in-frequency** algorithm. It takes exactly the same number of multiplications and additions/subtractions. So none of theses version has benefits over the other. It depends on the available computation hardware which version is easier to code. In the calculation example we will concentrate on the decimation-in-time version.

3.4.1 Optimizing the Algorithm

The basic operation in FFT is the “butterfly” operation, calculating a pair of data for the next stage.

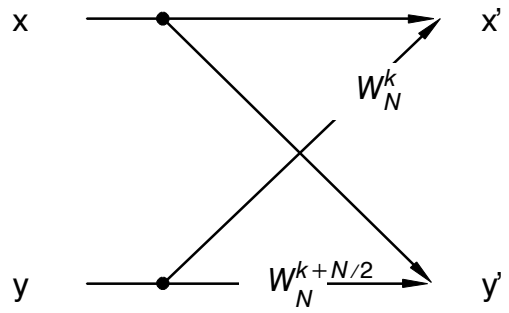


Figure 1.42: Basic butterfly structure for a pair of complex data

Since

$$W_N^{k+N/2} = - W_N^k \tag{1.32}$$

we arrive at fig. 1.43.

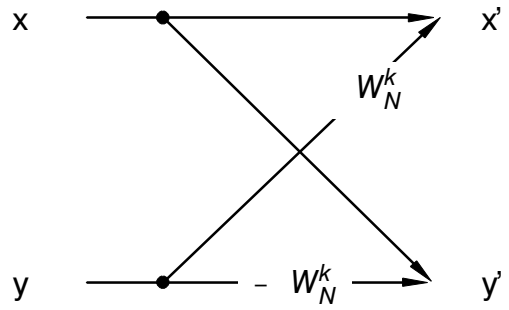


Figure 1.43: Simplified butterfly structure for a pair of complex data

An optimized structure is shown below which avoids one multiplication.

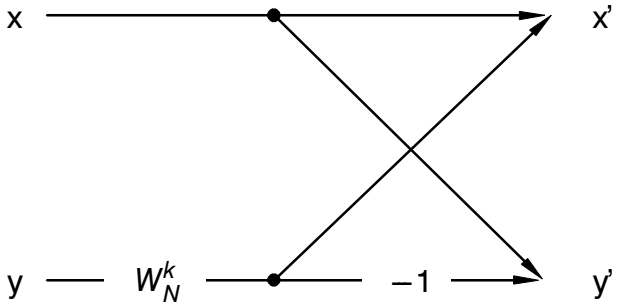


Figure 1.44: Optimized butterfly structure for a pair of complex data

According to fig. 1.44 the equations are

$$x' = x + W_N^k y \tag{1.33}$$

and

$$y' = x - W_N^k y\,. \tag{1.34}$$

Lab #02

4 Diagram Exercise (Pre FFT Lab)

Before coding the FFT algorithm the following lab exercise demonstrates to use diagrams for the graphical output of the results.

- Create a new Netbeans project (Java Application) with the name:
  JChartTst
- Copy the provided source code to you application and run the program.

```
package jchart2dtst;

import info.monitorenter.gui.chart.Chart2D;
import info.monitorenter.gui.chart.ITrace2D;
import info.monitorenter.gui.chart.traces.Trace2DSimple;
import java.awt.Color;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JFrame;

public class JchartTst {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
    int k;
    double fx;

    // Create a chart:
    Chart2D chart = new Chart2D();
    // Create an ITrace:
    ITrace2D trace = new Trace2DSimple();
    // Add the trace to the chart.
    chart.addTrace(trace);
    trace.setColor(Color.blue);
    trace.setName("x[k]");
    // always start in origin
    trace.addPoint(0.0, 0.0);
    // Add all points, as it is static:
    for (k = 0; k < N_POINTS; k++) {
        trace.addPoint(k, k);
```

```
        }
        // Make it visible:
        // Create a frame.
        JFrame frame = new JFrame("Cosine Samples");
        // add the chart to the frame:
        frame.getContentPane().add(chart);
        frame.setSize(400,300);
        // Enable the termination button [cross on the upper right edge]:
        frame.addWindowListener(
                new WindowAdapter() {
                    @Override
                    public void windowClosing(WindowEvent e){
                        System.exit(0);
                    }
                }
                );
        frame.setVisible(true);
        }

        static final int N_POINTS = 16;
}
```

▶ This result should look as follows:

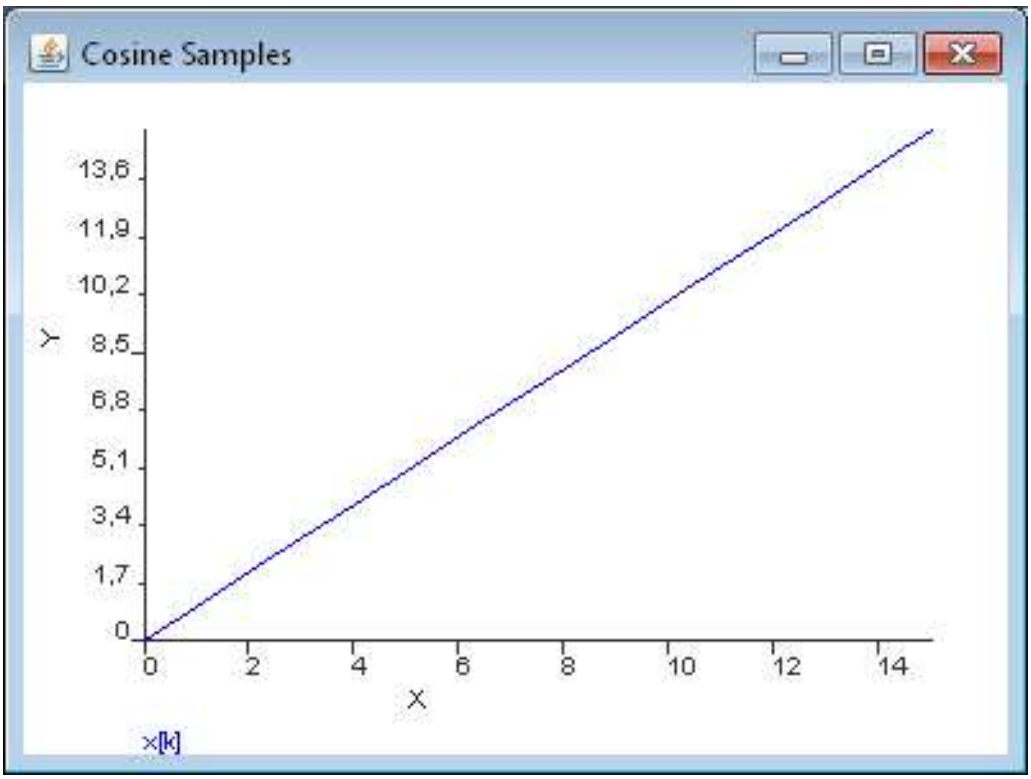


Figure 1.45: JChartTst output

- Modify the source code to display cosine samples of one period. This result should be as follows:

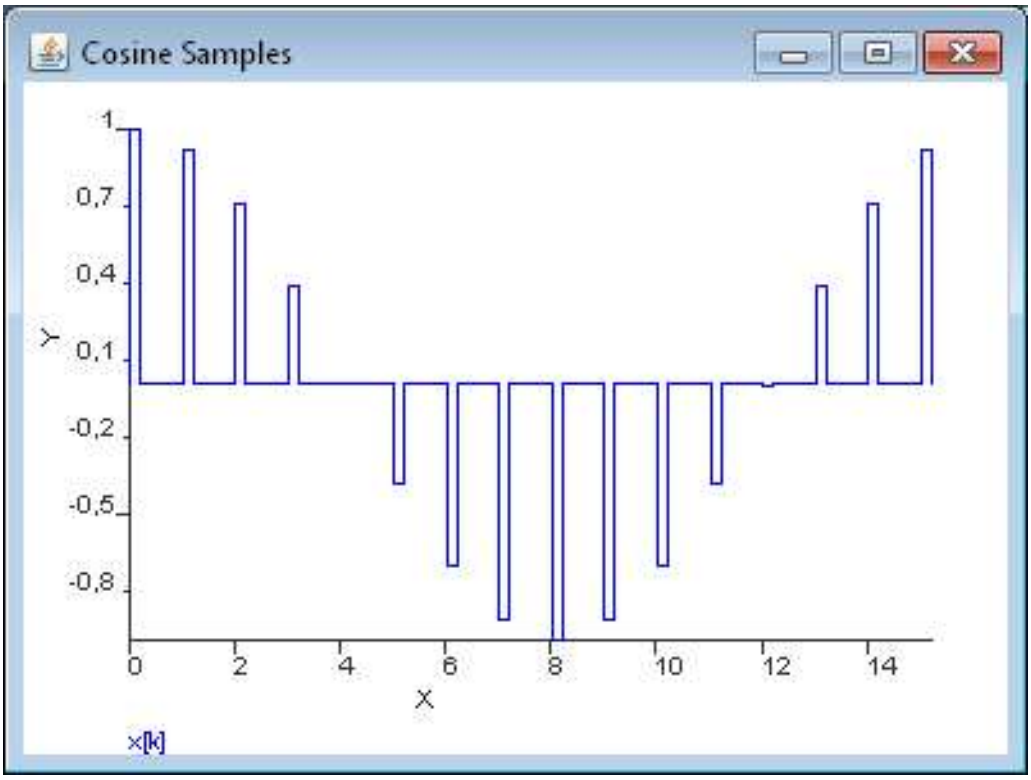


Figure 1.46: Modified JChartTst output with cosine samples

Lab #03

5 FFT (Cooley-Tukey) Coding

The FFT algorithm should be coded in Java. In order to test the computation stages we will use the test data (8-point)

$$x[k] = \sin(2\pi f_1 T_S k) + 0.5\sin\left(2\pi f_2 T_S k + \frac{3\pi}{4}\right), \quad 0 \le k \le 7. \tag{1.35}$$

The frequencies are $f_1 = 1$ kHz, $f_2 = 2$ kHz and the sampling period is $T_S = 1/8$kHz.

The 8-point FFT requires three stages. Your program should give the same results as shown below. The program must be able to handle smaller or greater numbers of data. However, the number is restricted to powers of two.

- Create a new Java Application with the class "FFTcta".
  This class has a main function to make it executable.

- Write functions to generate the test data, FFT initialization, and FFT stage.
  The function could be as follows:

```
// create test data
static void TestData()

display test data on screen
static void ShowTestData()

// bit reverse computation
static int BitRev(int k)

// initialize variables for FFT computation
static boolean FftInit(double[] x)

// copy output to input for next stage
static void CopyStage()

// compute butterflies
static void FftStage(int stage)

// display stage after one stage computation
static void ShowStage(int k)
```

```
// calculate FFT
static void FftComp()

// (optional) plot of result
static void FftPlot()
```

- Call the functions to loop over all stages, provide output for every stage (the output may differ from the output below)
- The inputs must be ordered in the bit-reversal manner to obtain order output for the frequency output.

The last two columns (real and imaginary parts) contain the final results.

```
--- FFT Cooley-Tukey Algorithm ---
  k       data
-----------------
  0       0,3536
  1       0,3536
  2       0,6464
  3       1,0607
  4       0,3536
  5      -1,0607
  6      -1,3536
  7      -0,3536
-----------------
compute WNn factors...
  n      Re{ WNn}      Im{ WNn}
------------------------------
  0      1,0000     -0,0000
  1      0,7071     -0,7071
  2      0,0000     -1,0000
  3     -0,7071     -0,7071
------------------------------

Computing 8-point FFT
  Stages: 3
p1=0  p2=1
p1=2  p2=3
p1=4  p2=5
p1=6  p2=7
------------------
---- stage:  0 ----
------------------
  k     Re{ si}      Im{ si}       Re{ so}      Im{ so}
-------------------------------------------------------
  0     0,3536     0,0000       0,7071     0,0000
  1     0,3536     0,0000      -0,0000     0,0000
  2     0,6464     0,0000      -0,7071     0,0000
  3    -1,3536     0,0000       2,0000     0,0000
  4     0,3536     0,0000      -0,7071     0,0000
```

```
  5    -1,0607     0,0000      1,4142     0,0000
  6     1,0607     0,0000      0,7071     0,0000
  7    -0,3536     0,0000      1,4142     0,0000
-------------------------------------------------
p1=0  p2=2
p1=1  p2=3
p1=4  p2=6
p1=5  p2=7
-------------------
---- stage:  1 ----
-------------------
  k      Re{ si}     Im{ si}      Re{ so}     Im{ so}
-------------------------------------------------------
  0     0,7071      0,0000       0,0000      0,0000
  1    -0,0000      0,0000      -0,0000     -2,0000
  2    -0,7071      0,0000       1,4142      0,0000
  3     2,0000      0,0000      -0,0000      2,0000
  4    -0,7071      0,0000      -0,0000      0,0000
  5     1,4142      0,0000       1,4142     -1,4142
  6     0,7071      0,0000      -1,4142      0,0000
  7     1,4142      0,0000       1,4142      1,4142
-------------------------------------------------------
p1=0  p2=4
p1=1  p2=5
p1=2  p2=6
p1=3  p2=7
-------------------
---- stage:  2 ----
-------------------
  k      Re{ si}     Im{ si}      Re{ so}     Im{ so}
-------------------------------------------------------
  0     0,0000      0,0000      -0,0000      0,0000
  1    -0,0000     -2,0000       0,0000     -4,0000
  2     1,4142      0,0000       1,4142      1,4142
  3    -0,0000      2,0000      -0,0000      0,0000
  4    -0,0000      0,0000       0,0000      0,0000
  5     1,4142     -1,4142      -0,0000     -0,0000
  6    -1,4142      0,0000       1,4142     -1,4142
  7     1,4142      1,4142      -0,0000      4,0000
-------------------------------------------------------
Thank you for using FFTcta.
BUILD SUCCESSFUL (total time: 1 second)
```

- Plot the data

  Plotting requires the graphics library “jchart2D”

  By importing

```
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.swing.JFrame;
import info.monitorenter.gui.chart.*;
```

```
import info.monitorenter.gui.chart.traces.Trace2DSimple;
import java.awt.Color;
```

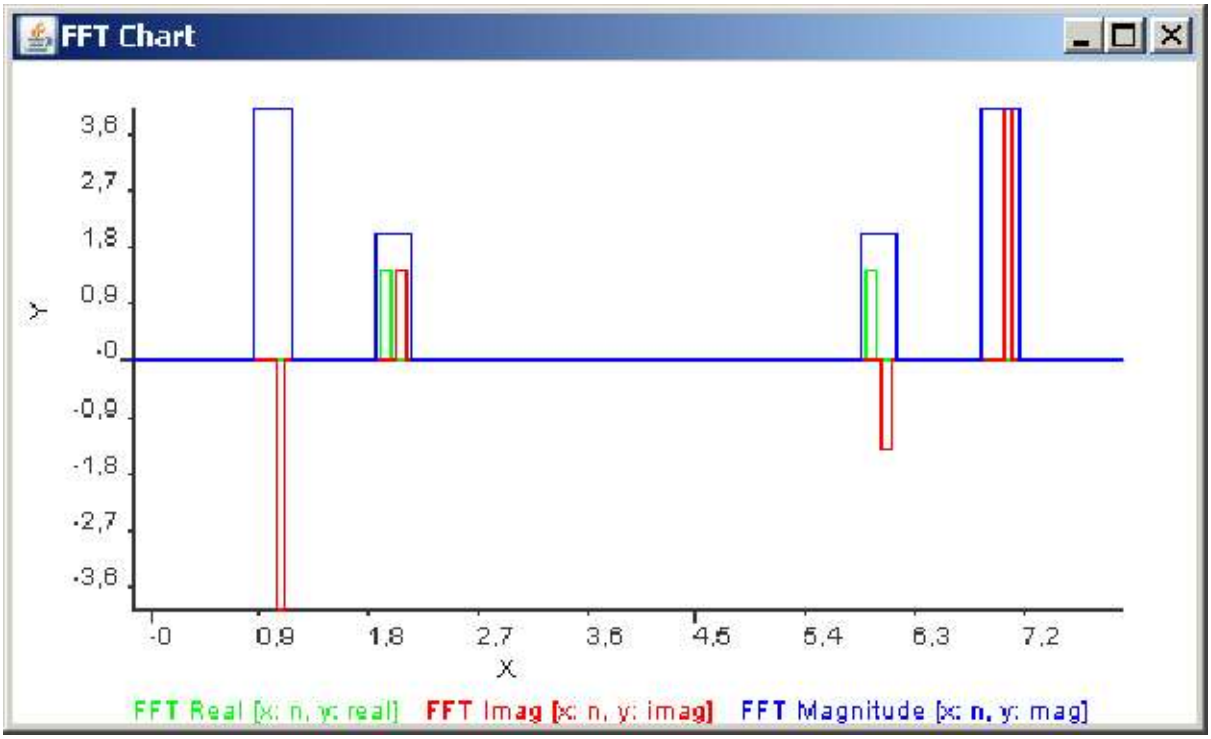
The graphics library can be used. It requires the variables

```
static Chart2D fftchart = new Chart2D();
static ITrace2D signtrace = new Trace2DSimple();
static JFrame fftframe = new JFrame("FFT Chart");
```

and the initializations

```
// graphics stuff
signtrace.setColor(Color.blue);
signtrace.setName("FFT Magnitude");
signtrace.setPhysicalUnits("n", "mag");
fftchart.addTrace(signtrace);
fftframe.getContentPane().add(fftchart);
fftframe.setSize(400, 300);
fftframe.addWindowListener(new WindowAdapter(){
    @Override
    public void windowClosing(WindowEvent e){
        System.exit(0);
    }
}
);
fftframe.setVisible(true);
```

Plotting of the magnitudes results in the following window:



6 Digital Filters

What is commonly called digital filters are time-discrete filters for sampled signals. It is assumed that the samples are taken every sampling interval (periodic sampling). The interval is calls sampling time T_S. This is the reciprocal of the sampling frequency f_S.

The digital filter (DSP = digital signal processing) builds the core of a typical system to process measured data (fig.).

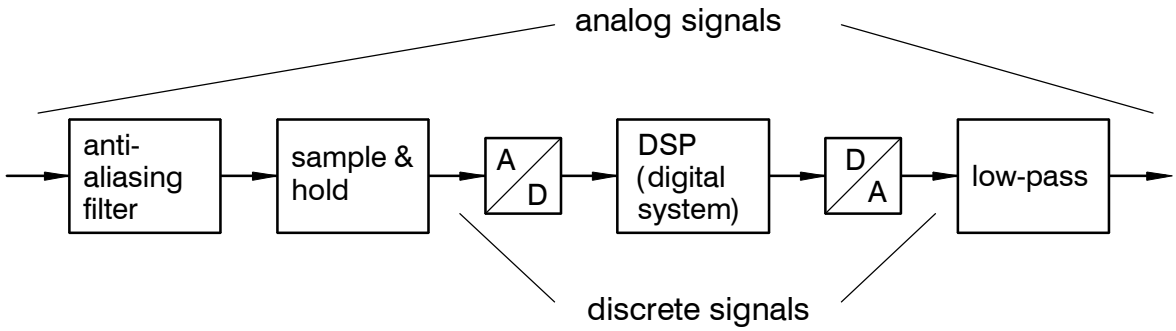


Figure 1.47: Typical system for signal processing

In this section we will focus on the DSP part. The necessity of the other elements will become apparent from the principles of the digital filters. It is assumed that the A/D and D/A converters provide enough precision for filter calculations. A bipolar 16 bit converter for instance offers a dynamic range (ratio of greatest to smallest amplitude) of

$$D = 20\log(2^{15}) = 90\text{dB}\,. \tag{1.36}$$

The sample & hold element captures the signal at every sampling time and keeps the signal constant during the sampling interval.

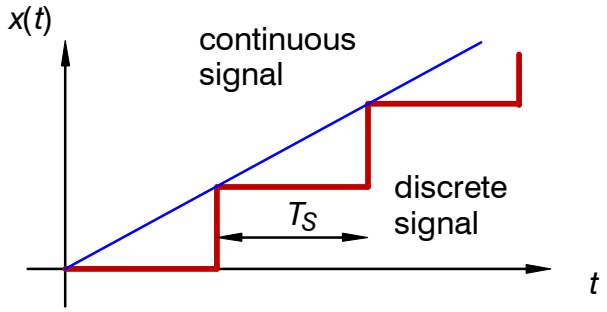


Figure 1.48: Sample and hold function

The act of sampling is mathematically the product of a series of impulses

$$\delta_{SS} = \sum_{k=-\infty}^{\infty} \delta(t - kT_S)\,, \tag{1.37}$$

where $\delta(t)$ is the (Dirac) impulse. This "chain" of impulse is zero except for multiples of the sampling period. It's magnitude is infinite but the "area", i.e. the integral over time is 1 for every single impulse.

If we use a modified impulse

$$\delta_{T0} = \frac{1}{T_0} \quad \text{for} - \frac{T_0}{2} \le t \le \frac{T_0}{2}, \tag{1.38}$$

the complex Taylor series coefficients c_k can be calculated. Obviously

$$\lim_{T_0 \to 0} \delta_{T0}(t) = \delta(t) \tag{1.39}$$

is valid. We will use (1.39) to recover the Taylor series coefficients for δ_{SS}.

Since (1.37) is periodic with T_S the Taylor series gives

$$c_k = \frac{1}{T_S} \int_{-\frac{T_S}{2}}^{\frac{T_S}{2}} x(t) e^{-j\frac{k2\pi t}{T_S}} dt = \lim_{T_0 \to 0} \frac{1}{T_S} \int_{-\frac{T_0}{2}}^{\frac{T_0}{2}} \frac{1}{T_0} e^{-j\frac{k2\pi t}{T_S}} dt . \tag{1.40}$$

With

$$\omega_s = \frac{2\pi}{T_S} \tag{1.41}$$

we obtain

$$c_k = \lim_{T_0 \to 0} \frac{1}{T_S} \int_{-\frac{T_0}{2}}^{\frac{T_0}{2}} \frac{1}{T_0} e^{-jk\omega_s t} dt . \tag{1.42}$$

Integration and carrying out the limes operation gives

$$c_k = \lim_{T_0 \to 0} \frac{1}{T_S} \frac{1}{T_0} \frac{-1}{jk\omega_s} \left[-2j \sin\left(k\omega_s \frac{T_0}{2} \right) \right] = \frac{1}{T_S} . \tag{1.43}$$

This shows that the impulse has uniform (constant) spectral distribution. The multiplication with $x(t)$ does not change the frequencies contained in x. The chain of Dirac impulses can be written as a Fourier series with $T = T_S$

$$\delta_{SS} = \sum_{k=-\infty}^{\infty} \delta(t - kT) = \sum_{k=-\infty}^{\infty} c_k e^{jk\omega_s t} = \frac{1}{T} \sum_{k=-\infty}^{\infty} e^{jk\omega_s t} . \tag{1.44}$$

Thus, the multiplication with x becomes

$$x^*(t) := x(t)\,\frac{1}{T}\sum_{k=-\infty}^{\infty} e^{jk\omega_s t} = x(t)\,\frac{1}{T}\sum_{k=-\infty}^{\infty} e^{-jk\omega_s t}\,. \tag{1.45}$$

The Fourier transform of (1.45) is

$$X^*(j\omega) = \int_{-\infty}^{\infty} x^*(t)e^{-j\omega t}dt = \frac{1}{T}\int_{-\infty}^{\infty} x(t)\sum_{k=-\infty}^{\infty} e^{-jk\omega_s t}\,e^{-j\omega t}dt\,. \tag{1.46}$$

Since the integral over a sum is the sum of integrals we can write

$$X^*(j\omega) = \frac{1}{T}\sum_{k=-\infty}^{\infty}\int_{-\infty}^{\infty} x(t)e^{-j(\omega+k\omega_s)t}\,dt = \frac{1}{T}\sum_{k=-\infty}^{\infty} X(j\omega + jk\omega_s)\,. \tag{1.47}$$

The spectrum of $X^*(j\omega)$ is an infinite sum of the spectrum of $X(j\omega)$ shifted by $k\omega_s$.

The following figure shows an example of the spectrum for $X(j\omega)$ and $X^*(j\omega)$.

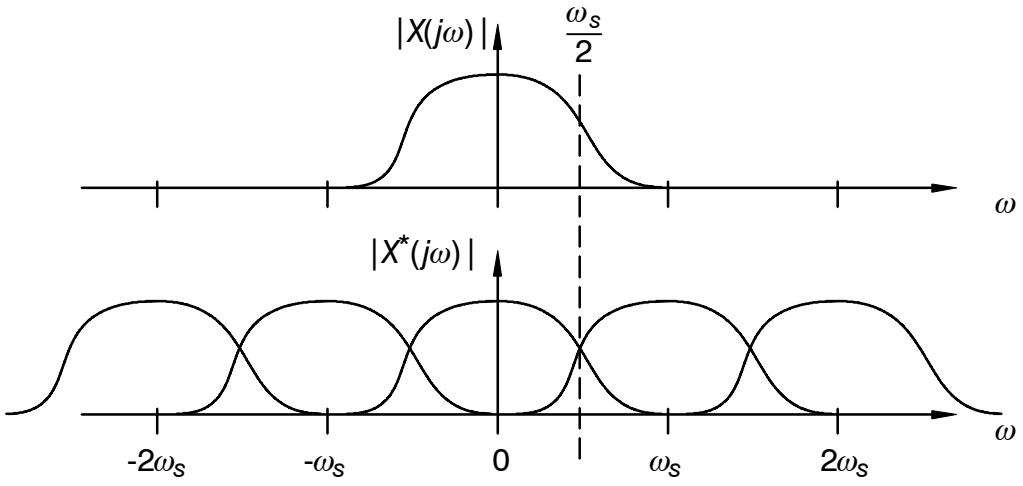


Figure 1.49: Spectrum of $X(j\omega)$ and $X^*(j\omega)$

Fig. 1.49 illustrates that a signal $x(t)$ can be recovered from the sampled system x*(t) if and only if the highest signal frequency in x does not exceed $\omega_s/2$.

Shannon theorem: The maximum signal frequency must not exceed $\omega_s/2$ (one half of the sampling frequency).

Therefore it is mandatory for almost all application to provide an analog low pass filter (**anti-aliasing filter**) with a pass band limited to $\omega_s/2$. A violation to the Shannon theorem is shown in fig. 1.50.

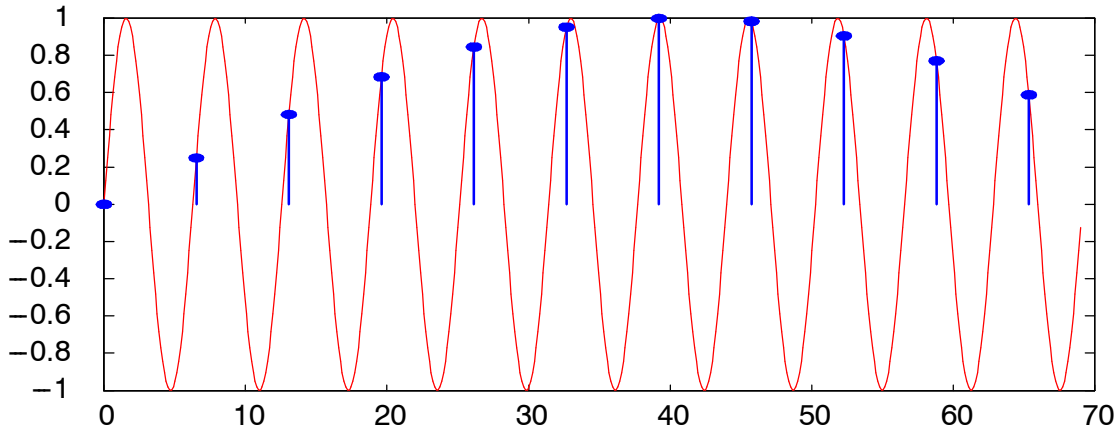


Figure 1.50: Violation of the Shannon theorem

The sine signal has a frequency of $1.04\omega_s$. The sampling frequency appears to be a slow sine oscillation of $0.04\omega_s$ (the difference of $1.04\omega_s$ and ω_s). This effect is called signal ambiguity and can be explained by the Fourier transform of the sampled signal.

7 Z-Transform

The discrete Fourier transform (DFT) or the equivalent FFT provide a frequency description of a signal. The signals are assumed to be periodical. A more general description in frequency domain is the Laplace transform

$$X(s) = \int_0^\infty x(t)e^{-st}dt \,. \tag{1.48}$$

This transform is limited to the positive time axis, but it can be used for all exponentially bounded functions. A special dual side Laplace transform exists if the Laplace transform is required for a signal which is valid on the whole time axis. This topic will not be covered in this section.

Since $\mathcal{F}\{\delta(t)\} = 1$ the Laplace transform of the sampled signal (multiplied with the sequence of impulses becomes

$$X^*(s) = \sum_{k=0}^{\infty} x(kT_S)e^{-skT_S} = \sum_{k=0}^{\infty} x[k]\, e^{-skT_S} \,. \tag{1.49}$$

By introducing the discrete frequency variable

$$z := e^{sT_S} \tag{1.50}$$

we obtain for (1.49) the z-transform

$$X(z) = \sum_{k=0}^{\infty} x(kT_S)\, z^{-k} \tag{1.51}$$

or shorter

$$X(z) = \sum_{k=0}^{\infty} x[k]\, z^{-k}\,. \tag{1.52}$$

The result is a real rational function in z. The main benefit of the z-transform is that input-output relations of signals in z domain are also real-rations functions called transfer functions or *discrete filters*.

Example: z-transform of the step function $\sigma[k] = 1$ for $k \geq 0$

$$X(z) = \sum_{k=0}^{\infty} z^{-k} = \frac{1}{1 - \frac{1}{z}} = \frac{z}{z - 1}\,. \tag{1.53}$$

This is because

$$\sum_{k=0}^{N} z^{-k} = z^{-1} \sum_{k=0}^{N} z^{-k} + 1 - z^{-(N+1)} \tag{1.54}$$

and therefore

$$\left(1 - z^{-1}\right) \sum_{k=0}^{N} z^{-k} = +1 - z^{-(N+1)}\,. \tag{1.55}$$

The sum becomes the algebraic expression

$$\sum_{k=0}^{N} z^{-k} = \frac{1 - z^{-(N+1)}}{1 - z^{-1}}\,. \tag{1.56}$$

The limes for $N \to \infty$ gives

$$\sum_{k=0}^{\infty} z^{-k} = \frac{1}{1 - z^{-1}}\,, \qquad \text{q.e.d.} \tag{1.57}$$

The series converges only if $|z| > 1$, i.e. if z is *outside* the unit circle. Z-transforms for many signals are found in the literature. The most common signals are shown in the table below.

$x[k]$	$\mathcal{Z}\{x[k]\} = X(z)$
$\sigma(kT)$ bzw. 1	$\frac{z}{z-1}$
kT	$\frac{Tz}{(z-1)^2}$
$(kT)^2$	$\frac{T^2z(z-1)}{(z-1)^3}$
e^{-akT}	$\frac{z}{z-e^{-aT}}$
kTe^{-akT}	$\frac{Tze^{-aT}}{\left(z-e^{-aT}\right)^2}$
$\sin(\omega_0 kT)$	$\frac{z\sin(\omega_0 T)}{z^2-2z\cos(\omega_0 T)+1}$
$\cos(\omega_0 kT)$	$\frac{z\left(z-\cos(\omega_0 T)\right)}{z^2-2z\cos(\omega_0 T)+1}$

8 Sample&Hold (1st Order Interpolation)

The signals in a discrete filter are impulses modulated by the value of the functions at each sampling event. The physical signals however are “staircase” functions (constant over one sampling interval). This is achieved by the 1st order hold function

$$H(s) = \frac{1-e^{-Ts}}{s} . \tag{1.58}$$

This is an integrator with a dead-time of T_S.

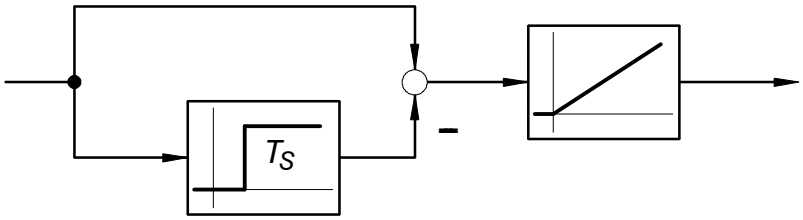


Figure 1.51: 1st order hold

The impulse response is shown in fig. 1.52. It converts an impulse to a pulse of length T_S.

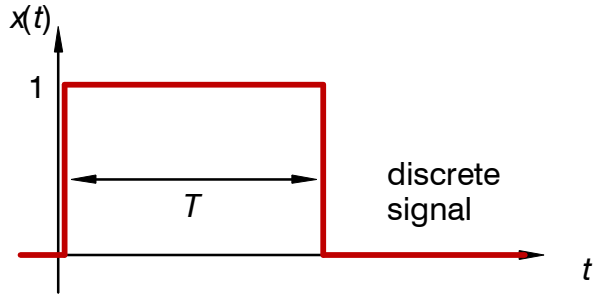


Figure 1.52: Impulse response of the 1st order hold

Since the 1 order hold is part of almost all discrete systems, it is important to understand the effects. In frequency domain

$$H(j\omega) = \frac{1 - e^{-j\omega T_S}}{j\omega} = \frac{e^{j\frac{\omega T_S}{2}} - e^{-j\frac{\omega T_S}{2}}}{j\omega} e^{-j\frac{\omega T_S}{2}}$$

$$= T_S \frac{\sin\left(\frac{\omega T_S}{2}\right)}{\frac{\omega T_S}{2}} e^{-j\frac{\omega T_{subS}}{2}} = T_S \,\mathrm{si}\left(\frac{\omega T_S}{2}\right) e^{-j\frac{\omega T_S}{2}}. \tag{1.59}$$

It can be seen that it changes gain and phase over frequency. If the sampling period is small with respect to ω the gain does not change much.

The phase change is the same as a dead-time element with dead time $T_S / 2$.

9 Discrete Transfer Function

The discrete transfer function or filter $G(z)$ describes the input-output behavior of signals in z-domain

$$y(z) = G(z)\, u(z)\,. \tag{1.60}$$

This is valid only for LTI (linear time-invariant) systems, i.e. real rational functions in z. A linear function

- does not depend on the amplitude of signals,
- does not change any frequency, i.e. the output must contains the same frequencies that the input signal has.

A counterexample is the square of an input signal $u[k] = \sin(\omega T_S k)$

$$y[k] = \sin^2(\omega T_S k) = \frac{1}{2}\left(1 - \cos(2\omega T_S k)\right). \tag{1.61}$$

The result is a DC component (1 / 2) and a cosine of twice of the input frequency, a violation for a linear system.

The general form of the a discrete transfer function is

$$G(z) = \frac{b_m z_m + b_{m-1} z^{m-1} \ldots b_1 z + b_0}{z^n + a_{n-1} z^{n-1} \ldots a_1 z + a_0} = \frac{\sum_{\nu=0}^{m} b_\nu z^\nu}{\sum_{\mu=0}^{n} a_\mu z^\mu}, \quad a_n = 1 . \tag{1.62}$$

For any causal filter $m \leq n$. In some cases the index of the a_k and b_i coefficient have a reverse notation in order to keep the computation indices simpler.

From (1.60) it follows

$$G(z) = \frac{\sum_{\nu=0}^{m} b_\nu z^\nu}{\sum_{\mu=0}^{n} a_\mu z^\mu} = \frac{y(z)}{u(z)} . \tag{1.63}$$

or

$$\sum_{\mu=0}^{n} a_\mu z^\mu y(z) = \sum_{\nu=0}^{m} b_\nu z^\nu u(z) . \tag{1.64}$$

With $a_n = 1$ by definition we can solve (1.64) for

$$z^n y(z) = - \sum_{\mu=0}^{n-1} a_\mu z^\mu y(z) + \sum_{\nu=0}^{m} b_\nu z^\nu u(z) . \tag{1.65}$$

Dividing by z^n results in

$$y(z) = - \sum_{\mu=0}^{n-1} a_\mu z^{-(n-\mu)} y(z) + \sum_{\nu=0}^{m} b_\nu z^{-(n-\nu)} u(z) . \tag{1.66}$$

From the definition if z (1.50) it follows

$$\frac{1}{z} = e^{-sT_S} . \tag{1.67}$$

This is a dead-time of T_S. So $1/z$ is regarded as a shift operator which shifts data by one sampling interval. The factor $1/z^2$ shifts by two sampling intervals and so on. Therefore eq. (1.66) can be written in time-domain

$$y[k] = - \sum_{\mu=0}^{n-1} a_\mu y[k - n + \mu] + \sum_{\nu=0}^{m} b_\nu u[k - n + \nu] . \tag{1.68}$$

This is a difference equation (equivalent to a differential equation for continuous systems). It is called *convolution*.

Discrete transfer function can be derived from analog transfer functions, so that the response to the same input signal is identical at the sampling points. Since this includes a 1st order hold it is calculated according to well known formula

$$G(z) = \frac{z-1}{z}\mathcal{Z}\left\{\frac{G(s)}{s}\right\}, \tag{1.69}$$

where

$$\mathcal{Z}\left\{\frac{G(s)}{s}\right\} \tag{1.70}$$

is the z-transform of $G(s)/s$. The following example illustrates this. For the continuous transfer function

$$G(s) = \frac{2}{s+3} \tag{1.71}$$

we have

$$\frac{G(s)}{s} = \frac{2}{s(s+3)} = \frac{\frac{2}{3}}{s} - \frac{\frac{2}{3}}{s+3}. \tag{1.72}$$

In time domain this is the signal

$$v(t) = \frac{2}{3}\left(1 - e^{-3t}\right). \tag{1.73}$$

The corresponding z-transform according to the table above gives

$$V(z) = \frac{2}{3}\left(\frac{z}{z-1} - \frac{z}{z-e^{-3T}}\right). \tag{1.74}$$

Thus, the discrete version of (1.71) becomes

$$G(z) = \frac{z-1}{z}V_1(z) = \frac{2}{3}\left(1 - \frac{z-1}{z-e^{-3T}}\right) = \frac{2}{3}\frac{1-e^{-3T}}{z-e^{-3T}}. \tag{1.75}$$

With $T_S = 0.5$ we obtain

$$G(z) = \frac{0.5179}{z-0.2231}. \tag{1.76}$$

The step response show a perfect match between the continuous and the discrete system at it's sampling points.

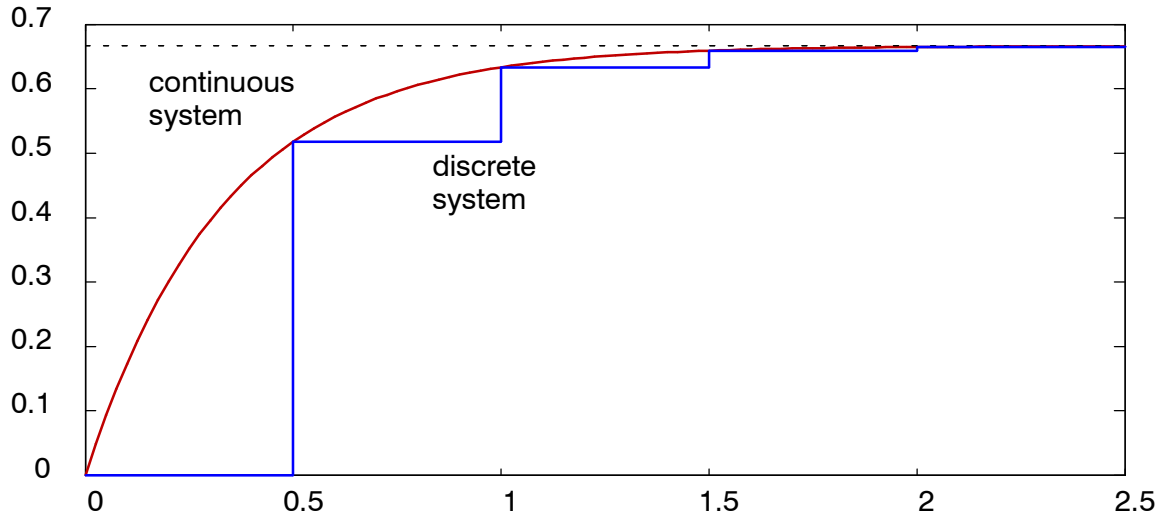


Figure 1.53: Continuous and discrete step response

9.1 Useful Forms for Real-Time Computations (SISO)

A suitable form for hardware accelerated DSP is handled in chapter 11.2.

The notation of a discrete transfer function is taken from (1.62).

Canonical forms are state-space forms where the numerator polynomial coefficients are identical to those of the B or C matrices and the denominator polynomial coefficients are found in the A matrix. This form allows the filter computation in real-time. Of course, these forms are not the only used structures for computation.

All forms can be converted from any other form by the similarity transform

$$x^* = Tx, \quad |T| \neq 0. \tag{1.77}$$

This a coordinate transform in the state-space variables. If the determinant of T is non zero, the inverse of T exists, so we can rewrite the state equations

$$A^* = TAT^{-1}, \; B^* = TB, \; C^* = CT^{-1}. \tag{1.78}$$

You should be able to verify (1.78) by the definition (1.77). This means that the following systems have identical transfer behavior:

$$x[k+1] = Ax[k] + Bu[k], \quad y[k] = Cx[k], \tag{1.79}$$

$$x^*[k+1] = A^*x^*[k] + B^*u[k], \quad y[k] = C^*x^*[k]. \tag{1.80}$$

Since the D matrix is not affected, it has been omitted here.

9.1.1 Controller Canonical Form

The name is derived from the fact that the transformation matrix can be calculated from the inverse of the controllability matrix

$$Q_c = \left[B \; AB \; A^2B \; ... \; A^{n-1}B \right]. \tag{1.81}$$

The system needs to be controllable, due to the required inverse of (1.81). Moreover, this form can be used to directly design state-feedback control.

The matrix A becomes a *Frobenius* structure

$$A = \begin{bmatrix} 0 & 1 & & 0 \\ & & \ddots & \\ & & & 1 \\ -a_0 & -a_1 & \cdots\cdots & -a_{n-1} \end{bmatrix}. \tag{1.82}$$

It can be seen that the (negative) denominator coefficients are part of the matrix A.

The B matrix simply has the structure

$$B = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{bmatrix}, \tag{1.83}$$

while the matrix C contains the numerator coefficients

$$C = \left[b_0 \, b_1 \ldots b_{n-1} \right]. \tag{1.84}$$

Due to its special form of the state-space matrices the block diagram has a very simple structure.

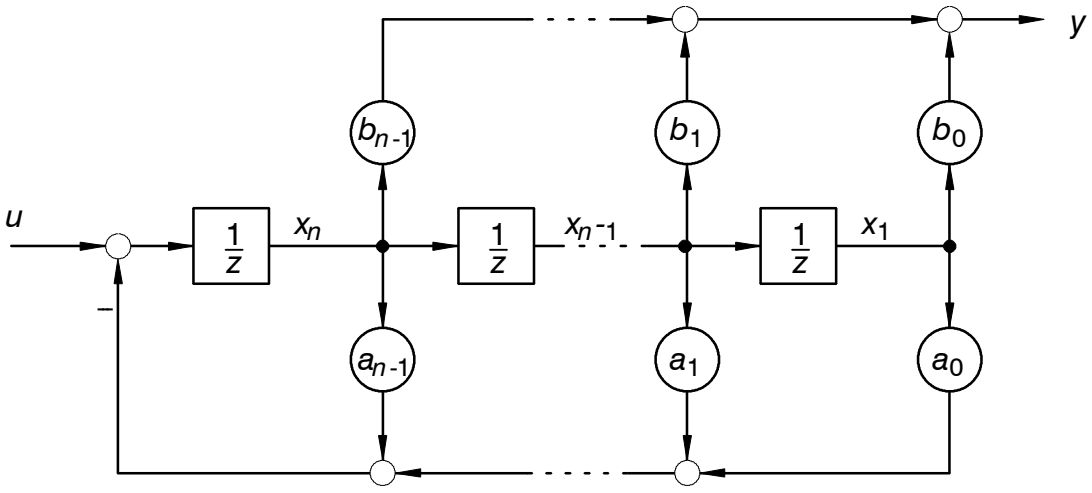


Figure 1.54: Controller canonical form

9.1.2 Observer Canonical Form

The observer canonical form is *dual* to the controller canonical form. The name is derived from the fact that the transformation matrix can be calculated from the inverse of the observability matrix

$$Q_o = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix} . \tag{1.85}$$

Any observable system can thus be transformed into the observer canonical form.

The matrix A becomes also a *Frobenius* structure

$$A = \begin{bmatrix} 0 & \cdots & 0 & -a_0 \\ 1 & & & -a_1 \\ & \ddots & & \vdots \\ & & & \\ 0 & & 1 & -a_{n-1} \end{bmatrix} . \tag{1.86}$$

It can be seen that the (negative) denominator coefficients are part of the matrix A.

The B matrix contains the denominator polynomial coefficients

$$B = \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_{n-1} \end{bmatrix} , \tag{1.87}$$

while the matrix C shows the simple form

$$C = [\, 0\, 0 \ldots 1 \,] . \tag{1.88}$$

Due to its special form of the state-space matrices the block diagram has the simple structure.

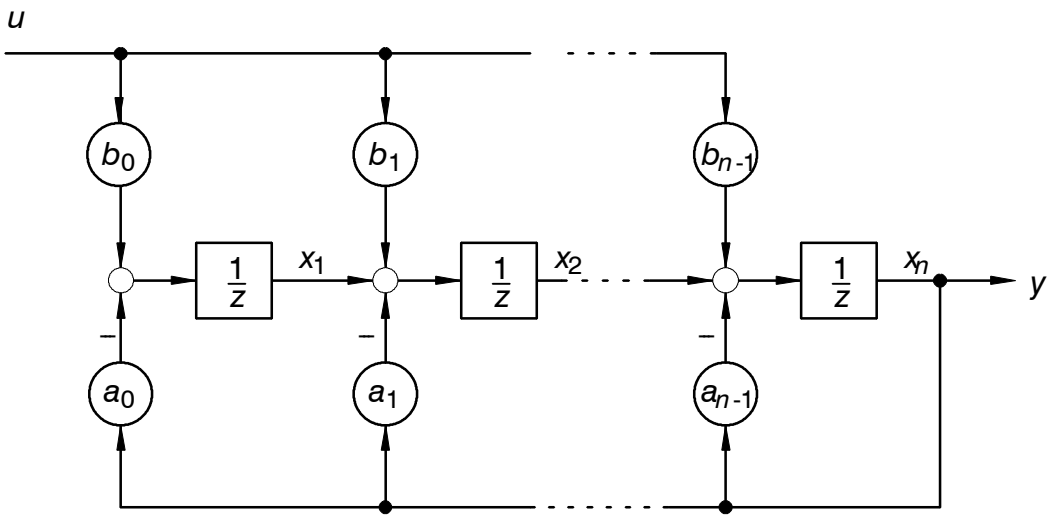


Figure 1.55: Observer canonical form

9.1.3 Exercise 1: Conversion from Transfer Function to State-Space and Verification (Matlab)

- Create the discrete transfer function

$$G(z) = \frac{-0.4702z^2 + 0.5199z + 0.4295}{z^3 - 1.28z^2 + 1.006z - 0.2466}.$$

- Convert the transfer function to controller canonical state-space form by inserting the polynomial coefficients into the appropriate matrices. Verify that the step responses are identical.

- Convert the transfer function to observer canonical state-space form by inserting the polynomial coefficients into the appropriate matrices. Verify that the step responses are identical.

- What form produces Matlab's ss(Gxx) ?

9.1.4 Exercise 2: Filter Algorithm for Controller Canonical Form

The controller canonical form – also known as Direct Form II filter – is widely used for filter implementation. For the filter in 9.1.3 the filter algorithm should be coded in Matlab.

- Create a new m-file "df2filter.m". The first line of this m-file should be

  function y = d2filter(b, a, u)

  where *a* is the vector of the denominator coefficients, *b* is the vector of the denominator coefficients and *u* is the input signal vector. The sampling period is not required for filter computation.

- Provide basic error checking on *a* and *b* dimensions.
- Verify proper operation by computing the step response (all values in u are "1") and compare it to Matlab's step(). Note that Matlab stores numerator and denominator values for transfer functions in reverse order. For instance for a 3. order system the coefficient b_0 is stored in b[3] and b_2 is stored in b(1).

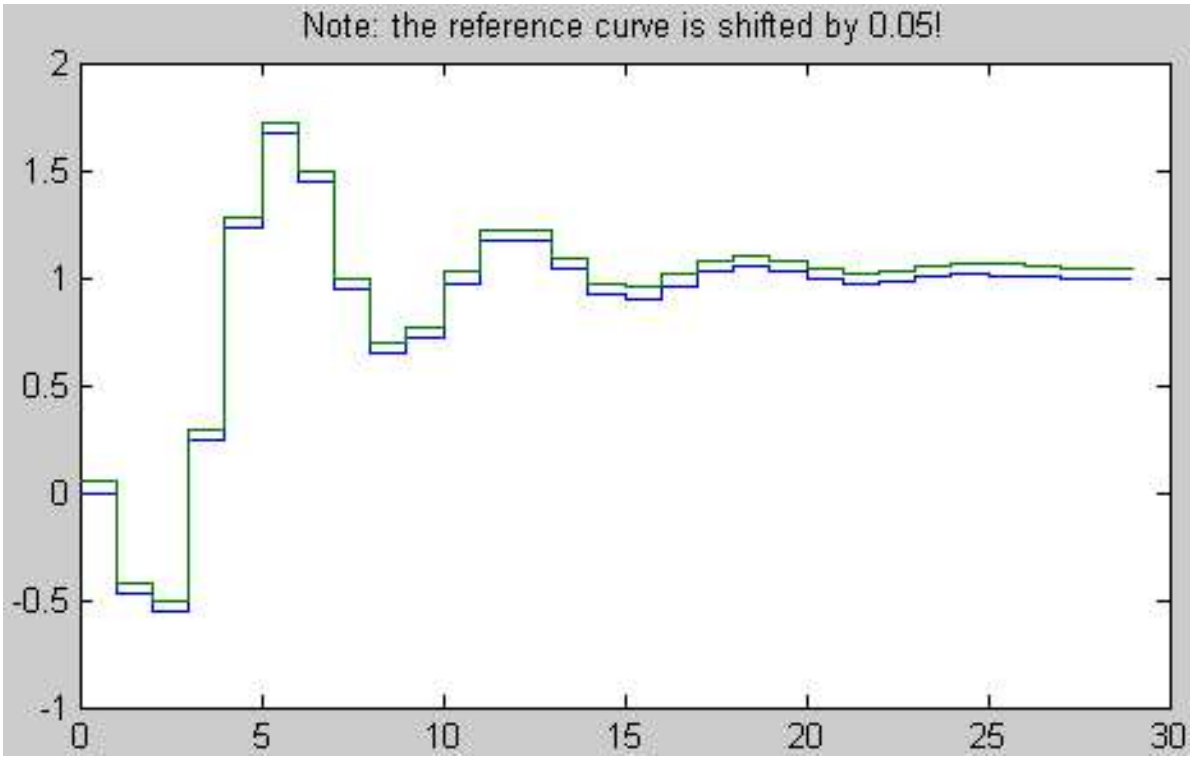


Figure 1.56: Filter calculation result

9.2 Example: Band-Limited Derivative Filter

In many cases the derivative of a sensor signal is required for control purposes. If for example a position signal is available the speed can be calculated by taking its derivative

$$v = \frac{\mathrm{d}x}{\mathrm{d}t} . \tag{1.89}$$

This operation amplifies possible noise in high frequency range as can be seen from

$$x = \cos(\omega_1 t) \quad => \quad \dot{x} = \omega_1 \sin(\omega_1 t) . \tag{1.90}$$

To avoid the negative effects of high frequency noise a band-limited derivative can be used. The ideal derivative filter is $G_{di}(s) = s$, which is a non causal function anyway. A band-limited version is

$$G_d(s) = \frac{s}{(T_1 s + 1)^2} . \tag{1.91}$$

For $T_1 \to 0$ the ideal derivative filter is recovered. With $T_S = 1\text{ms}$ and a corner frequency of 200Hz we obtain

$$T_1 = \frac{1}{2\,\pi\,200\text{Hz}} = 7.9577\,10^{-4}\,. \tag{1.92}$$

First $v(s) = G(s)/s$ needs to be calculated

$$v(s) = \frac{G(s)}{s} = \frac{1}{\left(T_1 s + 1\right)^2} = \frac{1}{T_1^2}\,\frac{1}{\left(s + \frac{1}{T_1}\right)^2}\,. \tag{1.93}$$

In time domain this corresponds to

$$v(t) = \frac{t}{T_1^2}\,e^{-\frac{t}{T_1}}\,. \tag{1.94}$$

The table of corresponding z-transform with $a = 1/T_1$ gives

$$V(z) = \frac{T_S}{T_1^2}\,\frac{z\,e^{-\frac{T_S}{T_1}}}{\left(z - e^{-\frac{T_S}{T_1}}\right)^2} \tag{1.95}$$

Finally the resulting discrete transfer function is

$$G_d(z) = \frac{z-1}{z}\,V(z) = \frac{T_S\,e^{-\frac{T_S}{T_1}}}{T_1^2}\,\frac{z-1}{\left(z - e^{-\frac{T_S}{T_1}}\right)^2}\,. \tag{1.96}$$

With numerical data we obtain

$$G_d = 449.437\,\frac{z-1}{(z-0.2846)^2}\,. \tag{1.97}$$

A better matching of the frequency domain is obtained by the bilinear transform where s is replaced in G(s) by (Padé approximation)

$$s = \frac{2}{T_S}\,\frac{z-1}{z+1}\,. \tag{1.98}$$

The result of the bilinear transform is

$$G_{d,b} = 297.791\,\frac{z^2-1}{(z-0.2283)^2}\,. \tag{1.99}$$

The comparison of all filters for a 25Hz sine signal is shown in fig. 1.57.

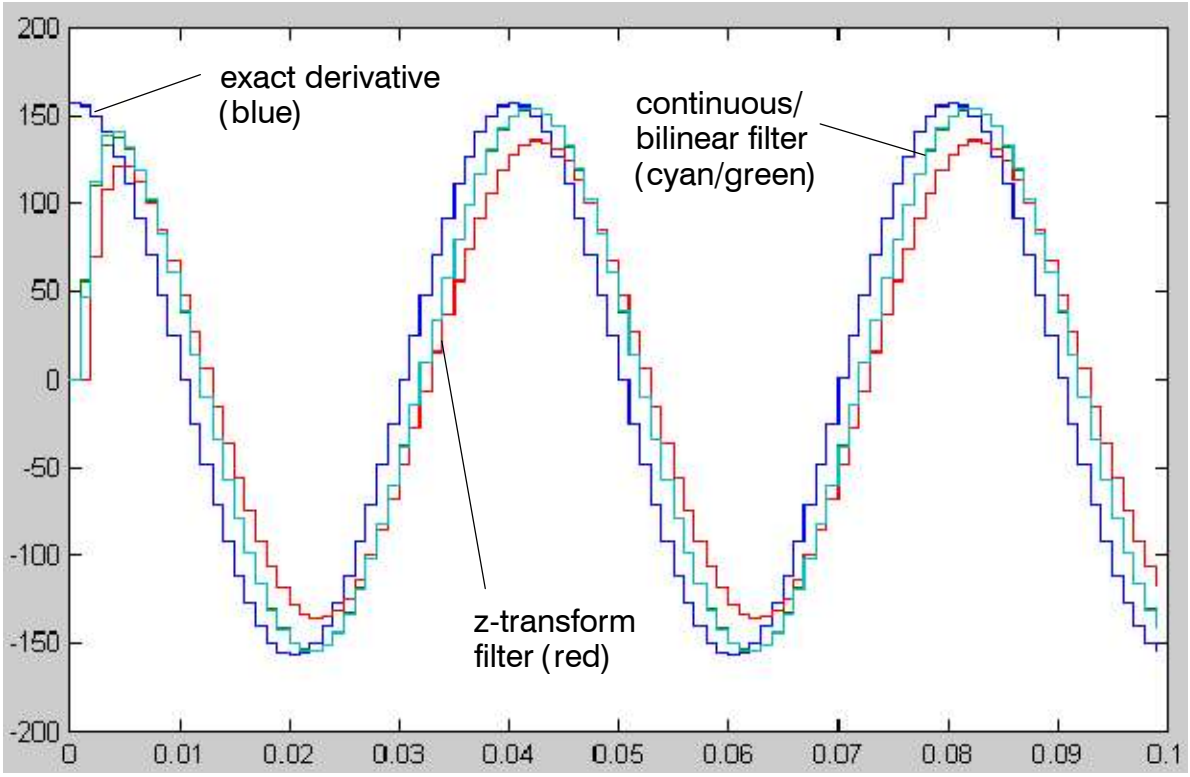


Figure 1.57: Filter responses to a 25Hz sine signal

All filters give satisfactory results but the bilinear filter output matches almost exactly the output of the continuous filter.

10 Filter Design

Filter design can be reduced to low pass filter since the filter types can be derived from low pass filters by transformations. The simplest low pass (1. order) is given by

$$G_1 = \frac{1}{1+s}\,. \tag{1.100}$$

The frequency behavior is evaluated for $s = j\omega$

$$G_1(j\omega) = \frac{1}{1+j\omega} \tag{1.101}$$

and we obtain for the magnitude (squared)

$$|G_1(j\omega)|^2 = \frac{1}{1+\omega^2} \tag{1.102}$$

The low frequency gain thus is 1 ($\lim \omega \to 0$) while at high frequency ($\omega \to \infty$) the gain is proportional to $1/\omega$, i.e. 20bB per decade. Higher order low pass filters are obtained by serial connection of low pass filters of 1st order

$$G_{lp} = \frac{1}{(1 + \alpha_1 s)(1 + \alpha_2 s)...(1 + \alpha_n s)} . \tag{1.103}$$

If all α_k are made identical the filter has n poles at $1/\alpha_k$

$$G_{lp} = \frac{1}{(1 + \alpha_k s)^n} . \tag{1.104}$$

This is called the critical damping (aperiodical filter). The corner frequency is defined as the frequency where the gain changes by $\sqrt{2}$, i.e.

$$\left|G_{lp}(j\omega_c)\right|^2 = \frac{1}{\left(1 + (\alpha_k \omega_c)^2\right)^n} \stackrel{!}{=} \frac{1}{2} . \tag{1.105}$$

The corner frequency remains unchanged $(\omega_c = 1)$ if

$$\alpha_k = \sqrt{\sqrt[n]{2} - 1} . \tag{1.106}$$

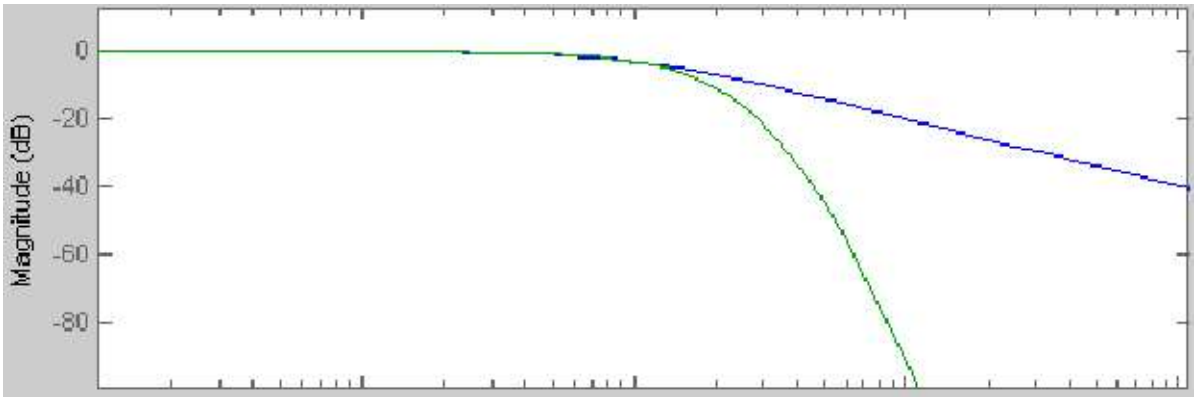


Figure 1.58: 1. and 10. order aperiodical low pass filter $(\omega_c = 1)$

Much better results are achieved when other coefficients in the general form of the low pass filter

$$G_{lp}(s) = \frac{A_0}{1 + c_1 s_n + c_2 s_n^2 + ... + c_n s_n^n} \tag{1.107}$$

are used. Here

$$s_n := \frac{s}{\omega_c} \tag{1.108}$$

is the normalized frequency variable (normalized with respect to the corner frequency).

For the design a factorized version like (1.103) is much better suited. Since poles can be complex (and are complex for most filters!) the factorization should be carried out in quadratic terms (or a possible single pole for odd orders, i.e. $b_1 = 0$)

$$G_{lp}(s) = \frac{A_0}{\left(1 + a_1 s_n + b_1 s_n^2\right)\left(1 + a_2 s_n + b_2 s_n^2\right)...} . \tag{1.109}$$

There exits no “optimal” low pass. Filter can be only optimized with respect to certain objectives. The major types of filters are:

- Aperiodical filter (see above filter with real poles)
- Butterworth filter: optimized for flat pass-band and sharp cut-off frequency. The filter shows significant overshoot in time domain.
- Chebychev filter: steeper descent than Butterworth but with oscillatory gain (pass band ripple) in the pass-through frequency range. Large overshoot property in time domain.
- Bessel filter: optimized for signal transmission of step type functions. This is achieved if the phase change is proportional to frequency (flat group delay).

10.1 Butterworth Filter

For several application the Butterworth filter is an accepted compromise. The magnitude depends only on the square of ω. Therefore, it can be written as

$$|G|^2 = \frac{A_0^2}{1 + k_2\omega_n^2 + \ldots + k_{2n}\omega_{2n}^{2n}} . \tag{1.110}$$

If ω_n is small the biggest impact have low exponents on ω. The best possible performance is achieved if the magnitude becomes

$$|G|^2 = \frac{A_0^2}{1 + k_{2n}\omega_{2n}^{2n}} . \tag{1.111}$$

The first Butterworth polynomials are listed below:

$$n = 1: \quad 1 + s_n \tag{1.112}$$

$$n = 2: \quad 1 + \sqrt{2}\, s_n + s_n^2 \tag{1.113}$$

$$n = 3: \quad 1 + 2\, s_n + 2s_n^2 + s_n^3 \tag{1.114}$$

$$n = 4: \quad 1 + 2.613\, s_n + 3.1414 s_n^2 + 2.613 s_n^3 + s_n^4 \tag{1.115}$$

Note that the coefficients are symmetrical with respect to $n/2$.

More general the coefficient of the quadratic form (1.109) are explicitly for even orders n:

$$a_k = 2\cos\frac{(2k - 1)\pi}{2n} , \quad 1 \le k \le \frac{n}{2} , \tag{1.116}$$

$$b_k = 1 , \quad 1 \le k \le \frac{n}{2} \tag{1.117}$$

and for odd orders n:

$$a_1 = 1 , \tag{1.118}$$

$$b_1 = 0\,, \tag{1.119}$$

$$a_k = 2\cos\frac{(2k-1)\pi}{2n}\,, \quad 2 \le k \le \frac{n+1}{2}\,, \tag{1.120}$$

$$b_k = 1\,, \quad 2 \le k \le \frac{n+1}{2}\,. \tag{1.121}$$

This allows to calculate any desired order of Butterworth low pass filters.

10.2 Chebychev Filter

The Chebychev filter offers the sharpest cut-off behavior compared to the Butterworth and the Bessel filter. The price for the sharp decay is a passband ripple determined by the parameter ε. Chebychev polynomials are bounded to ± 1 in the range $0 \le x \le 1$ and the magnitude increases monotonically for $x > 1$. The Chebychev polynomials are

$$T_n(x) = \begin{cases} \cos(n\ \mathrm{acos}x) & \text{for } 0 \le x \le 1 \\ \cosh(n\ \mathrm{acosh}x) & \text{for } x > 1 \end{cases}\,. \tag{1.122}$$

The polynomials can be expressed without cos and cosh. The first 4 order polynomials are

$$T_1 = x\,, \tag{1.123}$$

$$T_2 = 2x^2 - 1\,, \tag{1.124}$$

$$T_3 = 4x^3 - 3x\,, \tag{1.125}$$

$$T_4 = 8x^4 - 8x^2 + 1\,. \tag{1.126}$$

With these polynomials the nth order Chebychev filter gain becomes

$$|A|^2 = \frac{kA_0^2}{1 + \epsilon^2 T_n^2(x)}\,. \tag{1.127}$$

The constant k is selected to make the gain for $x = 0$ equal to A_0:

$$k = \begin{cases} 1 & \text{for } n \text{ odd} \\ 1 + \epsilon^2 & \text{for } n \text{ even} \end{cases}\,. \tag{1.128}$$

The factor ε determines the degree of passband ripple

$$\frac{A_{\max}}{A_{\min}} = \sqrt{1 + \epsilon^2}\,. \tag{1.129}$$

For odd orders we have

$$A_{max} = A_0 , \quad A_{min} = \frac{A_0}{\sqrt{1 + \epsilon^2}} \tag{1.130}$$

and for even orders n

$$A_{max} = A_0 \sqrt{1 + \epsilon^2} , \quad A_{min} = A_0 . \tag{1.131}$$

With the abbreviation

$$\gamma := \frac{1}{n} \operatorname{asinh} \frac{1}{\epsilon} \tag{1.132}$$

the quadratic factors of the form (1.109) can be calculated explicitly. The Chebychev coefficients for even orders ($1 \le k \le n/2$) are

$$b_k = \frac{1}{\cosh^2 \gamma - \cos^2 \frac{(2k-1)\pi}{2n}} , \tag{1.133}$$

$$a_k = 2 b_k \sinh \gamma \cos \frac{(2k-1)\pi}{2n} . \tag{1.134}$$

The Chebychev coefficients for odd orders are similar

$$b_1 = 0 , \tag{1.135}$$

$$a_1 = \frac{1}{\sinh \gamma} \tag{1.136}$$

and for $2 \le k \le (n+1)/2$:

$$b_k = \frac{1}{\cosh^2 \gamma - \cos^2 \frac{(k-1)\pi}{n}} , \tag{1.137}$$

$$a_k = 2 b_k \sinh \gamma \cos \frac{(k-1)\pi}{n} . \tag{1.138}$$

10.3 Bessel Filter

Butterworth and Chebychev filters lead to distortions when signals are transmitted which contain several frequencies like for instance square signals. The Bessel filters are optimized for low distortions. This is a function of the *group delay* defined as

$$t_{gr} = - \frac{d\varphi}{d\omega} . \tag{1.139}$$

A square signal does not become distorted when the group delay is constant, i.e. if the phase change is proportional to ω.

The theory of Bessel filters will be shown with a 2. order low pass since higher order Bessel filters lead to a set nonlinear equations for the coefficients. For higher order Bessel filters

tables with pre-calculated coefficients can be used. The 2. order low pass is (with gain $A_0 = 1$)

$$G_{lp} = \frac{1}{1 + a_1 s + b_1 s^2} . \tag{1.140}$$

The behavior with respect to harmonic functions is obtained for $s = j\omega_n$

$$G_{lp}(j\omega_n) = \frac{1}{1 - b_1\omega_n^2 + ja_1\omega_n} . \tag{1.141}$$

The phase (lag, since negative) is

$$\varphi = - \operatorname{atan}\frac{a_1\omega_n}{1 - b_1\omega_n^2} . \tag{1.142}$$

With the corner frequency ω_c we define a normalized group delay

$$T_{gr} := t_{gr}\,\omega_c . \tag{1.143}$$

This allows the use of (1.142) for calculations since

$$T_{gr} = - \omega_c \frac{d\varphi}{d\omega} = - \frac{d\varphi}{d\omega_n} . \tag{1.144}$$

The group delay with (1.142) (derivative) becomes [check this!]

$$T_{gr} = \frac{a_1(1 + b_1\omega_n^2)}{1 + (a_1^2 - 2b_1)\omega_n^2 + b_1^2\omega_n^4} . \tag{1.145}$$

For "small" frequencies ($\omega_n{}^2 \ll 1$) the term $b_1{}^2\omega_n{}^4$ has no effect, so we have

$$T_{gr} \approx a_1 \frac{1 + b_1\omega_n^2}{1 + (a_1^2 - 2b_1)\omega_n^2} . \tag{1.146}$$

This expression does not depend on ω_n if

$$b_1 \stackrel{!}{=} a_1^2 - 2b_1 \tag{1.147}$$

or if we set

$$b_1 = \frac{a_1^2}{3} . \tag{1.148}$$

For ω_n the gain must decrease by 3dB which is identical to

$$\left|G_{lp}(j\omega_n = 1)\right|^2 \stackrel{!}{=} \frac{1}{2} . \tag{1.149}$$

With (1.141) the condition (1.149) becomes

$$\left|G_{lp}(j1)\right|^2 = \frac{1}{(1 - b_1)^2 + a_1^2} = \frac{1}{2} . \tag{1.150}$$

The two nonlinear equations (1.148) and (1.150) are satisfied by

$$a_1 = 1.3617\,, \quad b_1 = 0.618\,. \tag{1.151}$$

Higher order Bessel filter coefficients are complicated to calculate and can be taken from tables in the literature.

10.4 Filter Comparison for 8th Order Filters

Butterworth, Chebychev and Bessel filters are compared with the help of Matlab's signal processing tools.

```
% filter comparison (6th order filer)

NN = 8;
fc = 100.0;
wc = 2.0 * pi * fc;

% Butterworth
[B_bu, A_bu] = butter(NN, wc, 's');
G_bu = tf(B_bu, A_bu);

% Chebivev
[B_ch, A_ch] = cheby1(NN, 2.0, wc, 's');
G_ch = tf(B_ch, A_ch);  G_ch = G_ch / dcgain(G_ch);

% Bessel
[B_be, A_be] = besself(NN, wc);
G_be = tf(B_be, A_be);

% all
figure(1);
step(G_bu, G_ch, G_be, 0.1);
figure(2);
bode(G_bu, G_ch, G_be, {1.0e2, 2.0e3});
```

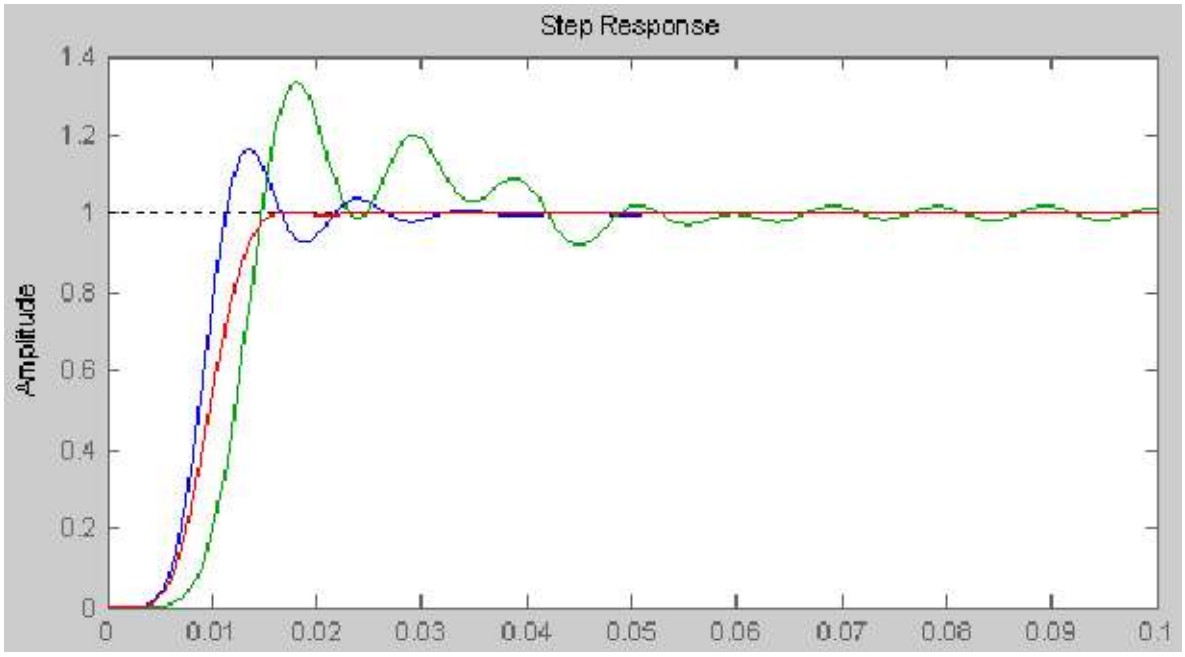


Figure 1.59: Step responses of Butterworth (blue), Chebychev (green) and Bessel (red) filters

As expected the Chebychev filter shows the worst step response property while Bessel offers best filter performance.

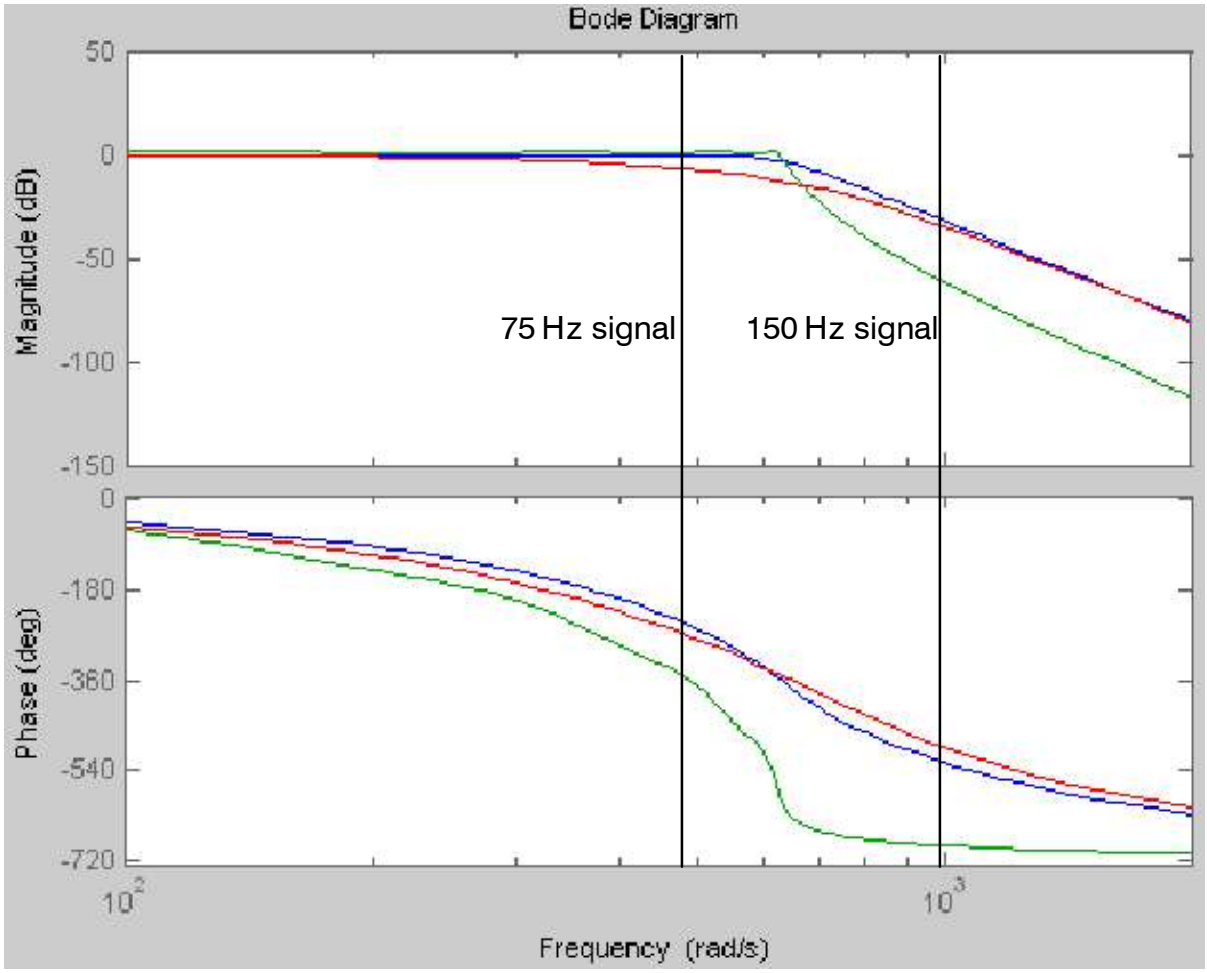


Figure 1.60: Bode diagrams of Butterworth (blue), Chebychev (green) and Bessel (red) filters

The 75Hz signal is in the passband range while the 150Hz signal should be removed by filtering.

```
T_s = 5.0e-4;
t = 0:199;
t = t' * T_s;
x1 = sin(2.0*pi*75.0*t);   x2 = cos(2.0*pi*150.0*t);
x = x1 + x2;
figure(3);
plot(t, [x1 x]);
title('Signal and distorted signal');
```

The figure 1.61 show distorted signal and the original 75Hz sine wave.

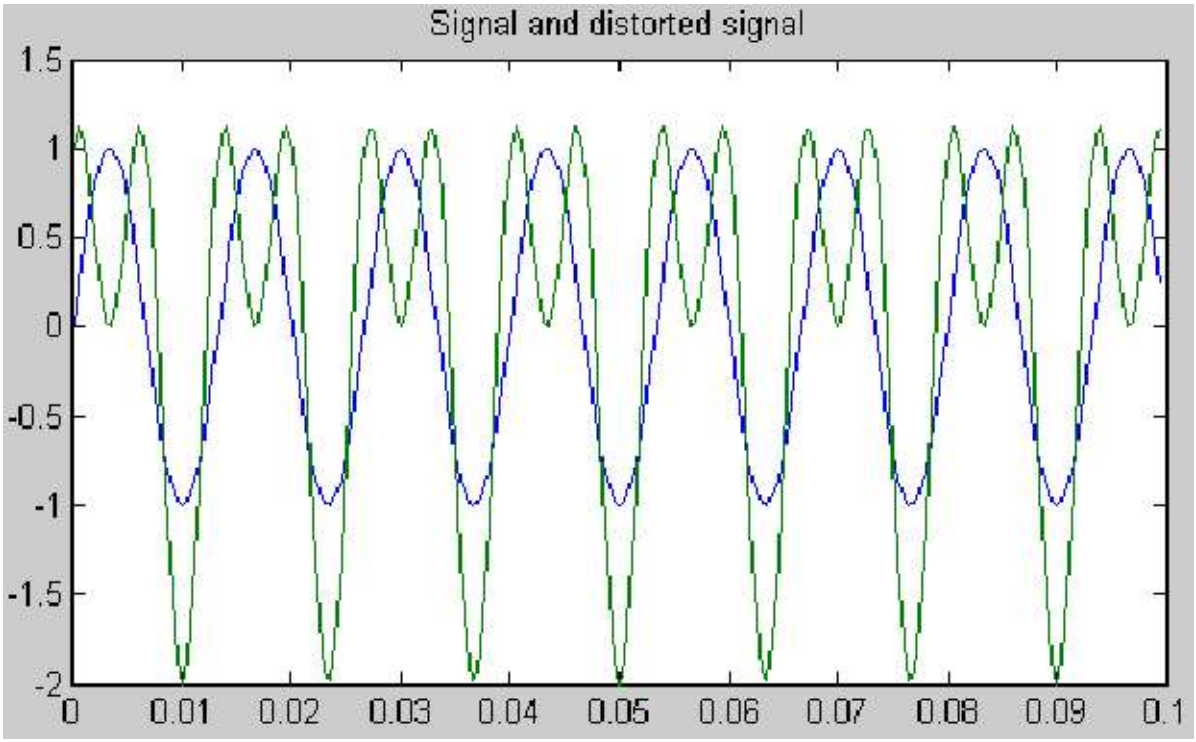


Figure 1.61: Original signal (blue) and distorted (green) signal

```
figure(4);
y_bu = lsim(G_bu, x, t);
y_ch = lsim(G_ch, x, t);
y_be = lsim(G_be, x, t);
plot(t, [y_bu y_ch y_be x1]);
title('Filter results: 75Hz and 150Hz signal');
```

All filters remove the 150Hz signal. The Bessel filter gain is too low, since it the passband gain reduces the 75Hz sine wave. Best performance offers the Chebychev filter. The Butterworth filter gives almost the same result for this application.

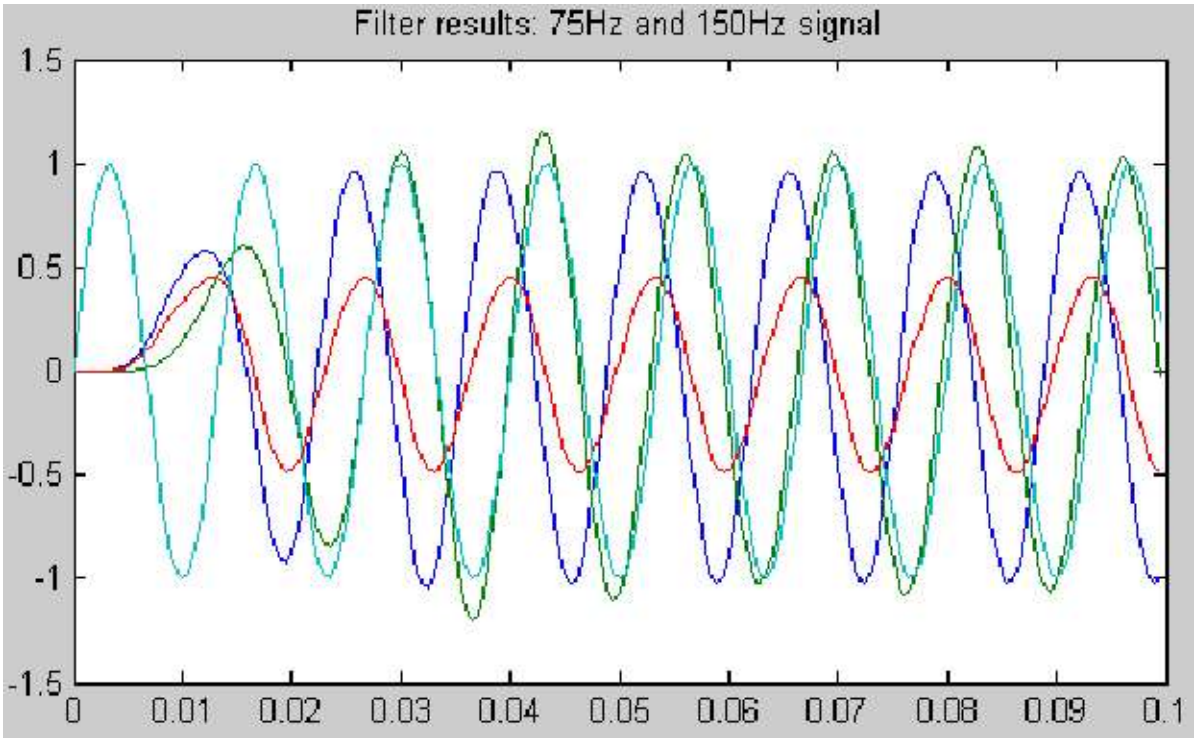


Figure 1.62: Filter results for Butterworth (blue), Chebychev (green) and Bessel (red) filters

The right filter type clearly depend on the application. A lower order Chebychev filter might beat a higher order Bessel filter but the filtered signal might be distorted.

Lab #04

11 ECG Signal Filters and Frequency Tracking

11.1 Matlab DSP

If ECG signals have low frequency noise from a floating ground signal it should be removed by an appropriate high pass filter. Signals below 1 Hz can be safely removed since they are not generated by the human body. The following figure illustrates this

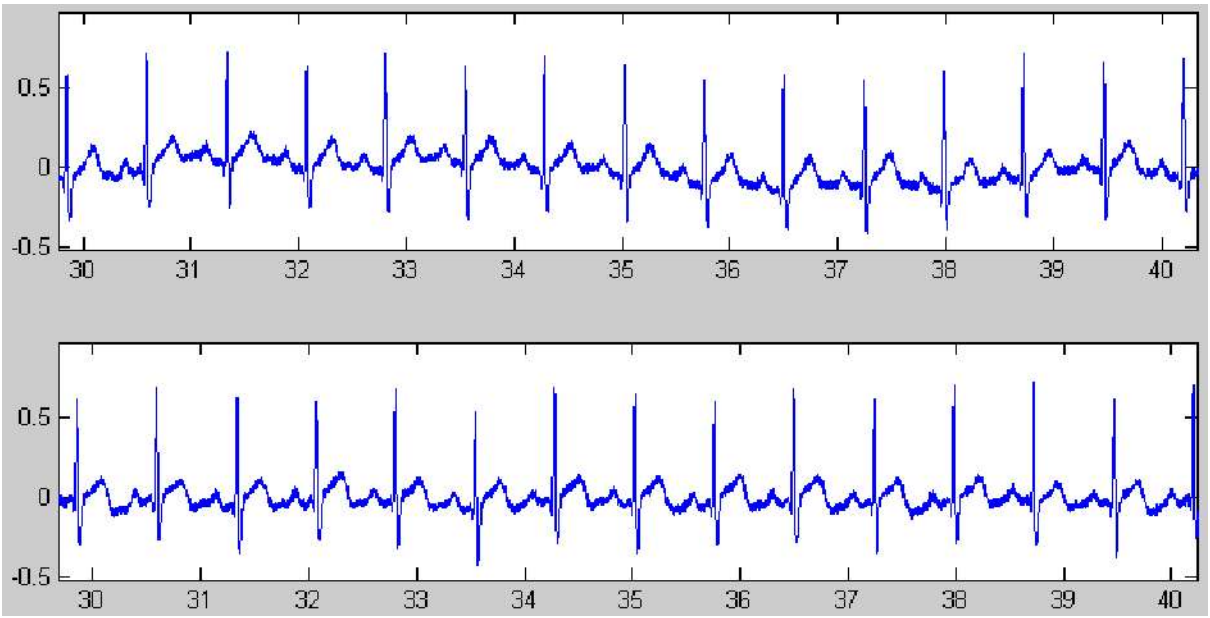


Figure 1.63: Effect of high pass filtering ECG data (low frequency removed from lower diagram)

- Load the 60s "s0020brem60" signal and extract the lead 11 signal.
- Design a. 0.4Hz 4th order Butterworth high pass with Matlab's Butterworth design function `butter`. The filter should be discrete (default). The Sampling period is $T_S = 1\text{ms}$.
- Perform filtering with the filter function `filter`. Verify the above result.

The signal still contains 50Hz noise from the grid. This can be easily removed by a 40Hz Butterworth low pass function (6th order is enough).

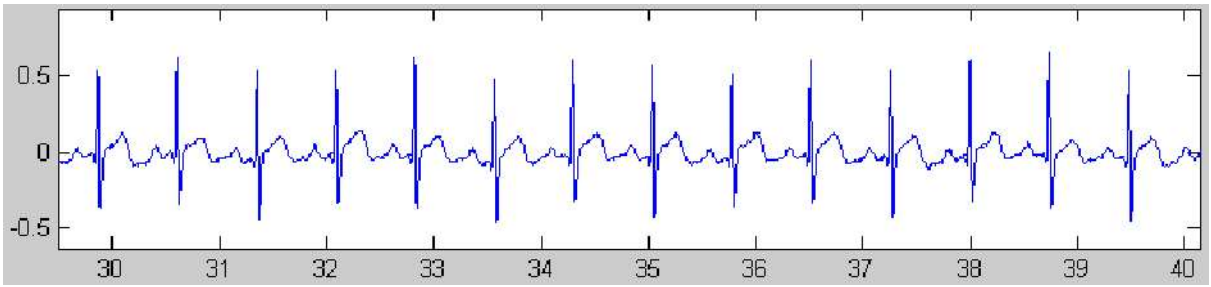


Figure 1.64: ECG data after 40Hz low pass filtering

- Design a 6th order discrete Butterworth low pass filter with 60Hz corner frequency; the sampling period is still $T_S = 1\text{ms}$.
- Perform filtering and plot the result to verify fig. 1.64. The input of the filter should be the high pass filtered signal from fig. 1.63.

11.2 DSP in "C"

The same filters should be applied as C code program. A commonly used form is the Transposed Direct-Form-II which require a minimum of storage (for the past states only).

The computation becomes easier it the transfer function is written in powers of z^{-k}:

$$H(z) = \frac{b_0 + b_1 z^{-1} + \ldots + b_n z^{-n}}{1 + a_1 z^{-1} + \ldots + a_n z^{-n}} . \qquad (1.152)$$

Note that a_0 is 1. If this is not the case numerator an denominator can be divided by a_0. The algorithm for computation follows from the TDF-II block diagram.

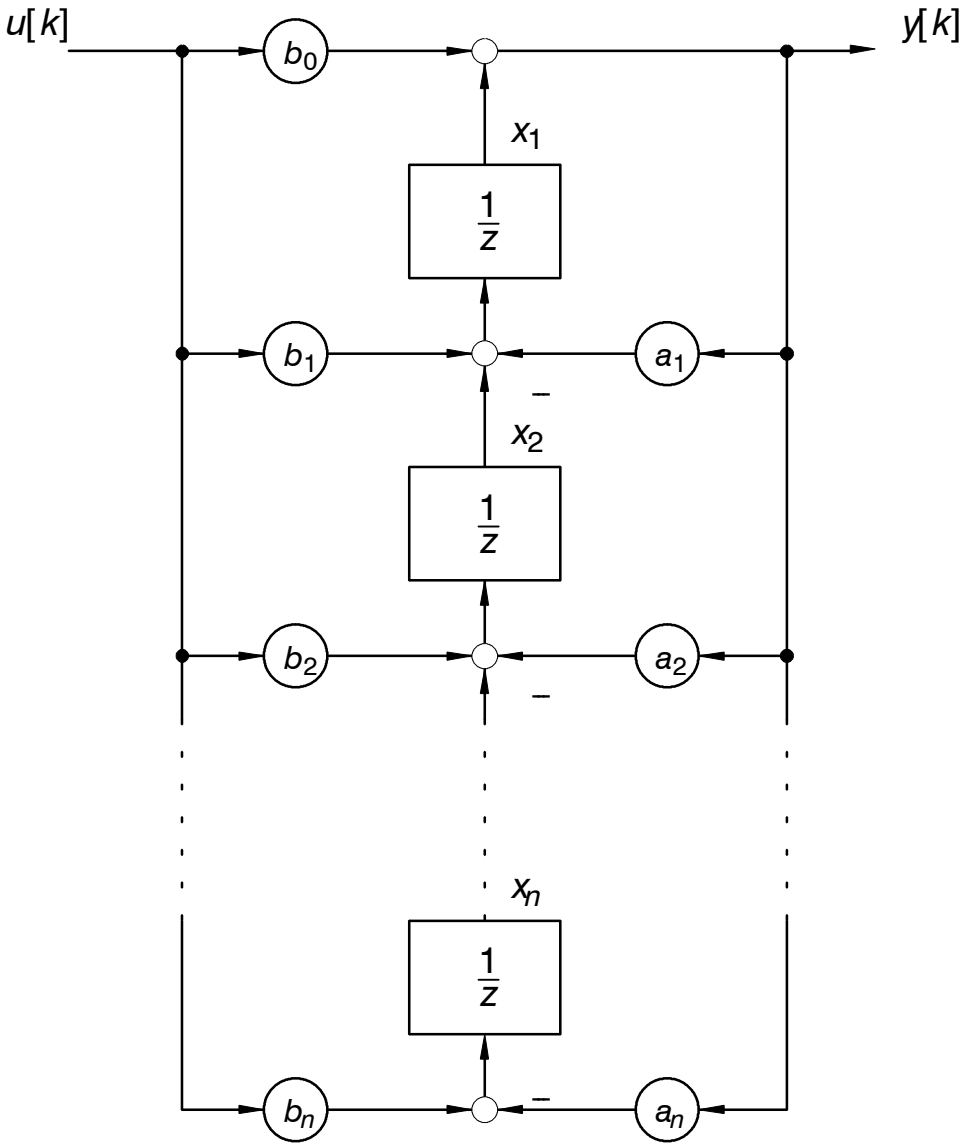


Figure 1.65: Transposed Direct Form II (TDF-II) discrete filter

The n-th order discrete filter has $2 \times (n+1)$ coefficients; a_0 should be made equal to 1 if this is not the case.

- Create the input data file from Matlab with (the variable u must exist in the Matlab workspace)

```
save('u.dat', 'u', '-ascii');
```

- The template file "ecgfilt.c" contains the filter object for order (n), numerator polynomial coefficients (num), denominator coefficients (den) and the state variables (x).

```
#define  IIR_NMAX 10
typedef struct {
        int n;
        double  num[IIR_NMAX];
```

```
        double  den[IIR_NMAX];
        double  x[IIR_NMAX];
} IIR_object;
```

- Take the low pass filter design from Matlab and copy the coefficients to the C code filter init function InitIIR() .

- Complete the filter algorithm in the following function using the TDF-II structure. The function receives the current input $u[k]$ and computes the output $y[k]$.

```
static double IIRdfII(double u)
{
        int k, n;
        double  y;

        n = IIR_Filt.n;
        y = 1.0;

        return y;
}
```

- Verify the result by importing the data to Matlab. You can load the results into Matlab with

```
yc = load('y.dat','-ascii');
```

Lab #04a

11.3 DSP in Synthesizable VHDL (System Generator)

Fast signal processing can be carried out in hardware. This can be easily done by the "System Generator" (see section 15 for detailed information). The DSP can be simulated and fully synthesized on top of Matlab Simulink.

- Create a new and unique folder `lpfilt` .
- Design a 2. order Butterworth low pass filter in Matlab using "butter". The normalized cut-off frequency is (with respect to Nyquist frequency f_{nyq})

  $$W_n = 0.2 \; .$$

  Matlab's discrete function generator for sine and cosine has a "Samples per period" parameter for specifying the frequency. If samples per period is n_SP then the following equation holds:

  $$\frac{T_{signal}}{T_{nyq}} = \frac{f_{nyq}}{f_{signal}} = \frac{n\_SP}{2\,SP} \; . \tag{1.153}$$

  The number of samples per period is then

  $$n\_SP = 2\,SP\,\frac{f_{nyq}}{f_{signal}} \; . \tag{1.154}$$

  For the cut-off frequency $f_c = W_n\, f_{nyq}$ we obtain

  $$n\_SP_c = \frac{2\,SP}{W_n} \; . \tag{1.155}$$

  If W_n is selected to be $W_n = 0.2$ we have $n\_SP_c = 10$. The filter outputs for the Matlab simulation and the hardware (VHDL) simulation are given below. It is almost impossible to recognize any visual difference between double precision computation in Matlab and the fixed-point computation from the System Generator.

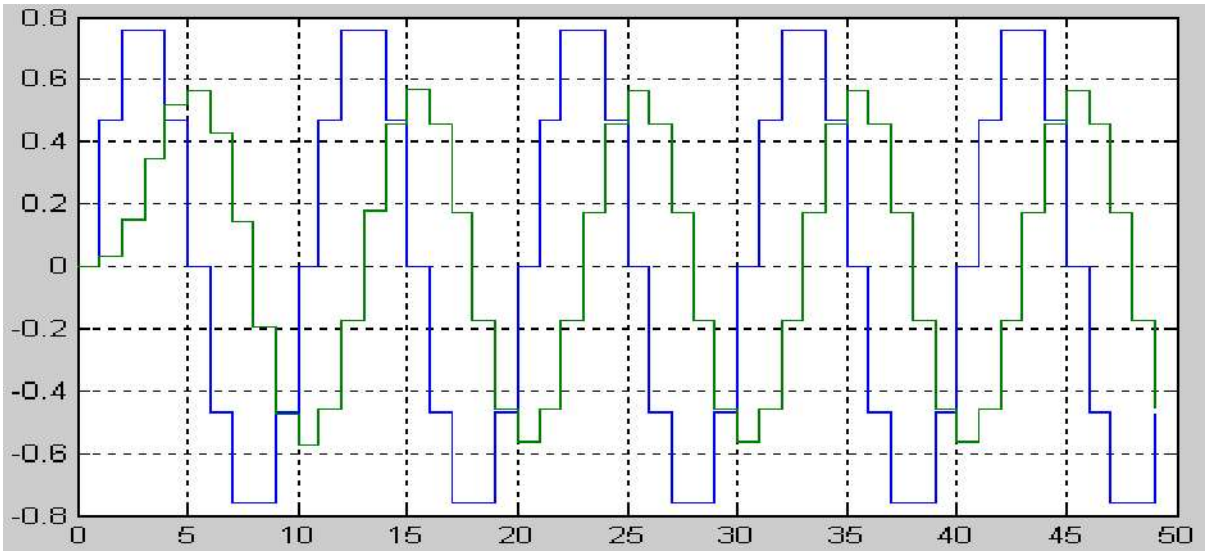


Figure 1.66: Matlab simulation of a $W_n = 0.2$ Butterworth low pass filter

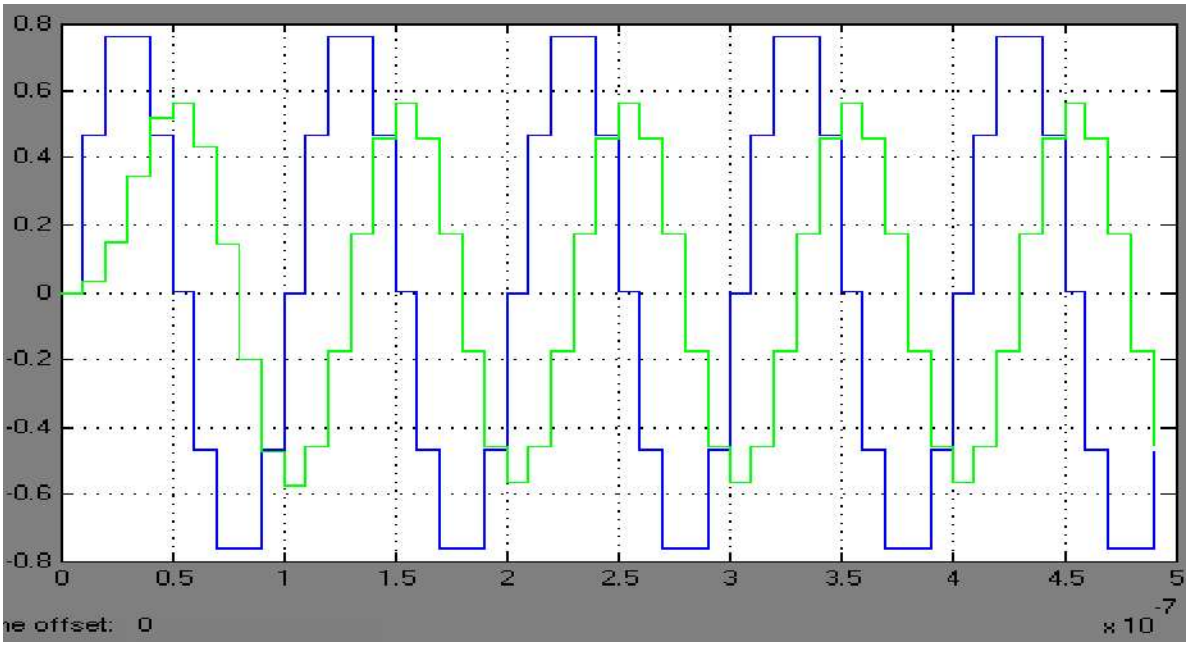
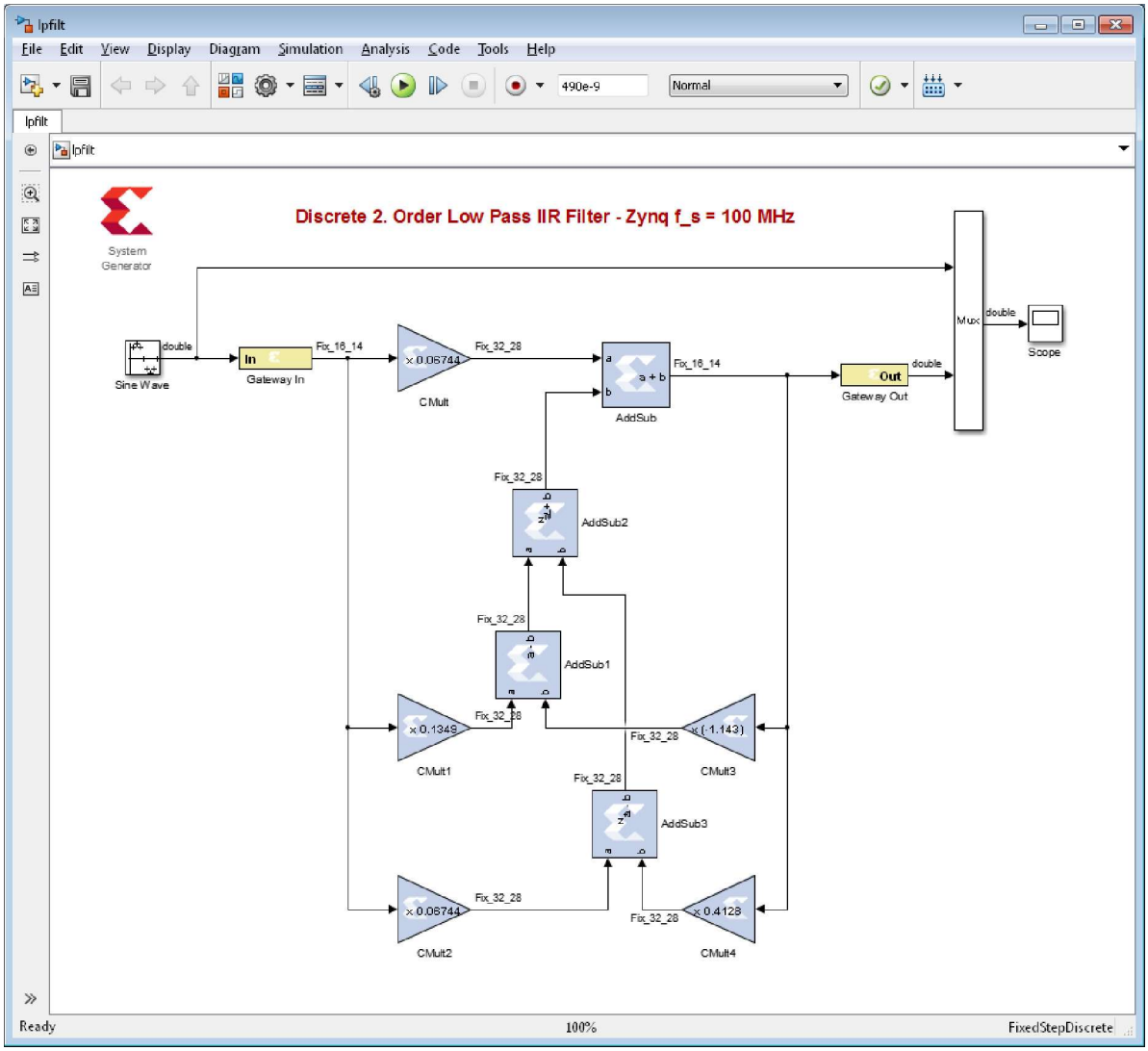


Figure 1.67: System Generator output of the Butterworth low pass filter in fixed-point

- Create a Matlab script file for the 2. order discrete Butterworth low pass and print the numerator and denominator coefficient with 16 fractional digits.
- Simulate the filter and compare the results to fig. 1.66.
- Create a new Simulink model and build a synthesizable system `lpfilt`.
  You can only use the colored blocks from the Xilinx repository.
  The system should use the transposed direct form II (TDF-II) as in fig. 1.65.
  The final system might have a similar structure like this:



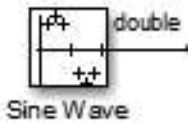
- Blocks you need:

This is a “meta” block, i.e. is contains only control information for the design and allows to generate VHDL code.

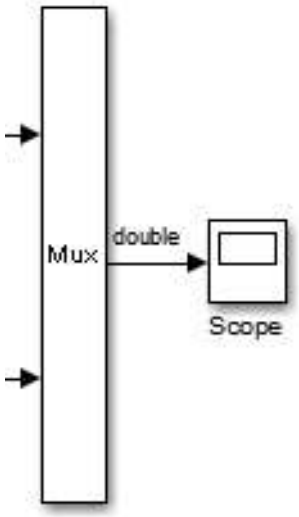
The tabs should be configured as follows:


System Generator: Ipfilt
Compilation
Clocking
General
Compilation :
HDL Netlist
Settings ...
Part :
Zynq xc7z020-1clg484
Hardware description language :
VHDL
VHDL library :
xil_defaultlib
Target directory :
./netlist
Browse...
Synthesis strategy :
Vivado Synthesis Defaults
Implementation strategy :
Vivado Implementation Defaults
Create interface document
Create testbench
Model upgrade...
Performance Tips
Generate
OK
Apply
Cancel
Help

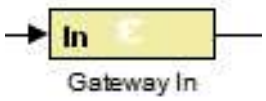
Matlab simulator should be configured as 10 ns step size and as a fully discrete system. **The simulation should stop at 490 ns!**



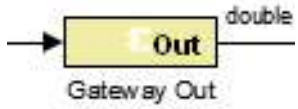
(Matlab) discrete sine wave generator



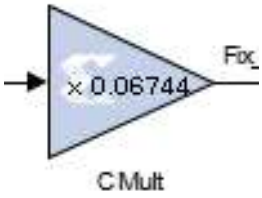
(Matlab) Multiplexer and scope output.



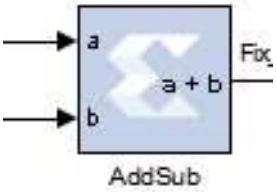
Gateway In: allows to specify physical device pins and acts as a simulation gateway to Simulink.



Gateway Out: allows to specify physical device pins and acts as a simulation gateway to Simulink.



Constant Multiplier, copy & paste coefficients from Matlab script file into configuration.
Select suitable fixed-point formats!



Adder / Subtractor; must be configured to be
an adder or a subtractor

Optional register/ delay element;
1 / z is usually part of the above elements!

- Verify proper operation by simulation under Matlab/Simulink. Do not generate VHDL!

For maximum performance transposed direct form I filter from figure 1.68 can be used. The critical path is short so that all multiply/add operations plus multiplication (MAC) can be executed in parallel.

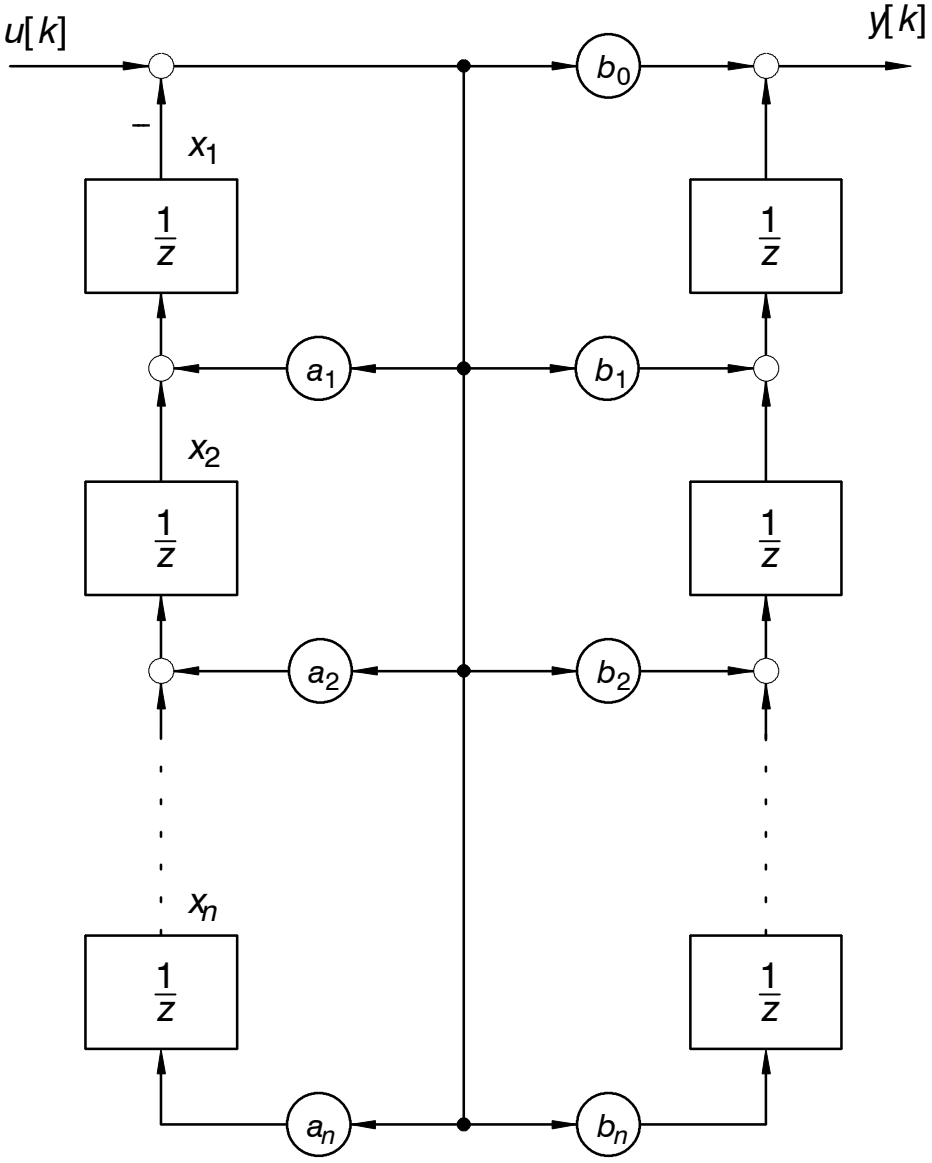


Figure 1.68: Transposed Direct Form I (TDF-I) discrete filter

11.4 Tracking Heart Beat Rate

The ECG data is analyzed in a band with of 1...150Hz. The heart beat rate is defined as the fundamental frequency contained in the measured data. The simplest way to detect the fundamental frequency is counting the samples between two peaks and setting

$$f_{heart} = \frac{1}{n\,T_S} \tag{1.156}$$

where n is the number of samples. This method has some major sources of errors:

- Peaks are not clearly visible in the data. Therefore it is difficult to define a threshold for a peak.
- The signals may contain error-peaks or peaks are missing due to measurement errors.

This problems are avoided if the frequency measurement does not depend on single measurements. A PLL (phase locked loop) is well suited for this purpose. It allows precise and permanent frequency detection and allows to track slow changes in the frequency. In a PLL the phase angle between two signals is detected and a controller changes the frequency of an oscillator so that the phase angle becomes constant. This ensures that the two frequencies are identical.

It is not required that the signal under investigation is pure sinuosity, since the phase detector determines the phase difference between the same frequency as from the oscillator.

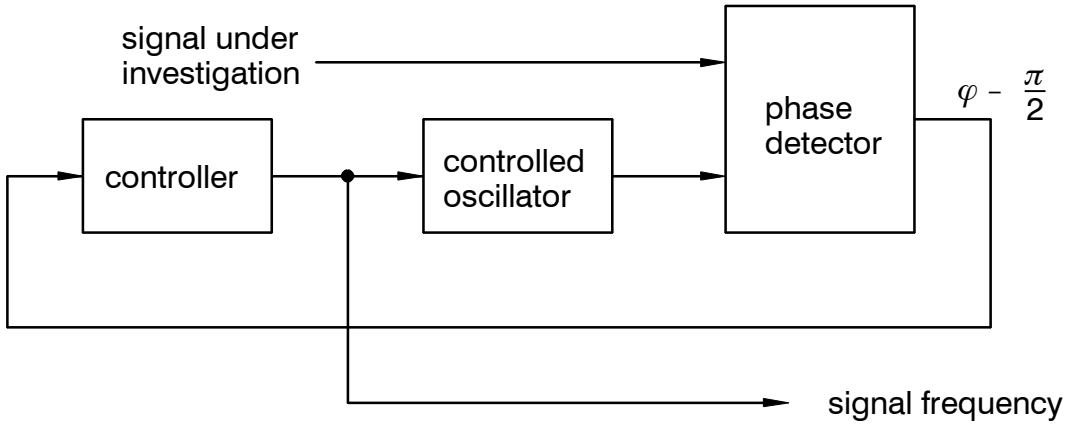


Figure 1.69: Structure of a PLL

The phase detector simply multiplies signals and performs a low pass filtering of the result. The product of two cosine frequencies

$$x_1 = \hat{x}_1 \cos(\omega_1 t) \quad , \quad x_2 = \hat{x}_2 \cos(\omega_2 t + \varphi) \tag{1.157}$$

is

$$x_{pd} = x_1 x_2 = \frac{\hat{x}_1 \hat{x}_2}{2} \cos\big((\omega_1 + \omega_2)t + \varphi\big) + \frac{\hat{x}_1 \hat{x}_2}{2} \cos\big((\omega_1 - \omega_2)t - \varphi\big) . \tag{1.158}$$

The sum of the frequencies can be eliminated by low pass filtering and we obtain

$$x_{pd,lp} = \frac{\hat{x}_1 \hat{x}_2}{2} \cos\big((\omega_1 - \omega_2)t - \varphi\big) = \frac{\hat{x}_1 \hat{x}_2}{2} \cos(\Delta\omega t - \varphi) . \tag{1.159}$$

If $\omega_1 = \omega_2$ or $\Delta\omega = 0$ the output of the phase detector becomes

$$x_{pd,lp} = \frac{\hat{x}_1 \hat{x}_2}{2} \cos\varphi . \tag{1.160}$$

For $\varphi = \pm \frac{\pi}{2}$ the output will be zero. If the controller becomes zero both frequencies are identical and the phase difference will be 90° (negative or positive).

> The initial phase argument in (1.159) must not exceed a certain value, otherwise the PLL will not lock to the correct fundamental frequency.

The block diagram of the phase detector is shown below. It simply multiplies the signals and perform low pass filtering to eliminate the "double frequency".

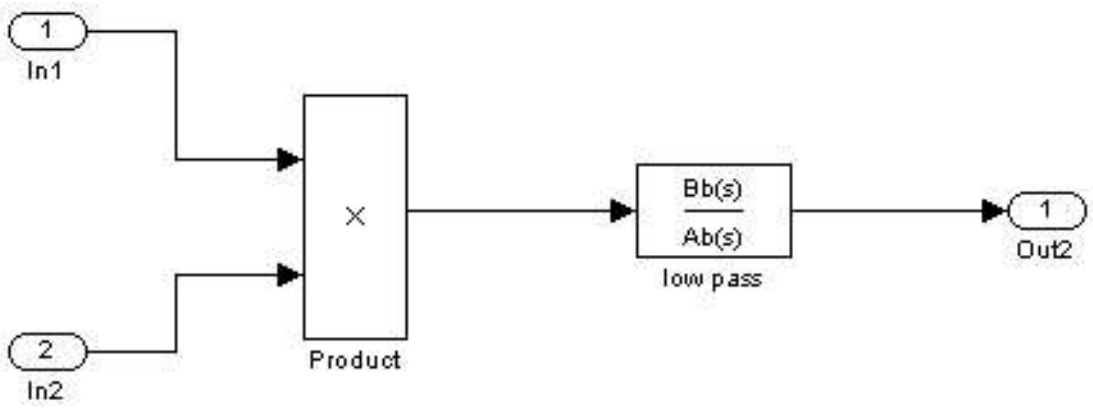


Figure 1.70: Phase detector block diagram

The controlled oscillator is more complicated. In order to simplify the design the continuous version is used. In a practical implementation the discrete version is required.

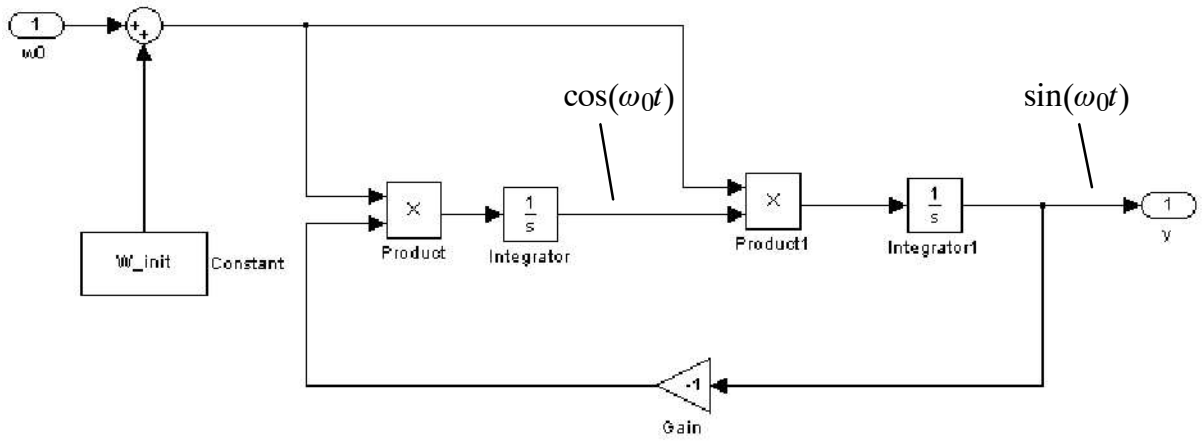


Figure 1.71: Controlled oscillator

This block diagram implements the differential equation for a sine or cosine function

$$\frac{\mathrm{d}^2 y}{\mathrm{d}t^2} = - \omega_0^2 y \,. \tag{1.161}$$

It it obvious that for instance

$$y(t) = \cos(\omega_0 t) \tag{1.162}$$

satisfies (1.161).

The controller in fig. 1.69 is a standard PI controller tuned to the low pass from fig. 1.70 since the low pass determines process dynamics.

The following result shows frequency tracking from 78/60Hz to 84/60Hz.

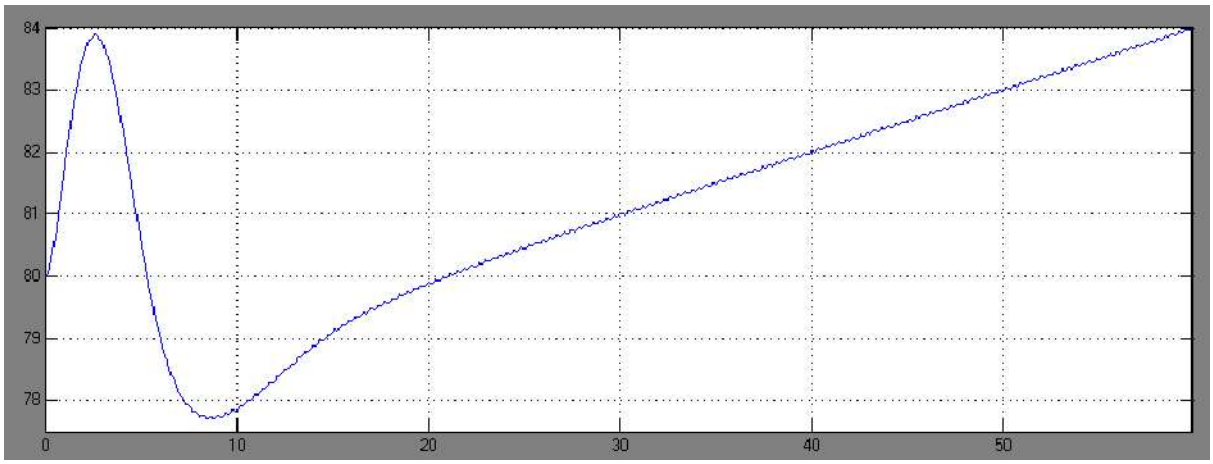


Figure 1.72: Frequency tracking by PLL

It takes approximately 20 seconds to lock into the measured data frequency. After this time the phase is locked (constant) and the frequency measurement give precise results.

12 FIR Filter

A general discrete transfer function of the form

$$G(z) = \frac{b_m z_m + b_{m-1} z^{m-1} \ldots b_1 z + b_0}{z^n + a_{n-1} z^{n-1} \ldots a_1 z + a_0} = \frac{\sum_{\nu=0}^{m} b_\nu z^\nu}{\sum_{\mu=0}^{n} a_\mu z^\mu}, \quad a_n = 1 . \tag{1.163}$$

can be written as a partial sum

$$G(z) = \frac{R_1}{z - z_1} + \frac{R_2}{z - z_2} + \ldots + \frac{R_n}{z - z_n} = \sum_{k=0}^{n} \frac{R_k}{z - z_k} . \tag{1.164}$$

For calculating the impulse response it is therefore sufficient to look at one of the terms in (1.164)

$$G_1 = \frac{R_1}{z - z_1} . \tag{1.165}$$

The corresponding difference equation is

$$y[k] = R_1 u[k - 1] + z_1 y[k - 1] . \tag{1.166}$$

The discrete unit impulse is shown in fig. 1.73.

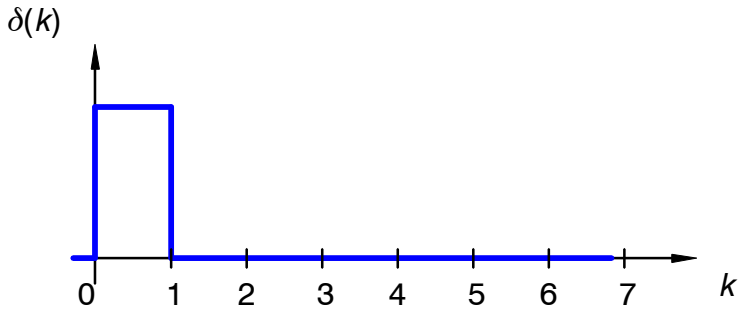


Figure 1.73: Discrete unit impulse $\delta[k]$

The z-transform (1.51) of the unit impulse is simply $\delta(z) = 1$. Therefore the impulse response in z domain is the same as the transfer function. In time domain the impulse response is

$$g[k] = \mathcal{Z}^{-1}\{G_1(z)\} . \tag{1.167}$$

This is not directly possible from tables in literature. The following modification allows back transformation from z to time domain

$$\frac{R_1}{z - z_1} = \frac{R_1}{z_1}\left(\frac{z}{z - z_1} - 1\right) . \tag{1.168}$$

The transformation to time domain becomes

$$g[k] = \frac{R_1}{z_1}\left(z_1^k - \delta[k]\right) = R_1\left(z_1^{k-1} - z_1^{-1}\delta[k]\right) . \tag{1.169}$$

or easier

$$g[k] = R_1 z_1^{k-1} \quad \text{for} \quad k \geq 1 , \quad g[0] = 0 . \tag{1.170}$$

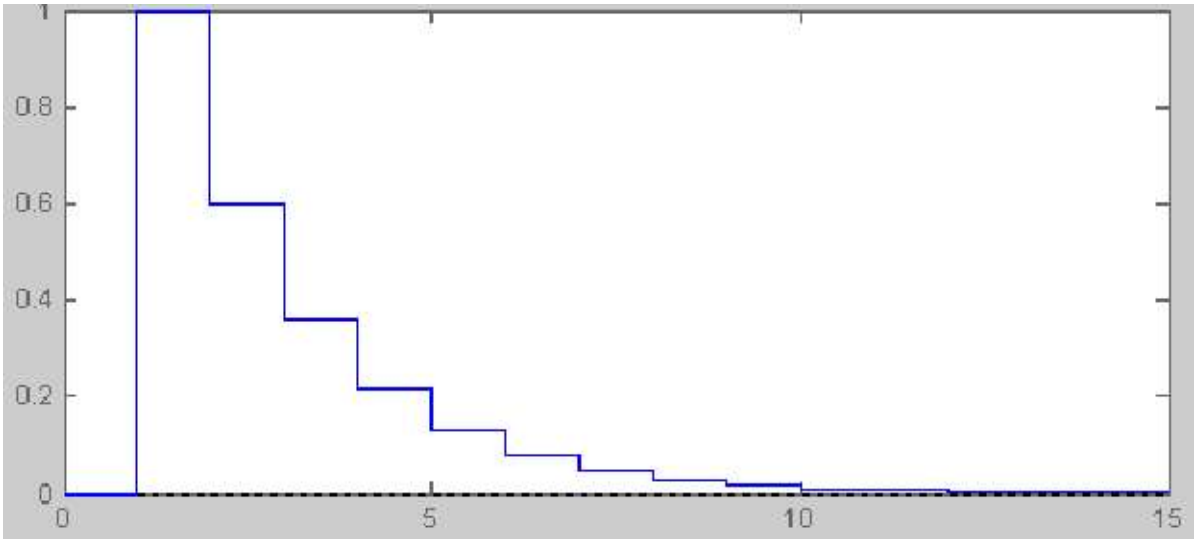


Figure 1.74: Impulse response of $G_1 = \frac{1}{z - 0.6}$

Although the impulse response tends to zero for $k \to \infty$, it never reaches zero exactly. These filters are called IIR filters (**i**nfinite **i**mpulse **r**esponse filter).

> If all poles are located at $z_k = 0$, the filter is called FIR filter (**f**inite **i**mpulse **r**esponse filter).

The filters have the general form

$$G(z) = \frac{b_n z_n + b_{n-1} z^{n-1} \ldots b_1 z + b_0}{z^n} = \frac{\sum_{k=0}^{n} b_k z^k}{z_n} . \qquad (1.171)$$

No equivalent time domain filter exists for this structure. These filters are very useful and are widely used in digital systems. The main benefits are

- powerful filtering capabilities
- filters can be designed to be linear phase
- no stability problems (always stable!)

The drawbacks are

- larger number of coefficients compared to IIR filters
- n sample periods pass until the filter result is available

The last point is critical for using and FIR filter in feedback systems.

As an example we look at the widely used *moving average* filter (all coefficients are identical). If the order is 2^k-1 it's computation requires only add and shift operations. The following filter is a 7th order moving average filter

$$G_{ma} = \frac{1}{8}\frac{z^7 + z^6 + z^5 + z^4 + z^3 + z^2 + z^1 + 1}{z^7} \,. \tag{1.172}$$

The filter computes the average of the current and the past 7 measurements. The result is a lowpass filter with discrete transmission zeros.

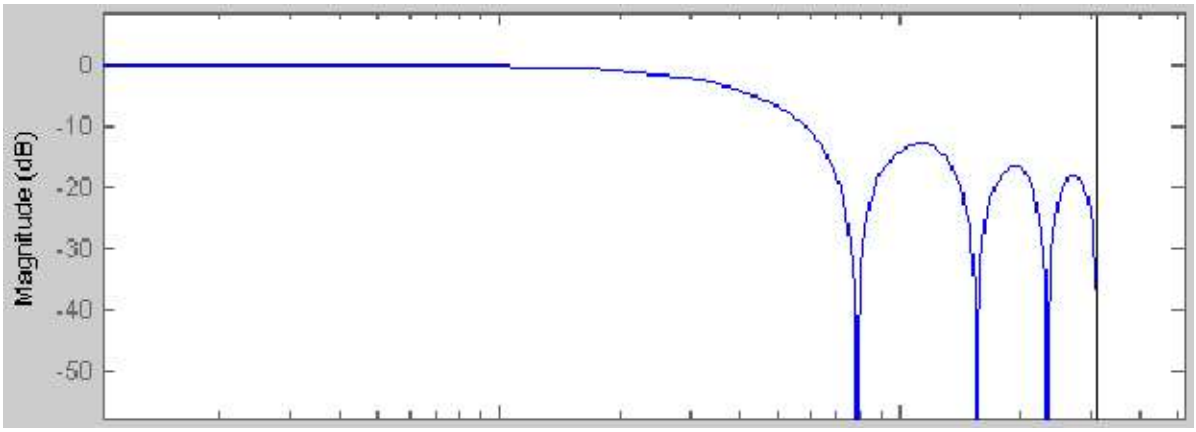


Figure 1.75: Frequency response of the moving average filter

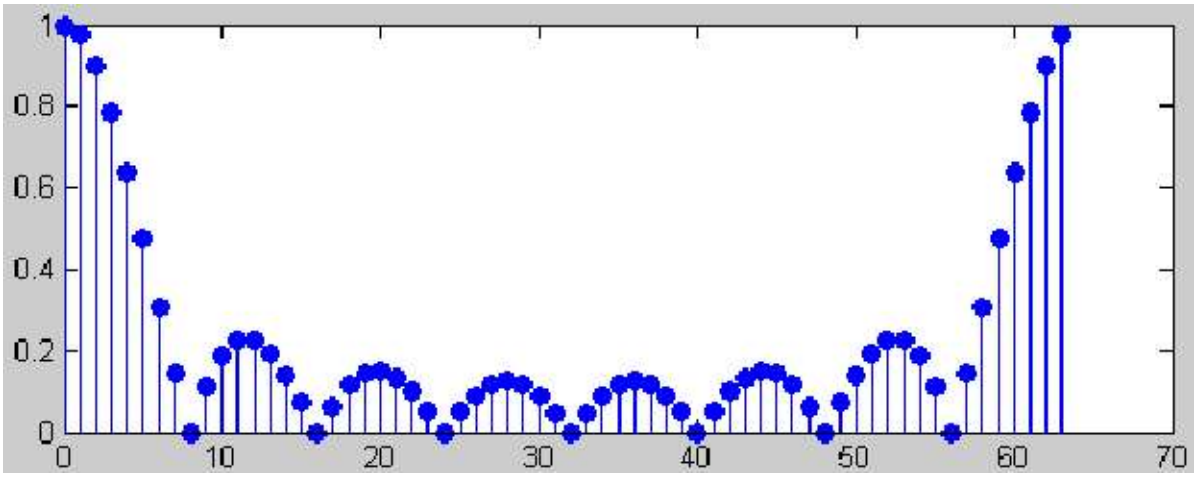


Figure 1.76: 64 point DFT of the impulse response

The DFT exposes the same filter characteristic for the linear discrete frequency axis.

With the sampling period $T_S = 1\text{ms}$ the first transmission zero is located at 125 Hz. The filter removes the frequency by 100% as shown in fig. 1.77. From this figure it is easy to explain why this happens.

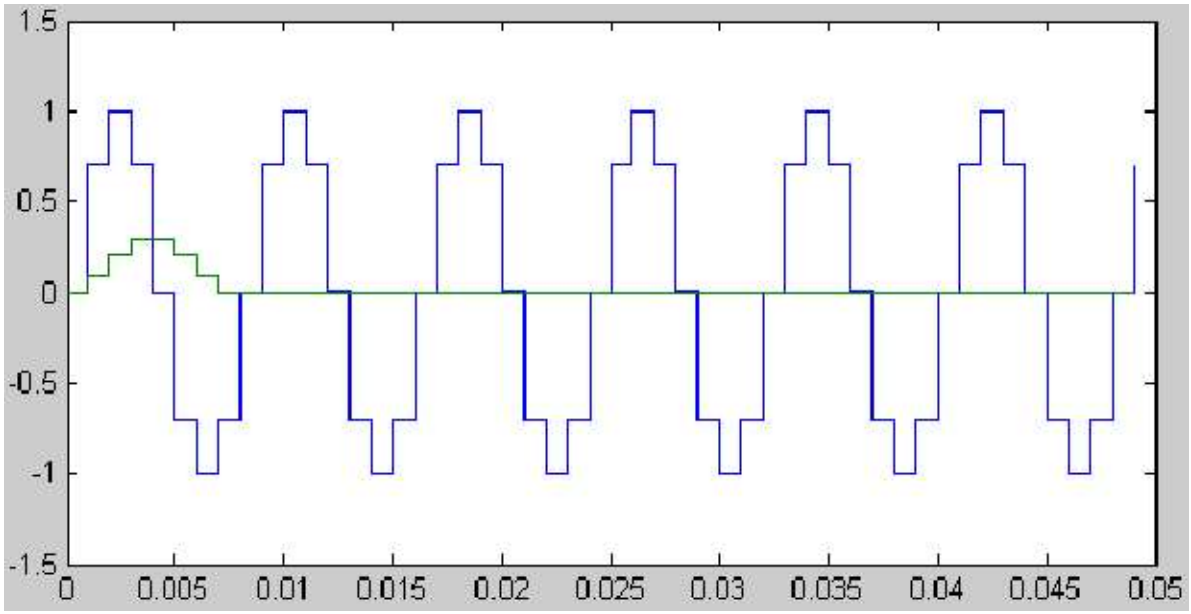


Figure 1.77: Moving average filter with 125Hz sine input (T_S = 1ms)

12.1 FIR Filter Design

FIR filter design is based on the inverse DFT. Although several methods have been published all of these methods share the same base principle. It is an interesting historical fact that one of the best papers (Parks and McClellan, 1971) was first rejected for publication!

The desired frequency characteristic can be specified by designing appropriate DFT frequencies (discrete frequencies). The inverse DFT then gives the impulses response and therefore the filter coefficients (numerator polynomial coefficients). The filter order (called the number of *taps*) depends on the number of DFT points. Of course, a higher order FIR filter shows better performance.

For FIR filter design different time and frequency axes are used. This is possible because signals are assumed to be periodical. The frequency axis can be written in $-N/2+1 \leq m \leq N/2$ instead of $0 \leq m \leq N-1$. This implies negative frequencies. It is the same notation as for the complex Fourier transformation. We can assume a real DFT for synthesis since we are not analyzing measured data. A 16 point frequency response with lowpass characteristic to $f_s/8$ would look as follows.

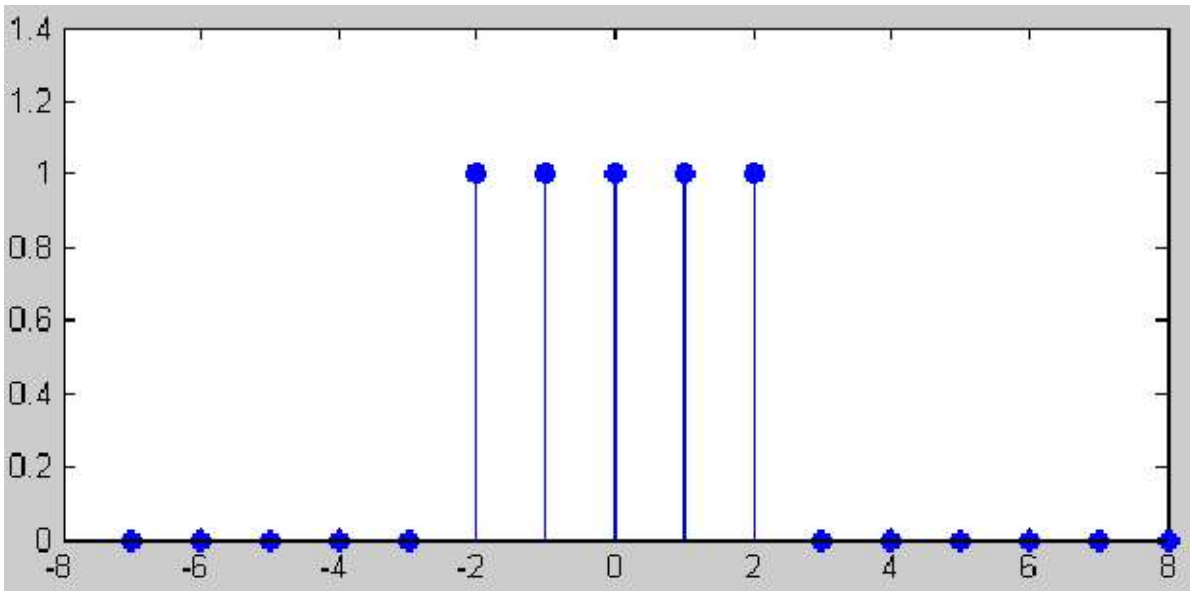


Figure 1.78: Frequency response $H(m)$ of a $f_s/8$ lowpass

The inverse DFT is a si [(sin x) / x)] function. The Fourier transform and the inverse Fourier transform of a rectangular function becomes always an si function. It is therefore no surprise that the inverse DFT has the shape of fig. 1.79.

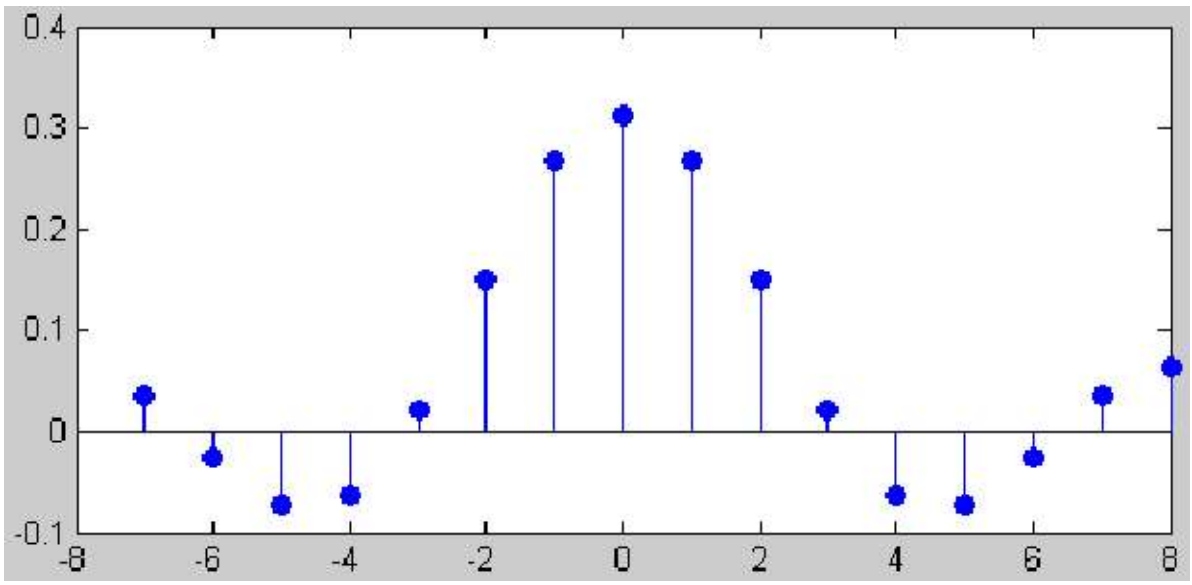


Figure 1.79: Inverse DFT of the frequency response from fig. 1.78

This is the impulse response (finite impulse response) of a filter with the desired frequency response. However, the filter works only exactly for the discrete frequencies of the frequency response. In this special case the frequencies 0, $f_s/16$, $2f_s/16$ pass the filter, all other discrete frequencies are perfectly rejected.

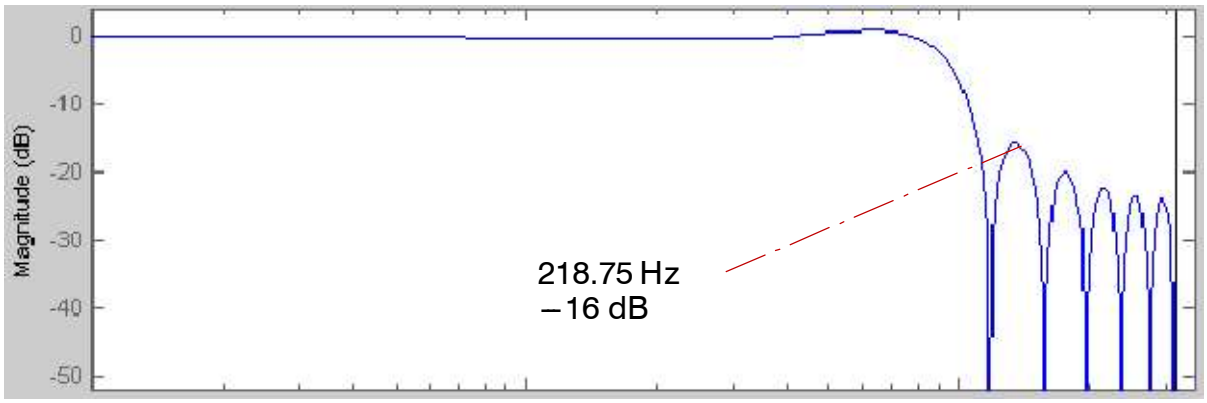


Figure 1.80: Frequency response (all frequencies up to Nyquist frequency)

The filter is only perfect for the discrete frequencies given by $H(m)$. The frequency response for the continuous range of frequencies shows (small) passband ripple and stopband ripple. The discrete frequencies are rejected due to transmission zeros. A frequency between the discrete frequencies (i.e. 218.75 Hz) the rejection is only -16dB.

The properties are demonstrated by filtering 62.5 Hz, 100 Hz and 175 Hz. The first frequency is a discrete passband frequency. In this case a gain of exactly one is guaranteed. The other frequencies are below and beyond of the corner frequency (125 Hz).

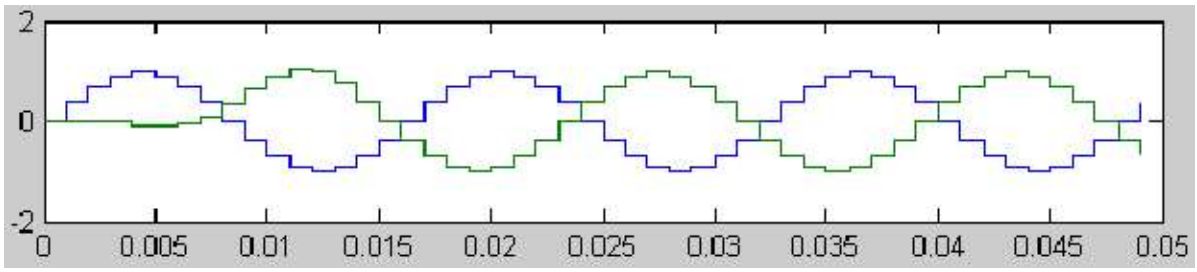


Figure 1.81: 16-tap FIR filter 65.2 Hz signal

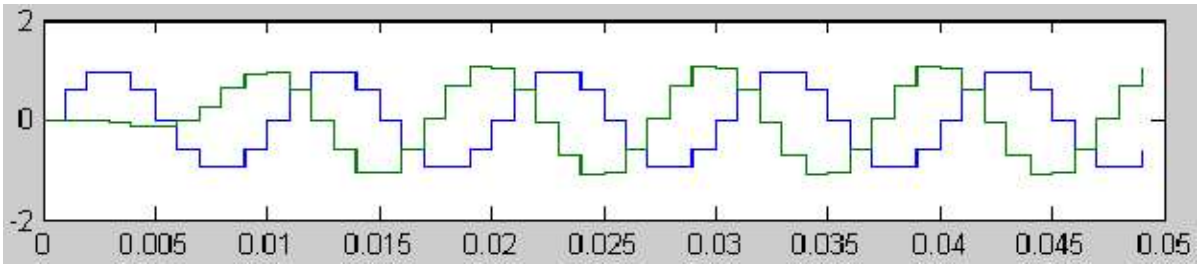


Figure 1.82: 16-tap FIR filter 100 Hz signal

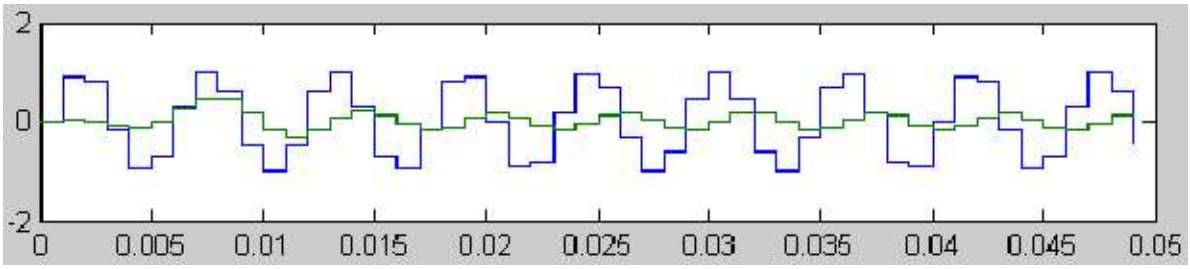


Figure 1.83: 16-tap FIR filter 175 Hz signal

The filter has sufficient passband and stopband properties. The rejection for frequencies not equal to any discrete frequency is not perfect (see fig. 1.83).

Several improvements have been developed to decrease passband ripple and to improve stopband rejection. The main design principle is still the finite impulse response of the inverse DFT.

12.2 Windowing for FIR Design

The ideal rectangular frequency response requires an unlimited sin(x) / x (si) function as impulse response. Since the impulse response must be truncated to a reasonable integer a passband ripple cannot be avoided completely and the stop band is never perfect.

Windowing the impulse response can significantly improve the filter properties. The filter order is not necessarily the full order of the inverse DFT. The following example shows that a reduced order filter still has similar performance. Here a rectangular window for the selection of the impulse response values is used.

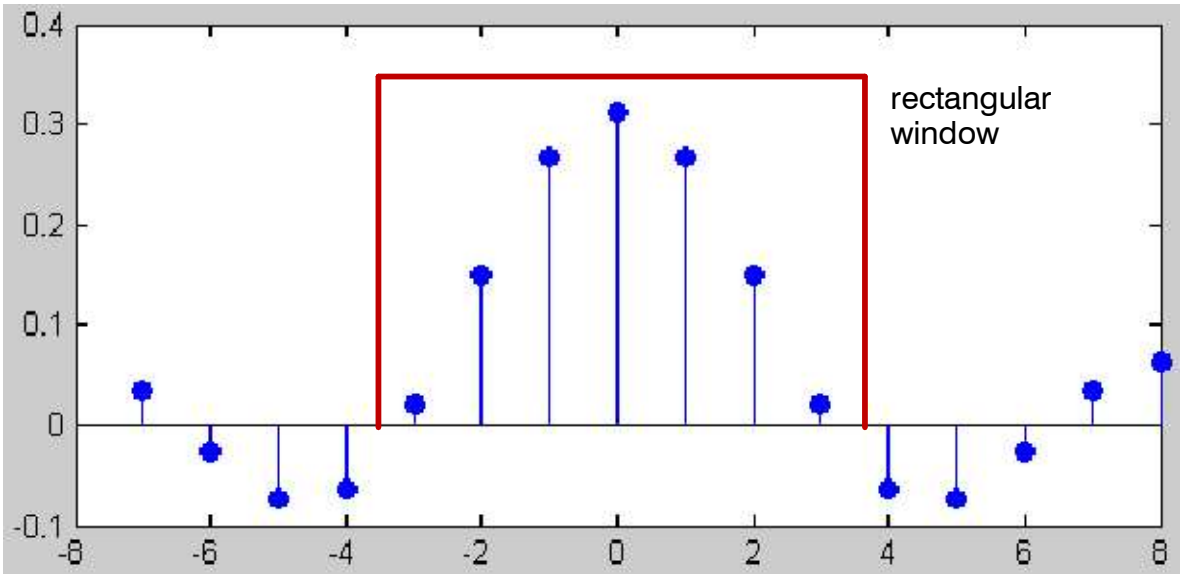


Figure 1.84: Inverse DFT with rectangular window for impulse response

Since $\sin(x)/x$ decreases with increasing magnitude of k, we restrict the filter to the coefficients within the rectangular window. Windowed FIR filters have thus odd numbered orders. We obtain a 6. order FIR filter with 7 coefficients (7-tap FIR filter). As can be seen from the frequency response that FIR filter also works as required.

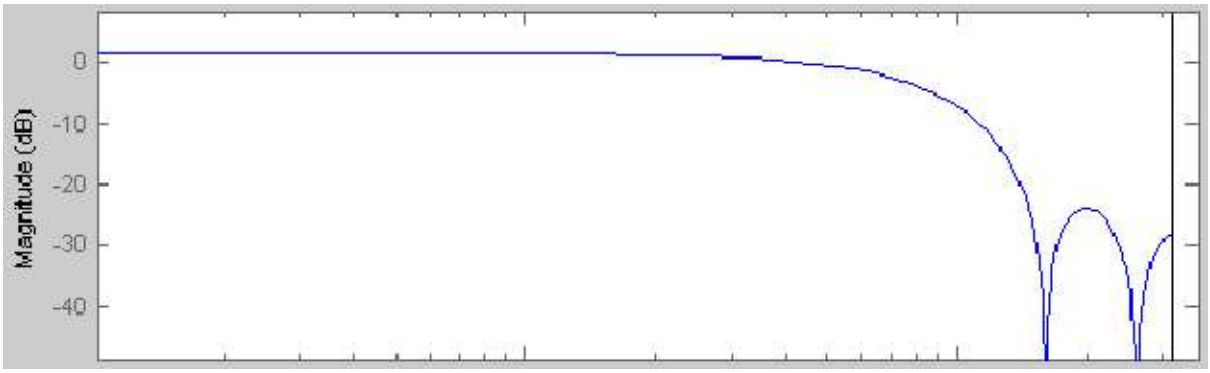


Figure 1.85: Frequency response of the 7-tap FIR lowpass filter

There are only two pairs of complex transmission zeros but the overall stopband performance has not degraded. The change from passband to stopband is not as steep as with the 16-tap FIR filter, the price for a lower order filter.

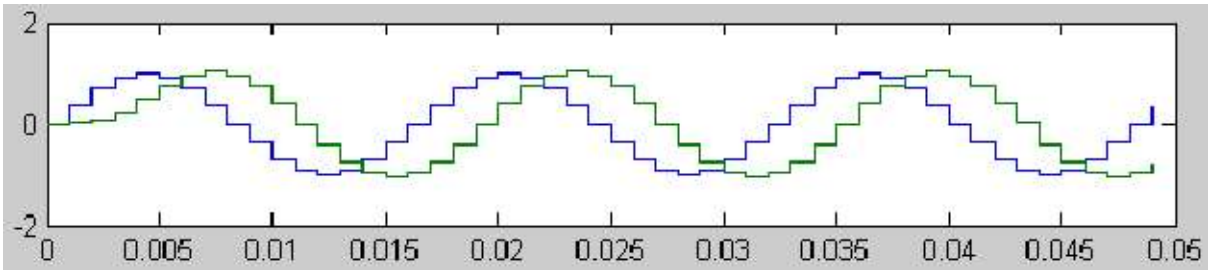


Figure 1.86: 7-tap FIR filter 65.2 Hz signal

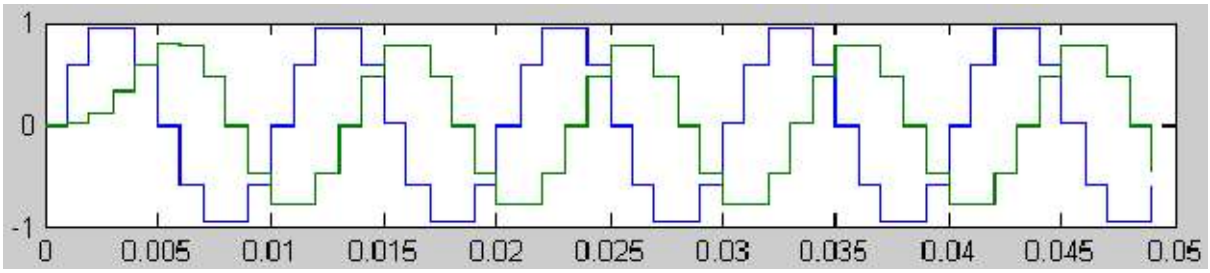


Figure 1.87: 7-tap FIR filter 100 Hz signal

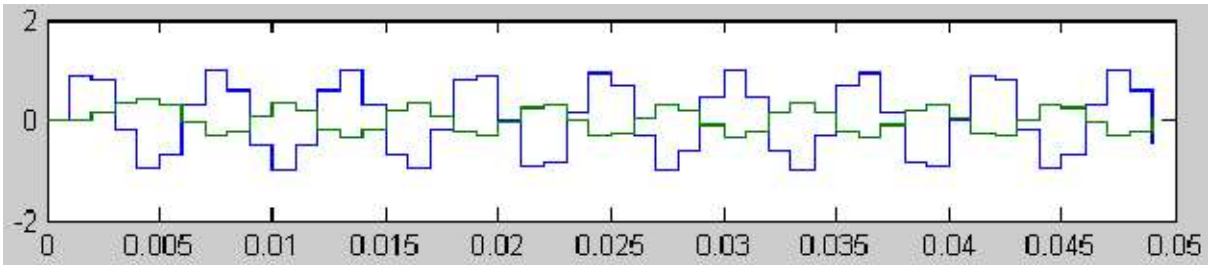


Figure 1.88: 7-tap FIR filter 175 Hz signal

Lab #05

13 Design of a Notch FIR Filter

A notch filter deletes a small band of frequencies from a signal. With 1 KHz sampling frequency f_s the frequency 62.5 Hz should blocked. All other frequencies should pass the filter. The FIR filter should have 16 coefficients, which calls for a 15th order FIR.

- Start with $H_m(m) = 1$ for a filter to pass all frequencies.

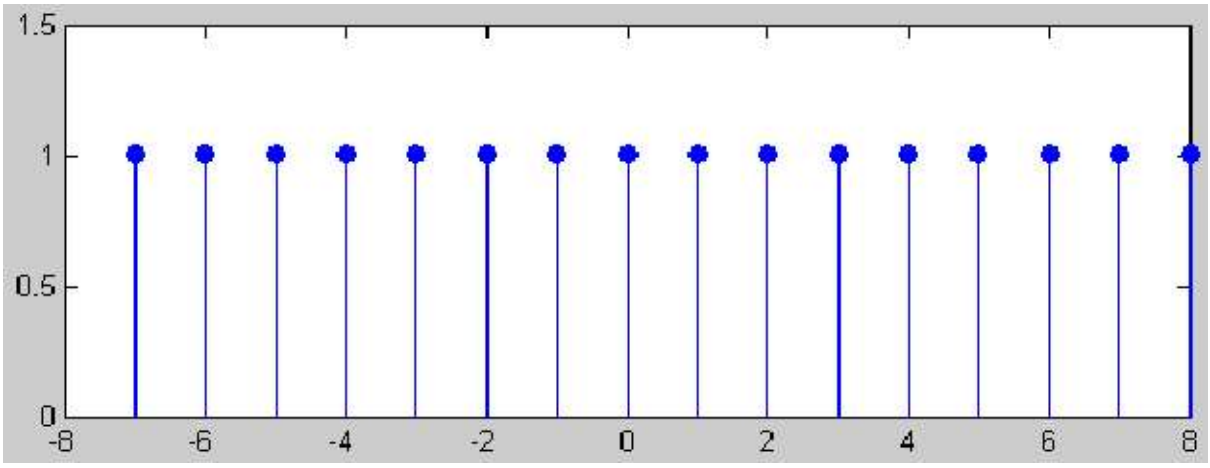


Figure 1.89: Flat frequency response

- Clear the frequency 1 / 16 * 1 KHz = 62.5 Hz

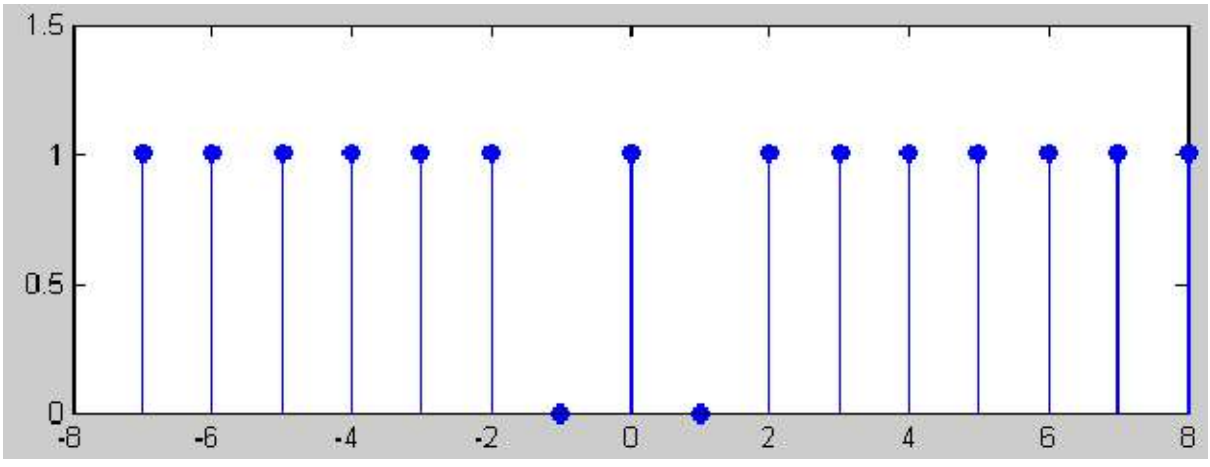


Figure 1.90: Frequency 62.5 Hz ($m = \pm 1$) blocked

- Compute the (shifted) inverse DFT. You can use the convenience file ifft_shift.m for this purpose:

```
function hk = ifft_shift(Hm)

N = length(Hm);
N2 = N / 2;
xx = ifft([Hm(N2:N); Hm(1:N2-1)]);
hk = [xx(N2+2:N); xx(1:N2+1)];
```

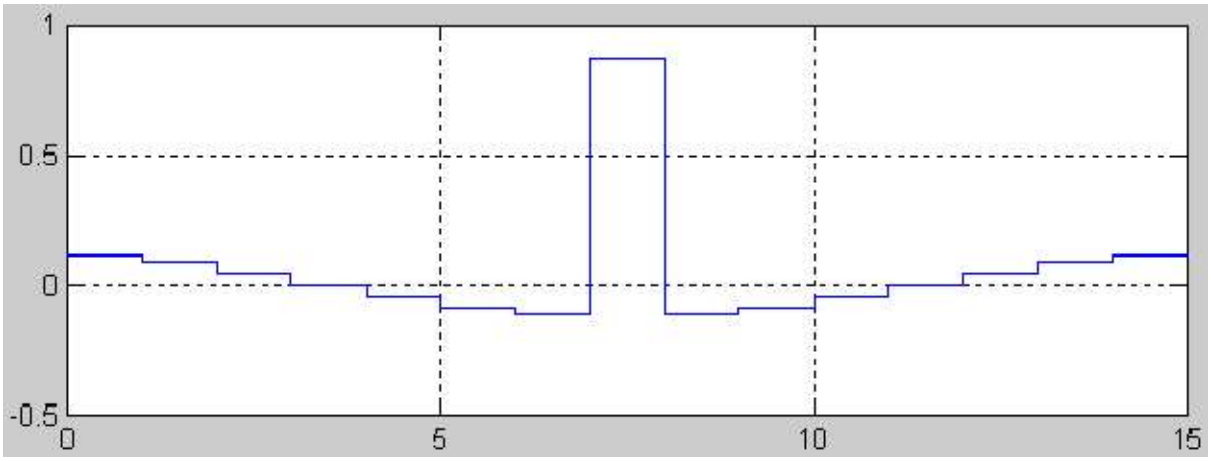


Figure 1.91: Inverse DFT (real!) of frequency response from fig. 1.90 = impulse response

- Verify that the frequency response meets the requirements.

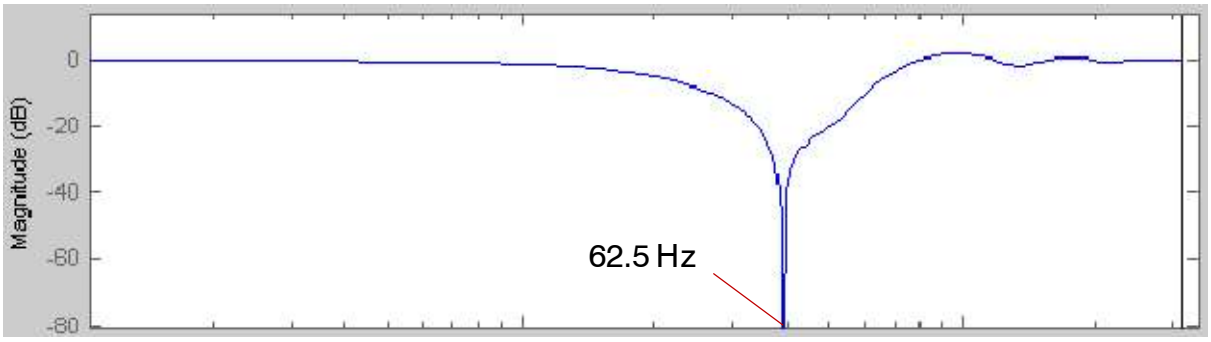


Figure 1.92: Continuous frequency response

- Compute filter response to the frequencies 15.625 Hz, 62.5 Hz, 125.0 Hz. Does the filter work as expected?

13.1 FIR Filter Optimization

An ideal lowpass filter with a rectangular Frequency response as in fig. 1.78 has an infinite impulse response of type $\sin(x)/x$. By using a rectangular window to select the filter coefficients the "side lobes" of $\sin(x)/x$ are no longer part of the FIR filter. This is responsible for the passband ripples. Only for the discrete frequences from the (DFT) frequency response this filter does not show passband ripples.

By using other than rectangular windows for the impulse response the passband and the stopband ripples can be reduced significantly. Not optimal general purpose window function exists, instead the suitable window depends on the application. In addition, the popular window functions have parameters to select. This makes FIR design not a as easy as it may first seem. The popular window function are:

- **Rectangular** window: the window function that we already used
- **Blackman** window: widely used window to reduce ripples
- **Chebychev** window: better cut-off frequency response with more ripples the Blackman. Mathematically more challenging as Blackman
- **Kaiser** window (also called Kaiser-Bessel): sharp cut-off properties (better than Chebychev) but more ripples in the stopband region.

13.1.1 Blackman Windowing

The Blackman window is defined as

$$w(k) = 0.42 - 0.5\cos\left(\frac{2\pi k}{N-1}\right) + 0.08\cos\left(\frac{4\pi k}{N-1}\right) \tag{1.173}$$

for

$$k = 0, 1, 2, \ldots, N-1 . \tag{1.174}$$

The constants 0.42, 0.5 and 0.08 are not mathematically optimized. It turned out that these coefficients give satisfactory results. These coefficients can be changed, but the sum should be 1 so that $w(N/2) = 1$ holds. For $N = 31$ the Blackman window is shown in fig. 1.93.

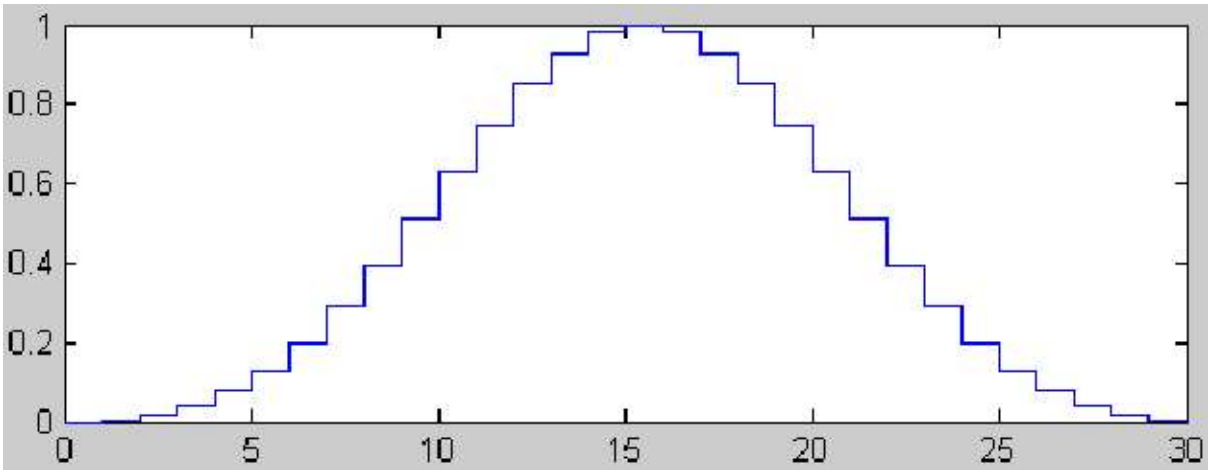


Figure 1.93: Blackman window for $N = 31$

The ripple reduction due to the Blackman window are demonstrated by the lowpass filter of section 12.1. The following figure shows the inverse DFT and the corresponding Blackman window coefficients.

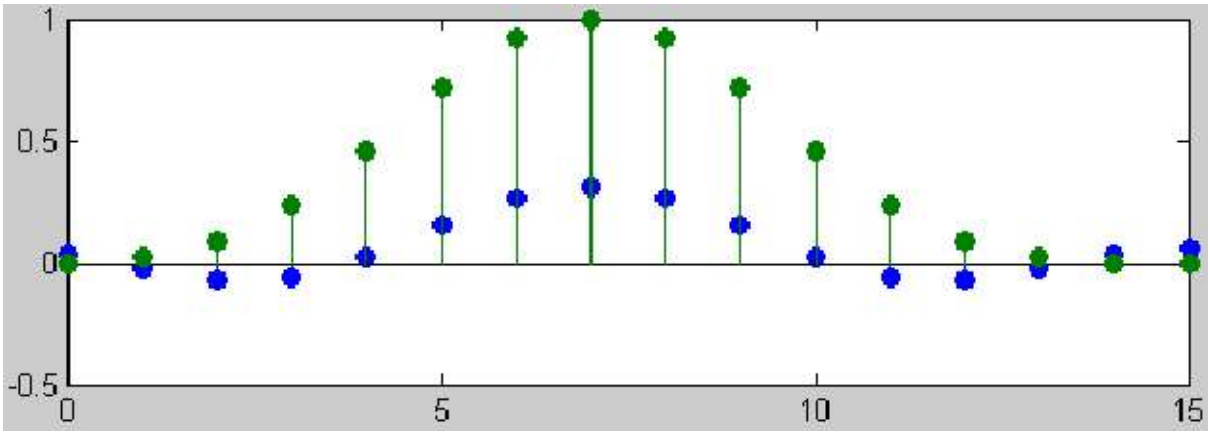


Figure 1.94: Inverse DFT (blue) and Blackman window coefficients (green)

The product of both functions are the numerator coefficients of the FIR filter. Note that the first and the two last coefficients are zero. This reduces the required order of the FIR filter.

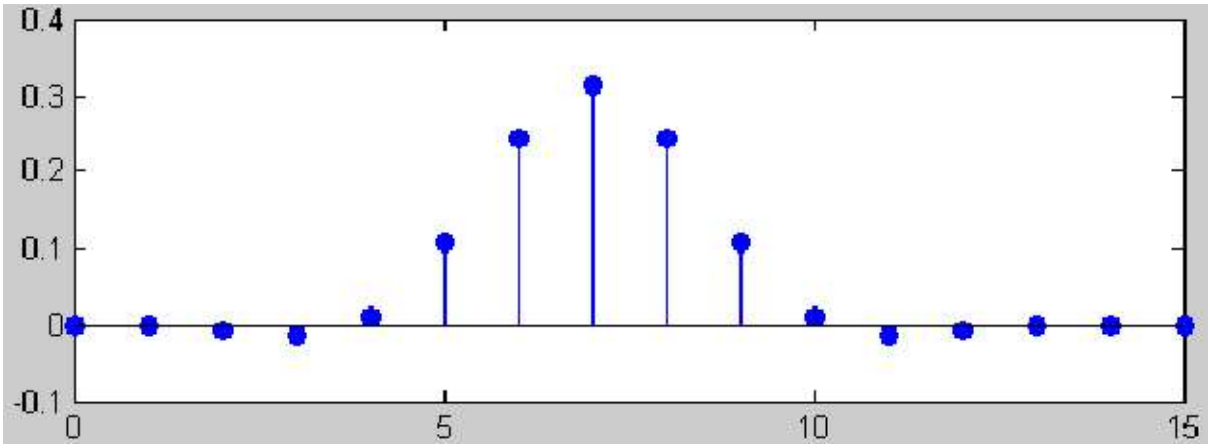


Figure 1.95: Product of inverse DFT and Blackman window

The frequency responses for logarithmic and linear scales are shown in the next figures. The passband and stopband ripples are reduced for the Blackman windowed FIR filter.

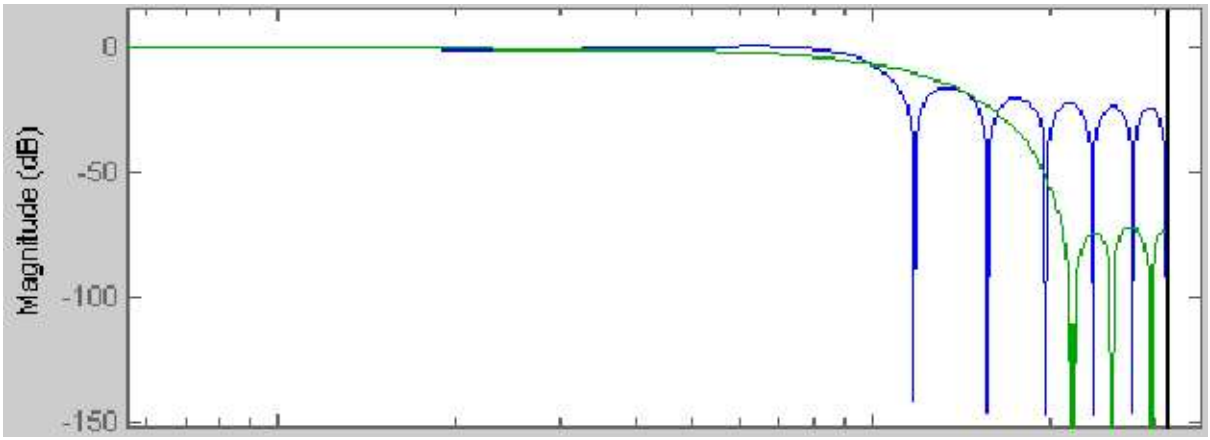


Figure 1.96: Frequency response (logarithmic scales) of rectangular and Blackman FIR

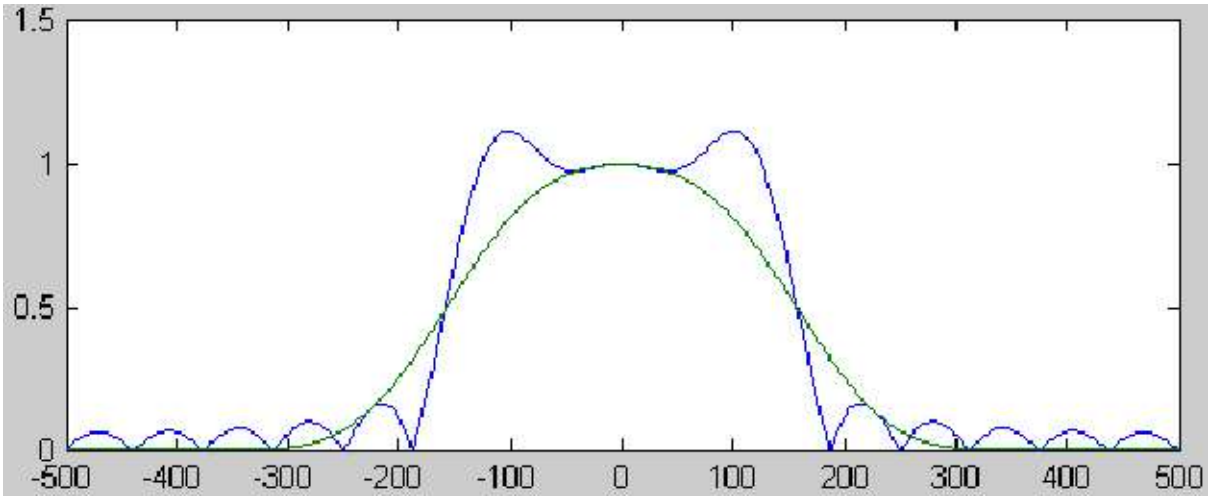


Figure 1.97: Frequency response (linear scales) of rectangular and Blackman FIR

13.1.2 Kaiser (-Bessel) Windowing

A very flexible but more complex method is the Kaiser windowing. It has a free parameter β which shapes the Kaiser window. This window is defined as

$$w(k) = \frac{I_0\left[\beta\sqrt{1-\left(\frac{k-p}{p}\right)^2}\right]}{I_0(\beta)}, \quad k = 1,2,...,N-1, \quad p = \frac{N-1}{2}, \qquad (1.175)$$

where $I_0(x)$ is the zeroth order Bessel function

$$I_0(x) = \sum_{q=0}^{\infty} \frac{x^{2q}}{4^q(q!)^2}. \qquad (1.176)$$

For numerical computation only the first 25 terms are used [6], so (1.176) becomes

$$I_0(x) \approx \sum_{q=0}^{24} \frac{x^{2q}}{4^q(q!)^2}. \qquad (1.177)$$

For $\beta = 3$ we obtain the window (green) in fig. 1.98.

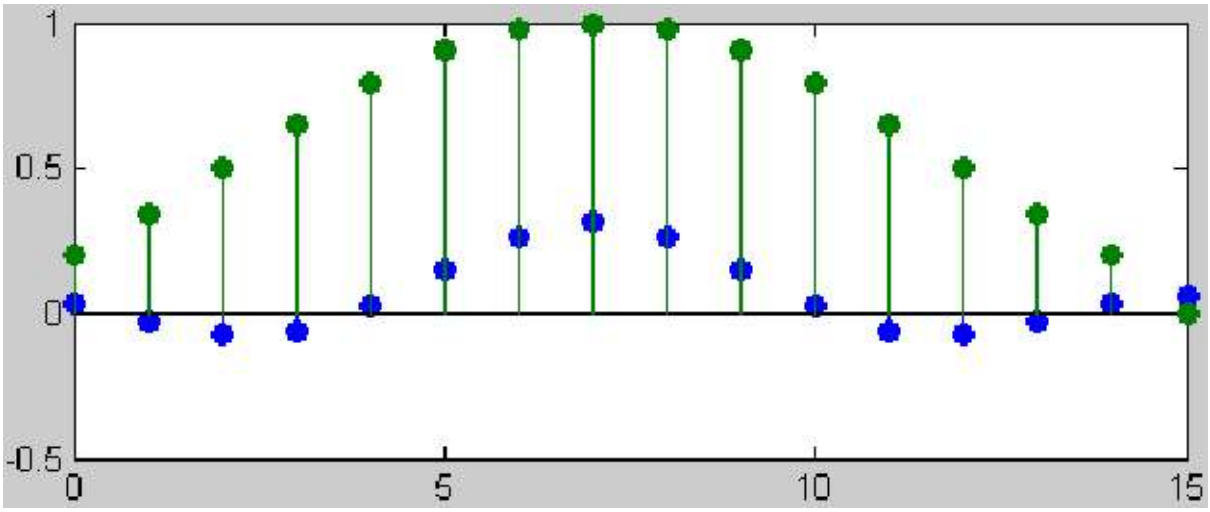


Figure 1.98: Kaiser window ($\beta = 3$) and impulse response

For this particular β quite satisfactory filtering results are obtained.

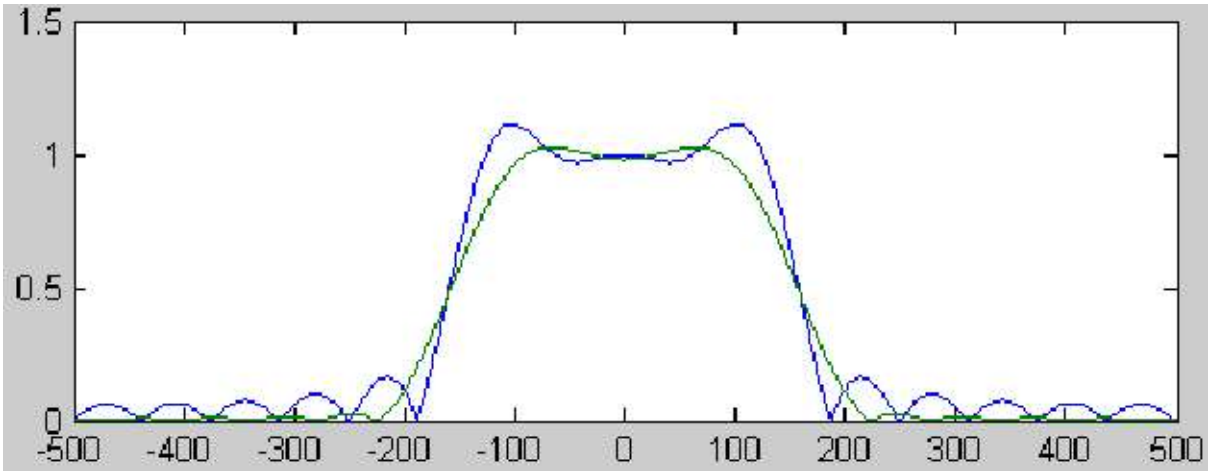


Figure 1.99: Frequency response (linear scales) of rectangular and Kaiser FIR

The results for filtering frequencies of 100 Hz and 200 Hz are shown below.

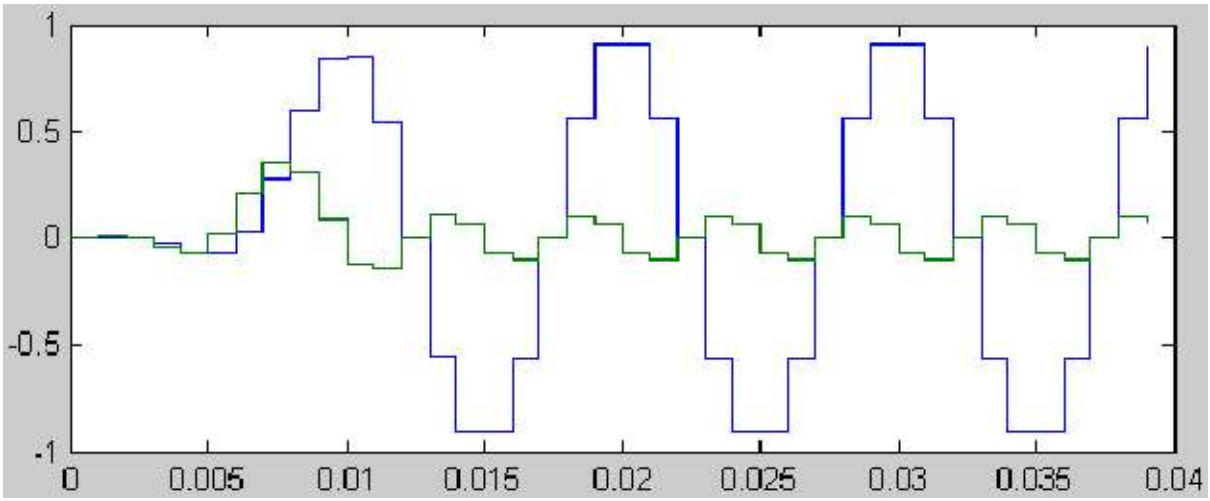


Figure 1.100: Filter outputs of 15-tap Kaiser windowed FIR filter (100Hz and 200Hz)

14 Bandpass and Highpass Filter Design

The design bandpass and highpass FIR filters can start from lowpass filters which are shifted in frequency. Since the multiplication of two sinusoidal signals give the sum and the difference of the two frequencies, the impulse response (i.e. the filter coefficients) can be multiplied with the required *bandpass center frequency*.

For a highpass filter this frequency is $f_s / 2$. If the shift frequency is a multiple of f_s / N several filter coefficients become zero. This is often exploited to speed filter computation. If the center frequency does not match these discrete frequencies all coefficients are required.

The frequency shift multiplier is given by (for even numbers of N)

$$s_{shift}(k) = \cos\left(2\pi f_{shift}\left(k - \frac{N}{2} + 1\right)T_S\right), \quad 0 \le k \le N - 1 \tag{1.178}$$

or (for odd N)

$$s_{shift}(k) = \cos\left(2\pi f_{shift}\left(k - \frac{N}{2}\right)T_S\right), \quad 0 \le k \le N - 1\,. \tag{1.179}$$

The shift frequency f_{shift} can be also expressed by the relative frequency index n_{shift}:

$$f_{shift} = \frac{n_{shift}}{N} f_s = \frac{n_{shift}}{N} T_S\,. \tag{1.180}$$

In this case (1.179) becomes

$$s_{shift}(k) = \cos\left(2\pi \frac{n_{shift}}{N}\left(k - \frac{N}{2}\right)\right). \tag{1.181}$$

The index n_{shift} ist not restricted to integer values. For $n_{shift} = N/2$ a lowpass filter will be converted to a high pass filter (see 14.2).

14.1 Bandpass Filter Example

We consider a lowpass filter which will be converted to a bandpass for $N = 32$. The frequency response of the low pass is shown in fig. 1.101.

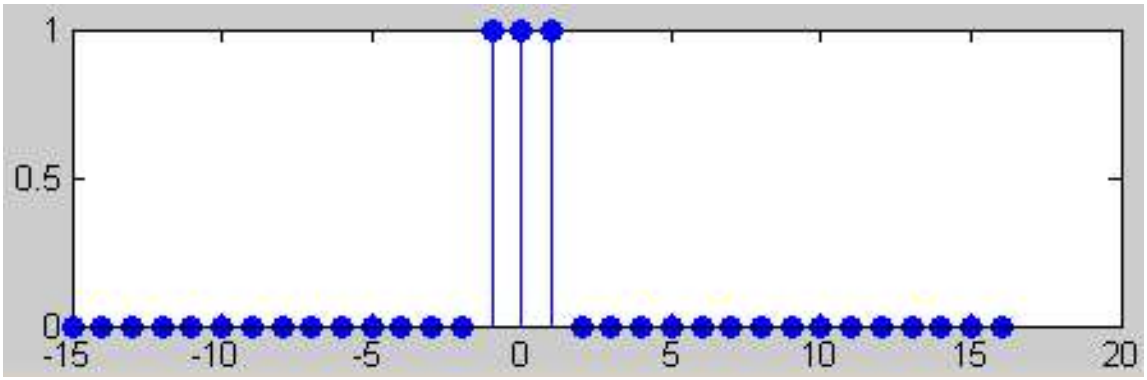


Figure 1.101: Lowpass discrete frequency response

The linear axis frequency response was not optimized with respect to passband and stopband ripples but it is valid at least for the discrete frequencies.

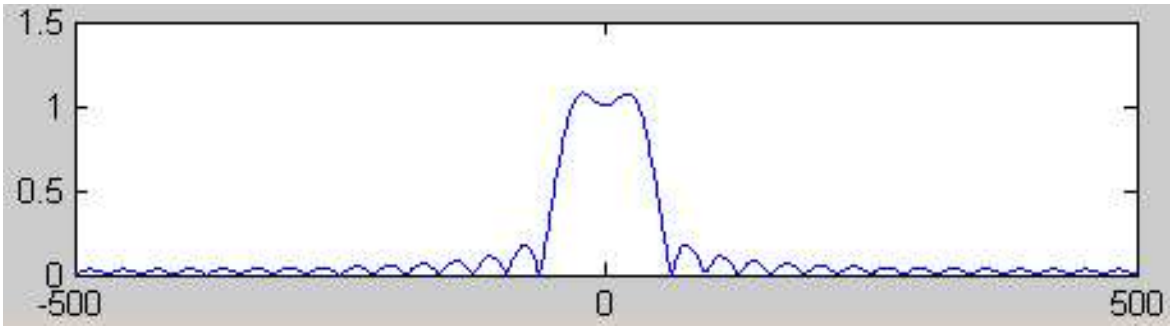


Figure 1.102: Linear axis FIR frequency response ($T_S = 1$ ms)

The transformation to bandpass is carried out by multiplying the impulse response by s_{shift} from (1.178).

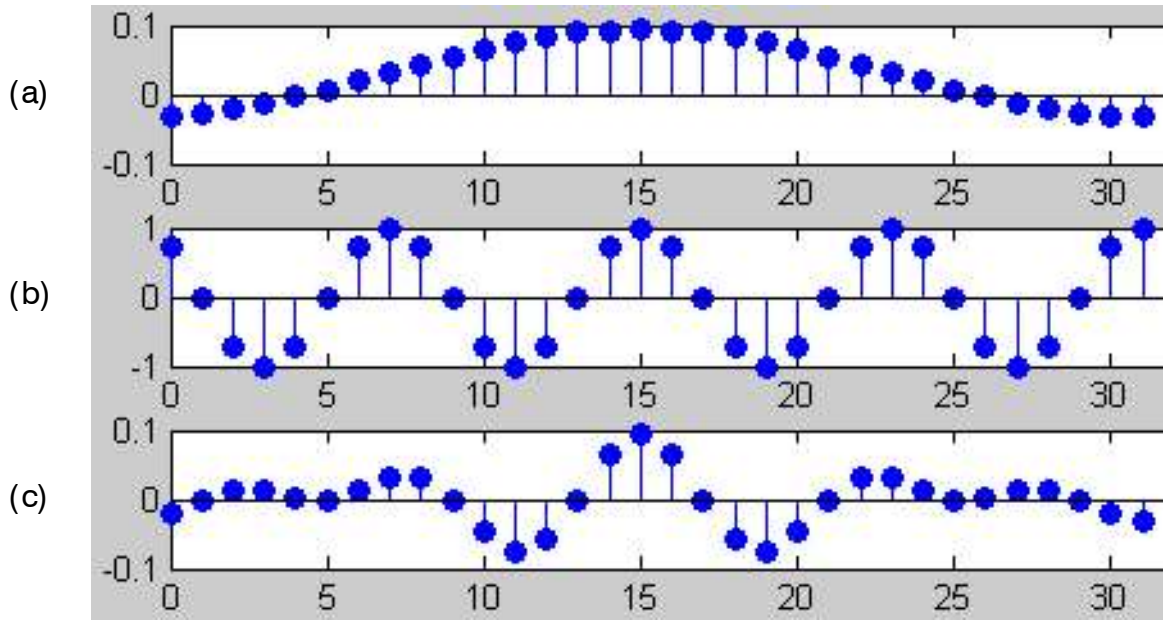


Figure 1.103: (a) impulse response (lowpass),
(b) s_{shift} coefficients by frequency shift of $4*f_s/N$,
(c) bandpass impulse response after multiplication

Since some of the coefficients in fig. 1.103 (b) are less than one in magnitude the frequency response of the FIR in fig. 1.103 (c) is not 0 dB for the frequency $4*f_s/N$.

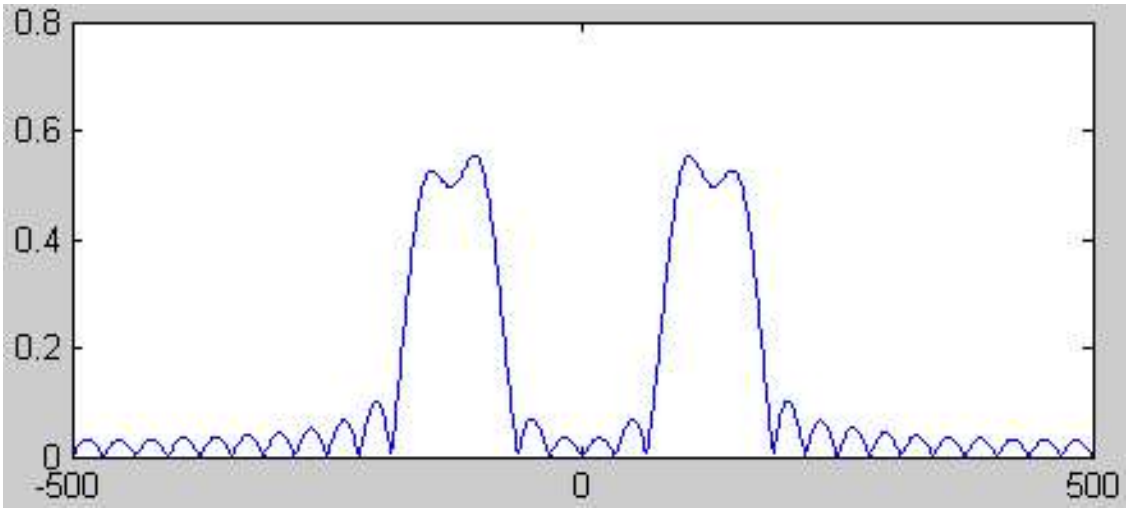


Figure 1.104: Frequency response of the bandpass filter ($4*f_s/N = 125$Hz center frequency)

The gain at $4*f_s/N$ is reduced by 0.5. This can be normalized back to one y a factor of 2.

14.2 Highpass Filter Example

The highpass filter is a special case of the bandpass. In this case the shift frequency becomes $f_{shift} = f_s / 2$. The coefficients of (1.178) are given by

$$s_{shift}(k) = \cos\left(\pi\left(k - \frac{N}{2} + 1\right)\right), \quad 0 \le k \le N - 1, \tag{1.182}$$

since $f_{shift}\, T_S = f_s/2\, T_S = 1/2$. If N is odd the sequence becomes

$$s_{shift}(k) = \cos\left(\pi\left(k - \frac{N}{2}\right)\right), \quad 0 \le k \le N - 1. \tag{1.183}$$

This is a sequence of -1, +1, -1, +1, ..., i.e. every second coefficient needs to be inverted. No gain adjustment is required since the magnitude of all coefficients is one.

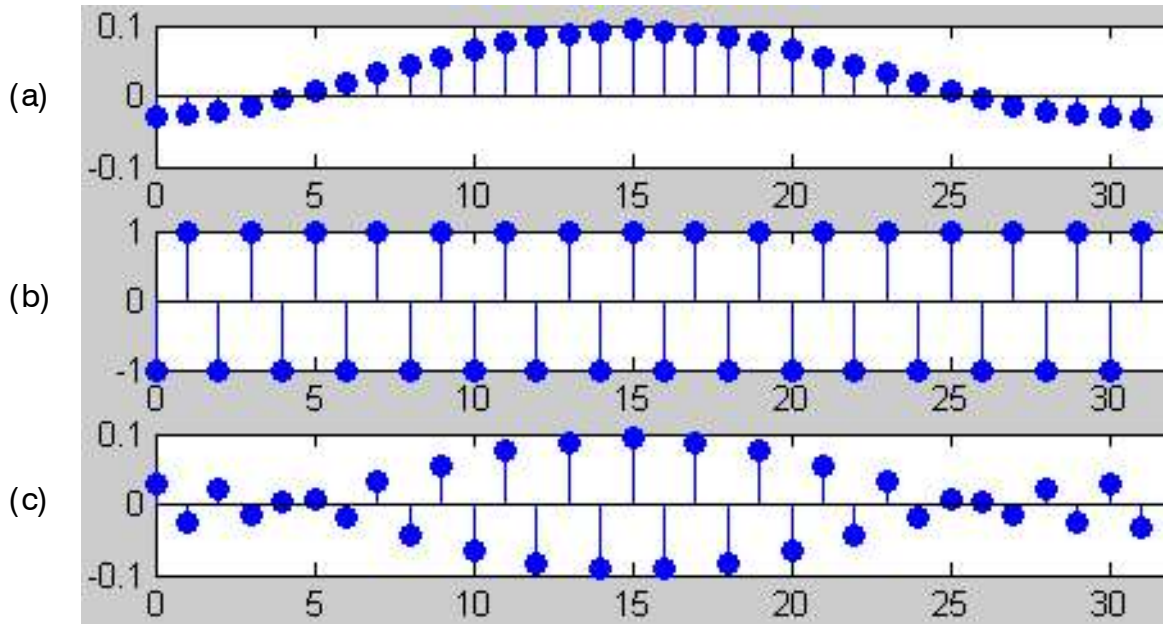


Figure 1.105: (a) impulse response (lowpass),
(b) s_{shift} coefficients by frequency shift of $f_s/2$,
(c) highpass impulse response after multiplication

The resulting frequency response is show below.

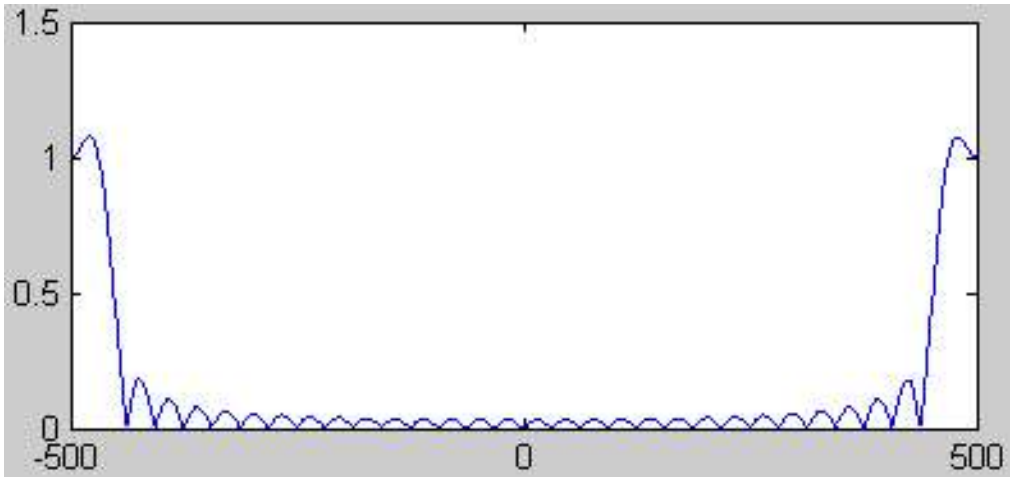


Figure 1.106: Highpass frequency response

Lab #06: Highpass

- Design a $N = 16$ lowpass filter (passband for $-2 \leq m \leq 2$)
- Minimize ripples for pass and stopband with a kaiser window (select $\beta = 3$ and restrict it to the first 15 values of the impulse response).
- Verify that frequency responses are as required.
- Calculate the s_{shift} coefficients and convert the lowpass to a highpass filter.
- Verify that you created a highpass filter by reviewing the frequency response.
- Use the test signal

$$x[k] = 1 + 0.25\sin(2\pi\ 50\ T_S\ k) + 0.5\sin(2\pi\ 400\ T_S\ k) \tag{1.184}$$

and verify that the first two terms are eliminate from the filtered result. The plot should look similar to the following diagram.

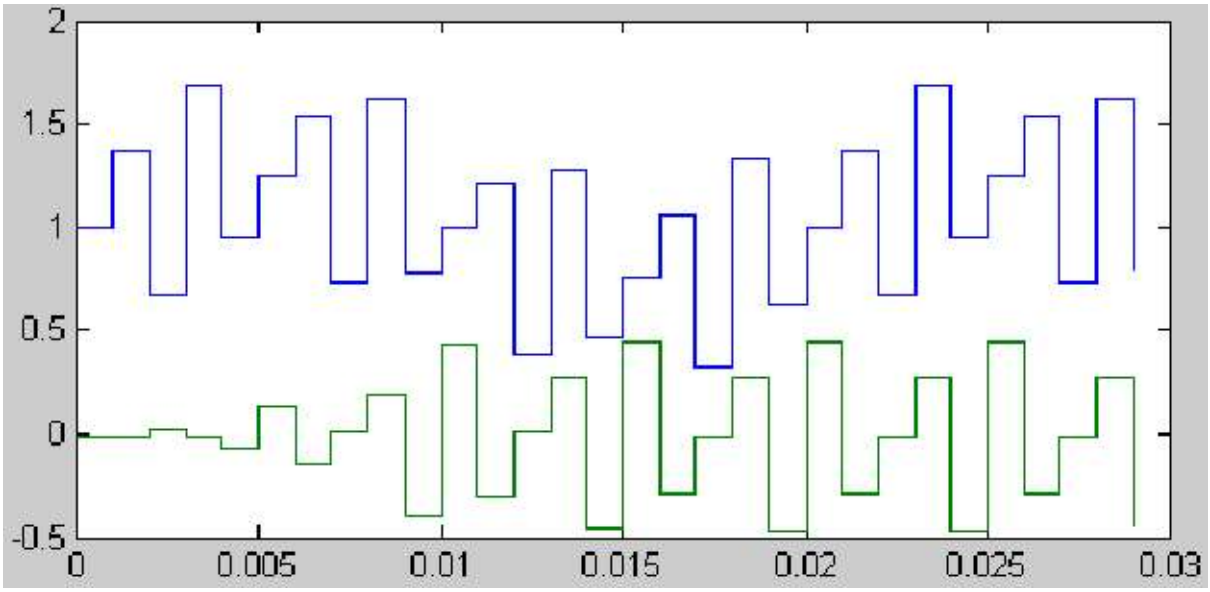


Figure 1.107: Input and output of the highpass filter (low frequency offsets removed)

Lab #07: 50Hz Notch

The 50Hz noise from an ECG signal should be removed by a notch filter. All other frequencies should keep their values. Since the sampling frequency f_s is 1 KHz the number of discrete frequencies N must be selected so that

$$f_k = \frac{k}{N} f_s, \quad \text{for} \quad k \in \{0, 1, ..., N - 1\} \tag{1.185}$$

matches 50 Hz for one k. We will choose N = 100, so the design has a 10 Hz resolution (40 Hz passes, 50 Hz is blocked, 60 Hz passes).

- Design a N= 100 notch filter.
- Verify by using `linfreqresp` that the filter works as required.
- Load the `s0020brem` ECG data set using `load_datm`.
- Filter the data of trace 11 with your designed notch filter and plot the result (compare unfiltered and filtered data). The result should look as follows:

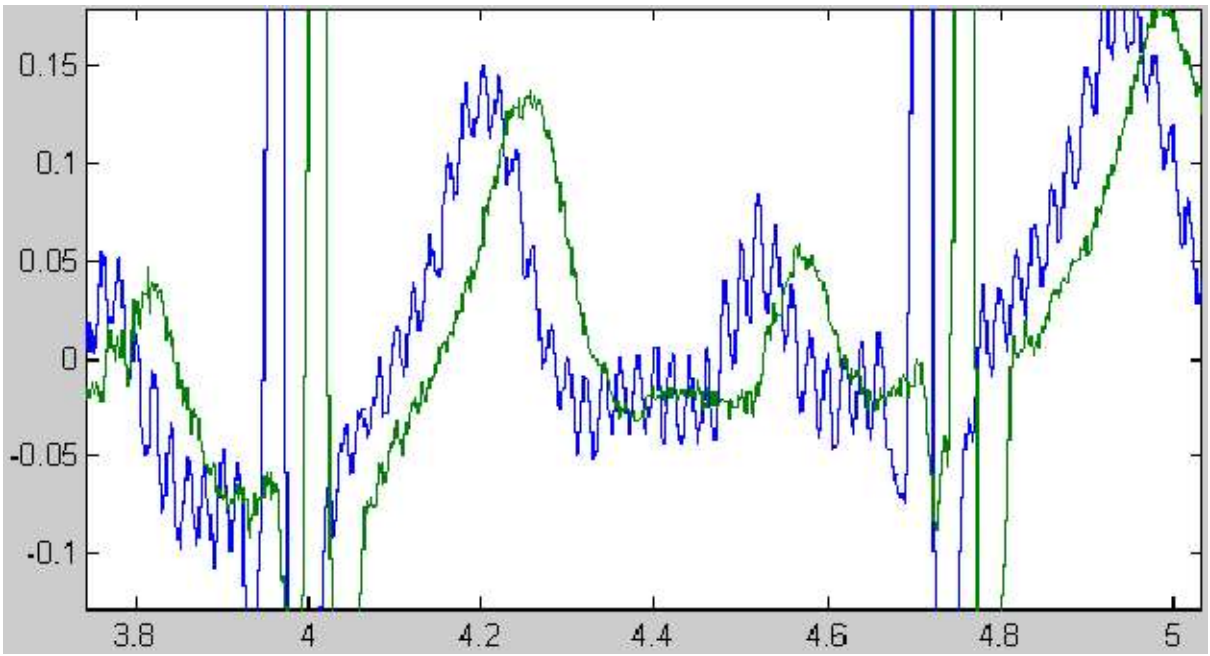


Figure 1.108: 50 Hz removal from ECG data

15 Fast Digital Signal Processing

Digital signal processing (DSP) can be carried out various digital devices.

- Slow DSP in microcontrollers
- Medium to high speed DSP on special DSP-processors
- Fast DSP on FPGAs or ASICs

While ASICs are useful for high volume applications (e.g. mobile phones) only FPGAs offer almost the same speed for any application that require high speed data processing. FPGAs are thus the best choice for high performance application at low hardware cost. However, a very simple formula reveals the true cost of a development

$$\text{design labor cost} = \frac{\text{(labor unit cost) x (design complexity)}}{\text{designer productivity}} . \qquad (1.186)$$

Since design complexity is considered to be higher for hardware (FPGA) DSP implementation, FPGAs have been used in the past only it the applications speed requirements were met only by FPGAs.

The key to the successful use of FPGAs is thus the designer productivity. Several tools have been recently developed for this propose. The most popular tools for DSP are National Instruments LabVIEW™ and Matlab's Simulink™ . While LabVIEW focuses on Xilinx FPGAs only the Matlab approach generates general purpose VHDL code. Probably the most efficient tool is Xilinx's SYSTEM GENERATOR™ base in Matlab Simulink. The reason is obvious: Since the code generator knows exactly the internal structure of the FPGA type in use it can exploit the special features of the individual (DSP blocks, RAM, Shift registers and so on).

The design flow is outlined below. The System Generator acts as the design entry tool. The complete design, simulation, verification can be carried out in Matlab Simulink. It generates all source files for the underlying ISE tool suite including ISim, ChipScope (integrated logic analyzer ILA) and Impact for FPGA configuration. Compilation, place and route is carried out with ISE and is usually not part of the System Generator (although it is possible). The ISE tools suite offers better support for checking the user constraints, timing requirements and implementation details (FPGA editor). This is much different from LabVIEW which hides all these steps from the user (except for unexpected errors).

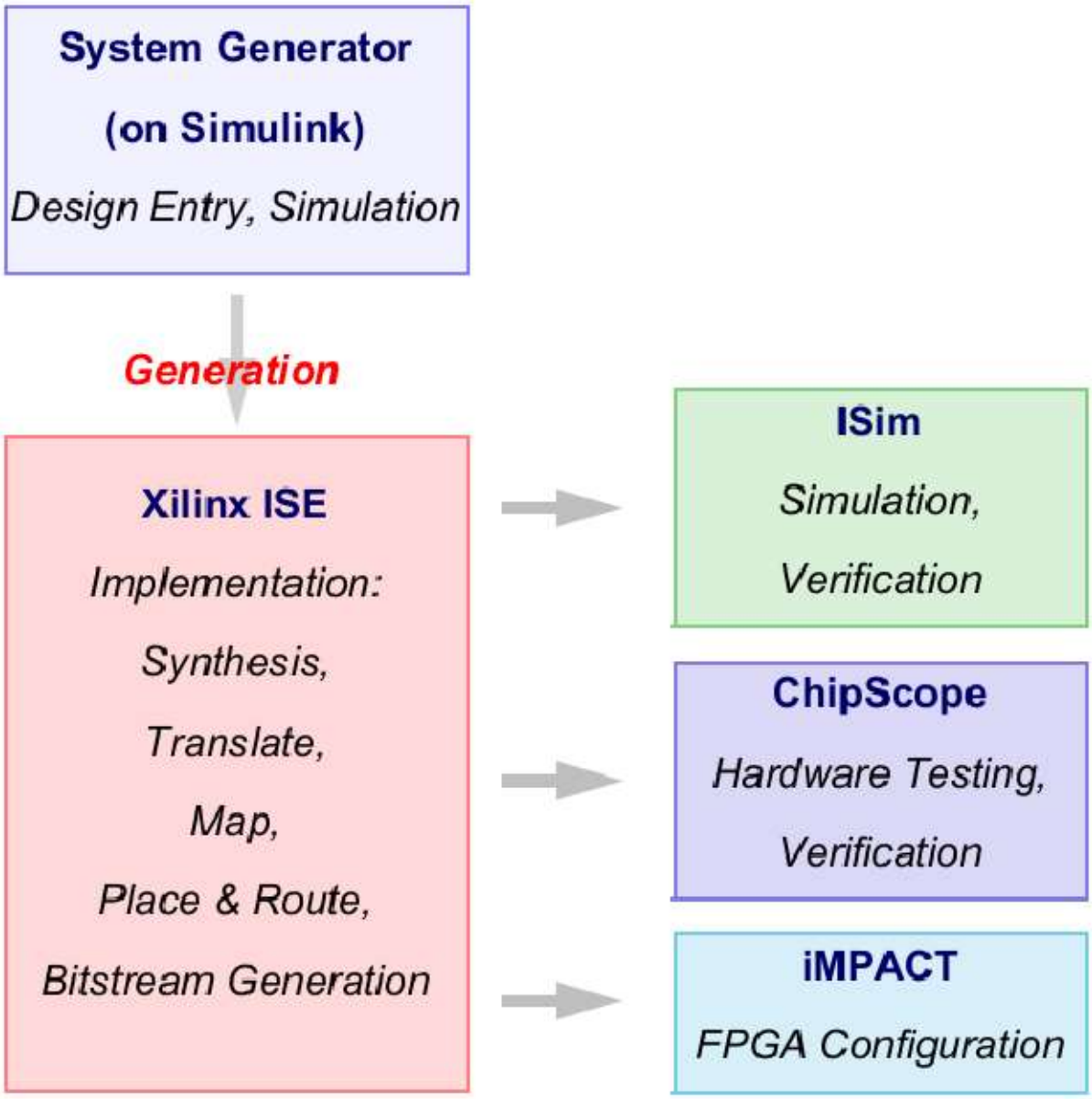


Figure 1.109: System Generator design flow

In theory a user of the System Generator can design DSP systems in hardware with unbeatable performance. However, sound knowledge of the FPGA internals and resources and VHDL knowledge is essential for an efficient implementation, since DSP is only a part of a real embedded system. But often it is the most complex part.

Lab #08: System Generator

With a simple application the design flow will be shown (not really DSP).

A four samples delay according to the following block diagram will be created using System Generator.

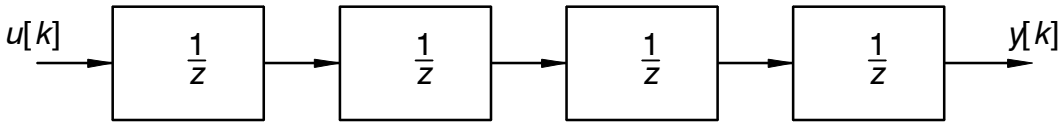
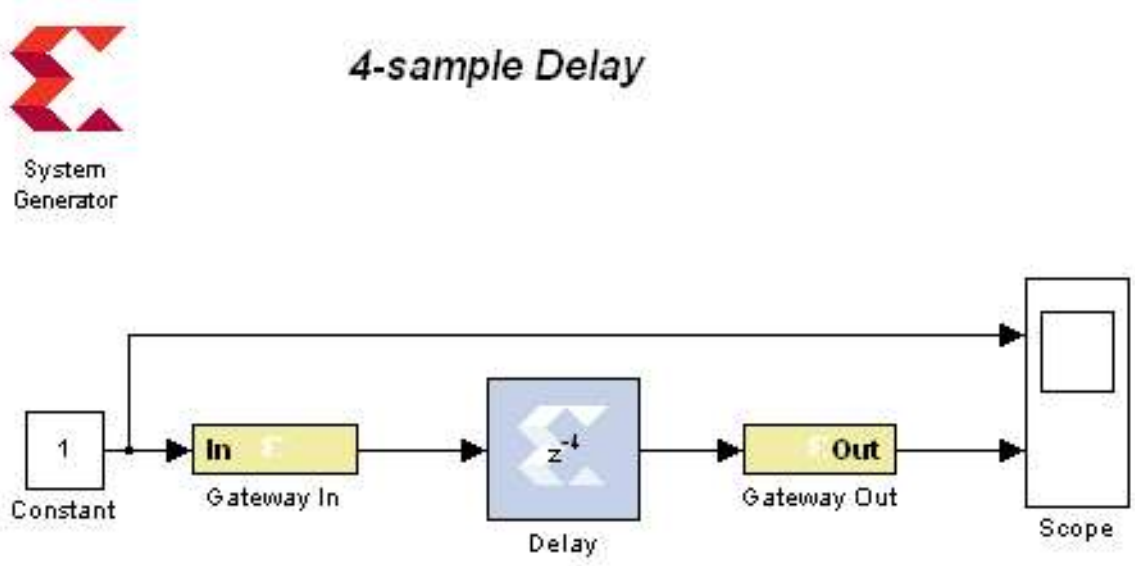


Figure 1.110: 4 sample delay

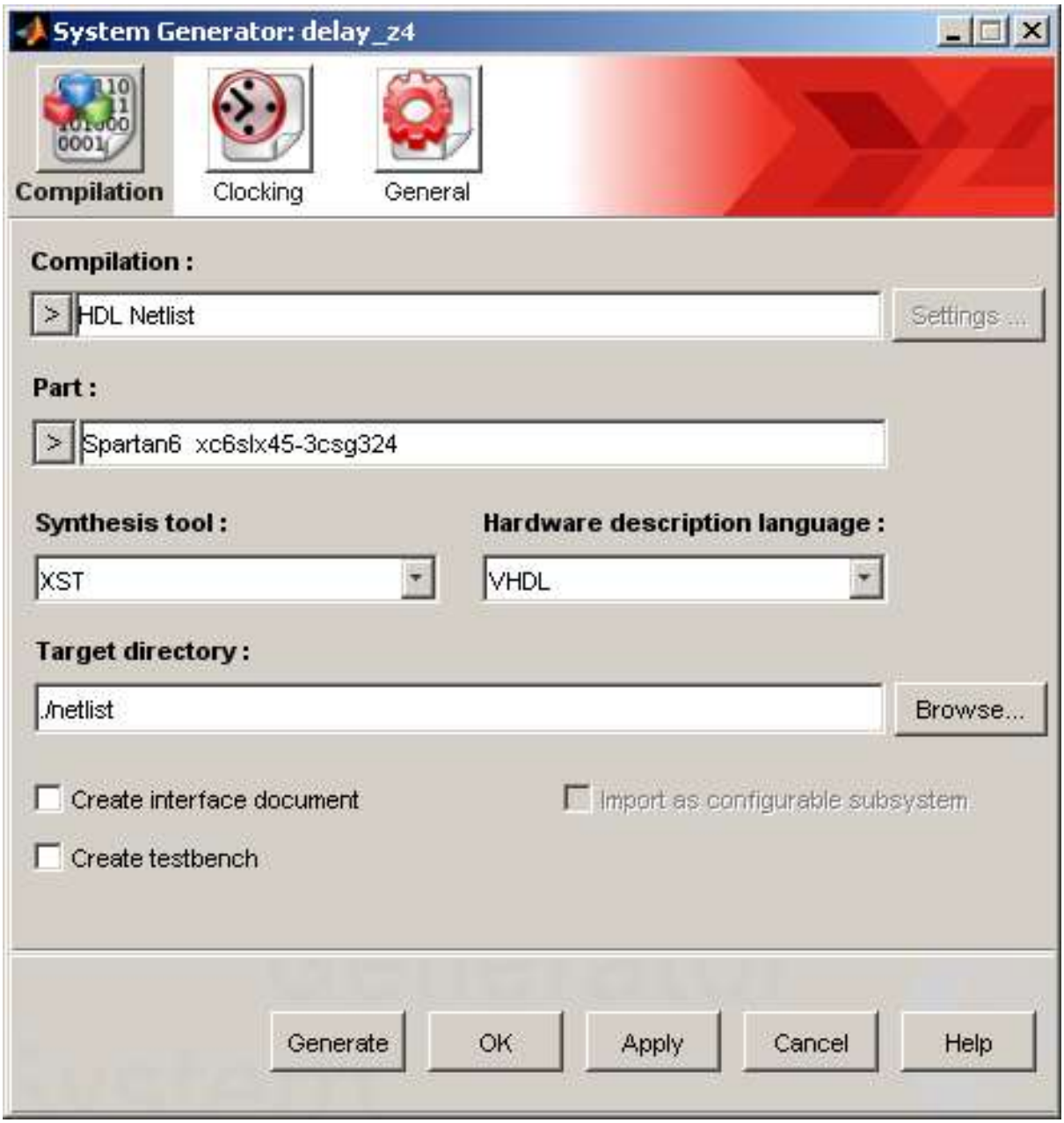
Input and output is only 1 bit wide. It is obvious in the design process that it becomes very easy to change the signal resolution (number of bits).

- Create a new! directory called "delay_4z".
- Create an empty Simulink model in this folder and name it "delay_4z".
- From the Xilinx Blockset insert the blocks as outlined below.



All block that do not have the Xilinx shaded logo will not be synthesized. They are for Simulink simulation and versification only. The input and output signals are taken from signals or pads in the synthesized design.

- The System Generator block is not connected to anything. It is a control block required for synthesis. This block needs to be parametrized to match the target FPGA, the clock system and the Simulink fundamental sampling period. All other sampling periods must be a multiple of this sampling period.
  Set the parameters for the compilaton tab as follows:

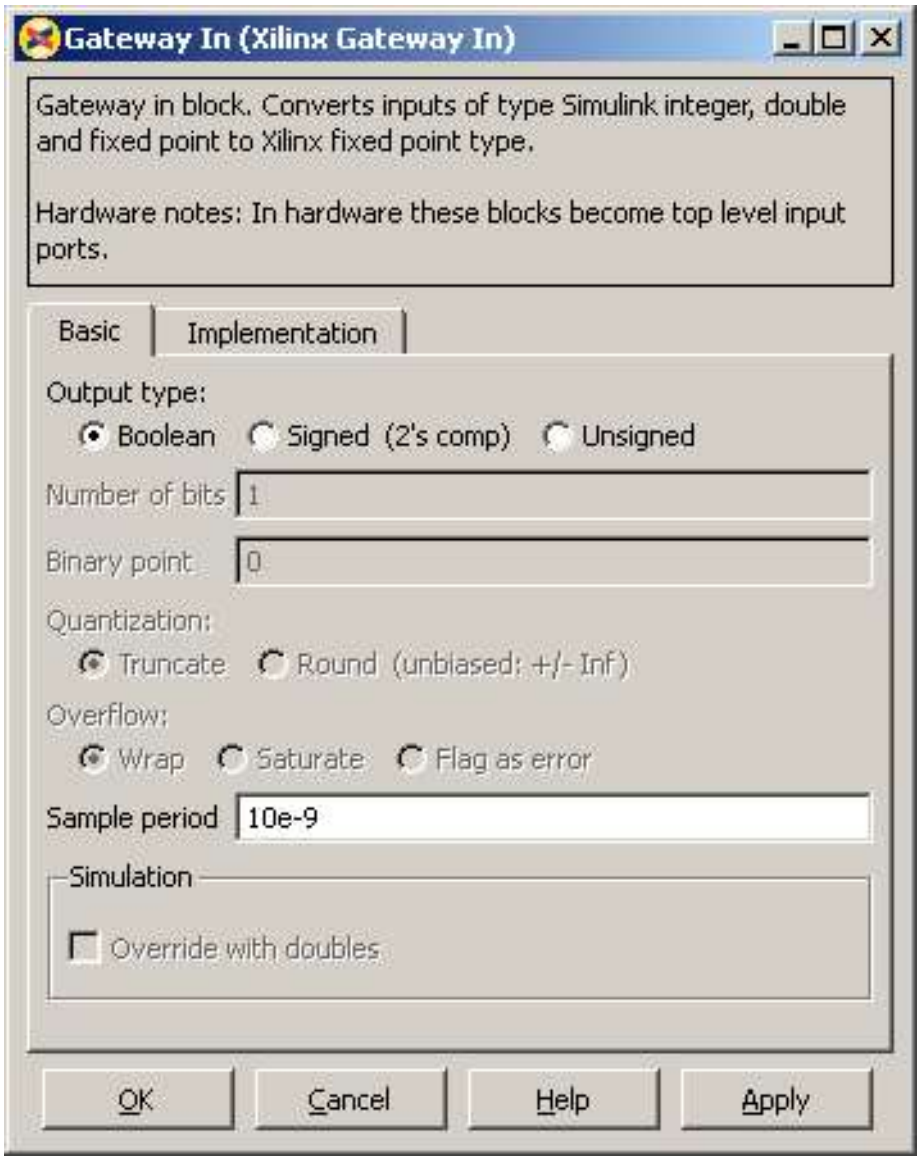


Adjust parameters of the clocking tab as follows:

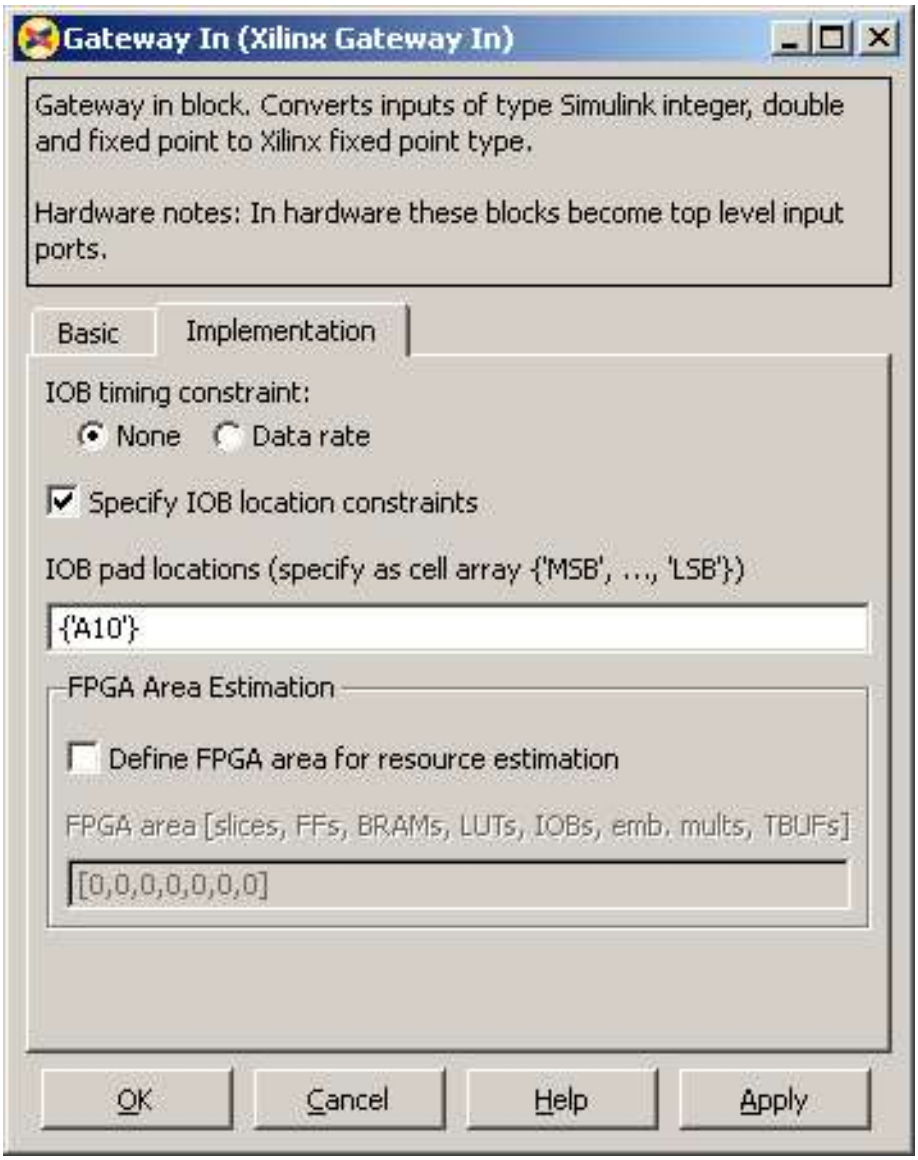
These setting match with the Atlys Spartan-6 board (100 MHz clock source on L15):

- Change the latency (number of flip-flops / registers) to 4 in the delay properties.
- The Gateway In and Gateway Out blocks define the systems input and output signals.

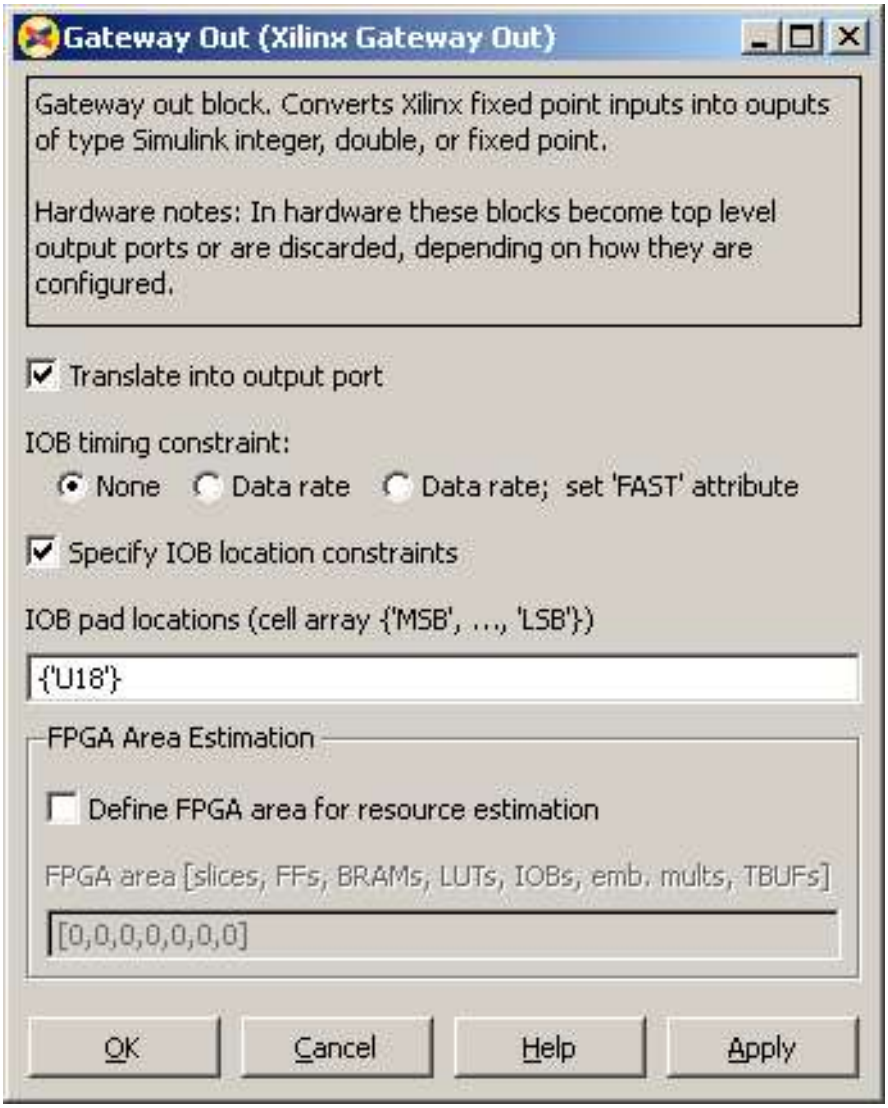
Gateway In / Basic settings:



Gateway Out / Implementation settings [the input A10 is attached to the switch(0)] :



- Set the Gateway out parameters according to the following figure [the pad U18 is connected to LED(0)].



- In the Simulink Configuration Parameters set
  Stop time: 100ns and use Solver: discrete (no continuous states). Leave the rest unchanged (defaults). Select OK.
- Start the simulation (please wait for addition simulation setup computation) and verify that the systems works as expected.
- Open the System Generator and select Generate. Please wait for completion and watch possible error messages that might come up.

Now you have created an executable FPGA configuration. The rest is carried out in ISE.

- Start ISE and open the delay_z4_cw.xise project description file in the ./netlist folder.
- Open the generated VHDL file and search for "gateway" to inspect how the input and output signals are processed.

- We can now tune the design to optimize it to a particular application.
  Open the Synthesize–XST property dialog and select the HDL Options tab.
  Uncheck Shift Register Extraction. What does that mean?
  It the Xilinx Specific Options change Pack I/O Registers into IOBs to YES.

- Start synthesis by selecting Implement Design –> run
  (you can safely ignore the warnings – most of them result from the simple design)

- In the Design Summary select the Post–PAR Static Timing Report

  What is the minimum period? _____

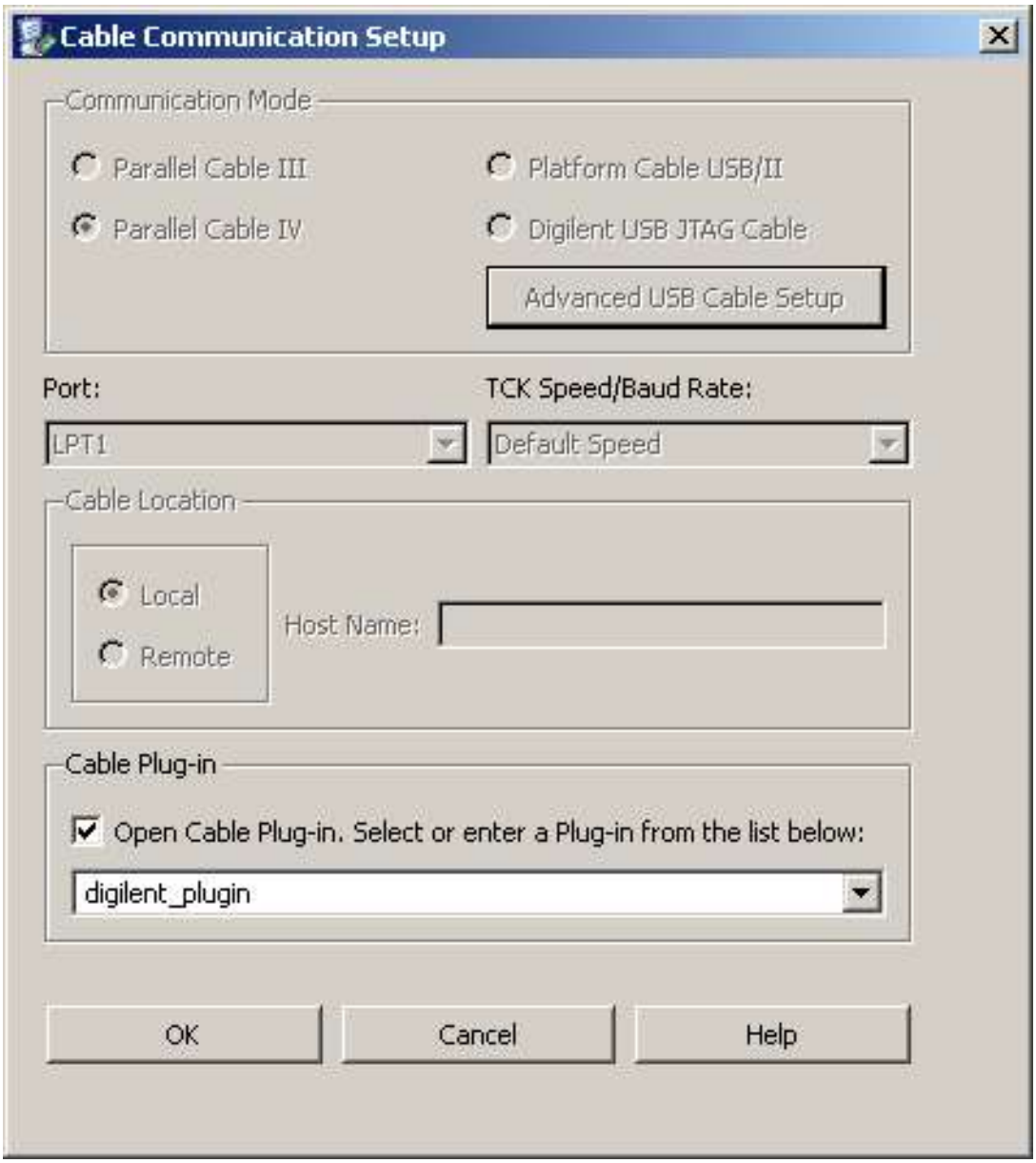
  What is the maximum clock frequency? _____

  Is the timing constraint (100 MHz clock) satisfied? _____

- Inspect the Device Utilization Summary and verify that the resources are allocated as expected.

- Start the FPGA Editor.
  Inspect the realization of gateway_in, gateway_out and delay_z4.

- Select Manage Configuration Project (IMPACT) –> run
  Double–Click on Boundary Scan
  Select Output –> Cable Setup from the Impact Menu and enter digilent_plugin into the Cable plug-in field. Check Open Cable plug-in.



- Right-Click Initialize JTAG Chain.
  Auto–Assign the configuration file delay_z4_cw.bit.
  Select NO Attach SPI or BPI PROM!
  Accept the defaults to the Device Programming Properties.

- Right-click on the Spartan device and select Program.

- Verify on the Atlys board the the configuration executes properly.

16 Logic Analyzer

From the previous lab it becomes obvious that the design cannot be tested completely since internal signals are not accessible for measurement. Moreover, the measurement of a number of digital signals or busses (or integer signals) is a tremendous effort, require cost and time. Modern FPGAs thus have their integrated logic analyzers (ILA). For Xilinx this tool is denoted ChipScope. It also works in combination with external logic analyzers or MSOs (Mixed Signal Oscilloscopes). For most application ChipScope works without additional hardware by using the JTAG port. Figure 1.111 outlines the ChipScope block diagram.

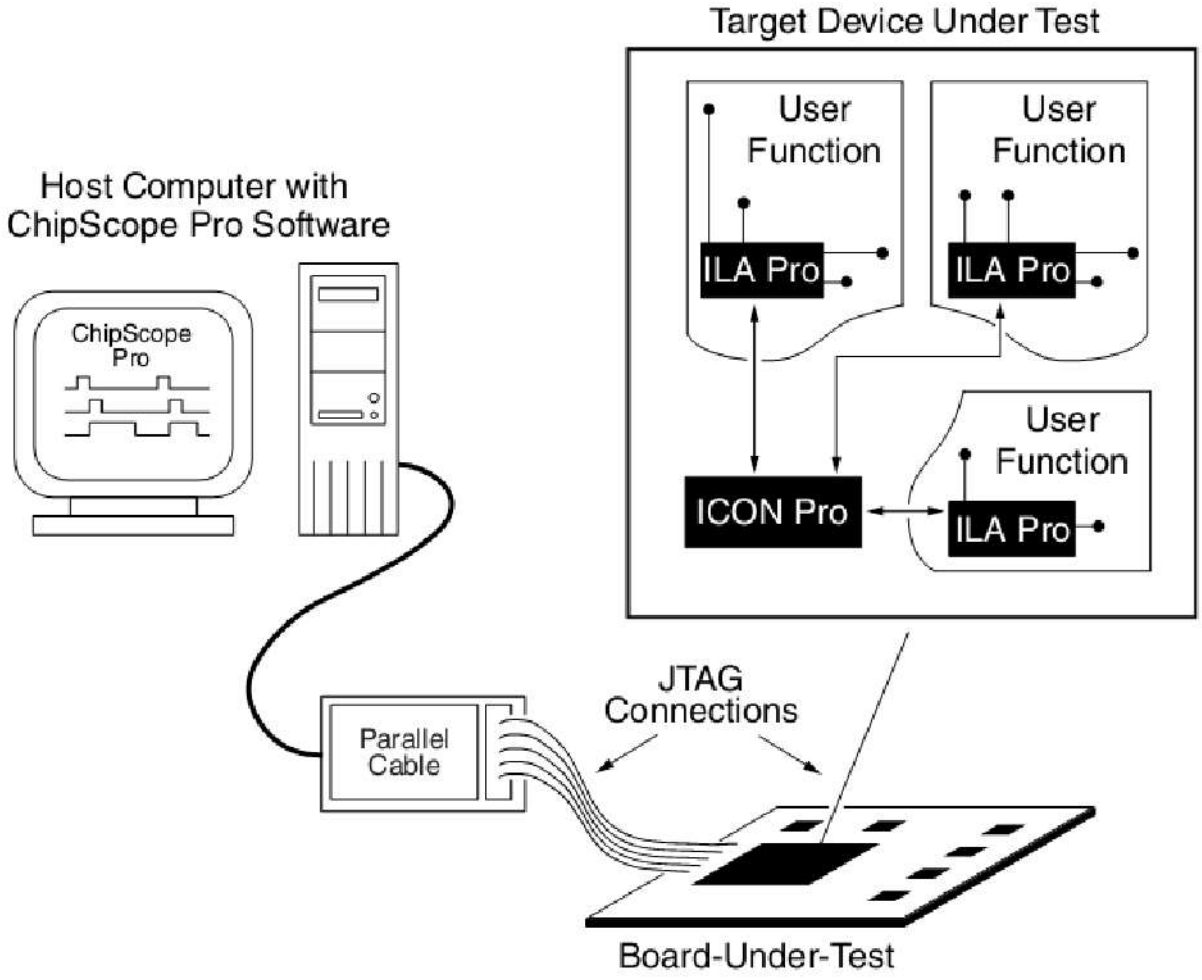


Figure 1.111: ChipScope block diagram, © Xilinx 2011

The ILA becomes part of the FPGA configuration. Using an ILA causes some overhead due to additional slices, routes and buffer memory (usually Block RAM) to store the measurements after trigger conditions have met. This means that an FPGA cannot include an ILA if most of the resources are used up by the design.

The ILA limitations are:

- 4096 data channels
- Buffer size from 256 up to 131k (possible buffer size depends on the FPGA type)
- 16 trigger ports of up to 256 channels
- 16 separate match units per trigger with variable complexity (complex match conditions require more FPGA resources)
- Synchronous operations up to 500 MHz
- Triggering by boolean expressions or a sequence of algebraic conditions
- Boolean storage qualification conditions
- Graphical display of signals, busses, and integer values
- Virtual input/output
- Agilent logic analyzer support (ATC2 = Agilent Trace Core 2)
- Up to 15 independent ILAs, VIOs or ATC2s per design

Even with this limitation the integrated logic analyzer is more powerful than any available logic analyzer device.

Lab #09: VIO / ILA (Update to Vivado)

16.1 VIO / ILA Lab

The proper timing if the design can be verified by the interated logic analyzer (ILA) in every FPGA. However, this requires some free ressources (logic cells and block ram) for trigger functions and data storage in real time.

All signals under investigation are connected with VIOs (virtual IOs). Several ways exist to include the ILA into a design.

Previous versions of the System Generator had an ILA block in Simulink. The Vivado version of the System Generator does not offer this feature. The ILA need to be added within Vivado.

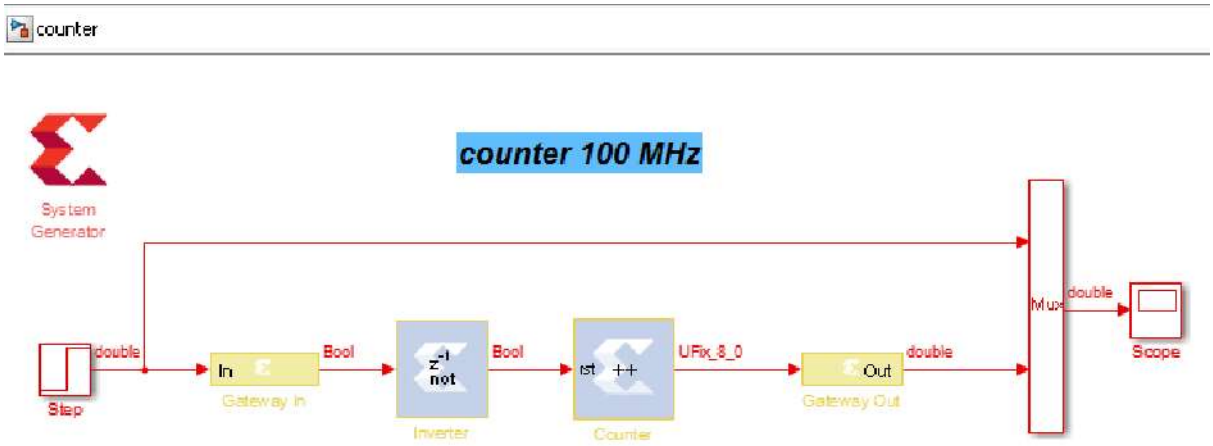


Figure 1.112: 100 MHz 8 bit counter

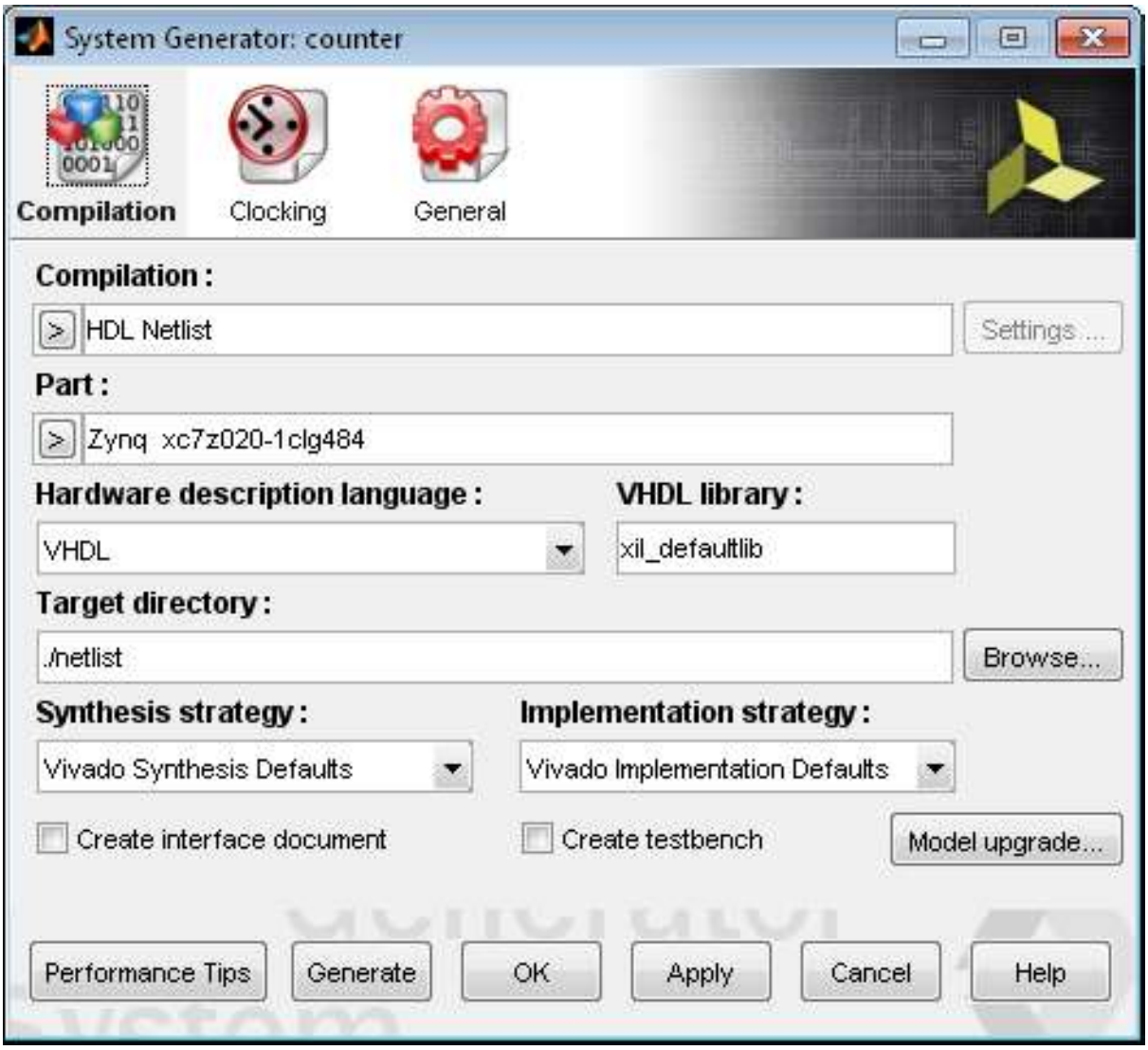


Figure 1.113: System Generator block configuration (1)

Figure 1.114: System Generator block configuration (2)

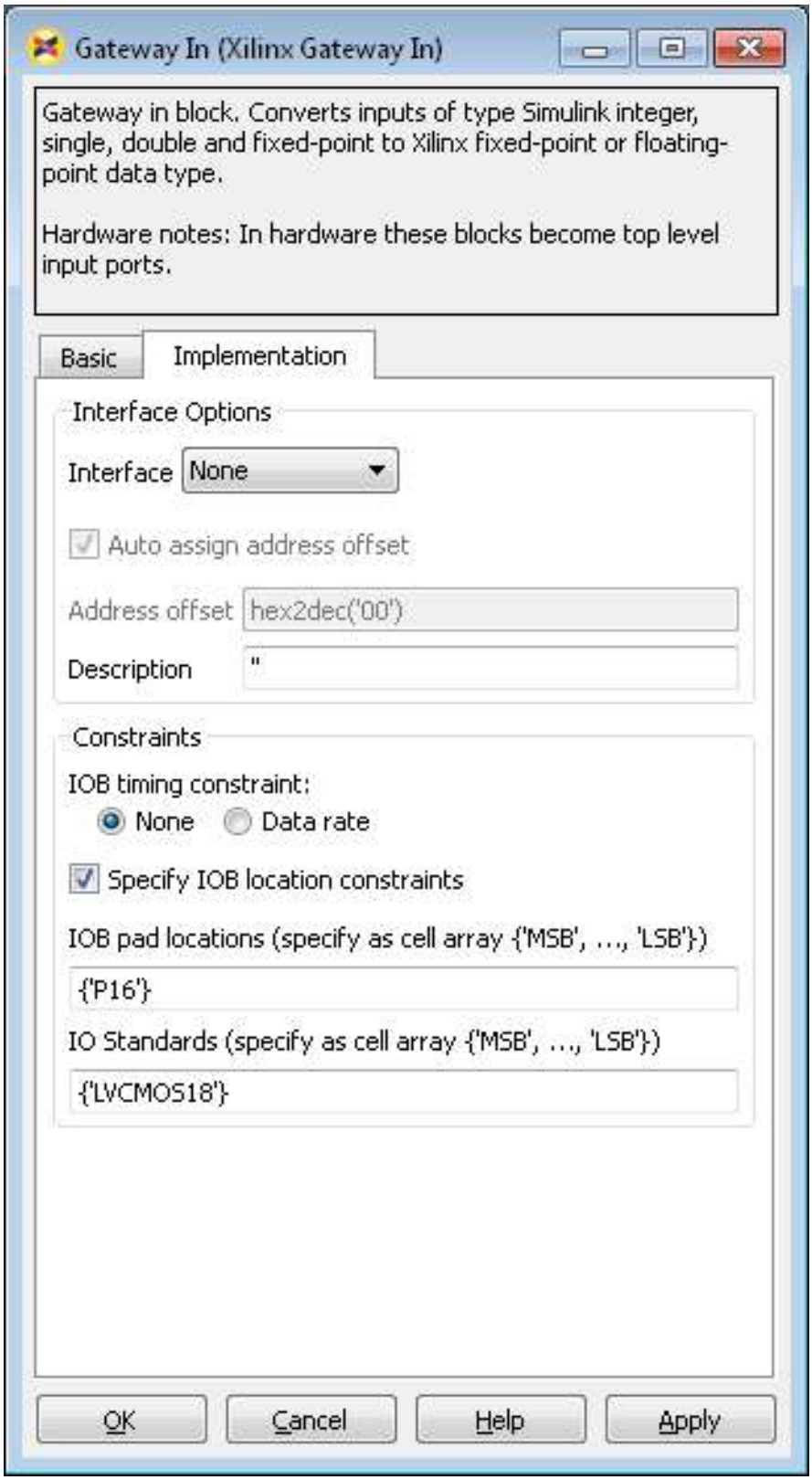


Figure 1.115: Gateway in block configuration (input is “button center”)

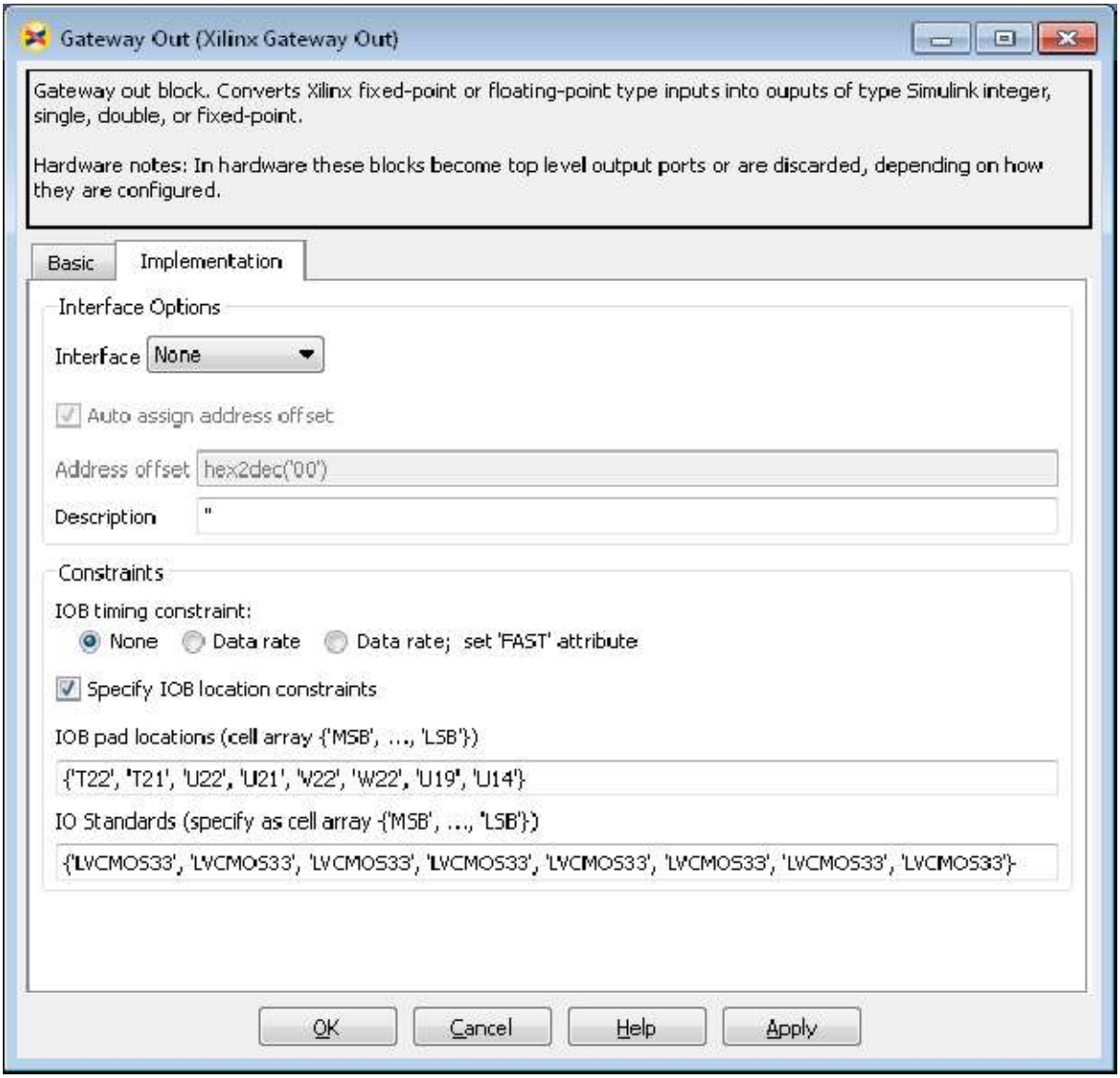


Figure 1.116: Gateway Out block configuration (outputs are “leds[7:0]”)

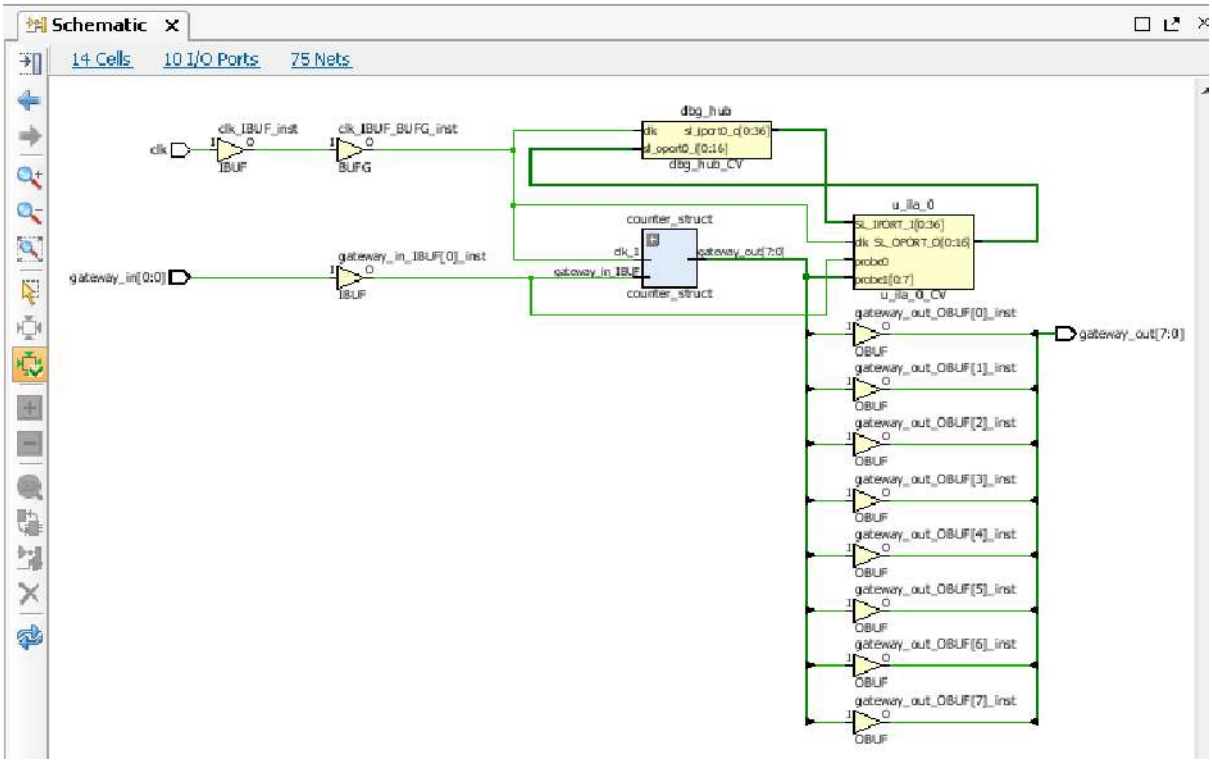


Figure 1.117: Debug schematic view in Vivado

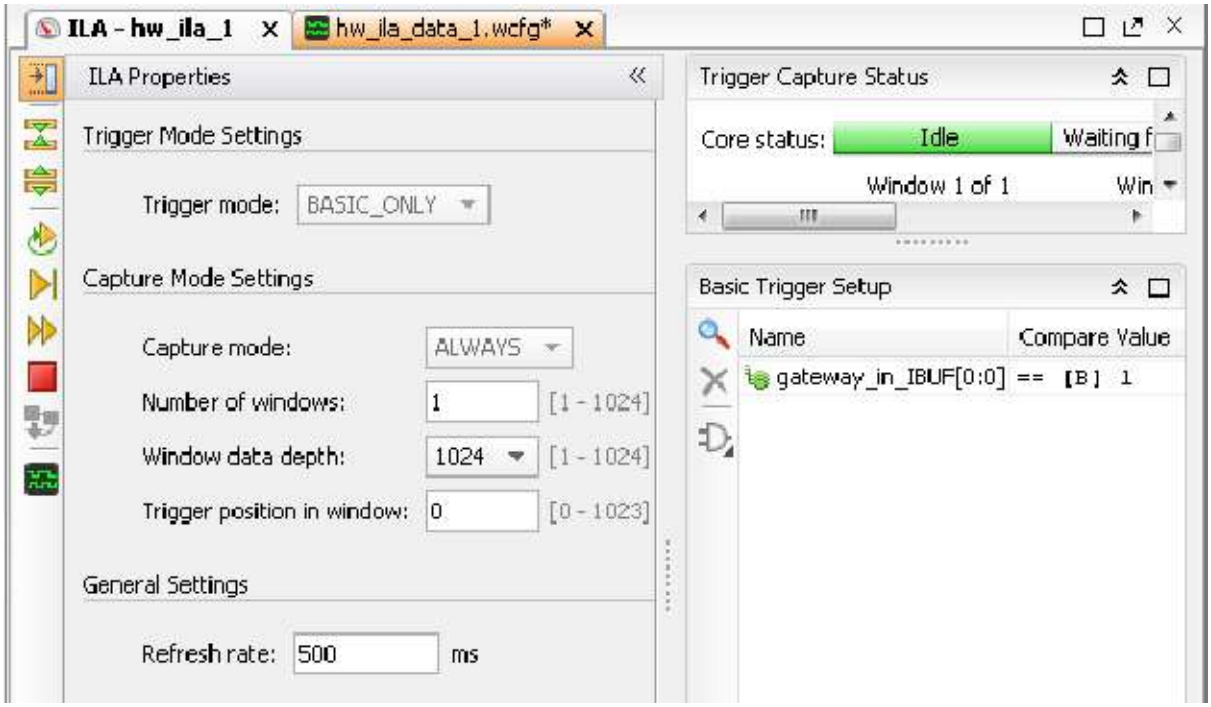


Figure 1.118: ILA trigger and configuration

Figure 1.119: Counter output in ILA (real-time)

- Create a new folder "delay_z4Cs" and create a "delay_z4cs.mdl" Simulink model. Add a ChipScope probe to the design according to fig. 1.120. Note that the signal names are propagated to ChipScope. This is recommended for analysis in the ChipScope waveform diagram.

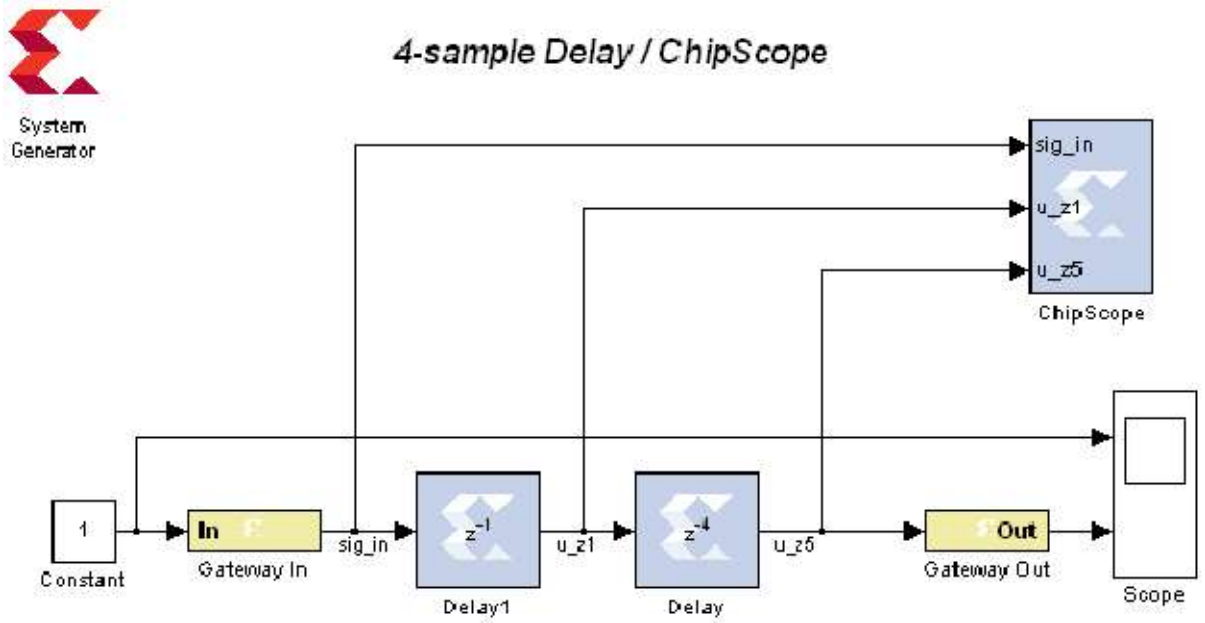
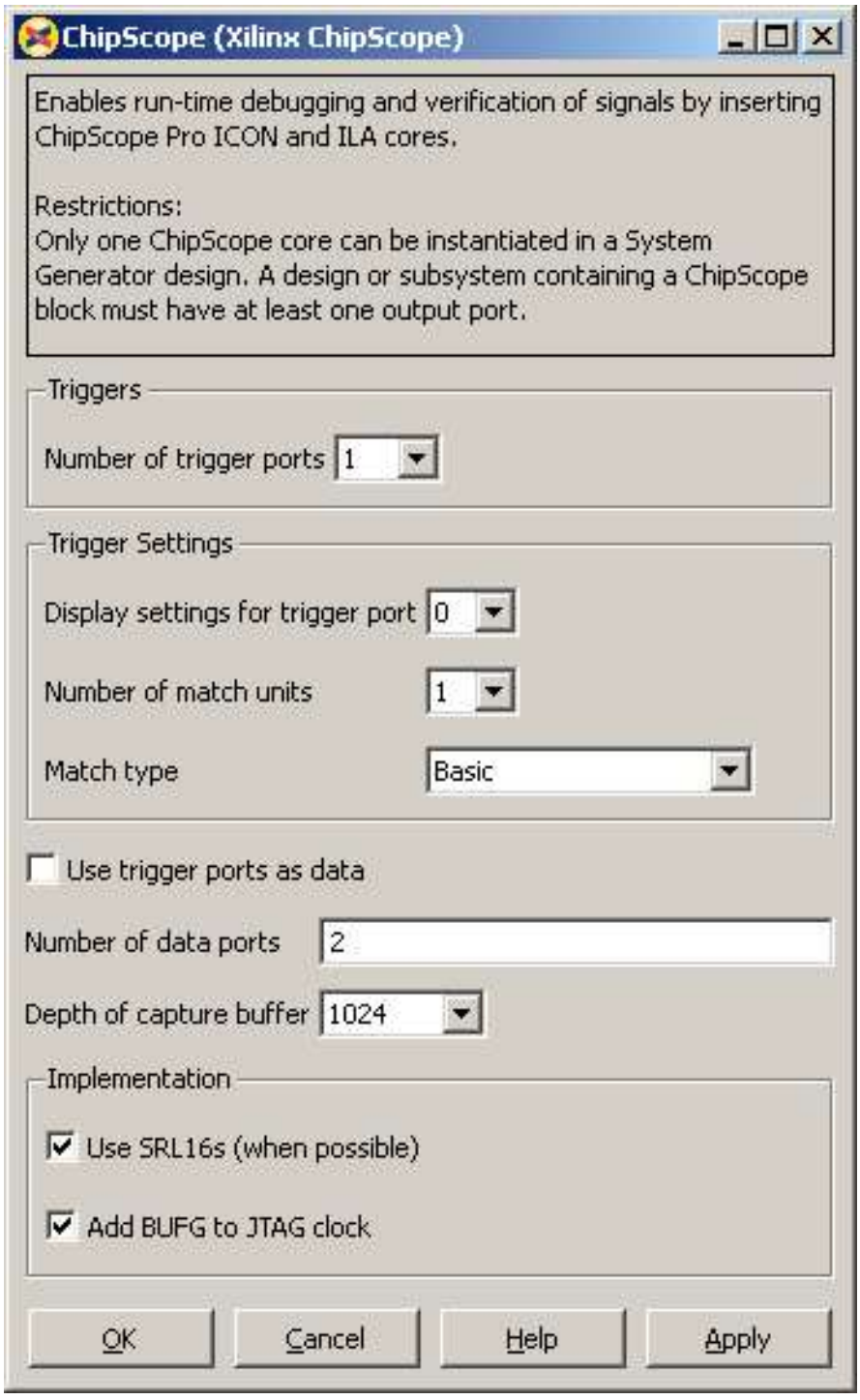


Figure 1.120: ChipScope application

The input signal triggers the analyzer.

- Set up the ChipScope property sheet according to the following figure.

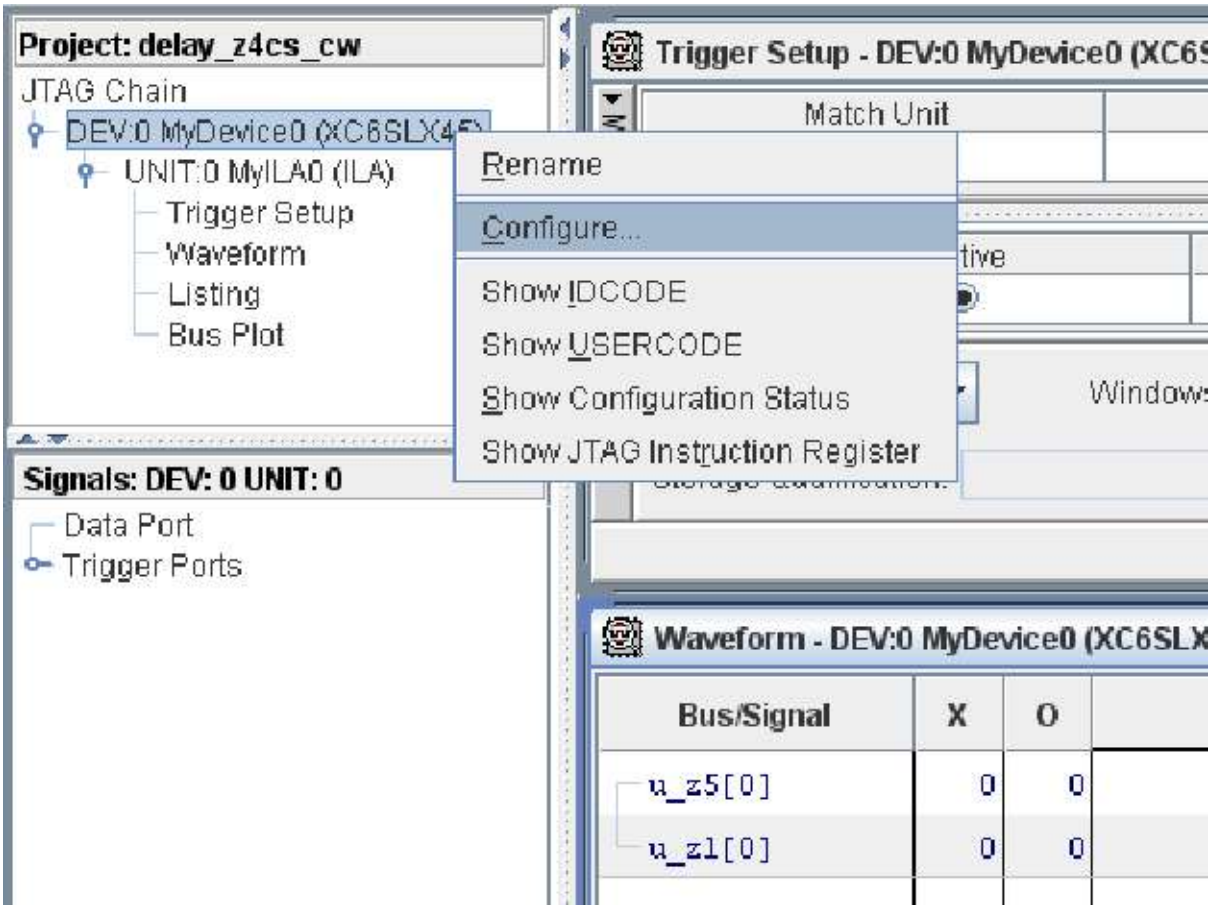


For Spartan-6 the minimum capture buffer depth is 1024 since data is stored in block ram.

- Configure the System Generator with 10ns clock rate (on pin L15) and generate the design.

- Open the project file “delay_z4cs_cw.xise” in ISE.

- Accept all defaults and select "Analyze Design Using ChipScope" with right mouse button and click "Run". Wait for ChipScope to come up.
- In ChipScope select from the menu JTAG Chain –> Open Plug-in...
  Enter digilent_plugin in the Plug-In Parameters dialog. Select OK.
- Select the FPGA device and select Configure...



Select the "delay_z4cs_cw.bit" file and configure the FPGA.

- Signal are defined by a ".cdc" file (ChipScope Definition and Connection File) which is required for signal display and diagrams. The System Generator automatically creates such a file from the Simulink model. From the main menu open File –> Import...
  Locate the file "delay_z4cs.cdc" as Import File and select OK.
- Set the trigger condition to M0:sig_in to '1' (this trigger the ILA if switch '0' is set to '1').
- Arm the ILA by pressing the ▶-symbol. Switching on (SW0) should start the capture process. Verify that you see a plot similar to fig. 1.121.

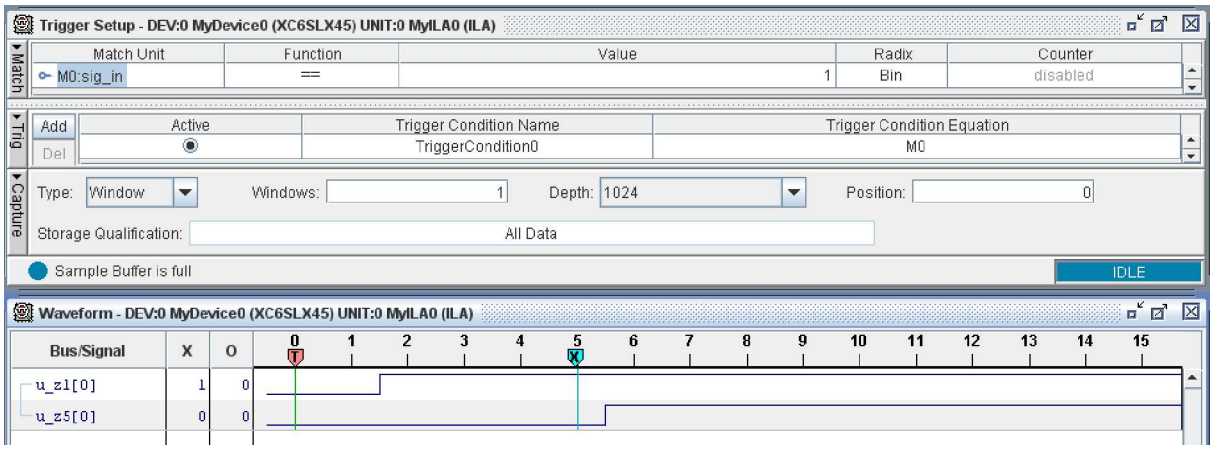


Figure 1.121: Waveform after trigger event

Lab #10: 10 MHz FIR Filter

16.2 Hardware FIR Filter Lab

A 10 MHz FIR filter cannot be realized by a microcontroller. The proper timing of the design can be verified by the logic analyzer. An 8 bit counter acts as a sawtooth generator. The following 7-tap lowpass FIR filter removes higher order harmonics to obtain a sinusoidal output (also 8 bits). The FIR coefficients are given as parameters to the FIR compiler block. The upper Gateway Out block is only required for plotting since Simulink Scopes should never access internal signals directly.

- Create a new folder "sin_gen" and a Simulink model with the same name.

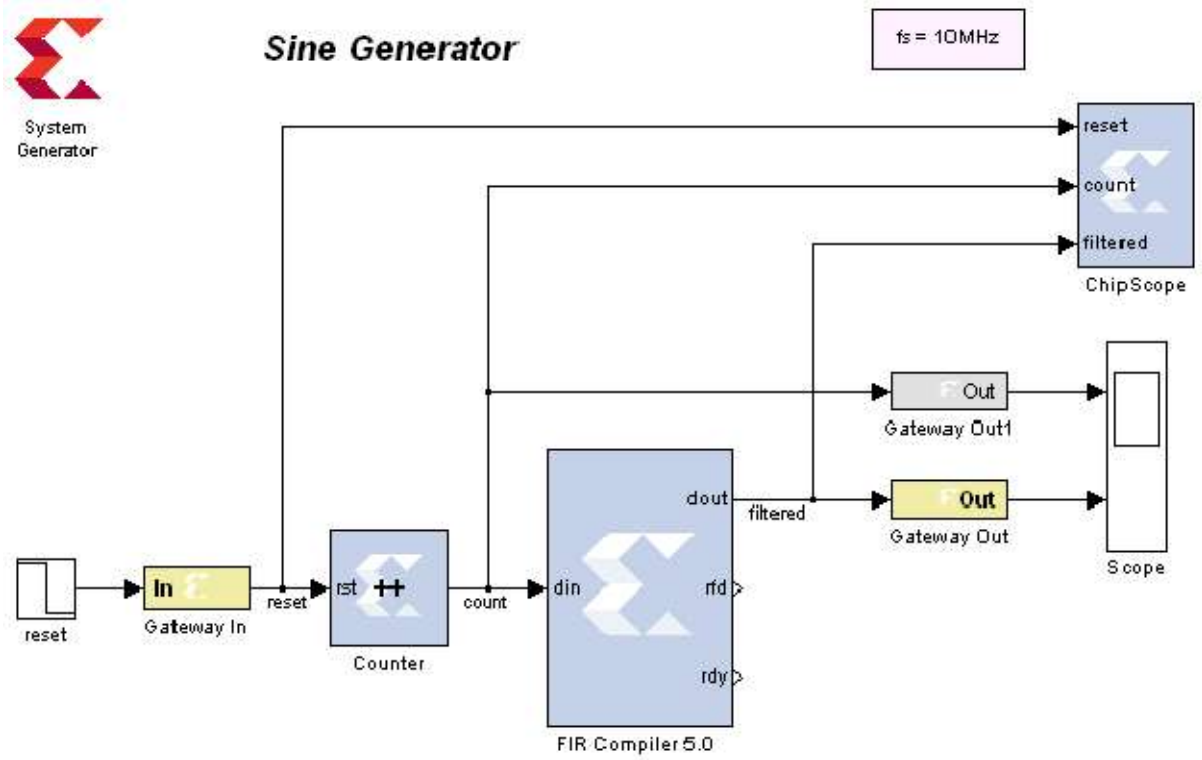
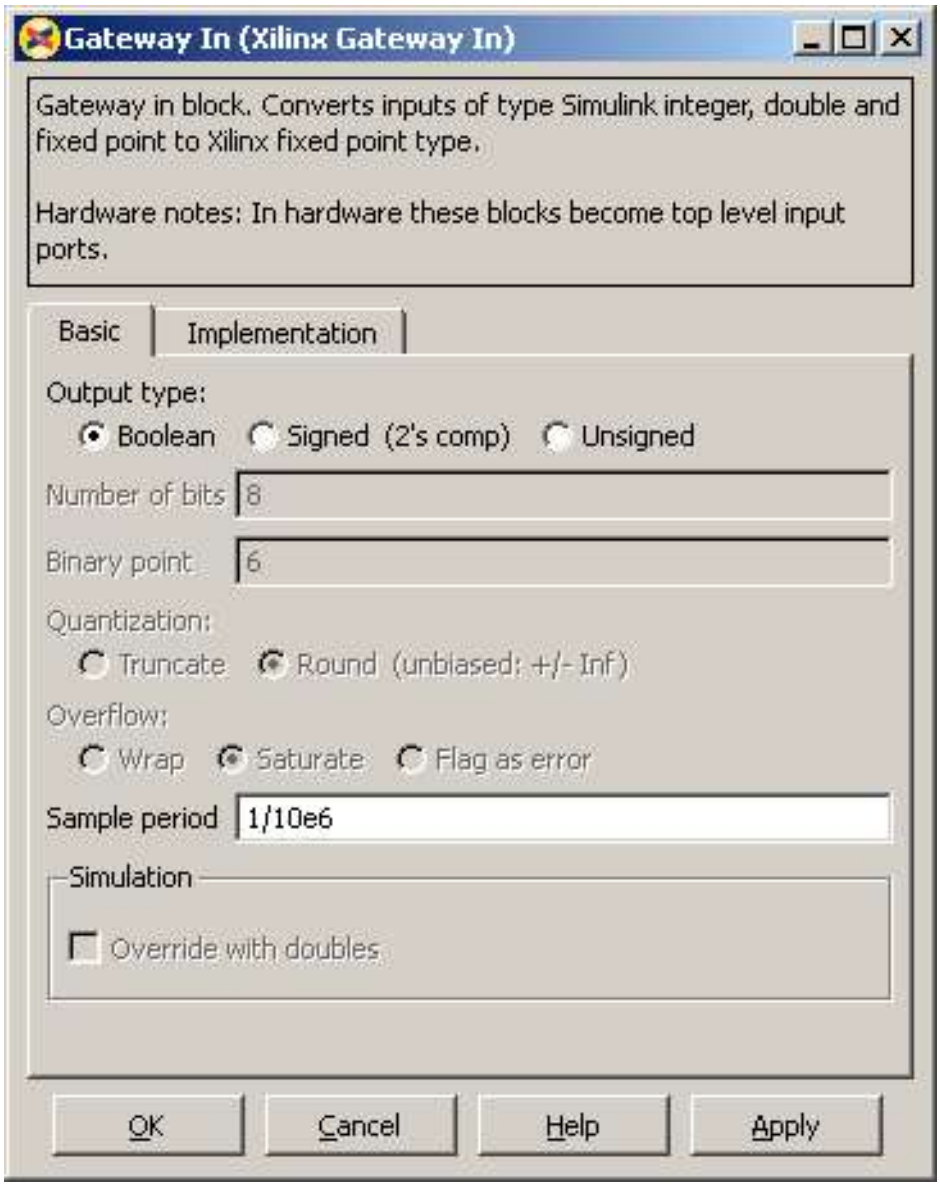
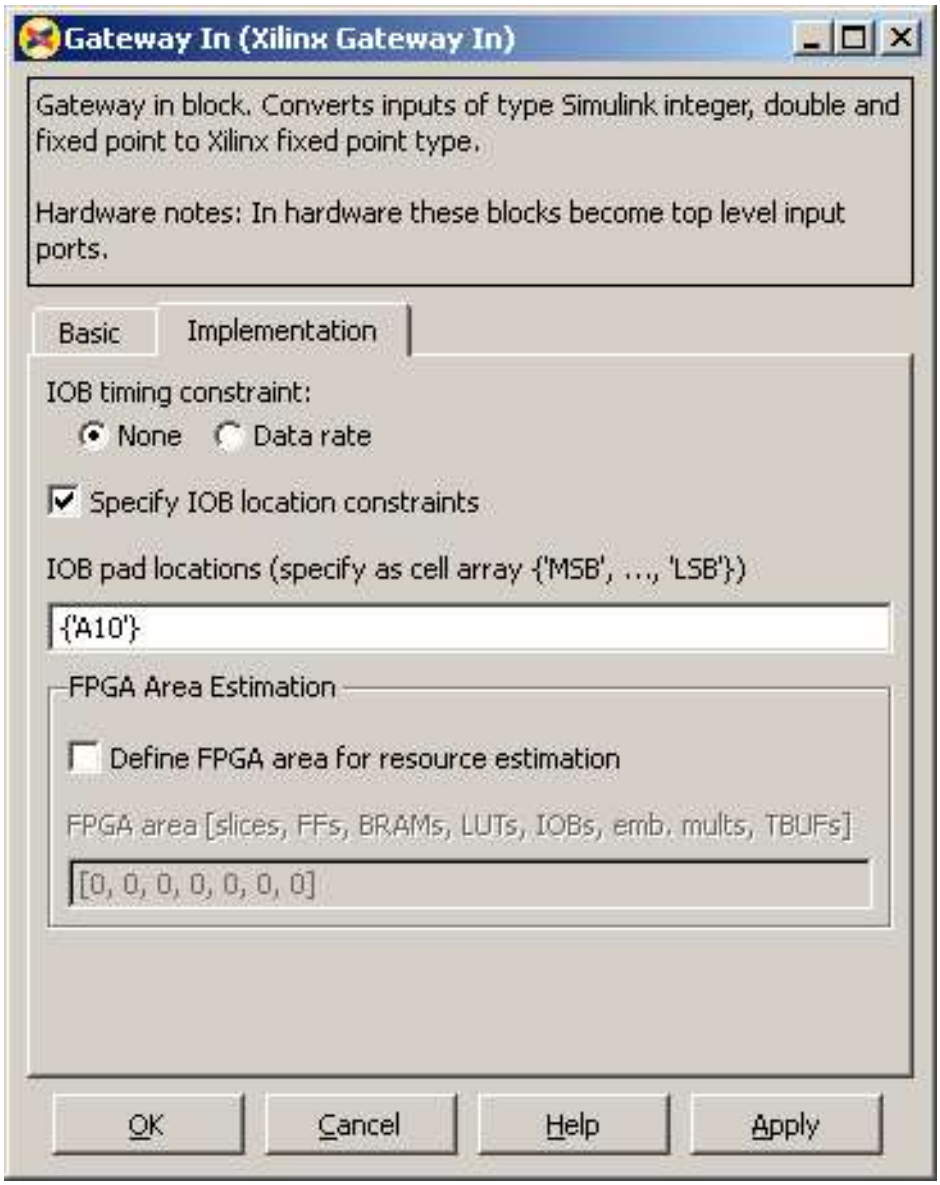


Figure 1.122: Sine generator model

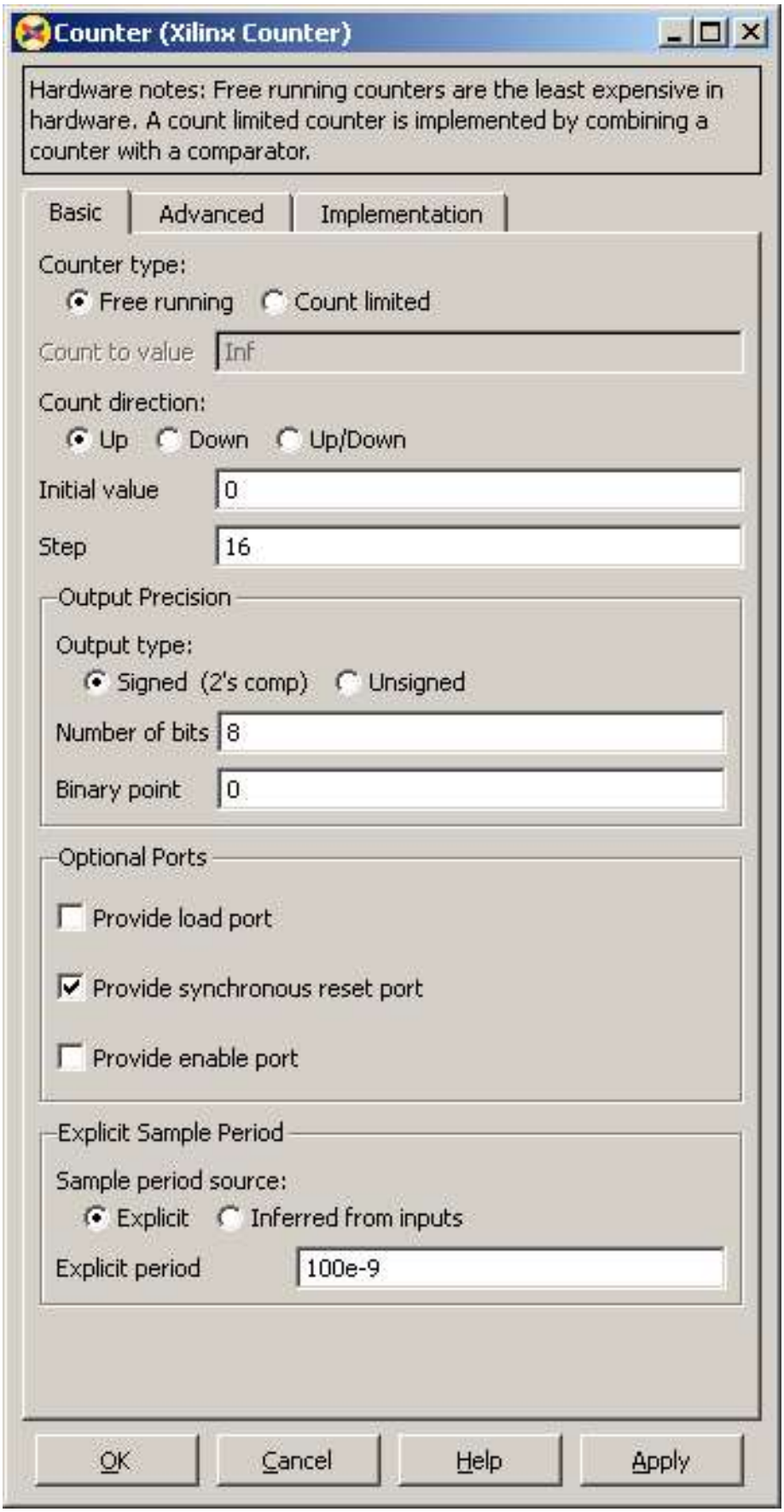
- The sampling rate is determined by Gateway In (see below).



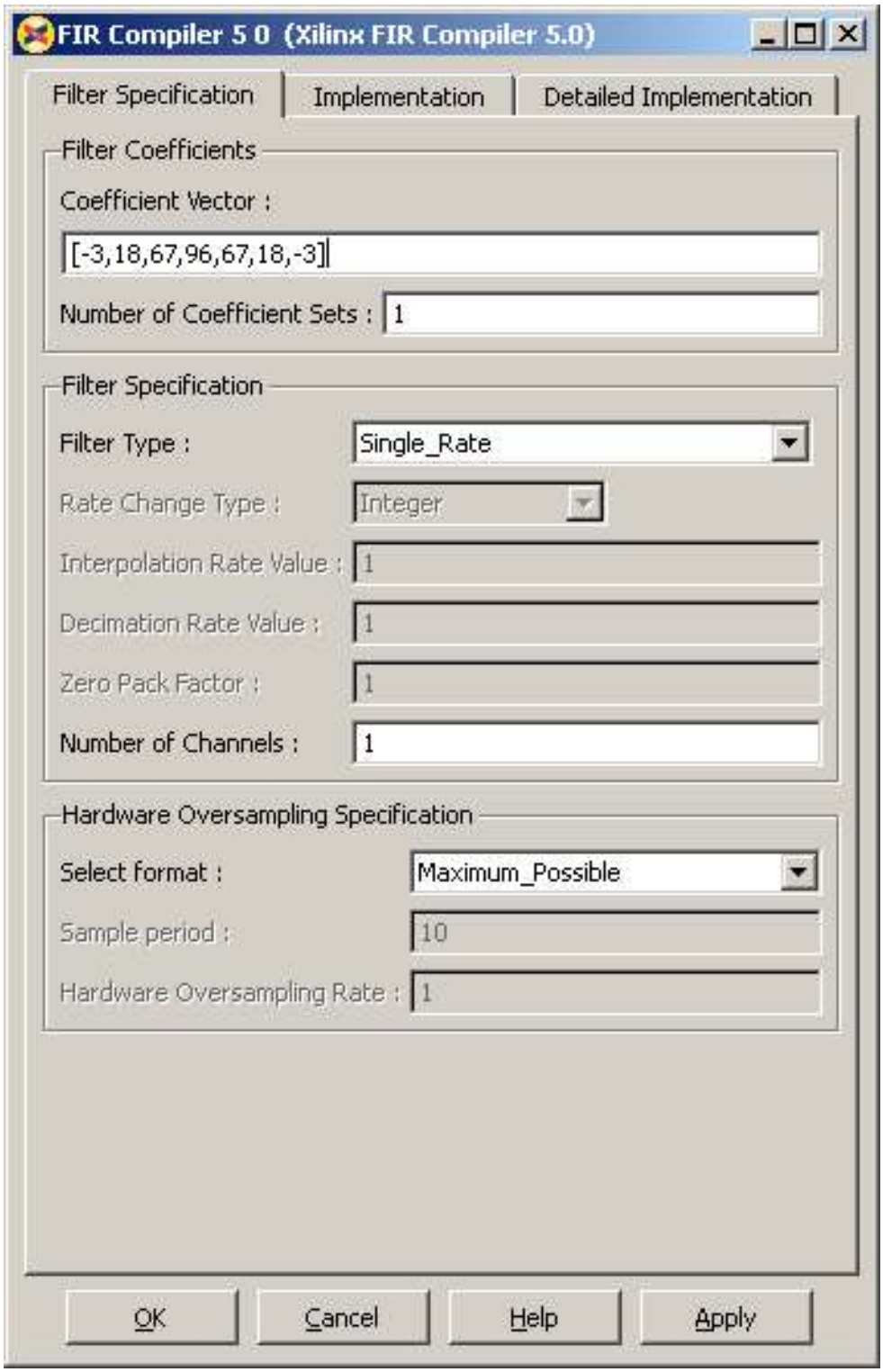
- The input is assigned to switch 0 (pin 'A10').



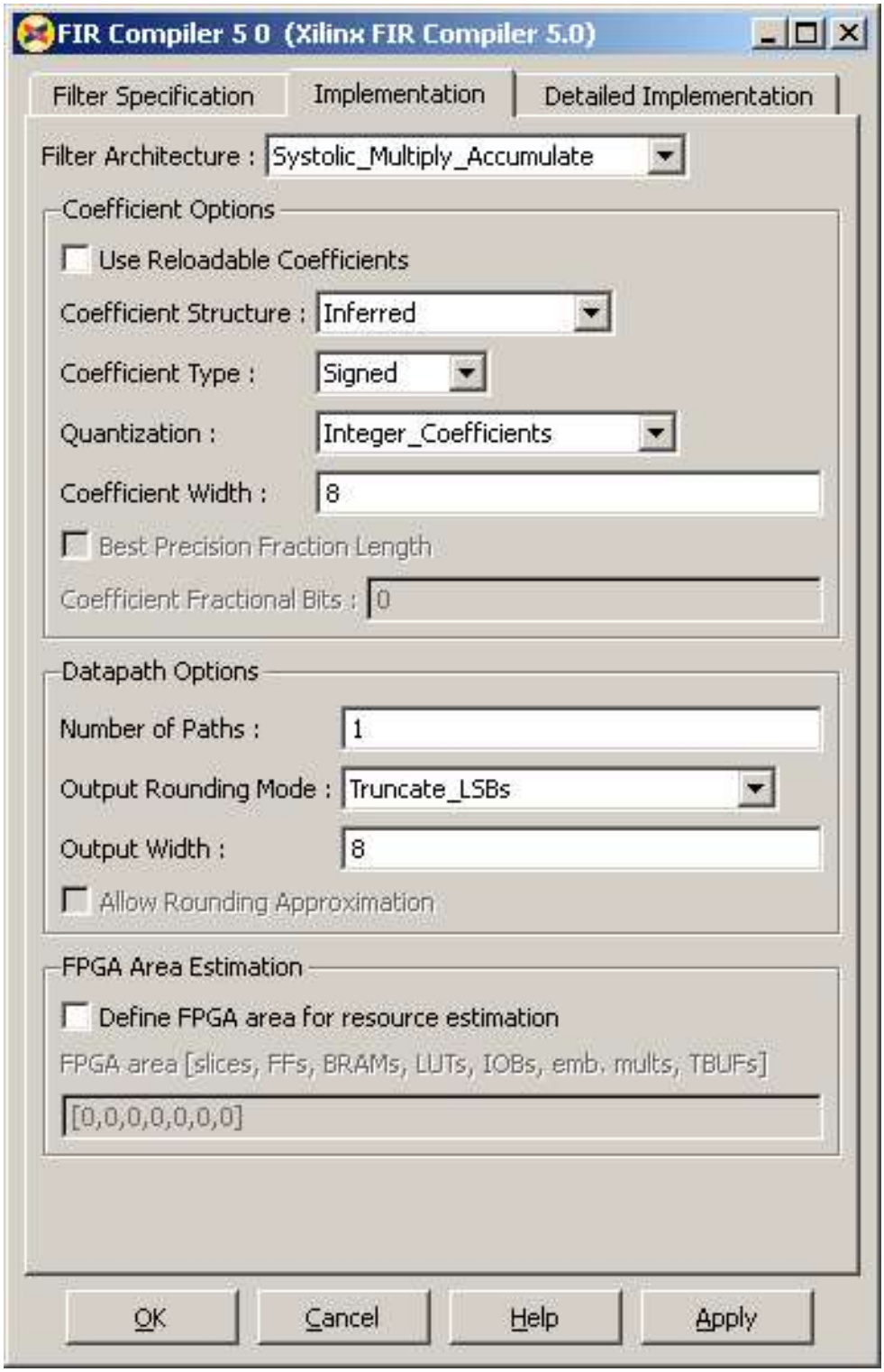
- The counter is a free running (least expensive) 8 bit counter with signed integer output. Every sampling period the integer 16 is added to the counter. Thus the sawtooth's frequency becomes 16 / 256 * 10 MHz = 625 KHz.


Counter (Xilinx Counter)
Hardware notes: Free running counters are the least expensive in hardware. A count limited counter is implemented by combining a counter with a comparator.
Basic
Advanced
Implementation
Counter type:
Free running
Count limited
Count to value
Inf
Count direction:
Up
Down
Up/Down
Initial value
0
Step
16
Output Precision
Output type:
Signed (2's comp)
Unsigned
Number of bits
8
Binary point
0
Optional Ports
Provide load port
Provide synchronous reset port
Provide enable port
Explicit Sample Period
Sample period source:
Explicit
Inferred from inputs
Explicit period
100e-9
OK
Cancel
Help
Apply

- The lowpass filter is a 7-tap Kaiser windowed FIR. Add the coefficients to the FIR compiler block properties.



- In the implementation tab set the Coefficient Options and the Datapath Options to 8 bits. The filter implements 8 x 8 multiplications (signed). The sum is truncated to 8 bits to fit Gateway Out block.



- Select Generate from the System Generator and synthesize the design in ISE.
- Select Analyze Design Using ChipScope with right mouse button and click on Run.
- Set up ChipScope and trigger data capture with switch_0 (from '1' to '0' like in the Simulink simulation).

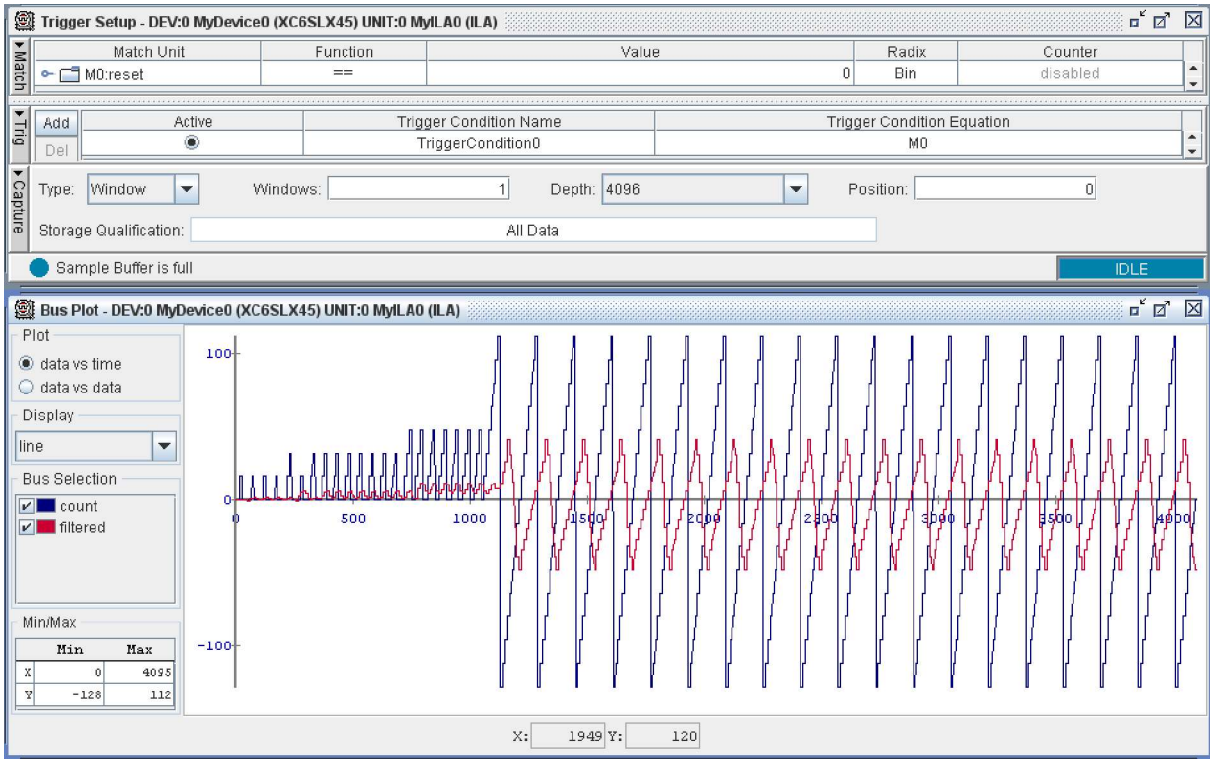


Figure 1.123: ChipScope result from the running FPGA

Your results should look similar to the diagram above.

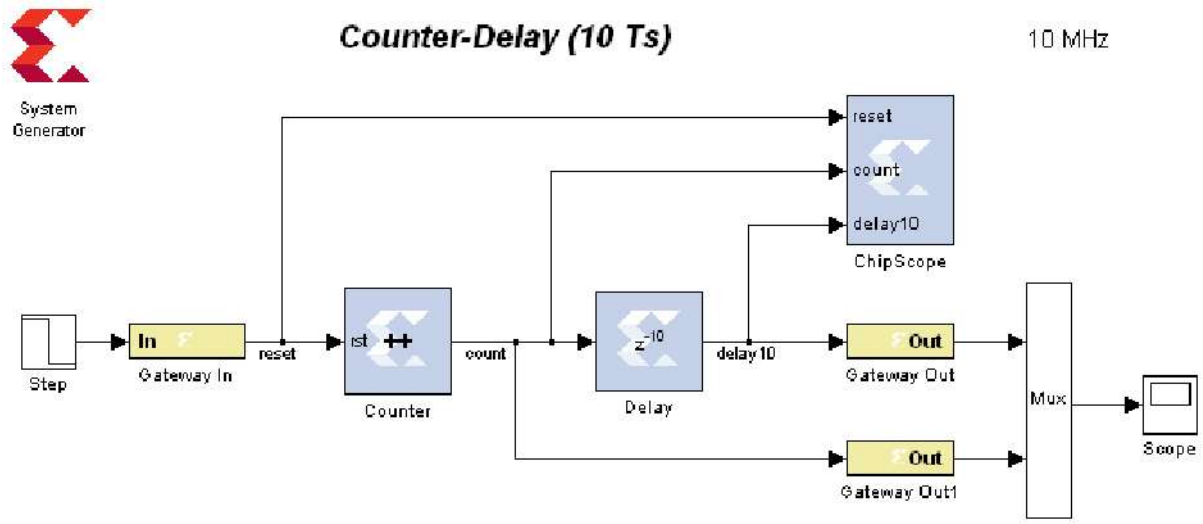
- Explain the "distortions" in the beginning of the diagram. What is the physical reason for this effect?

Lab #11: Counter-Delay

16.3 Counter-Delay Lab

A counter-delay application should be analyzed by ChipScope on a FPGA system at 10MHz sampling period. An 8-bit unsigned counter should be delayed by 10 sampling periods. This requires 10 8-bit registers which will be created automatically by the System Generator.

- Create a new folder with name "cntdelay" and set up a model with the same name. It should look like the following model.



- Parameterize the System Generator block by the following data:
  Part: Spartan6 xc6slx45–3csg324
  FPGA Clock period (ns): 10
  Clock pin location: L15
  Simulink system period (sec): 100e–9 (this means 10MHz sampling clock)

- Gateway In block:
  Output type: boolean
  Sample period: 100e–9
  [Implementation] IOB pad locations: {'A10'} (switch 0)

- Counter block:
  Output type: Unsigned, Number of bits: 8, Binary point: 0

[x] Provide synchronous reset port
Explicit Sample Period: 100e–9
[Implementation]: [x] Use behavioral HDL

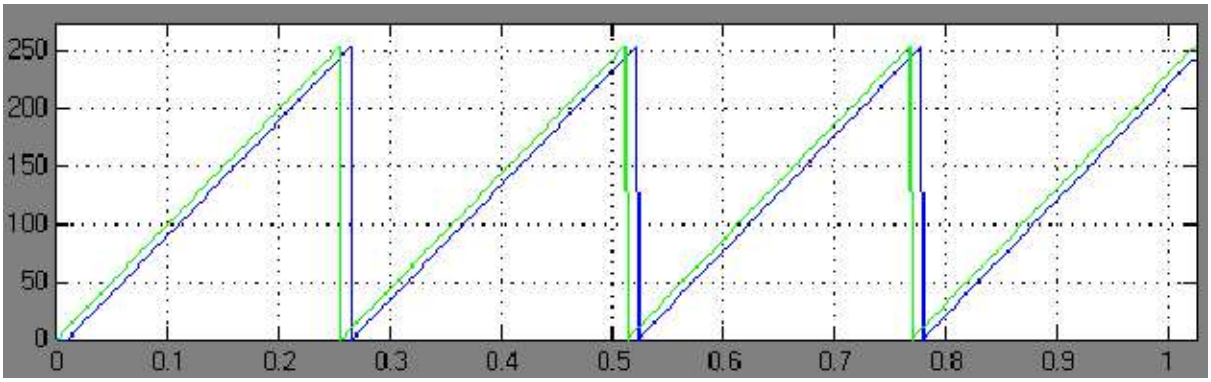
- Delay block:
  Latency: 10

- Name all signals entering the ChipScope block for better readability later in ChipScope.

- ChipScope block:
  Number of trigger ports: 1
  Number of data ports: 2
  Depth of capture buffer: 1024 (or bigger)

- Select in the Simulink Configurations Configuration Parameters
  Stop Time: 1024*100e–9
  Solver: discrete (no continuous states)
  Select OK

- Simulate in Simulink and you should observe the following diagram.

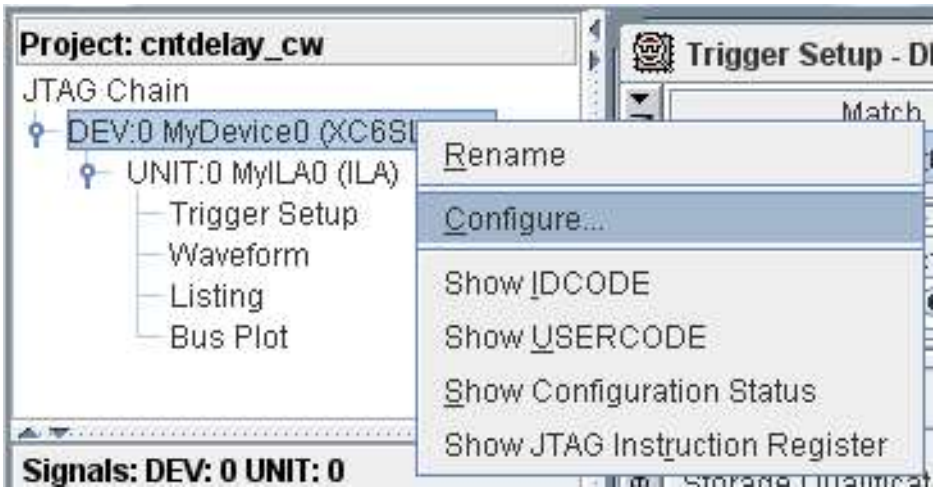


You should see the same behavior with ChipScope on the real system.

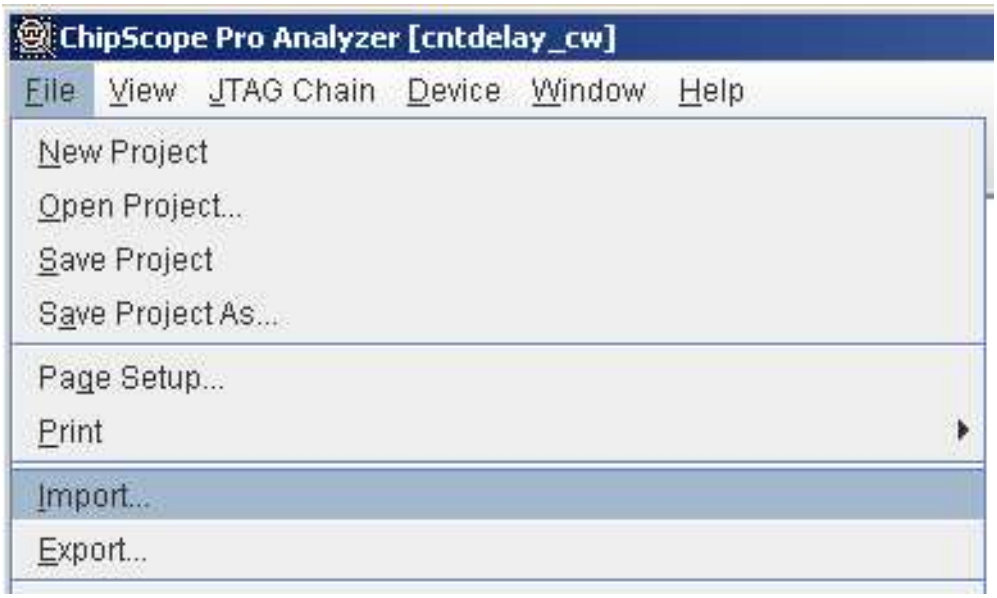
- Select Generate in the System Generator – and have a break...

- Open ISE and close any open project.
- Navigate to the .\cntdelay\netlist folder and open "count_delay_cw.xise".
- In the Processes window select
  Implement Design –> Right mouse button –> run
- Locate the Post-PAR Static Timing Report and write in the possible maximum

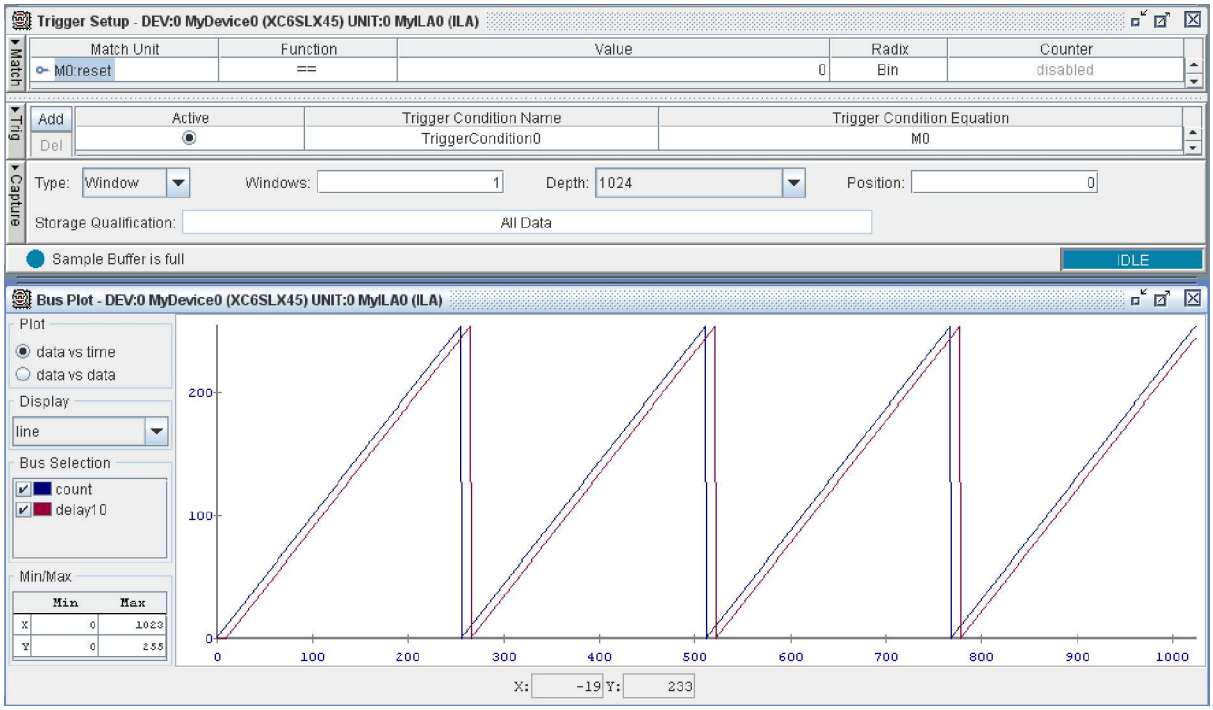
  clock frequency: _____ (MHz)
  Is the timing constraint satisfied?
- Select
  Analyze Design Using ChipScope –> RIght mouse button –> run
  ... wait for ChipScope coming up...
- In ChipScope select from the main menu
  JTAG Chain –> Digilent USB JTAG Cable...
  (or Open Plug-in... for older versions of ISE)
- Verify that the device name XC6LX45 appears and select OK.
- Select DEV:0 –> Configure and choose the correct xxx.bit file.



- Select File –> Import...
  and choose the cntdelay.cdc definition file.



- Open the trigger setup and the bus plot window.
- Set the trigger value to 0 (reset off)
- In the Bus Plot window select both "count" and "delay10" and choose different colors for plotting.
- Slide switch 0 to the "on" position and arm the trigger (▶ button)
- Slide the switch 0 to the "off" position and inspect the bus plot window.



Lab #12: Complex Multiplier

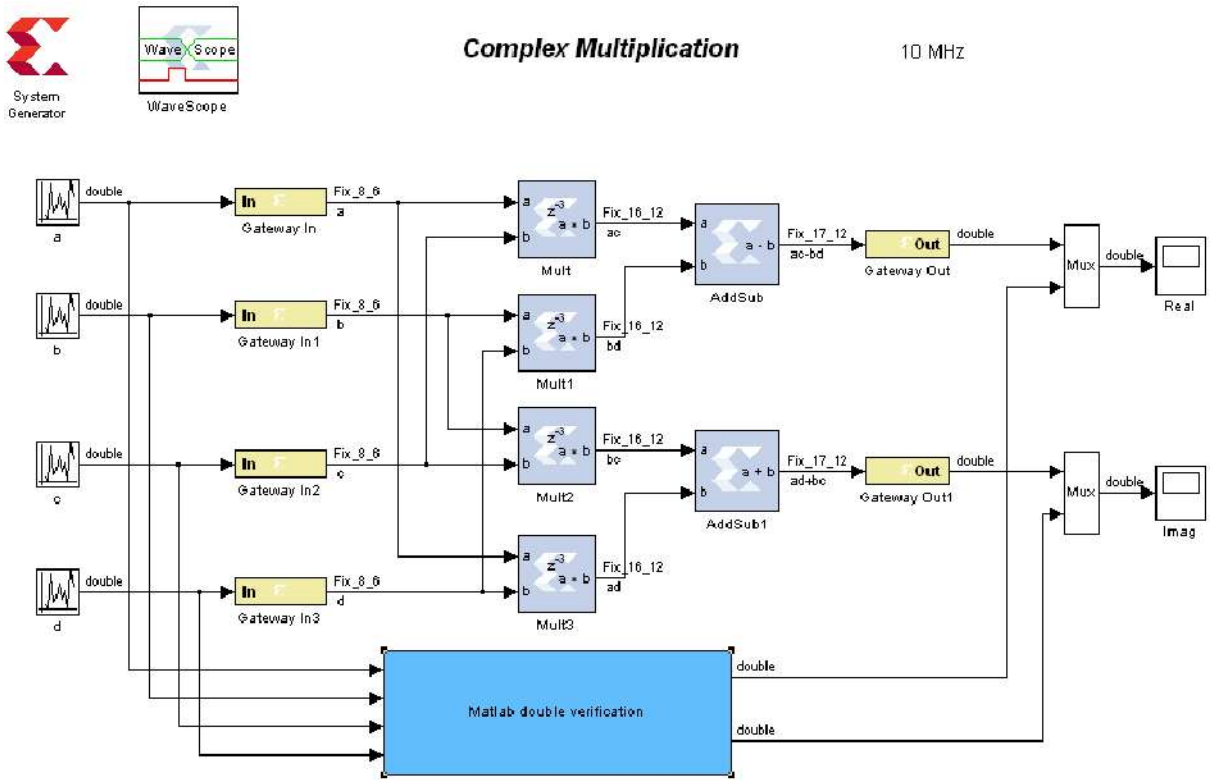
16.4 Complex Multiplier Lab

Complex multiplications a basic operation for many algorithms (i.e. FFT). A complex multiplier in different number formats should be implemented in System Generator. The multiplication with $x = a + jb$ and $y = c + jd$ gives

$$xy = (a + jb)(c + jd) = ac - bd + j(ad + bc). \quad (1.187)$$

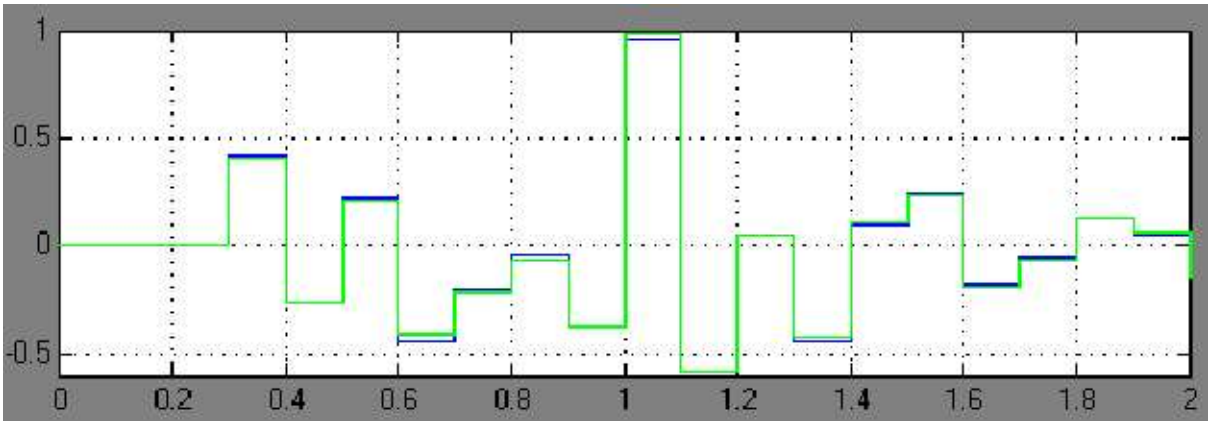
A complex multiplier in hardware thus comprises four real multiplications, one subtraction and one adder. The complex multiplier can be used as a subsystem to create advanced algorithms.

- Create a new folder and model "comp_mult". Implement the algorithm (1.187). Provide 8 bit input signal width (int_8.6 format). You can accept maximum precision settings (default) for all subsequent blocks.



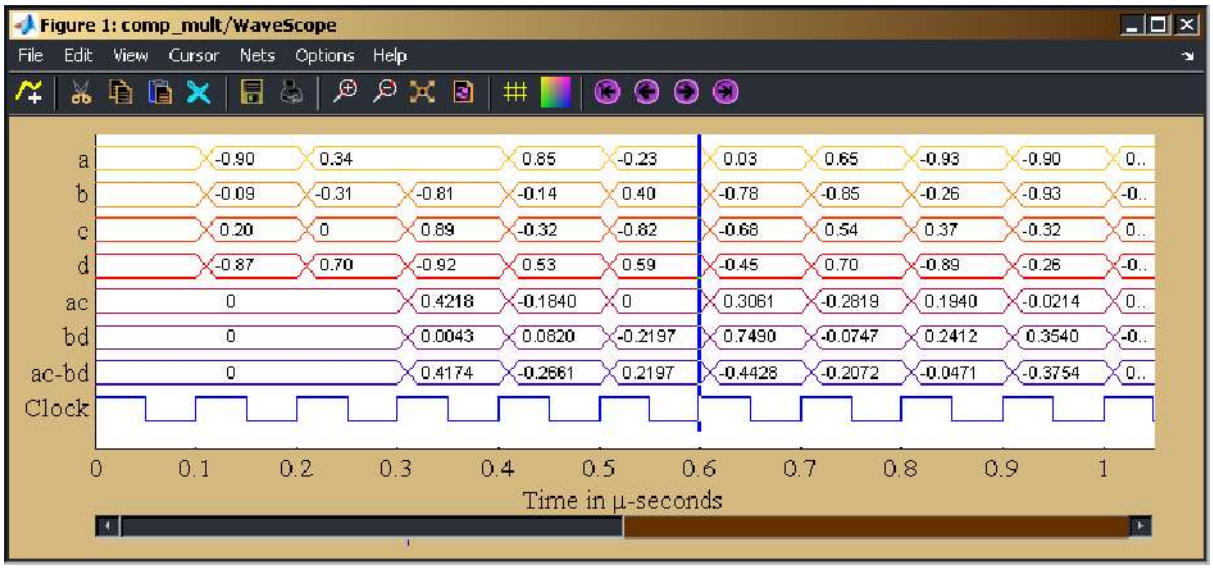
- Make the usual setting in the System Generator block and provide an Simulink system period of 100e.9 (10 MHz). The simulation time should be 20 clock cycles.

- Simulation inputs are (double) uniform random numbers in the range -1...+1.
- Add a WaveScope block for advanced display of simulation results. Include all signals of interest to the WaveScope. The WaveScope is located in
  Xilinx Blockset –> Tools.
- Provide a Simulink (double) verification block to compare the results with the FPGA version (masked block in the above model). This block comprises only Simulink elements!
- Verify proper function of the system. The Simulink Scopes should give a similar output as the following diagram. Only the real output is shown.



- Explain the differences between the FPGA output and the Simulink verification block.

- Inspect the WaveScope output and verify the computed values. The WaveScope output should look similar to the following diagram.

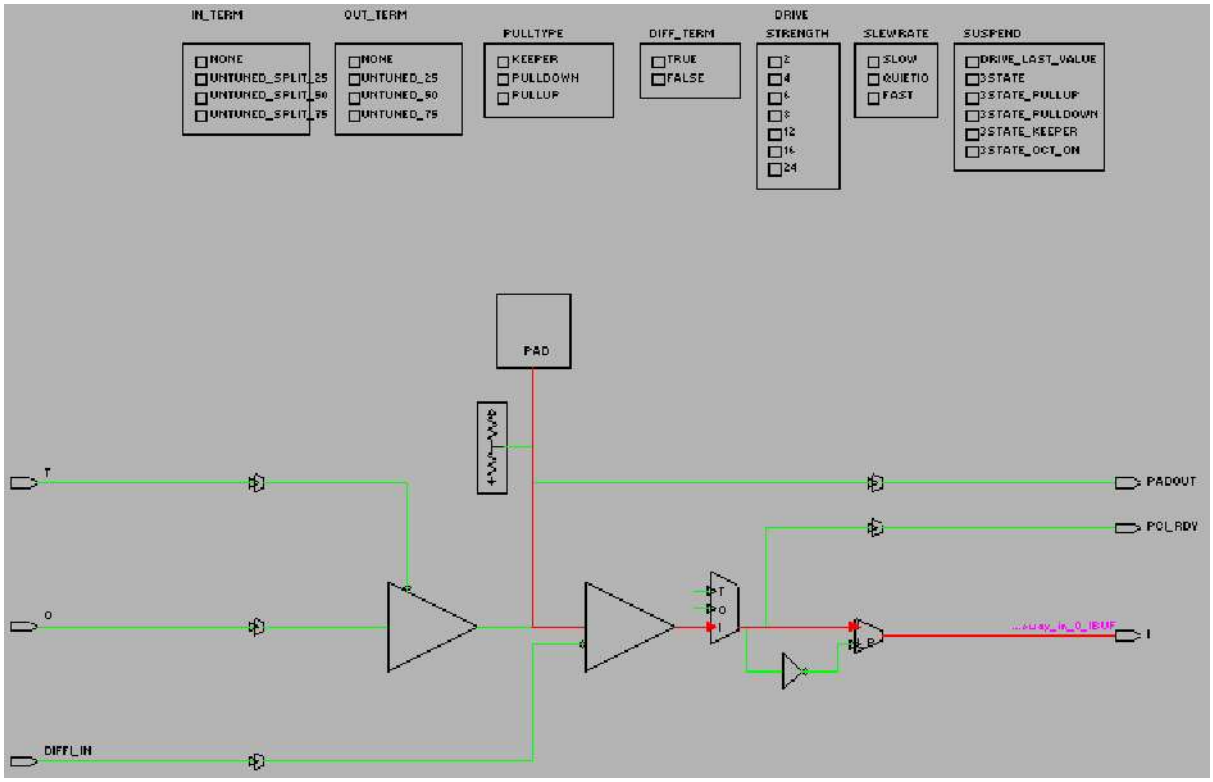


The numbers in the diagram depend on the seeds of the random number generators.

16.5 Number Format Optimization and Hardware

The precision of the Multiplier can be improved by selecting suitable number formats.

- Change the input signal format (Gateway In blocks) to 16 bits / 14 fractional bits (int_16.14).
- Change the output width to the same format (int_16.14) with the least hardware effort (truncation without rounding and without saturation). With this format selections input and output data have the same format. and a general purpose complex multiplier is created.
- Verify the port formats and perform simulation to prove that not visible errors occur.
- Generate the system in inspect the created hardware with the FPGA Editor.
- Start with the input signal from the first Gateway In block and follow the signals through the FPGA (see below).



- Follow this signal to the multiplier. What type of device is used for multiplication?

_____

- How many slices are required for subtractor and the adder?

_____

- What type of slices are used for this purpose?

_____

- What is the maximum of complex multiplications per second? Hint: use the Post-PAR Static Timing Report to answer this question

_____

***

17 CORDIC Algorithm

Multipliers and accumulators are available on modern hardware which allow almost any speed of real-time digital data processing. However, many application call for operations like $\sqrt{x^2 + y^2}$, cosine, sine, tangent, divisions, or logarithmic functions.

The question arises how these function can be realized in hardware. Look-up tables and interpolations are possible but this approach requires lots of memory and involves additional interpolation calculations. A very fast, efficient and precise solution is the CORDIC (**CO**rdinate **R**otation **DI**gital **C**omputer) algorithm. CORDIC allows to calculate many scientific functions by a sequence of shift, add and subtract operations (and one multiplication) These resources are almost unlimited on modern FPGAs.

It was derived from Jack E. Volder in 1959 and is widely used in digital systems. The basic operation is a rotation operation in the cartesian plane (the complex plane with real and imaginary axis can be used also). Any given point in the cartesian plane denoted by x_1 and x_2 can be rotated by an angle θ

$$\begin{bmatrix} x_2 \\ y_2 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} . \tag{1.188}$$

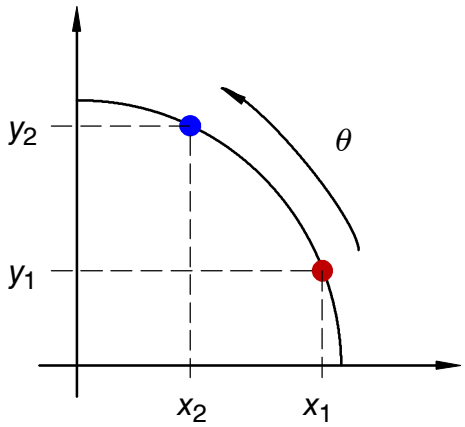


Figure 1.124: Rotation of a point in the cartesian plane

Since the determinant of the rotation matrix in (1.188) is one, the distance from the origin to the two points remains the same.

The rotation equations can be written as

$$x_2 = x_1 \cos\theta - y_1 \sin\theta = \cos\theta \left(x_1 - y_1 \tan\theta \right) , \tag{1.189}$$

$$y_2 = x_1 \sin\theta + y_1 \cos\theta = \cos\theta \left(y_1 + x_1 \tan\theta \right) . \tag{1.190}$$

Dropping the $\cos\theta$ term lead to the *pseudo-rotation*

$$\hat{x}_2 = x_1 - y_1 \tan\theta\,, \tag{1.191}$$

$$\hat{y}_2 = y_1 + x_1 \tan\theta\,. \tag{1.192}$$

Pseudo-rotations are correct with respect to the angle but the length of $\hat{x}_2$ and $\hat{y}_2$ are larger than the correct values (they are scaled by $1 / \cos\theta$). This will be corrected (easily) later.

Things are becoming elegant if we restrict ourselves to use only values for $\tan\theta$ which are powers of 2. In these cases the multiplication with $\tan\theta$ reduces to a binary shift operation.

i	$\tan\theta^i$	θ^i (degrees)
0	1	45.0
1	0.5	26.565
2	0.25	14.036
3	0.125	7.125
4	0.0625	3.576
5	.	.
.	.	.

If we want to rotate by an arbitrary angle θ it will achieved by a series of smaller rotations. If CORDIC is executed on hardware the number of iterations will be fixed. The result needs to be scaled by a constant since we dropped the cosines in the pseudo-rotations.

$$\frac{1}{K_n} = \frac{1}{\prod_{i=0}^{n-1} \frac{1}{\cos\theta^{(i)}}} = \prod_{i=0}^{n-1} \cos\theta^{(i)}\,. \tag{1.193}$$

The scaling factor is also given by

$$K_n = \prod_{i=o}^{n-1} \frac{1}{\cos\theta^{(i)}} = \prod_{i=0}^{n-1} \sqrt{1 + 2^{(-2i)}}\,. \tag{1.194}$$

If for instance three iterations are used, the result needs to be scaled by

$$\frac{1}{K_3} = \cos\theta^0 \cos\theta^1 \cos\theta^2 = 0.6136\,. \tag{1.195}$$

The worst case error is determined by the angle of the last iteration step

$$\theta_{err} = \operatorname{atan}\left(2^{-(n-1)}\right)\,. \tag{1.196}$$

If we want to rotate by 50° with only four iterations ($n = 4$) the result would look like this

i	$\Delta\theta$
0	+ 45.0
1	+ 26.565
2	−14.036
3	−7.125
sum	+ 50.404

The error is less than the value for $i = 3$.

The CORDIC algorithm can be written as follows

$$x^{(i+1)} = x^{(i)} - d_i\left(2^{-i} y^{(i)}\right), \tag{1.197}$$

$$y^{(i+1)} = y^{(i)} + d_i\left(2^{-i} x^{(i)}\right). \tag{1.198}$$

Where d_i is the *decision operator* with $d_i = \pm 1$. In order to determine d_i an angle accumulator is required

$$z^{(i+1)} = z^{(i)} - d_i\theta^{(i)}. \tag{1.199}$$

Different modes exist to set the decision variable d_i.

17.1 Rotation Mode

In rotation mode the input vector is rotated by the desired angle by performing CORDIC micro-rotations.

The value $z^{(0)}$ is set to θ for initialization. If $z^{(i)} > 0$ then $d_i = 1$ else d_i = -1. This means that the sign of $z^{(i)}$ determines the decision variable

$$d_i = \text{sign}\, z^{(i)}. \tag{1.200}$$

Rotation mode is applied, whenever the angle of rotation is known, i.e. when a sine or cosines of an angle has to be computed. See lab in section 17.4 for reference.

If we want to rotate 38° , the following sequence of micro-rotations occur:

```
i  d_i   theta_i      z_i
------------------------
0    1   45.0000  38.0000
1   -1  -26.5651  -7.0000
2    1   14.0362  19.5651
3    1    7.1250   5.5288
4   -1   -3.5763  -1.5962
```

```
5     1     1.7899    1.9801
6     1     0.8952    0.1902
7    -1    -0.4476   -0.7050
8    -1    -0.2238   -0.2573
9    -1    -0.1119   -0.0335
```

It can be seen that d_i depends only on the sign of $\theta^{(i)}$. The angle accumulator $z^{(i)}$ tends to zero.

17.2 Vectoring Mode

In vectoring mode the input vector is rotated onto the x-axis. This is required when the angle in not known, i.e. for an atan operation.

The angle accumulator is set to zero ($z^{(0)} = 0$). The decision operator depends on the sign of $y^{(i)}$

$$d_i = - \text{ sign } y^{(i)} . \tag{1.201}$$

In this way the vector is rotated towards the angle zero.

If we want to rotate the vector (1, 2) onto to x-axis, the following sequence of micro-rotations occur:

```
  i  d_i   theta_i      z_i      x_i      y_i
-------------------------------------------
  0   -1  -45.0000   0.0000   1.0000   2.0000
  1   -1  -26.5651  45.0000   3.0000   1.0000
  2    1   14.0362  71.5651   3.5000  -0.5000
  3   -1   -7.1250  57.5288   3.6250   0.3750
  4    1    3.5763  64.6538   3.6719  -0.0781
  5   -1   -1.7899  61.0775   3.6768   0.1514
  6   -1   -0.8952  62.8674   3.6815   0.0365
  7    1    0.4476  63.7626   3.6821  -0.0211
  8   -1   -0.2238  63.3150   3.6822   0.0077
  9    1    0.1119  63.5388   3.6823  -0.0067
```

It can be seen that d_i depends only on the sign of $y^{(i)}$. The angle accumulator $z^{(i)}$ shows the value $\text{atan}(2/1) = 63.43°$.

17.3 Hardware Implementation

CORDIC is ideal for hardware implementation. Depending on the required accuracy the number of CORDIC micro-rotations can be determined. This involves the pre-computation of the factor $1 / K_n$. Since a micro-rotation requires only multiplications with a power of 2, the multiplications are carried out using binary shift operations and adders/subtracters.

Additional logic is necessary to derive the decision operator d_i from the micro-rotation stages. The CORDIC block diagram is shown below.

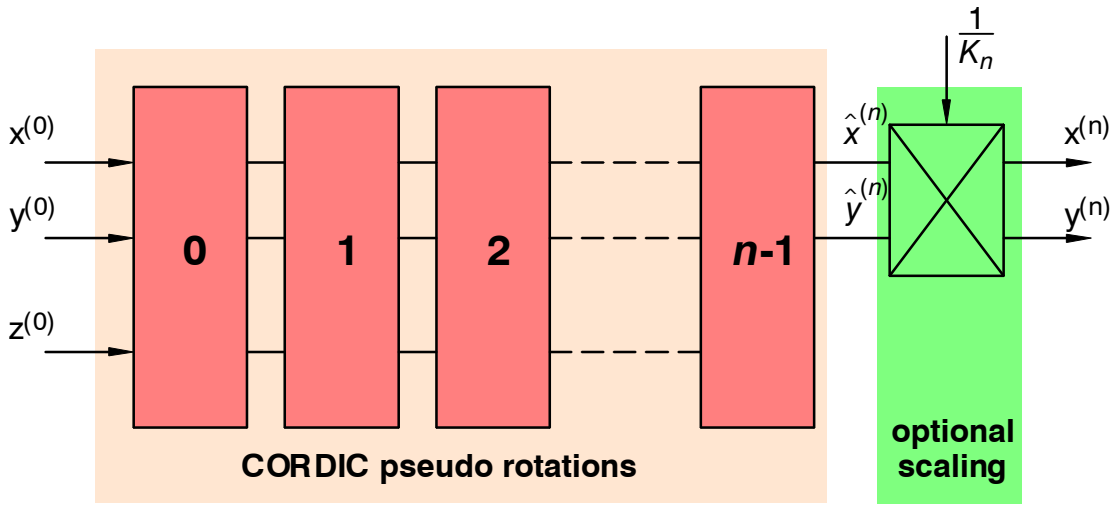


Figure 1.125: CORDIC hardware structure

Lab #13: CORDIC sine/cosine

17.4 CORDIC Sine / Cosine Computation

Sine and cosine should be calculated by a $n = 10$ stage CORDIC.

- Write a Matlab m-file to implement the CORDIC algorithm for simultaneous computation of sine and cosine.
- Compute K_n for $n = 10$. Set $x^{(0)} = 1 / K_n$ and $y^{(0)} = 0$.
- Precompute a lookup table for $\theta^{(i)}$.
- Complete the m-file to implement the algorithm according to (1.197)...(1.199) by applying rotation mode as of section 17.1.
- Compare your result with the exact result from Matlab's sin() and cos().

Lab #14: CORDIC atan

17.5 CORDIC Arcus Tangent Computation

The arcus tangent should be calculated by a $n = 10$ stage CORDIC. Vectoring mode applies here.

- Write a Matlab m-file to implement the CORDIC algorithm for computing the arcus tangent.
- Set $x^{(0)} = 1.0$ and $y^{(0)} = y$.
- Precompute a lookup table for $\theta^{(i)}$.
- Complete the m-file to implement the algorithm according to (1.197)...(1.199) as well as (1.201) (vectoring mode) as of section 17.2.
- Compare your result with the exact result from Matlab's atan().

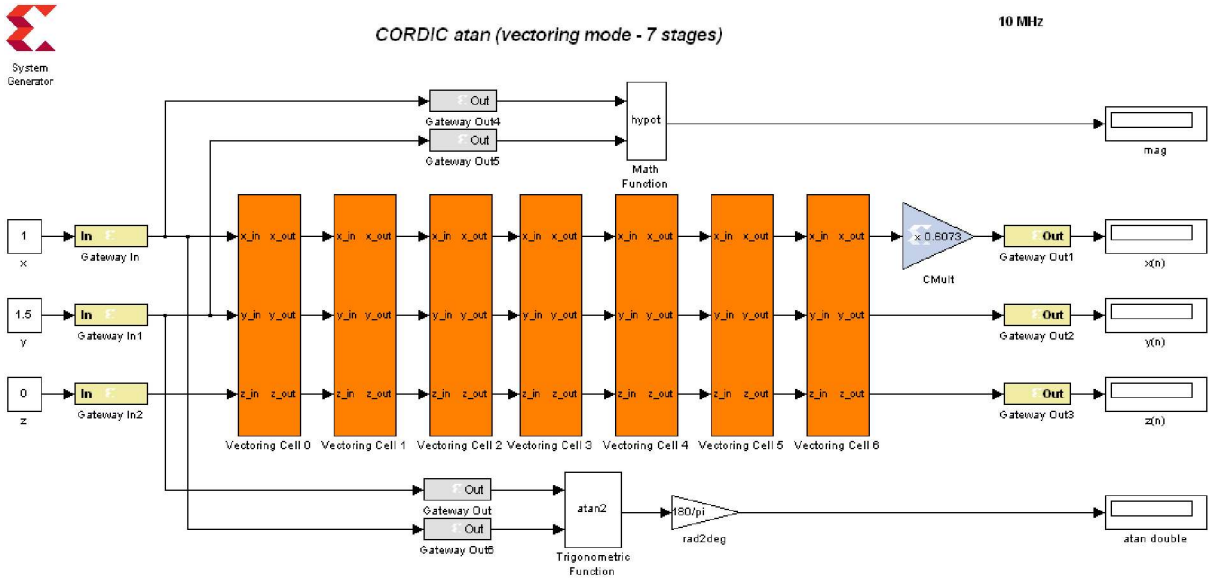
***

Lab #15: FPGA atan/magnitude

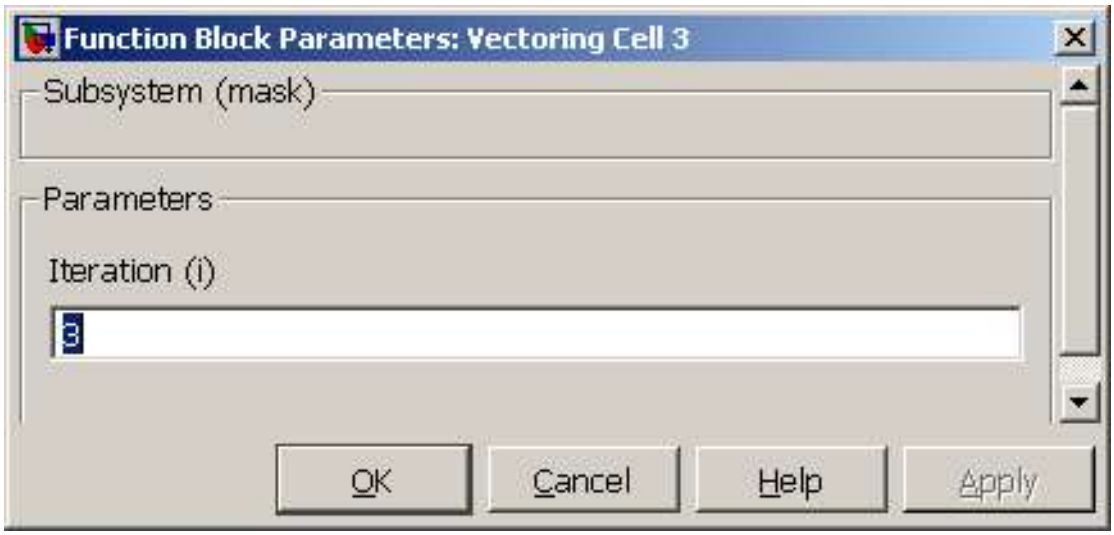
17.6 FPGA CORDIC Arcus Tangent / Magnitude Computation

The arcus tangent and magnitude should be calculated by a $n = 7$ stage CORDIC on the FPGA using System Generator. Vectoring mode applies here since the angle is unknown.

Since all CORDIC cells are identical a vectoring mode cell should be created as a *masked block* with one parameter i. The complete system is shown below.

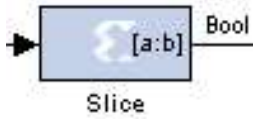


The masked blocks for a vectoring mode cell has a parameter i, which is accessible from the subsystem mask dialog (by double-clicking on the subsystem). This parameter is defined in the **Edit Mask** dialog for the subsystem. The parameter i determine all cell specific functions. Extending the number of cells becomes very easy by just copying the required number of blocks.

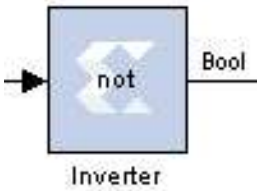


Here is a list of required blocks for the subsystem to implement vectoring mode in one stage of the CORDIC algorithm.

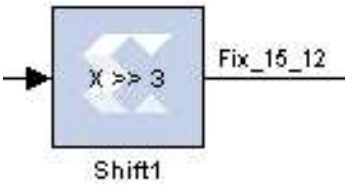
Access the sign bit:



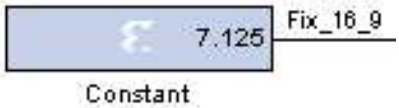
Invert a bit:



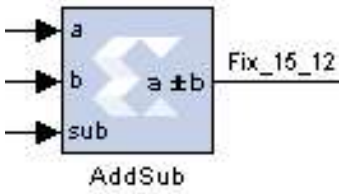
Shift (division by powers of 2) Note that the number of shifts is determined by the mask parameter i:



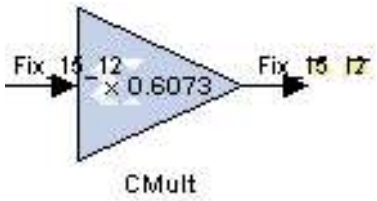
Constant (for angle $\theta^{(i)}$) Note that the angle is commuted from mask parameter i by the Simulink *atand* function:



Adder / Subtracter:



Scaling (not inside the subsystem – for obtaining the correct value of the magnitude):



- Create a Matlab/Simulink model "cordic_atan".
- Code a masked subsystem for one vectoring cell parametrized by one parameter *i*. The data format is – of course – fixed-point. The data width for all inputs and outputs is 16 bits.
- Test your system with only one cell (45° angle).
- Copy the subsystem 6 times and make the appropriate connections.
- Verify the results by comparing them to the Matlab double calculations. Give reasons for the differences.

***

Lab #16: FPGA sine/cosine

17.7 FPGA CORDIC Sine / Cosine Computation

The sine/cosine should be calculated by a $n = 7$ stage CORDIC on the FPGA using System Generator. Rotation mode applies here is $z^{(0)}$ initialized by the angle θ.

Since all CORDIC cells are identical a rotation mode cell should be created as a *masked block* with one parameter k. The complete system is shown below.

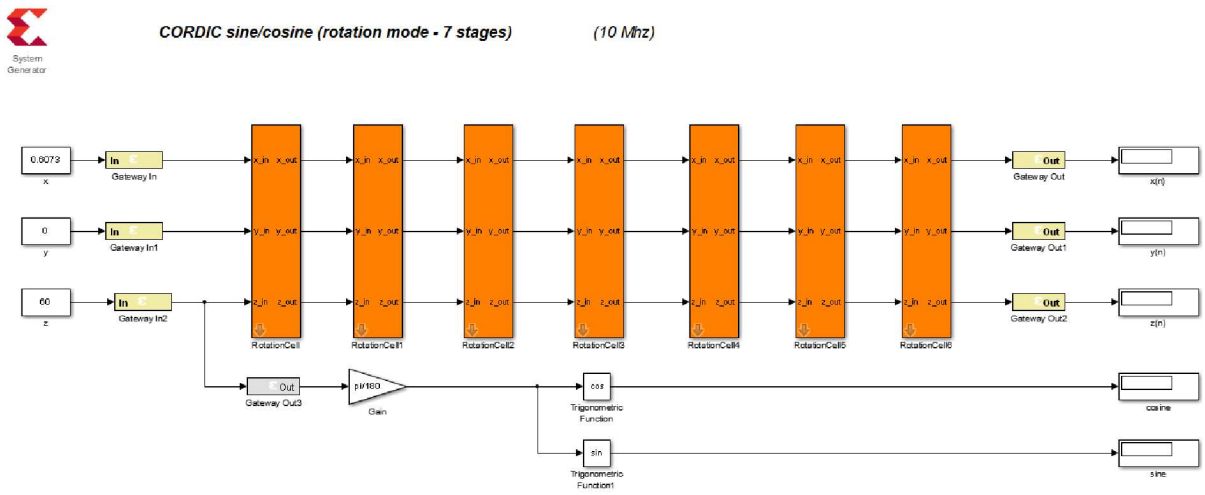


Figure 1.126: CORDIC sine / cosine computation for FPGA

- Create a *rotation* cell as a masked Simulink block with an editable parameter k. Select carefully appropriate fractional bits for maximum accuracy (signals and coefficients should have 16 bit formats).
- Copy the block 6 times and adjust k with the right stage number k.
- Verify the computation by comparing the CORDIC results with the exact Matlab solutions.

This lab has to be reported as part of the requirements for ES-MEDL. The report must include a detailed description of the design.

***

17.8 Generalized CORDIC

CORDIC allows to compute many algebraic functions – not to say almost everything required for technical computing. So far we have used the so-called **circular rotation** which allows to compute sine, cosine, atan, or magnitudes of a vector.

With other rotation types (**linear rotation, hyperbolic rotation**) several other functions can be commuted with the same efficiency. Let us summarize the circular rotation (rotating a vector by a given angle (on a circle).

In **rotation mode** for **circular rotation** the decision variable d_i depends on the condition

$$d_i = \text{sign}\,(z^{(i)})\,. \tag{1.202}$$

After n iterations we obtain

$$x^{(n)} \approx K_n\left(x^{(0)}\cos z^{(0)} - y^{(0)}\sin z^{(0)}\right), \tag{1.203}$$

$$y^{(n)} \approx K_n\left(x^{(0)}\sin z^{(0)} + y^{(0)}\cos z^{(0)}\right), \tag{1.204}$$

$$z^{(n)} \approx 0\,. \tag{1.205}$$

A graphical representation is shown in the next figure.

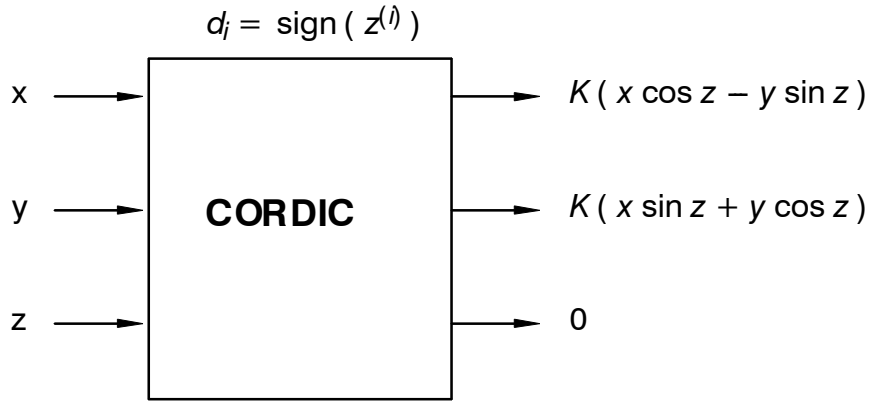


Figure 1.127: Rotation mode in the circular coordinate system

This mode can be used to compute sine and cosine by setting
$x = 1/K,\ y = 0.$

The outputs then become $\cos z$ and $\sin z$.

In **vectoring mode** for **circular rotation** the decision variable d_i depends on the condition

$$d_i = -\ \text{sign}\,(y^{(i)})\,. \tag{1.206}$$

After n iterations we obtain

$$x^{(n)} \approx K_n \sqrt{\left(x^{(0)}\right)^2 + \left(y^{(0)}\right)^2}\,, \tag{1.207}$$

$$y^{(n)} \approx 0\,, \tag{1.208}$$

$$z^{(n)} \approx z^{(0)} + \operatorname{atan}\left(\frac{y^{(0)}}{x^{(0)}}\right). \tag{1.209}$$

A graphical representation is shown in the next figure.

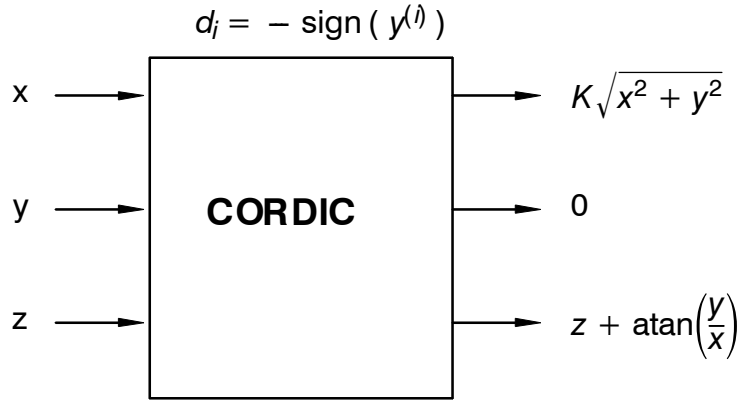


Figure 1.128: Vectoring mode in the circular coordinate system

This mode can be used to compute the magnitude

$$\sqrt{x^2 + y^2} \tag{1.210}$$

and the arcus tangent $\operatorname{atan} y$
by setting $x = 1$ and $z = 0$.

The **circular rotation** principle is shown the in fig. 1.129.

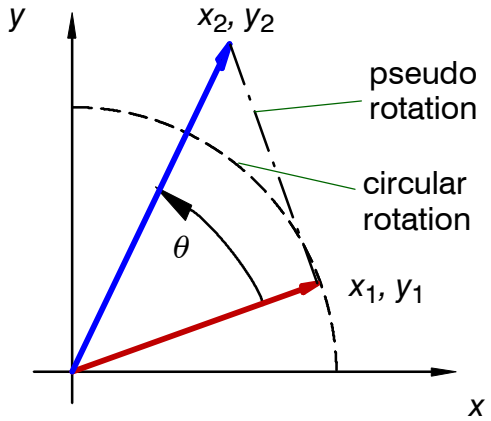


Figure 1.129: Circular rotation

The scaling factor for circular rotation is

$$K_n = \prod_{i=0}^{n-1} \sqrt{1 + 2^{-2i}} \; . \tag{1.211}$$

17.8.1 Coordinate Systems for Other Mathematical Functions

While the rotational coordinate system allows the calculation of trigonometric function and the square root, other coordinate system are required to calculate important additional mathematical functions.

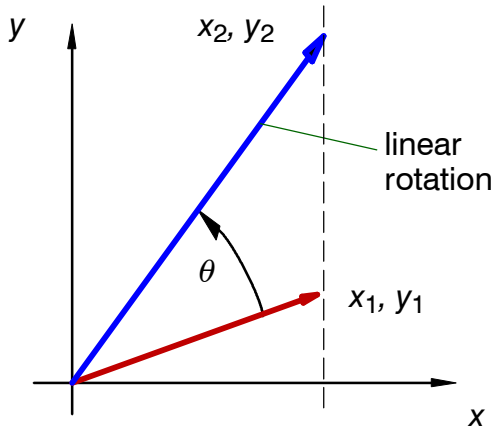


Figure 1.130: Linear rotation

The linear rotation coordinate system allows multiplication and – more important – division operations. The direct division operation in hardware costs a lot of resources while CORDIC offers a cheaper solution. No scaling factor is required ($K_n = 1$).

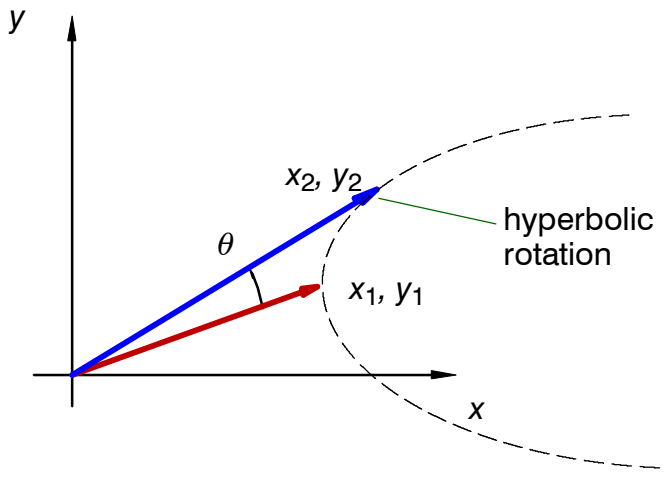


Figure 1.131: Hyperbolic rotation

The hyperbolic ration coordinate system is equivalent to the circular rotation but for the hyperbolic sinh, cosh, and atanh. The scaling factor

$$K_n^* = \prod_{i=0}^{n-1} \sqrt{1 - 2^{-2i}} \; . \tag{1.212}$$

applies. As $n \to \infty$ the factor becomes $K_n{}^* = 0.82816$. Since $K_n{}^*$ is less than 1, each microrotation results in a vector that is smaller as the exact rotation.

A common notation of the so-called **generalized CORDIC equations** exist.

$$x^{(i+1)} = x^{(i)} - \mu d_i \left(2^{-i} y^{(i)}\right) , \tag{1.213}$$

$$y^{(i+1)} = y^{(i)} + d_i \left(2^{-i} x^{(i)}\right) . \tag{1.214}$$

$$z^{(i+1)} = z^{(i)} - d_i e^{(i)} . \tag{1.215}$$

The parameters for the various orations are as follows:

Rotation	μ	$e^{(i)}$
circular	1	$\text{atan}\, 2^{-i}$
linear	0	2^{-i}
hyperbolic	–1	$\text{atanh}\, 2^{-i}$

The linear coordinate system CORDIC equations are thus

$$x^{(i+1)} = x^{(i)} = \text{const.} , \tag{1.216}$$

$$y^{(i+1)} = y^{(i)} + d_i \left(2^{-i} x^{(i)}\right) . \tag{1.217}$$

$$z^{(i+1)} = z^{(i)} - d_i 2^{-i} . \tag{1.218}$$

The linear coordinate rotation performs multiplication (rotation mode) and division (vectoring mode).

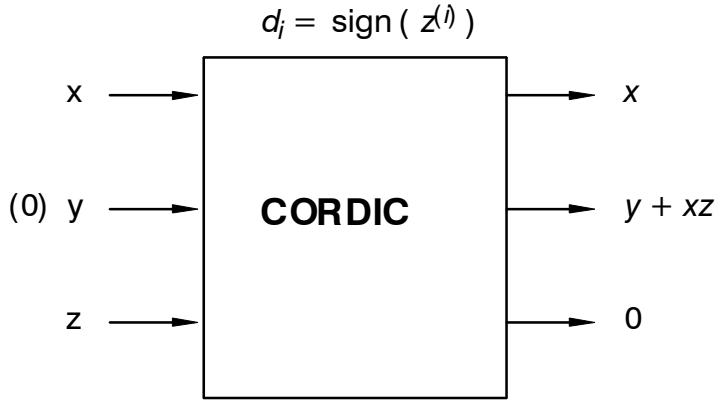


Figure 1.132: Rotation mode in the linear coordinate system

It it obvious that multiplication implies $y = 0$.

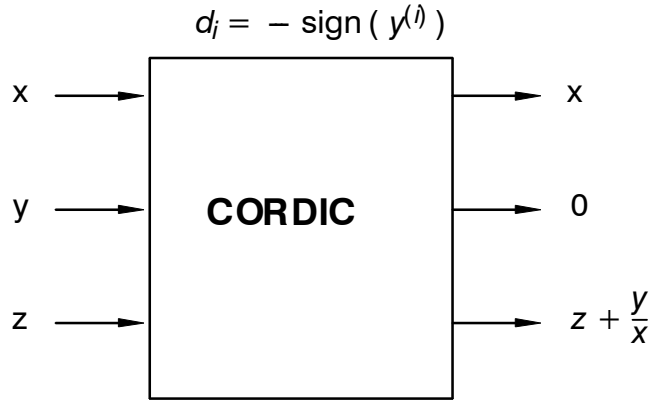


Figure 1.133: Vectoring mode in the linear coordinate system

For the division operation initialize $z = 0$.

Lab #17: CORDIC mult/div

17.9 CORDIC Multiplication and Division

Multiplication and Division is possible with linear coordinate system microrotations

- Write a Matlab m-file to implement the CORDIC algorithm for multiplication. This requires rotational mode. See sample output for the multiplication of 1.41 (x) and 0.88 (z) => 1.24201 (y).

```
>> cordic_mult(1.41, 0.88);
--- Rotation Mode ---
  k  d_k       e_k       z_k       x_k       y_k
-------------------------------------------------
  0    1    1.0000    0.8800    1.4100    0.0000
  1   -1   -0.5000   -0.1200    1.4100    1.4100
  2    1    0.2500    0.3800    1.4100    0.7050
  3    1    0.1250    0.1300    1.4100    1.0575
  4    1    0.0625    0.0050    1.4100    1.2337
  5   -1   -0.0313   -0.0575    1.4100    1.3219
  6   -1   -0.0156   -0.0262    1.4100    1.2778
  7   -1   -0.0078   -0.0106    1.4100    1.2558
  8   -1   -0.0039   -0.0028    1.4100    1.2448
  9    1    0.0020    0.0011    1.4100    1.2393
-------------------------------------------------
 ==> x*z =   1.24201 (  1.24080)
-------------------------------------------------
```

▶ Write a Matlab m-file to implement the CORDIC algorithm for division. This requires vectoring mode. See sample output for the division of 1.41 (y) and 0.88 (x) => 1.60352 (z).

```
>> cordic_div(1.41, 0.88);
--- Vectoring Mode ---
  k  d_k       e_k       z_k       x_k       y_k
-------------------------------------------------
  0   -1   -1.0000    0.0000    0.8800    1.4100
  1   -1   -0.5000    1.0000    0.8800    0.5300
  2   -1   -0.2500    1.5000    0.8800    0.0900
  3    1    0.1250    1.7500    0.8800   -0.1300
  4    1    0.0625    1.6250    0.8800   -0.0200
  5   -1   -0.0313    1.5625    0.8800    0.0350
  6   -1   -0.0156    1.5938    0.8800    0.0075
  7    1    0.0078    1.6094    0.8800   -0.0063
  8   -1   -0.0039    1.6016    0.8800    0.0006
  9    1    0.0020    1.6055    0.8800   -0.0028
-------------------------------------------------
 ==> y/x =   1.60352 (  1.60227)
-------------------------------------------------
```

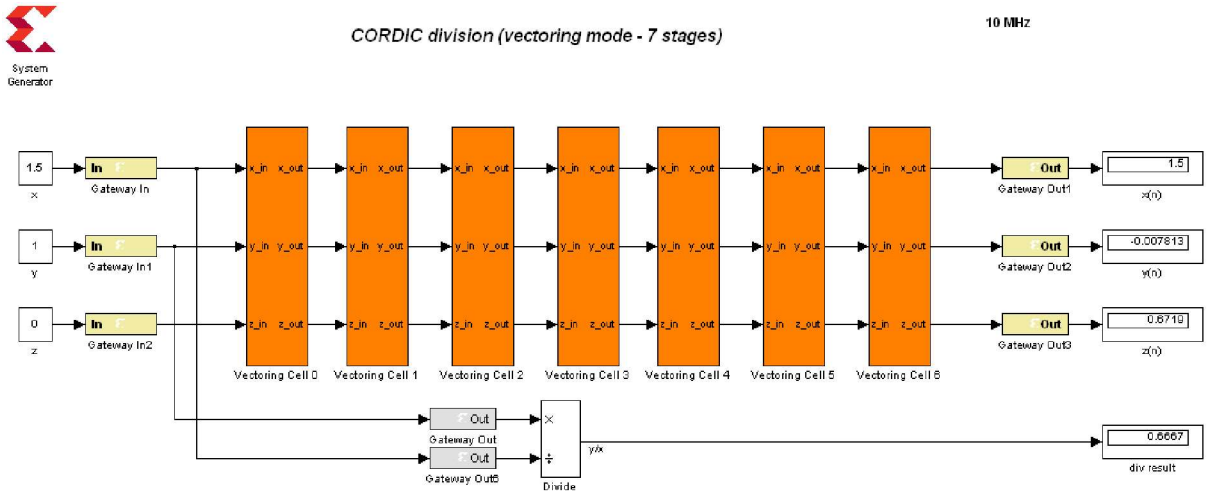
***

Lab #18: FPGA CORDIC div

17.10 CORDIC Division on Hardware

Division by conditional subtraction and shifting is a complex algorithm. Practical implementation by CORDIC linear microrotations in vectoring mode (linear rotation!) is often a good approach to compute hardware divisions with the required accuracy. Since linear rotation is simpler than circular rotations, the resources are quite moderate.

- Write a System Generator model in Matlab/Simulink to perform the division operation in 16 bits and with 7 stages. Provide appropriate number formats for the Vectoring cells.
- Verify the result by comparing with the Matlab calculation in double format. The model should look similar to the following block diagram.



***

Lab #19: Adding System Generator Hardware to MicroBlaze

18 Adding Hardware to MicroBlaze as AXI4 Bus Component

The tools to design components of an embedded system are optimal for specific purposes. However, for complex designs it is most efficient to take advantage of several tools. As an example for using two powerful design tools a hardware design in System Generator should be included in a MicroBlaze processor system.

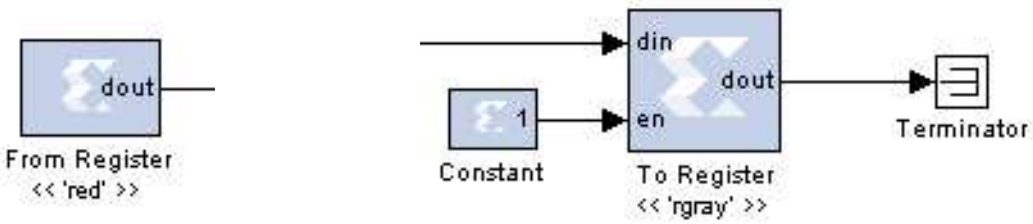
To satisfy speed requirements an RBG to grayscale converter should be created in hardware and added to an EDK/SDK project. RGB video signals are converted to a grayscale signal according to the formula

$$gray = 0.3R + 0.59G + 0.11B\,, \tag{1.219}$$

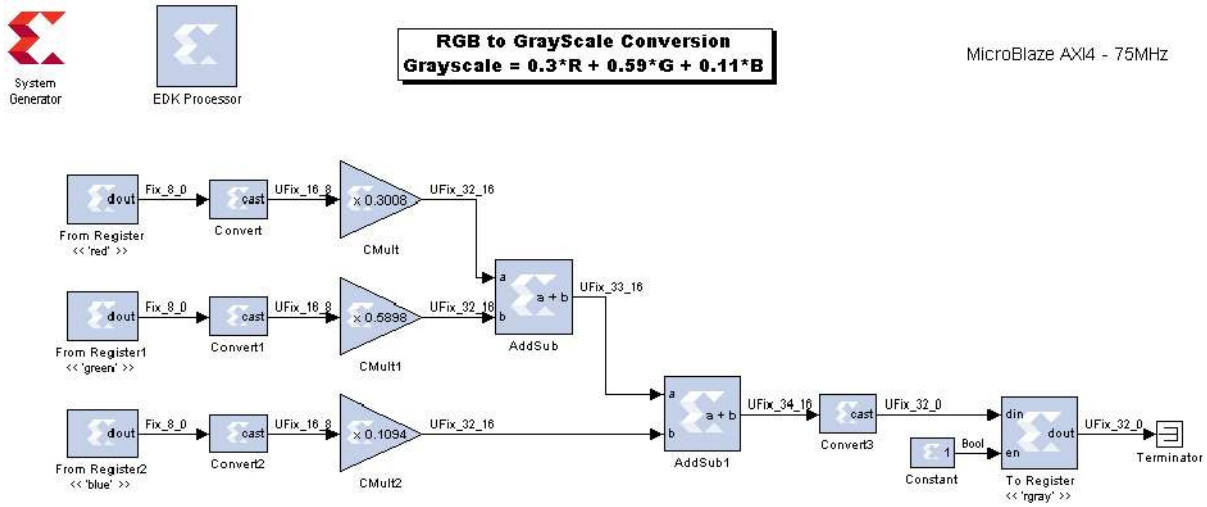
where RGB stands for **r**ed, **g**reen, and **b**lue colors. The algorithm requires three multiplications and to adders.

The MicroBlaze system can be imported to the System Generator or a System Generator design can be used as a processor component. The second approach is taken here to make EDK/SDK the main design application. In this case the System Generator design is exported to EDK as an AXI bus device.

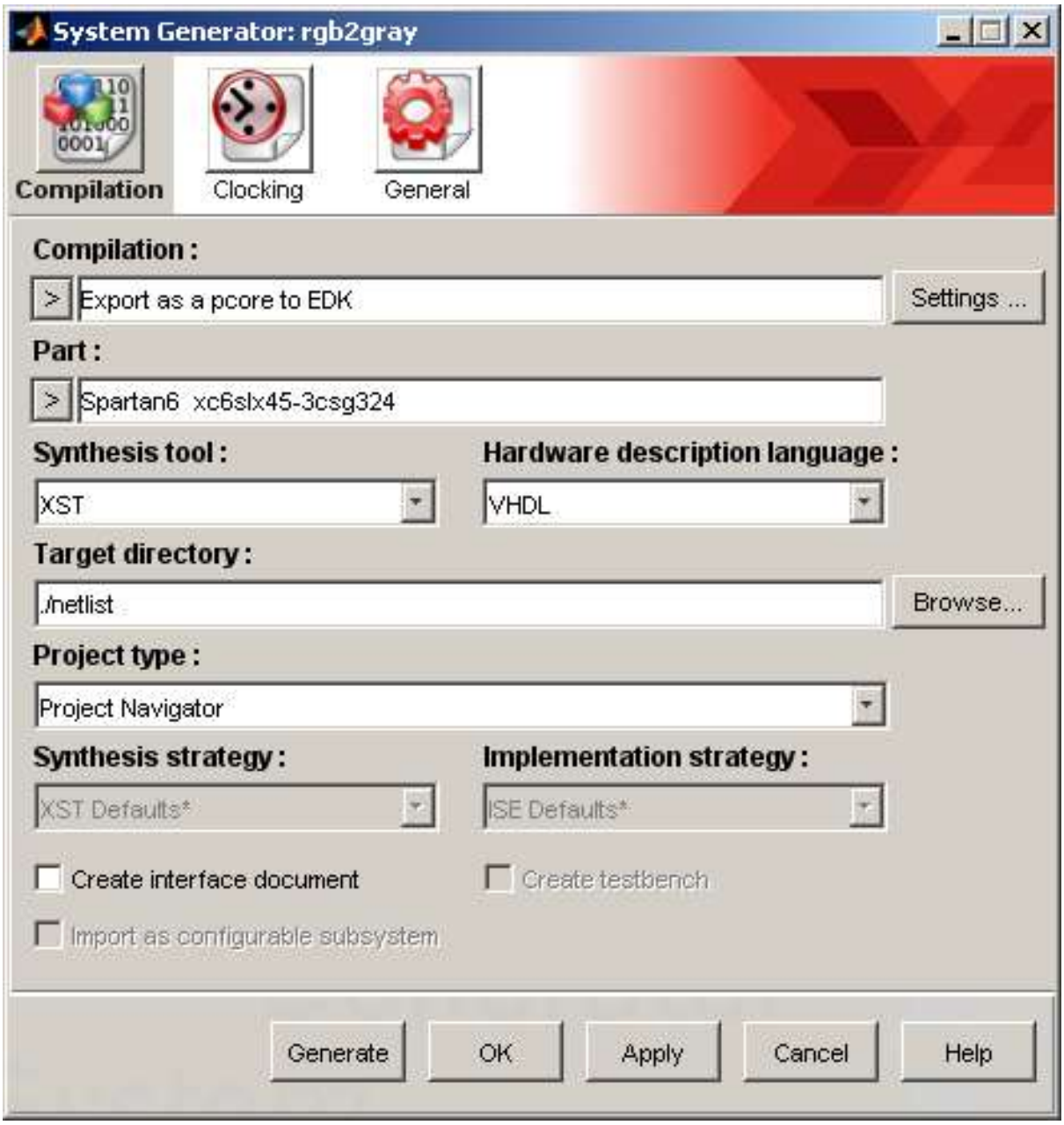
- Create a new folder "rgb2gray" and model the algorithm (1.219) in fixed-point. Please note that color values are unsigned. The RGB colors are 8 Bit, the output can be 32 bit (the size of an AXI bus register). Instead of the usual "Gateway In" and "Gateway Out" blocks for a top level design, "From Register" and "To Register" block are required. They will become AXI bus registers later.



Since the output of the "To Register" block will be read from the bus it must be terminated to avoid error messages from Simulink. The hole hardware could look as shown below.

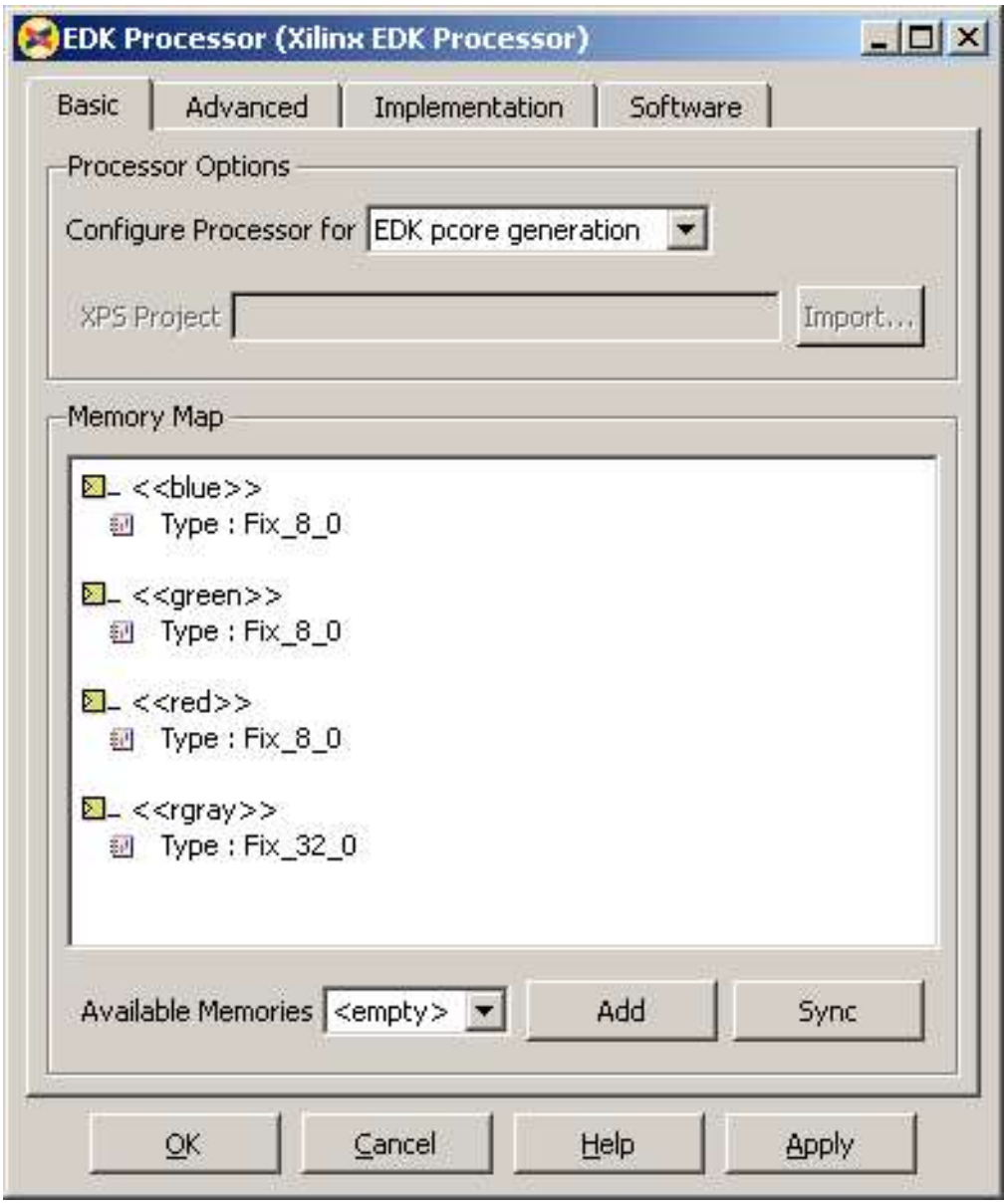


An EDK processor block is required as can be seen. This processor is not generated when "Export as a pcore to EDK" is selected in the System Generator block dialog (see below). Do not forget to specify the correct FPGA device here.



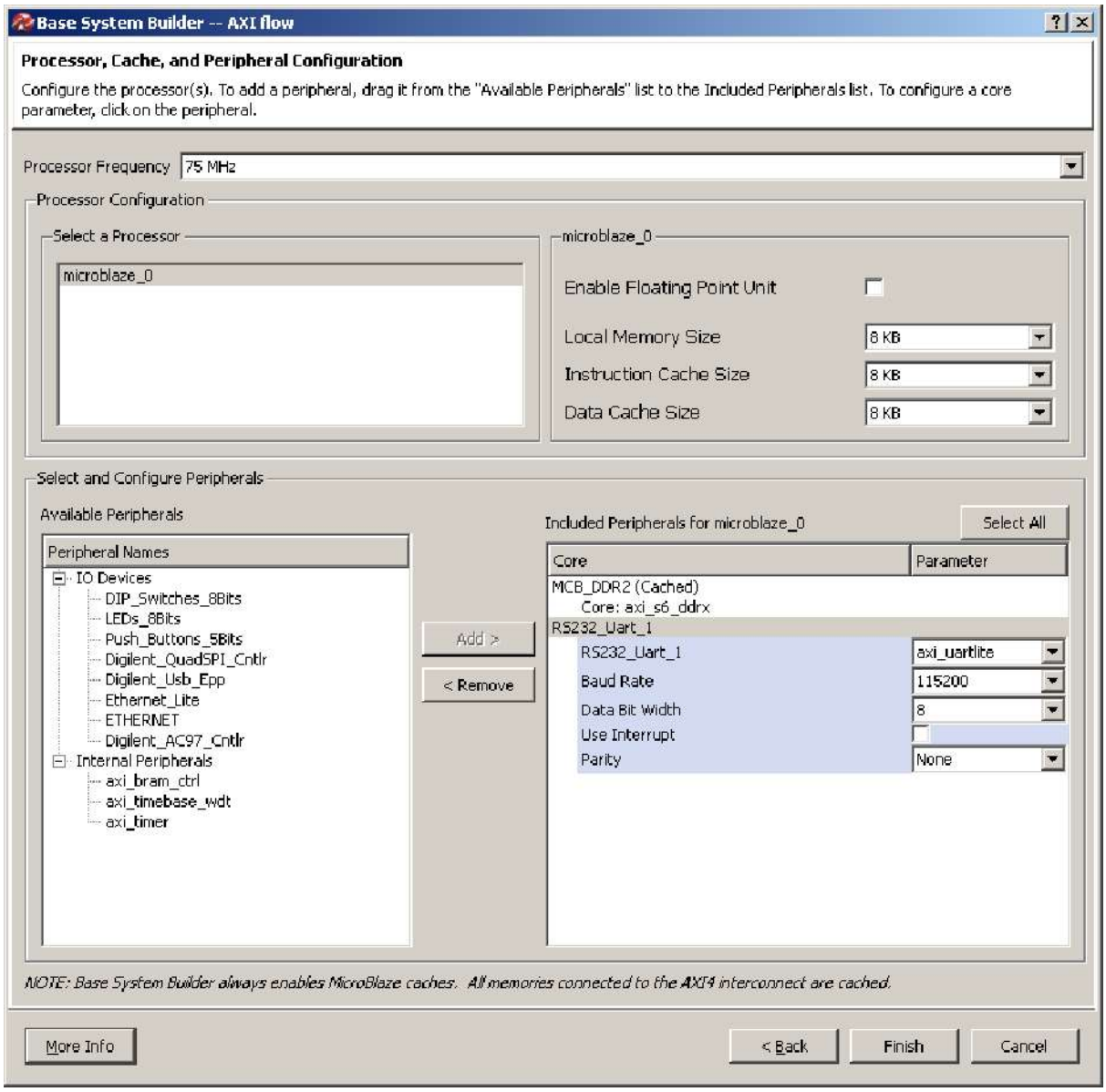
It is mandatory to specify input and output format explicitly in the Register blocks. These registers have to be imported in the memory map of the processor (block).

- Import the memory block in the EDK processor (see below). Take care that proper names are supplied as you will refer to them later.

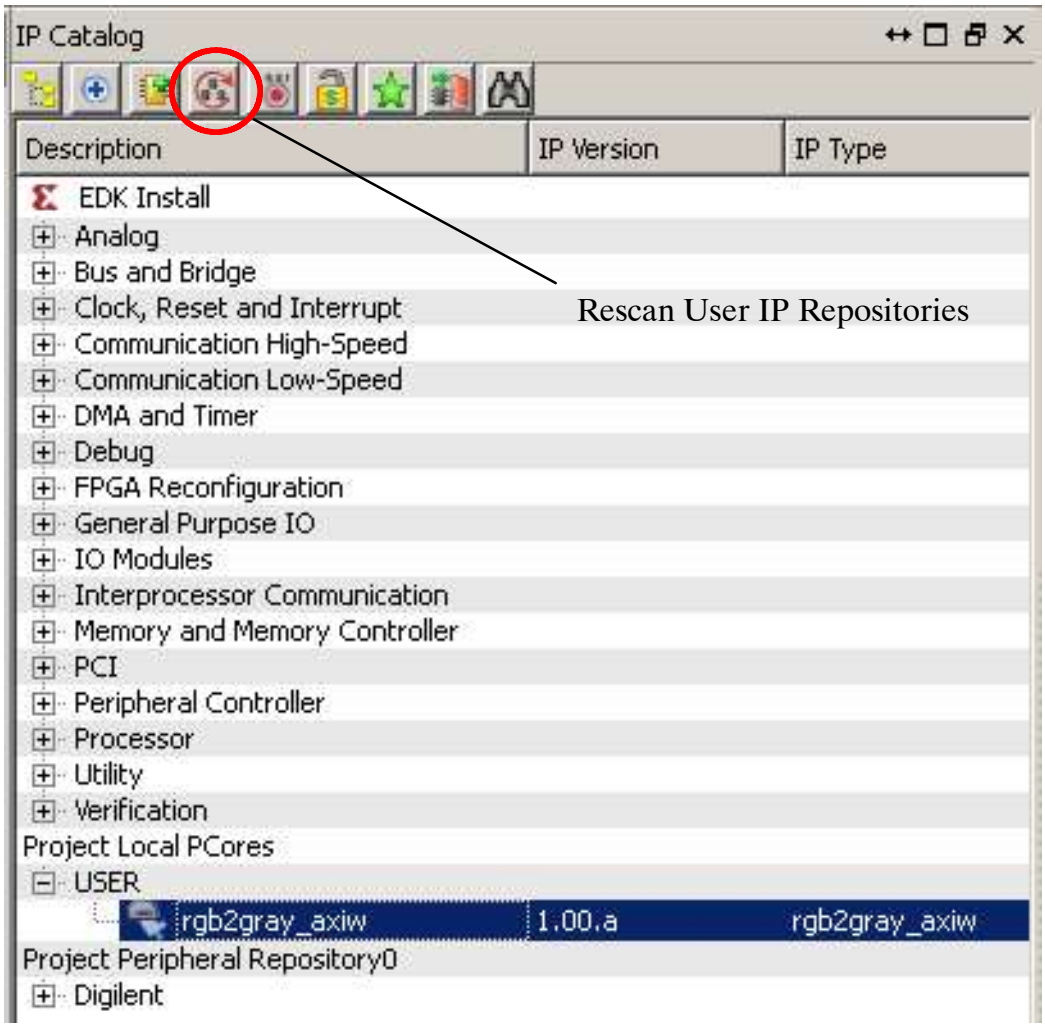


- If everything seems satisfactory select "Generate" from the System Generator block dialog. A new pcore will be created under ".\netlist". This creates an AXI compatible pcore for EDK. No more work has to be done with Matlab/Simulink.

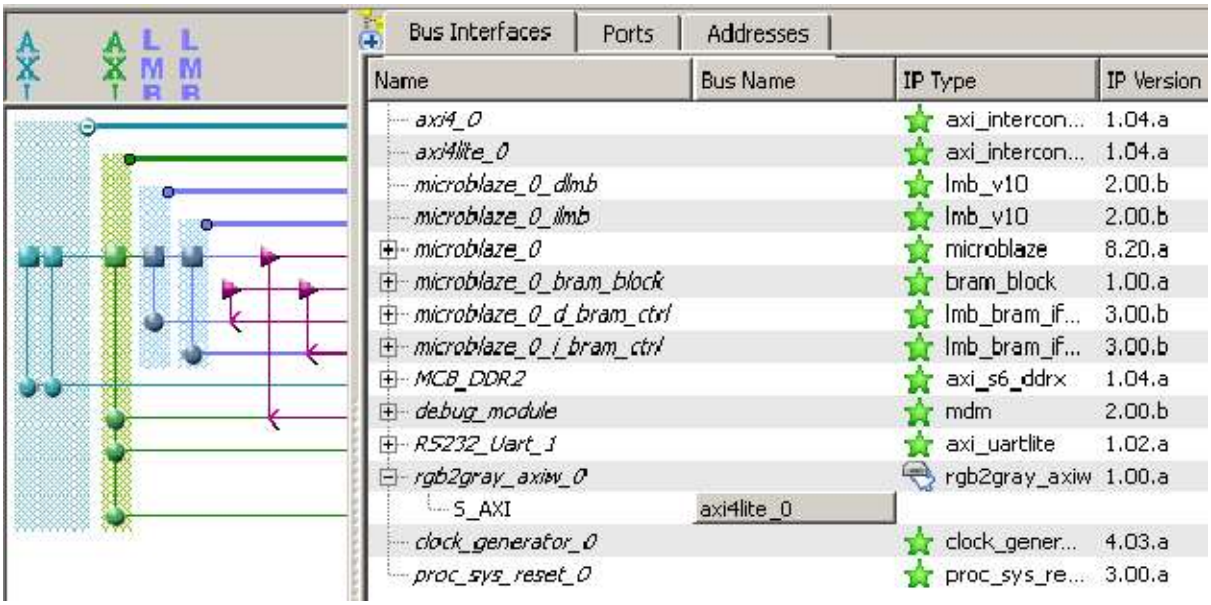
- Create a new folder for an EDK project.
  Create a new system with the wizard an select the following components for the Atlys board. Your newly created pcore is not yet visible.



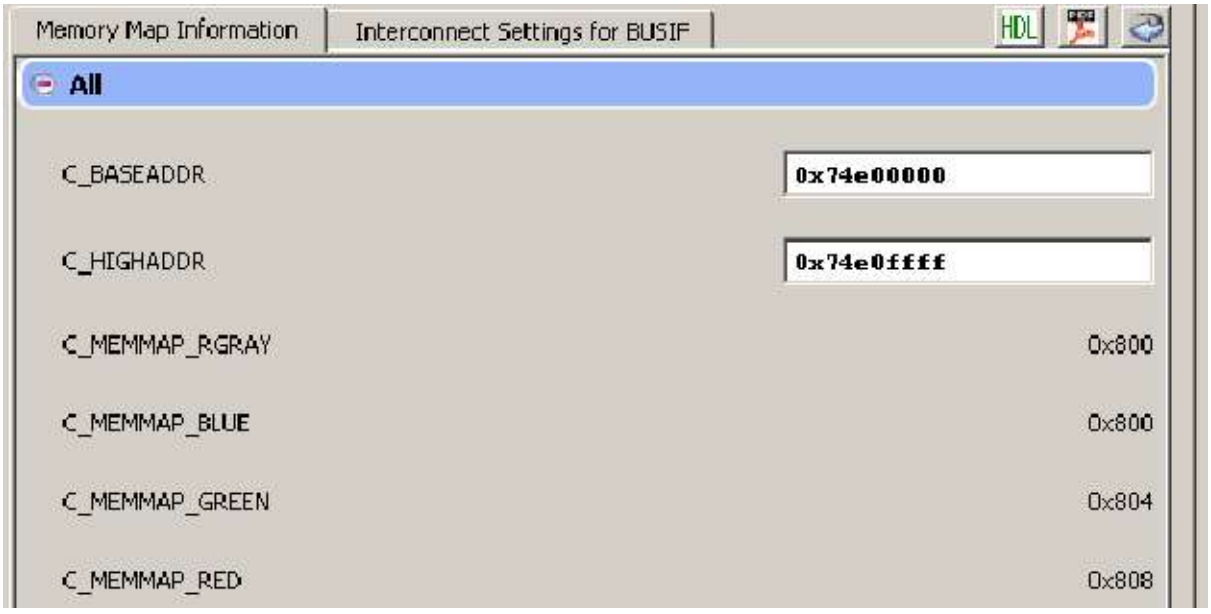
- Copy the whole directory “rgb2gray_axiw_v1_00_a” the .\netlist\pcores folder from the System generator to the appropriate pcores folder in EDK.
- Press “Rescan user IP Repositories” (see below) so that your new pcore becomes available.



- Add the IP by double-clicking on it and make the necessary connections (to AXI-lite) if it does not happen automatically. Verify that all addresses are created correctly. Your system should look similar to the following figure.



- Select Hardware –> Generate Bitstream from the main menu and wait for completion.
- Pass the design to SDK by selecting Project –> Export Hardware Design to SDK... and wait for SDK to come up.
- Before closing EDK inspect the pcore property sheet for memory map information. The memory offsets for each register (relative to the base address) is shown in the figure below. The values may differ for your design.



- Create a “Hello, World” application and modify it to verify the function of the RGB to grayscale hardware. Since the registers are memory mapped the following definition allows the register access to all inputs and the result (on a single line):

```
#define CI_REG(k) (*(volatile u32 *)
          (XPAR_RGB2GRAY_AXIW_0_BASEADDR+0x800+4*k))
```

- With the completed code fragment

```
    for (i = 15; i < 30; i++) {
        red = i;
        green = i + 10;
        blue = i + 20;
        .
        .
        .
    }
```

the terminal output should look similar to

```
-- Entering main() --
  red  green  blue   gray
------------------------------
    F     19    23 ==> 17 (17)
   10     1A    24 ==> 18 (18)
   11     1B    25 ==> 19 (19)
   12     1C    26 ==> 1A (1A)
   13     1D    27 ==> 1B (1B)
   14     1E    28 ==> 1C (1C)
   15     1F    29 ==> 1D (1D)
   16     20    2A ==> 1E (1E)
   17     21    2B ==> 1F (1F)
   18     22    2C ==> 20 (20)
   19     23    2D ==> 21 (21)
   1A     24    2E ==> 22 (22)
   1B     25    2F ==> 23 (23)
   1C     26    30 ==> 24 (24)
   1D     27    31 ==> 25 (25)
------------------------------
-- Exiting main() --
```

The values in brackets are from the software solution to verify proper hardware functionality:

```
// Verify hardware results
static u32 rgb2gVerify(u32 r, u32 g, u32 b) {
       return (77 * r + 151 * g + 28 * b) >> 8;
}
```

***

Lab Requirements

19 Required Reports for ES-MED Labs

In fulfillment of the grade for ES-MED labs the following lab reports have to be submitted:

- FFT solution according to section 5
- CORDIC solutions in Matlab (m-file) and System Generator implementations for
  – sine/cosin

All lab documentations must include algorithm, number format discussion, block diagrams, and software descriptions.

***

20 Bibliography

[1] Ashenden, Peter J.: The Designer's Guide to VHDL, 3rd. Ed.
Morgan Kaufmann, 2008

[2] Al-Hashimi.Bashir M.: System-on-Chip: Next Generation Electronics.
Institution of Electrical Engineers, 2006

[3] Below, Klaus and Dietrich, Karin: Medizinische Gerätetechnik (in German).
Europa-Lehrmittel, 2006

[4] Jennings, D., Fint, A., Turton, BCH. and Nokes, LDM: Introduction to Medical Electronics Applications.
Edward Arnold PLC, 1995

[5] Lipsett, Rger, Schaefer, Carls and Ussery Cary: VHDL Hardware Description and Design.
Kluwer Academic 1990

[6] Lyons, Richard G.: Understanding Digital Signal Processing.
Prentice Hall, 2011

[7] Predroni, Volnei A.: Circuit Design and Simulation with VHDL, 2nd. Ed.
MIT Press, 2010

[8] Prutchi, David and Norris, Michael: Design and Development of Medical Electronic Instrumentation.
John Wiley& Sons, 2005

[9] Reis, Ricardo, Lubaszewski, Marcelo and Jess, Jochen: Design of Systems on a Chip: Design and Test.
Springer, 2010

[10] Reichardt, J. und Schwarz, B.: VHDL-Synthese.
Oldenbourg, 2001

[11] Sass, Ron and Schmidt, Andrew G.: Embedded Systems Design with Platform FPGAs: Principles and Practices.
Elsevier Inc. 2010

[12] Wakerly, John F. : Digital Design, Principles & Practices.
Prentice Hall, 2001

***