## FP7- Grant Agreement no. 283393 - RadioNet3

Project name: Advanced Radio Astronomy in Europe
Funding scheme: Combination of CP \& CSA
Start date: 01 January 2012 Duration: 48 month


## Deliverable 8.16

Report on effectiveness of green measures: beam former

Due date of deliverable: 2015-08-31
Actual submission date: 2015-11-20
Deliverable Leading Partner: MAX PLANCK GESELLSCHAFT ZUR FOERDERUNG DER WISSENSCHAFTEN E.V. (MPG), Germany

Document name: Uniboard ${ }^{2}$ Beam Former Firmware Design Document including Green Measures
Type:
Revision: Document

WP:
Authors: 8

MPIfR Report:
Guenter Knittel
MPIfR-UB3-11/2015

### 1.1 Dissemination level

| Dissemination Level |  |  |
| :---: | :---: | :---: |
| PU | Public | $\mathbf{X}$ |
| PP | Restricted to other programme participants (including the Commission Services) |  |
| RE | Restricted to a group specified by the Consortium (including the Commission Services) |  |
| CO | Confidential, only for members of the Consortium (including the Commission Services) |  |

### 1.2 Document history

| Revision | Date | Author | Modification / Change |
| :---: | :---: | :---: | :---: |
| 1.0 | 31 Jan 2014 | G. Knittel | Initial Version |
| 2.0 | 31 Jul 2015 | G. Knittel | Revised Version |
| 3.0 | 18 Nov 2015 | G. Knittel | Revised Version including Green Measures <br> and Implementation Description |

### 1.3 Distribution list

ASTRON: Andre Gunst, Eric Kooistra, Sjouke Zwier, Daniel van der Schuur, Harm Jan Pepping
JIVE: Arpad Szomoru, Jonathan Hargreaves, Sergei Pogrebenko, Paul Boven, Harro Verkouter
UMAN: Ben Stappers
INAF: Gianni Comoretto
BORD: Benjamin Quertier, Alain Baudry, Stephane Gauffre
UORL: Cedric Dumez-Viou, Rodolphe Weber, Nicolas Grespier
MPG: Guenter Knittel, Gundolf Wieching, Bernd Klein, Udo Beckmann

### 1.4 Terminology

10GBASE-KR: 10Gigabit/s Ethernet interface for backplanes, using one differential signal pair in each direction (four PCB traces)
ADC: Analog to Digital Converter
ALM: Adaptive Logic Module
BF: Beamformer
bps: Bits per second
BW: Bandwidth
Channel: Frequency band, output of the (polyphase) filterbank
CORDIC: (for COordinate Rotation DIgital Computer), algorithm for computing trigonometric functions
cplx: complex, denoting a complex number
CXM: Complex multiplication, complex multiplier
DSP: Digital Signal Processing
FD: Full-duplex, bi-directional link offering full data rate in each direction simultaneously
Firmware: Collection of control codes close to the hardware, less likely and easy to change than software
FFT: Fast Fourier Transform
FPGA: Field Programmable Gate Array
VHDL: Very high-speed integrated circuit Hardware Description Language
Im: Imaginary component of a complex number
I/O: Input/Output
IP: Intellectual Property
LAB: Logic Array Blocks

LUT: Look-up Table
MLAB: $\quad$ LAB that also can be used as RAM
QSFP+: $\quad$ SFP for 40 Gb Ethernet
RAM: Random Access Memory
PAF: Phased Array Feed
PFB: Polyphase Filterbank
Re: Real component of a complex number
SFP: Small Form-factor Pluggable transceiver, an optical transceiver module
SFP+: $\quad$ SFP for 10 Gb Ethernet
SP-FP: Single-precision floating-point (number)
Subband: Frequency band, output of the (polyphase) filterbank
Transceiver: High-speed I/O-circuit with bi-directional data transfer, using serial differential signaling
XGMII: $\quad 10 \mathrm{Gigabit} / \mathrm{s}$ Media-Independent Interface

### 1.5 References

[1] W. C. Barott, O. Milgrome, M. Wright, D. MacMahon, T. Kilsdonk: „Real-Time Beamforming Using High-Speed FPGAs at the Allen Telescope Array", (2011). Department of Electrical, Computer, Software, \& Systems Engineering - Daytona Beach. Paper 2.
[2] G. Comoretto: „Deliverable 8.4 - Uniboard ${ }^{2}$ Digital Receiver Firmware Design Document", INAF report 01/2014
[3] G. Comoretto, G. Knittel, A. Russo: "Uniboard Pulsar Receiver Design document" (2012)
[4] P. E. Dewdney: „SKA1 System Baseline Design", SKA-TEL-SKO-DD-001, 2013-03-12
[5] G. Schoonderbeek: "Deliverable 8.2 - Hardware Design Document", ASTRON Doc. Nr. ASTRON-TN-040 1.0 (2014)
[6] A. Szomoru: "UniBoard ${ }^{2}$ Work Package description", RadioNet3 283393 (2011)
[7] A. Szomoru: "Beamforming specifications", „ADC-specifications", personal communication
[8] https://en.wikipedia.org/wiki/CORDIC
[9] https://www.altera.com/products/fpga/stratix-series/stratix-10/overview.html\#family-table, visited on July 62015
[10] G. Schoonderbeek, ASTRON, email communication
[11] suggested by G. Wieching
[12] UniBoard2 Schematic, Sjouke Zwier / Gijs Schoonderbeek, 12/12/2014
[13] John Bunton, CSIRO, "PAF Beamformer"
[14] ALTERA, "Reducing Power Consumption and Increasing Bandwidth on 28-nm FPGAs", WP-01148-2.0, March 2012
[15] M. Pedram and A. Abdollahi, "Low Power RT-Level Synthesis Techniques: A Tutorial", University of Southern California
[16] M. R. Stan and W. P. Burleson, "Bus-invert coding for low-power I/O," IEEE Transactions On VLSI Systems, Vol.3, No.1, pp.49-58, 1995
1 Document information ..... 2
1.1 Dissemination level ..... 2
1.2 Document history ..... 2
1.3 Distribution list ..... 2
1.4 Terminology ..... 2
1.5 References ..... 3
2 Introduction ..... 6
2.1 Document Organization ..... 6
3 System Architecture ..... 7
3.1 Filterbank ..... 7
3.2 Backplane ..... 8
3.2.1 Revised Architecture ..... 8
4 Architecture of the Beamformer FPGA ..... 9
4.1 Chip Resources ..... 9
4.1.1 Logic ..... 9
4.1.2 Arithmetic ..... 9
4.1.3 Memory ..... 9
4.2 Block Diagram ..... 9
4.3 Input Stage ..... 10
4.4 Input Fan-Out Register Tree ..... 11
4.5 Beamformer Unit ..... 11
4.5.1 Theory of Operation ..... 11
4.5.2 Beamformer Core Block Diagram ..... 12
4.5.3 Weightfactor and Index Multiplexer ..... 13
4.5.4 Control Unit ..... 13
4.5.5 Memory and CXM Stage ..... 14
4.5.6 Adder Tree ..... 15
4.5.7 Accumulator Stage ..... 16
4.6 Weightfactor Memory ..... 18
4.7 Weightfactor Fan-Out Register Tree ..... 18
4.8 Output Stage ..... 19
4.8.1 Output Unit ..... 19
4.8.2 Control Unit ..... 20
4.8.3 10Gb Ethernet Interface ..... 21
4.8.4 Data Transfer Control ..... 22
4.8.5 Complete Output Stage ..... 22
4.9 System Outline ..... 23
4.10 Design Summary ..... 24
5 Scalability ..... 24
5.1 Alternative Arithmetic ..... 24
5.1.1 The CORDIC Unit ..... 25
5.1.2 Conclusion ..... 27
5.2 Alternative FPGAs ..... 27
5.3 Increasing the Number of Beams ..... 27
5.3.1 Circuit-Level ..... 27
5.3.2 System-Level ..... 28
5.4 Increasing the Observing Bandwidth ..... 28
5.4.1 Circuit-Level ..... 28
5.4.2 System-Level ..... 30
5.5 Increasing the Number of Antennas ..... 30
5.5.1 Circuit-Level ..... 30
5.5.2 System-Level ..... 31
5.6 Scalability Summary ..... 33
5.7 Alternative Architectures ..... 33
6 Green Measures ..... 34
6.1 Introduction ..... 34
6.1.1 Choice and Design of the Platform, Chip Technology ..... 34
6.1.2 Required Computing Performance ..... 34
6.1.3 Efficiency of Algorithms ..... 34
6.1.4 Quality of Implementation of the Hardware, Gateware, Firmware, and Software ..... 34
6.1.5 Presence or Absence of Power Saving Provisions ..... 34
6.2 Tools and Procedures for Power Consumption Measurements ..... 34
6.3 Tool Flow ..... 35
6.3.1 Procedure ..... 35
6.4 Power-Efficient Gateware Implementation ..... 40
6.4.1 Traveling Enable-Signal on Pipeline Registers ..... 40
6.4.2 Organization of on-chip Memory Systems ..... 40
6.4.3 Low-Power Coding on Address Buses ..... 42
6.4.4 Low-Power Coding on Parallel Single-Endend (off-chip) Buses ..... 42
6.5 Architectural Changes for Increased Power Efficiency ..... 43
6.6 CORDIC Arithmetic ..... 43
6.7 Compiler Options for Increased Power Efficiency ..... 44
6.8 Scaling Performance vs. Power Consumption ..... 44
7 Implementation ..... 45
7.1 Implementation Restrictions ..... 45
7.2 Actual Architecture ..... 45
7.3 VHDL and QSys Source Files ..... 48
7.4 Results and Performance ..... 49
8 Summary ..... 49

## 2 INTRODUCTION

The presented beamformer is a narrow-band beamformer operating on frequency channels of around 1 MHz bandwidth. The specifications are oriented towards SKA phase 1 requirements [4], [7]: 384 MHz observing bandwidth, 256 dual-polarization receivers ( 512 antenna signals), and 64 beams. The fundamental mathematical operation that it performs is

$$
\begin{equation*}
y(n)=\sum_{m=0}^{M-1} w_{m} \cdot x_{m}(n) \tag{1}
\end{equation*}
$$

where $x_{m}(n)$ is the signal from the $m$-th antenna at sample time $n * T_{0}$, $w_{m}$ the weightfactor for the $m$-th antenna, and $y(n)$ the beam sample at time $n * T_{0}$. All of the quantities $y, x$, and $w$ are complex numbers. In (1), a single-beam system (many-to-one) is described.
A system computing several beams $b=0 \ldots B-1$ (many-to-many) can then be described as

$$
\begin{equation*}
y_{b}(n)=\sum_{m=0}^{M-1} w_{m, b} \cdot x_{m}(n) \tag{2}
\end{equation*}
$$

Finally, if the antenna signals have been split into a number of frequency channels $c=0 \ldots C$ - 1 the system can be described as

$$
\begin{equation*}
y_{b, c}(n)=\sum_{m=0}^{M-1} w_{m, b, c} \cdot x_{m, c}(n) \tag{3}
\end{equation*}
$$

The beamformer is polarization-agnostic, meaning, all 512 antenna signals can be used for the computation of a given beam. Polarization can be included by setting the weigths appropriately. For an estimate of the error arising from the channel bandwidth see [1].

The system must maintain a total of $M^{*} B^{*} C$ weightfactors. In this example design this amounts to $12,582,912$ complex numbers. The precision is set to 19 bits per component, as this is the maximum width of the hard-wired multipliers on the Arria-10 FPGAs. Thus, a total of $478,150,656$ bits must be provided for the weightfactors.
The total number of complex multiplications per second is given by $384 * 10^{6} * 512 * 64=1.2582912 * 10^{13}$. The FPGAs which are planned to be used for a low-cost version of the Uniboard ${ }^{2}$ provide a total of 759 hard-wired macros for complex multiplication. For a complex multiplication $(a+j b) *(c+j d)$, operand width for $a$ and $b$ can be up to 18 bits, whereas the width for $c$ and $d$ can be up to 19 bits.

For cost reasons this study uses the slowest production version of the FPGA, which allows the complex multipliers to be used at up to 360 MHz . The complex multipliers are grouped into sub-arrays of 16 units, so that 7 complex multipliers remain unused per FPGA. In order to meet the computing requirements, a total of 48 FPGAs on 12 Uniboards are used. Then the minimum clock frequency is

$$
\begin{equation*}
1,2582912 \cdot 10^{13} /(48 \cdot 752)=348,6 \mathrm{MHz} \tag{4}
\end{equation*}
$$

Thus, using a clock frequency of 360 MHz in conjunction with shallow buffers will provide some headroom for compensating any data rate variations. Each FPGA needs to process $384 / 48=8$ frequency channels across the 512 antenna signals. We assume that channelization is done by polyphase filterbanks, and that the complex output samples have a width of 16 bits per component. Thus, data rate into each beamformer FPGA is $8 * 512 * 10^{6} * 32=$ $1.31072 * 10^{11} \mathrm{bit} / \mathrm{s}$. We assume conservatively that a net data rate of $10 \mathrm{~Gb} / \mathrm{s}$ can be achieved per backplane lane. Thus, each beamformer FPGA needs at least 14 such interfaces. Each beamformer-FPGA provides a total of 72 such backplane interfaces, so this is not a limitation. However, the actual number of interfaces depends on the architecture of the polyphase filterbanks, and on how the 512 antenna signals are connected to the filterbankFPGAs.

These particular considerations shall now be used to derive architecture and size of the beamformer. Because of the multitude of hardware constraints it is very hard to give general rules for scaling the system. A change of the desired performance or availability of more powerful chips might very well require the design of a completely different architecture.

### 2.1 Document Organization

Sections 3 and 4 describe system and chip architecture on a fairly high level to effectively convey the basic ideas of the design. These sections only describe the data paths and arithmetic units, without going into details regarding control units and control interfaces. This basic infrastructure is assumed to simply „be there" and work in the expected way. It will mostly be described in general terms.

Also not covered in these sections are considerations regarding minimum operand precision across the many buses in the design. As general design guidelines, the maximum precision as provided by the available hardware resources (multipliers, RAM etc) is used for the operands. This also defines the width of register banks and other units. In many cases this kind of arithmetic precision might not be necessary.
Section 5 discusses ways to scale the presented architecture towards higher performance. The goal is to be able to process more antennas, wider bandwidth, more beams, or combinations thereof. Most likely this will require the use of more powerful FPGAs (Stratix-10), or a parallel arrangement of multiple systems.

Section 6 is about "green measures", which in this case means power efficiency. Several methods for reducing power consumption are discussed. These methods relate to specific implementation styles and algorithmic optimizations. However, savings that arise from unused antennas, or weightfactors being zero or negligible (in a potential PAF application), are not considered in this work.
Section 7 finally describes the VHDL implementation. The main purpose of the implementation is to verify the effectiveness of the "green measures", and not to verify the correctness of the circuitry itself. Thus, the VHDL code is just a skeleton of the final design which will not necessarily produce meaningful results. However, the design will include all relevant parts to prove that the chip will meet the performance requirements.

## 3 System Architecture

### 3.1 Filterbank

Since there are 512 antenna signals, the system needs to have 512 filterbanks. It is assumed that the actual implementation employs polyphase filterbanks with a channel bandwidth of about 1 MHz . It might be the case that the observing bandwidth is larger than 384 MHz [7], in which case it is assumed that 384 channels can be selected arbitrarily out of the larger set of filterbank output channels.
It is further assumed that the polyphase filterbank generates complex samples with 16 bits for both the real and imaginary components. This particular bit width is not a strict requirement, however, the components may not have more than 18 bits due to hardware limitations (multiplier port width).
The design of this filterbank is not part of this workpackage, so we cannot make any statements about the required hardware resources. From other projects [2], [3] we conservatively estimate that one FPGA can implement 32 filterbanks. Then, a total of 16 FPGAs are needed, or four Uniboards. In this case the signal processing system would take the shape of a standard midplane system with eight Uniboards to the left and eight to the right of a midplane (also called backplane).
In an SKA scenario it is most likely that the signal processing system is located in a central facility, and that the digitizers are located close to the receivers (dishes). Data is then transmitted in digital form over optical fibers. In the first instance the digitizer signals should be connected to those Uniboards that implement the filterbanks. In case there are not enough interface resources, digitizer signals can also be connected to the other Uniboards and passed on to the filterbank units via the backplane.
A block diagram showing the filterbank, its partitioning and the output distribution is given in Figure 1.


Figure 1: General Filterbank Architecture

### 3.2 Backplane

A decision was made to not include any local mesh on the Uniboard ${ }^{2}$, but to provide all interconnects on a backplane. Again, design of the backplane is not part of this workpackage. Therefore we list only basic requirements about the interconnect structure in order to support the beamformer, assuming a polyphase filterbank as described above.
According to the data rates given in Figure 1, any filterbank-FPGA needs to connect to any beamformer-FPGA via one 10Gbit/s backplane lane (transceiver channel). This occupies 48 transceivers on any filterbank-FPGA. Conversely, any beamformer-FPGA is connected to any filterbank-FPGA, which occupies 16 transceivers on any beamformer-FPGA. This basically constitutes the required interconnect structure on the backplane. We assume that the results of the beamformer (beam samples) are output via optical links local to each beamformer-FPGA to a GPU-cluster or similar external device.
A sketch of the required interconnect structure is shown in Figure 2. See also [5] for planned interconnect capabilities.


Figure 2: Interconnect Structure

### 3.2.1 Revised Architecture

The Uniboard ${ }^{2}$ architecture has been modified for the production version as shown in Figure 3 [12]. Each FPGA now has only 48 FD pairs towards the backplane because of routing constraints. However, this number is just sufficient for the beamformer architecture presented here. The remaining 24 FD pairs are used to construct a ring network, which might be used for sample distribution (as an example see Figure 26), or for fast update of weightfactors.


Figure 3: Interconnect Structure (Revised Version)

## 4 ARCHITECTURE OF THE BEAMFORMER FPGA

All 48 beamformer-FPGAs have an identical design. The specific role of an individual FPGA is defined by its place on the backplane, the programmable set of weight factors, and by further programmable parameters such as output destination.
The beamformer-FPGA consists mainly of six parts:

- sample input stage,
- input fan-out register tree,
- beamformer unit,
- weightfactor memory,
- weightfactor fan-out register tree, and
- output stage.

In addition to these units there are control and synchronization units at various places, whose functionality will be described only in general terms. Also, the general infrastructure for writing all programmable parameters including weightfactors will be described in a later document.

### 4.1 Chip Resources

For a better understanding of the following sections we shall give a coarse overview of available hardware resources. The specific FPGA that is planned to be used for the production version is an Altera Arria-10 GX 1150 (order code Altera 10AX115U4F45I3SG).

### 4.1.1 Logic

Configurable logic elements on an Arria-10 FPGA include so-called LABs (Logic Array Blocks) and MLABs (Logic Array Blocks that also can be used as RAM). Both are further divided into so-called ALMs (Adaptive Logic Modules). Each ALM provides two LUTs, a two-bit carry-chain for building adders, and 4 flipflops (registers). The LUTs allow computation of any boolean expression of up to 7 variables. The FPGA provides a total of 854,400 LUTs, and 1,708,800 single-bit registers.

### 4.1.2 Arithmetic

Each FPGA provides a total of 1,518 so-called „DSP Blocks", each providing two $18 \times 19$ bit multipliers among other things. These multipliers are implemented as specialized hardware units. The design tools allow two DSP Blocks to be used as one complex multiplier (CXM). Thus, there are 759 CXMs on the chip.

### 4.1.3 Memory

Each FPGA provides a total of 2,713 memory blocks of 20Kbits each. These specialized hardware units are called „M20K". They can be organized as $512 \times 40$ bits. Thus, for example, one entry can hold one complex weightfactor with 19 bits for both the real and imaginary component.
M20K blocks can be used as dual-port memory with one read and one write port. Reading from an entry that is simultaneously being written to has some implications, so it is best to avoid this situation.

### 4.2 Block Diagram

The block diagram of a beamformer-FPGA is shown in Figure 4.


Figure 4: Beamformer-FPGA Block Diagram

### 4.3 Input Stage

The input stage consists of 16 identical interfaces, each connecting to one filterbank-FPGA. Each interface can be implemented as a 10GBASE-KR transceiver channel ( 10 Gbps Ethernet for backplanes), and should provide a data rate of about 10 Gbps . In case of $10 \mathrm{GBASE}-\mathrm{KR}$, the interface towards the FPGA-logic is called XGMII (10 Gigabit Media Independent Interface) and is implemented as a parallel interface transfering 8 bytes in parallel. The clock rate at this stage is 156.25 MHz .
The samples enter a FIFO memory which should be large enough to provide buffer space for a number of data packets. For safety purposes we assume a buffer capacity of 16 KByte per interface, for a total of 256 KByte across the input stage. FIFO memories are typically implemented using the Altera design tools. These tools report a memory consumption of 7 M20K blocks per FIFO, for a total of 112 such blocks across the input stage. This represents about $4 \%$ of the available resources.
Towards the remaining logic on the beamformer-FPGA, a 32-bit data bus is needed. Thus, the read-port of each FIFO is carried out as 32-bit interface, and data width reduction is handled by internal control logic. Besides this, the FIFOs also provide translation between the I/O- and the system clock domains ( 156.25 MHz to 360 MHz ).
Accordingly, one complex sample ( $16+16$ bits) is read per read cycle (clock) from each FIFO, or 16 complex samples are read per clock across the entire input stage.
The samples need to be transmitted by the filterbank-FPGAs in a specific order, which is shown in Figure 5. See also Figure 1 for sample arrangement. Sample origin is given by [Antenna Number; Frequency Channel Number].


Figure 5: Input Stage Block Diagram

### 4.4 Input Fan-Out Register Tree

The FIFO of interface $n$ feeds the data to CXM unit $n$ in each beamformer core (see section 4.5.5). There are 47 beamformer cores. Thus, the fan-out is 47 as well, and to reduce wiring delays and meet the clock rate the input data is distributed by a fan-out register tree. We use a two-stage $6 \times 8$ register arrangement. Thus, this distribution stage consumes 27,648 flipflops. This is $1.6 \%$ out of $1,708,800$ available flipflops. The arrangement is sketched out in Figure 6. Note that the latency introduced by this structure is of no concern.


Figure 6: Input Fan-Out Register Tree

### 4.5 Beamformer Unit

The beamformer unit is divided into an array of independent beamformer cores in order to make better use of the available hardware resources. Each beamformer core operates on one sample across all 512 antennas in one frequency channel and generates all 64 beam samples from this input data set. We call this collection of input samples an input vector (see Figure 1).
A beamformer core uses 16 complex multipliers (CXM) in parallel. Accordingly, there are 47 beamformer cores on the chip, with a total of 7 CXMs, or 14 DSP Blocks going unused. This represents a resource utilization of $99.1 \%$.

A beamformer core will only start operating when all 512 complex samples are available. Thus, we need one M20K memory block in front of each CXM for data buffering. An input vector will only occupy 32 out of 512 entries of each M20K block, so there is ample room for buffering and data rate adjustments. A total of 752 M 20 K blocks are needed for this purpose, or about $27.7 \%$ of the available memory blocks. The weightfactors, on the other hand, are streamed in real-time from the weightfactor memory and don't need deep buffers. This will be explained in detail in the following section.

### 4.5.1 Theory of Operation

Whenever a beamformer core has enough free storage space for an input vector it will issue a work package request to the scheduler (see Figure 4 ). The scheduler will prioritize all requests and send an input vector to the selected beamformer core. This input vector is simply the next input vector in the input FIFOs and can be from any of the eight frequency channels. A read across all input FIFOs will yield 16 complex samples, which are written into the 16 memory blocks of the beamformer core in parallel. 32 such transfers are needed to move a complete input vector to the selected beamformer core.
Using such scheme it cannot be predicted which frequency channel any given beamformer core will receive, nor the exact point in time the complete input vector will be available. Still the beamformer core is required to run at $100 \%$ efficiency, i.e., no single cycle may be spent idle when input data is available.
For this purpose the weightfactor memory (see section 4.6) issues a constant stream of weightfactors for all eight frequency channels. The beamformer core utilizes a set of multiplexers to select the proper frequency channel. Most importantly, however, the stream of weightfactors also include the index of the current set of weightfactors, which consists of antenna and beam number. Using this index, a beamformer core can select the corresponding time samples from the memories and start processing immediately, even if it tapped the weightfactor stream in mid-sequence. It is obvious that the sum of products in (3) can be computed in any order.
A control unit local to each beamformer core only needs to determine when an input vector has been completed, which happens after 2048 clocks. It will then tag the corresponding partial sums as being the last ones, and switch to the next input vector in the sample memories, if present.

The „inner loop" iterates over beams, which reliefs timing requirements of the accumulator at the output of a beamformer core.

### 4.5.2 Beamformer Core Block Diagram

The beamformer unit consists of 47 identical beamformer cores. As explained above, this uneven number is given by the available chip resources and might be different for other FPGA variants. Nevertheless, the architecture should allow any number of beamformer cores to be operated at peak performance. A beamformer core consists of the following parts:

- weightfactor and index multiplexer,
- memory and CXM stage,
- adder tree, and
- accumulator.

These units are detailed in the following sections. The block diagram is shown in Figure 7.


Figure 7: Beamformer Core Block Diagram

### 4.5.3 Weightfactor and Index Multiplexer

The scheduler has written an input vector into the sample memories, and also the frequency channel number ( 0 .. 7 ) into a small FIFO. The frequency channel number is used to select one of eight weightfactor streams from the weightfactor memory system. Thus, this stage mainly consists of a very wide MUX and a set of registers. A control unit reads the channel FIFO and counts the number of cycles. In cycle 2047 it issues the flag „LAST" to the adder tree, to be passed on to the accumulator stage.


Figure 8: Weightfactor and Index Multiplexer

### 4.5.4 Control Unit

The control unit basically consists of a small state machine and a counter. It controls operation of the entire beamformer pipeline down to the output interface. Provided the channel FIFO does not run empty (and input vectors are available accordingly), uninterrupted operation is established.


Figure 9: Control Unit

### 4.5.5 Memory and CXM Stage

This unit receives a set of 16 complex weightfactors from the multiplexer stage each clock. In addition to that, it receives a memory address of the corresponding set of samples, and the beam number. Its sole task is to read the sample memory, and to pass the 16 complex samples together with the weightfactors to the complex multiplier (CXM). Therefore this unit is divided into 16 Memory and CXM units, as shown in Figure 10.


Figure 10: One Sample Memory and associated CXM Unit

The entire stage is shown in Figure 11. Products and beam number are passed on to the adder tree. Note that all 16 memory blocks, which consist of one M20K block each, receive the same read address. Writing samples from the input stage is always done in units of 16 complex samples, one from each input interface, per clock. Also, a transfer always includes one complete input vector. Therefore, we need only one write address counter for all 16 memories, which is incremented upon the Write Enable signal from the scheduler. The counter will endlessly wrap around.


Figure 11: Memory and CXM Stage

### 4.5.6 Adder Tree

The adder tree is a pipelined binary tree of adders, which compute one partial sum of 16 complex products each clock. Each adder is pipelined in itself for high performance. Operand precision starts at 35 bits and increases by one bit per stage. The tree has 4 stages. After the last stage the operands are truncated to 35 bits again.


Figure 12: Adder Tree

### 4.5.7 Accumulator Stage

The accumulator stage accumulates one complex sample for each of the 64 beams. Each component is maintained as a 40 -bit number. Format conversions or rounding can be done after the beam samples are complete.
The partial sum of 16 complex multiplies that has been produced by the adder tree arrives together with the beam number at this stage. The beam number is used as the read address, and later as the write address, for a memory system holding the beam sample as it has been computed so far. In a pipelined fashion the new partial sum will be added to the current value, and the updated value will be written back. Since the „inner loop" iterates over the beam number, read and write address will always be different and so read/write-hazards will be avoided.
For each beam, 32 partial products need to be accumulated.
The accumulator stage in its basic form is shown in Figure 13.


Figure 13: Accumulator Stage (Basic Form)

However, at some point in time the beam samples need to be read by the output stage, and the beam memory must be cleared for the next input vector. This might interrupt the operation of the beamformer core. In order to avoid this, the beam memory is implemented as a double-buffer. While one memory system is used for accumulation, the other is read and cleared. Buffer switching must be controlled by the beamformer core control unit, and synchronized with the output stage.

A block diagram of the double-buffered accumulator stage is shown in Figure 14. In this form the accumulator stage consumes four M20K blocks. For the entire beamformer unit this sums up to 188 M20K blocks, or about $7 \%$ of the available resources.


Figure 14: Double-Buffered Accumulator Stage

In Figure 14, note the flag „LAST", which identifies the last partial sum of an input vector, which in turn completes the last beam sample. The next partial sum will belong to the next input vector, and must therefore be accumulated into the other buffer. This buffer in turn must have been read and cleared by the output stage by then. Computing all beam samples for a given input vector will take 2048 clock cycles, reading and clearing the buffer will take at most 128 clock cycles. The control unit of the output stage also receives the LAST-flag.

The LAST-flag switches all memory ports as the data travels through the pipeline, so there is not a single cycle overhead when switching to a new input vector.

### 4.6 Weightfactor Memory

Each beamformer-FPGA processes 8 frequency channels from 512 antennas and produces 64 beams. Therefore it needs to store $8 * 512 * 64=262,144$ complex weights. Both real and imaginary components have a width of 19 bits and are two's-complement fixed-point numbers.
The weightfactor memory is organized as eight independent banks, one for each frequency channel. Thus, a bank holds $512 * 64=32,768$ weightfactors and occupies 64 M20K blocks. The entire weightfactor memory therefore occupies 512 M 20 K blocks (out of 2,713, or about $19 \%$ ).
In order to keep the beamformer core operating at maximum speed, each weightfactor memory bank outputs 16 weightfactors in parallel. Thus, each memory bank is physically organized as a $2048 \times 640$ bit memory, consisting of a $4 \times 16$ array of M20K blocks.
See Figure 15 for a block diagram of one weightfactor memory bank.


Figure 15: One Weightfactor Memory Bank

Note: a more flexible index management can be obtained by storing the indices along with the weightfactors in the weightfactor memory.

### 4.7 Weightfactor Fan-Out Register Tree

This distribution network is similar to and serves the same purpose as the register tree in Figure 6. It also uses a two-stage $6 \times 8$ register arrangement. Flipflop-consumption is 262,456 , or $15.4 \%$ of the available registers. Note that again the latency introduced by this structure is of no concern.

### 4.8 Output Stage

The task of the output stage is to monitor the „LAST"-flags from all beamformer cores, collect the beam samples, convert the operands to single-precision floating-point numbers and send the data to the final, in this scenario external destination.
Reading the beam samples sequentially from all accumulator memories takes $47 * 64=3008$ clocks, this is more than what is needed to compute a new set of beam samples on a given beamformer core ( 2048 clocks). Thus, the output stage needs to be implemented as several independent units, each serving only a subset of beamformer cores. For this example design we assume six such units, each serving 8 (7) beamformer cores. Clearing the beam sample buffers can be done mostly overlapping with the read-out.
The data rate produced by each individual output unit is as follows. One complex beam sample is computed in 32 clocks on each beamformer core. At 360 MHz , this amounts to 88.9 ns. Each beam sample is output as two SP-FP numbers, or 64 bits in total. The aggregate data stream from 8 beamformer cores adds up to $8 * 64$ bits $/ 88.9 \mathrm{~ns}=5.76 \mathrm{~Gb} / \mathrm{s}$. Thus, each output unit connects to one 10 GbE interface. This can either connect to an optical transceiver, or to the backplane. As for the input stage there is one output FIFO, translating between the clock domains and also the data bus widths.

### 4.8.1 Output Unit

The schematic diagram of one output unit is shown in Figure 16. The control unit receives the LAST-flags from the eight beamformer cores, prioritizes the signals and reads out the beam samples. A tree of multiplexers is used to pass the proper samples to the fixed-point to floating-point converters. The control unit will also have all required information to assemble a valid Ethernet packet including destination address.


Figure 16: Output Unit

### 4.8.2 Control Unit

The control unit latches all LAST-flags and performs a rotational scan to find an active bit. Then, the header is read ( 7 words, or 56 bytes) and passed to the Ethernet interface. Afterwards the beam samples are transferred, and during read-out the buffer is cleared. Finally an 8-byte placeholder for the CRC32 is read from the header memory (all zeros). The header circuitry needs to be adapted and most likely be extended to meet the final structure of the entire system. In this design a simplified version is implemented to facilitate basic testing.


Figure 17: Control Unit

### 4.8.3 $\quad$ 10Gb Ethernet Interface

There are six 10 Gb Ethernet ports in the output stage, which in this design are grouped in one transceiver bank. This configuration allows the transmit clock for all transmitters to be generated by a single ATX PLL.The ports are configured as Tx-only, the receiving device is assumed to be able to process the entire data stream without pause at all times. Each port includes a 10 Gb MAC, which generates the CRC for each frame. The MAC provides an Avalon Streaming Interface to the upstream circuitry.
For this design we assume that the data is sent via the optical transceivers to an external destination and thus we use 10GBASE-R interfaces.


Figure 18: 10 Gb Ethernet Output Interface

### 4.8.4 Data Transfer Control

It is assumed that an Ethernet frame consists of 576 bytes, or 7264 -bit words on the write-side, or 14432 -bit words on the read side of the FIFO. The entire frame including a 6 -byte destination address, 6 -byte source address, 2 -byte type field, 42 -byte protocol header (IP, UDP etc.), 512 bytes payload, and an 8 -byte placeholder (set to 0 in the FIFO packet) must be written into the FIFO. The placeholder will be replaced with the CRC32 by the MAC, with 4 bytes going unused. The controller waits for at least this number of words to be present in the output FIFO, and transfers the packet in one transaction. Afterwards it waits a minimum of four clock cycles to satisfy the IPG requirements.


Figure 19: Data Transfer Controller

### 4.8.5 Complete Output Stage

A block diagram of the output stage is shown in Figure 20.


Figure 20: Complete Output Stage

### 4.9 System Outline

Figure 21 shows a sketch of the Uniboard and a 3D rendering of how the complete beamformer system could look like. The dimensions of the Uniboard are $280 \mathrm{~mm} \times 366.7 \mathrm{~mm}$. The midplane is designed for a 19 " rack and is of size $428 \mathrm{~mm} \times 387 \mathrm{~mm}$. Thickness is 3.3 mm .


Figure 21: Sketches of the Components

### 4.10 Design Summary

We have presented system and chip architecture for a beamformer based on the Uniboard ${ }^{2}$ signal processing platform. The performance figures are oriented towards the SKA Phase 1 requirements: 384 MHz observing bandwidth, 256 dual-polarization receivers ( 512 antenna signals), and 64 beams. The design target was to achieve a high resource utilization and a high compute efficiency. This has been achieved by a special data flow architecture, in which the stream of weightfactors, rather than the stream of samples, controls the sequence of operations. This allows a high percentage of chip resources such as hardware-multipliers, even if being an uneven number, to be employed. Also, any expensive redundant storage of weightfactors is avoided. An overall pipeline structure together with redundant buffers allows uninterrupted operation without a single-cycle loss.
The system is of moderate size (16 Uniboards and a backplane) and uses FPGAs in the slowest speed grade. This should help to keep the hardware and power supply costs low. Further studies in this direction („Green Measures") are detailled in section 6.

## 5 SCALABILITY

The multidimensional design space that needs to be considered for the specific requirements of radio astronomical facilities is spanned by the following parameters:

- desired observing bandwidth,
- number of antennas,
- number of beams,
- required numerical precision,
- data rate of the surrounding infrastructure,
- recurring costs (power consumption, cooling, building costs, maintenance etc.)
- non-recurring costs (design effort, component costs, manufacturing, deployment etc.)

In the remainder of this section we will explore possibilities for increasing the performance of the system with respect to observing bandwidth, number of antennas and number of beams, necessarily trading in arithmetical precision and costs. We'll start on the micro-architectural level (FPGA circuitry) and work our way up to the system level.

### 5.1 Alternative Arithmetic

Recalling (1) we can see that the dominant operation is a complex multiply, which when performed in Cartesian coordinates requires four multiplies and two adds. Therefore, the number of multiplier hardmacros on the FPGA defines the upper limit in performance. The obvious idea is to perform this operation in polar coordinates, which then takes the form

$$
\begin{equation*}
r_{1} \cdot r_{2} \quad \angle\left(\varphi_{1}+\varphi_{2}\right) \tag{5}
\end{equation*}
$$

The expenses are then reduced to one multiply and one add.
However, the complex multiply is immediately followed by an add, which cannot be done (so easily) in polar coordinates. Thus, the result (scaled and phase-shifted phasor) must be transformed into Cartesian coordinates immediately after the complex multiply. It is obvious that neither any further multiply nor any trigonometric functions can be employed for this operation.
The method of choice is therefore the well-known CORDIC algorithm [8]. CORDIC can be used to transform a complex number from Cartesian to polar coordinates and back. It is a multiplier-free, iterative shift-and-add operation that can be implemented as a pipeline for high throughput. In principle, the architectural changes to the system would be as shown in Figure 22. Note that all other aspects of the beamformer as presented so far, such as data paths, memory systems and flow control, remain unaffected.


Figure 22: Beamformer System using Polar Coordinates

So, the general idea is to implement all CORDIC units using the regular FPGA fabric and to pair each multiplier hardmacro with one CORDIC unit. In this way, the number of CXM-macros (see section 4.5.5) would be increased by a factor of four compared to the previous approach. Likewise, the overall chip performance would be increased by four, which could be used, for example, to compute four times the number of beams.
However, the CORDIC unit requires a certain amount of logic resources, depending on the desired precision. Available chip resources may put an upper limit on this approach. This is detailed in the following section.

### 5.1.1 The CORDIC Unit

Figure 23 shows the pipeline for transforming a complex number from polar coordinates to Cartesian coordinates (real and imaginary part). The necessary scaling operation at the end is not accounted for. Instead, it can be pre-multiplied into the weightfactor magnitude (5) or be deferred to later stages in the processing pipeline. Also, it is assumed that the phase $\beta$ of the operand has been normalized to a range of $-\pi / 2 \leq \beta \leq \pi / 2$. All operands are processed as two's complement fixed-point numbers.


Figure 23: CORDIC Unit
As can be seen in Figure 23, both precision and expenses depend on the following quantities:

- the bitwidth $K$ of the phase angle,
- the bitwidth $L$ of real and imaginary components, and
- the number $N$ of pipeline stages (iterations).

To a certain degree these quantities are correlated with each other. A high number of stages requires a certain bitwidth of the real and imaginary components or otherwise all bits will be shifted out. Likewise a certain angular resolution must be provided or otherwise the comparisons will be meaningless.
In [8] we find: "For most ordinary purposes, 40 iterations $(n=40)$ is sufficient to obtain the correct result to the 10th decimal place." This would roughly correspond to IEEE754 SP-FP accuracy. Other than this it is hard to give an analytic error function over the above quantities. Therefore a bitaccurate simulator was written in C to give visual representations of the error distribution. This simulator compares the outputs of the CORDIC pipeline to DP-FP results of the C library.
Figure 24 shows the relative error in percent of four selected configurations. The parameters $K, L$ and $N$ are shown next to the plots. In general, the error is pronounced towards small magnitudes. With decreasing parameter values the error becomes larger, but more importantly, it is pronounced for angles close to $0^{\circ}$ or $90^{\circ}$ irrespective of magnitude.


Configuration $A: K=24, L=24, N=24$


Configuration $B: K=16, L=16, N=16$


Configuration $\mathrm{C}: \mathrm{K}=12, \mathrm{~L}=12, \mathrm{~N}=12$


Configuration $\mathrm{D}: \mathrm{K}=8, \mathrm{~L}=12, \mathrm{~N}=8$



Error for Magnitude $<10 \%$ artificially set to zero





Figure 24: CORDIC Relative Error Distribution
Resource consumption is most critical for the adders in the CORDIC pipeline. In the FPGA-fabric, adders are implemented using LUTs and the fast carry chains. Altera FPGAs provide a certain number of $A L M s$ (Adaptive Logic Module). Each ALM can be configured to provide two full-adders. Thus, a 16-bit adder consumes 8 ALMs.
However, as we recall from Figure 10, each CXM is also paired with one M20K memory block. Thus, either ALMs or M20K blocks may become the limiting factor for this approach. Table I lists the available resources for both the low-cost Arria-10 GX 1150 and the high-end Stratix-10 GX 2800 .

Table I: Available FPGA Resources

| FPGA | Adaptive Logic Modules ALM | 18x19 Multiplier Hardmacros | M20K Memory Blocks |
| :---: | :---: | :---: | :---: |
| Arria-10 GX 1150 | 427,700 | 3,036 | 2,713 |
| Stratix-10 GX 2800 | 933,120 | 11,520 | 11,721 |

Table II and Table III list the FPGA resources required for different configurations. Fundamental quantity is the number of implemented beamformer cores, which each contain 16 CXMs and 20 M 20 K memory blocks (see section 4.5.2). In our traditional approach an Arria-10 GX 1150 allows 47 beamformer cores to be implemented. So we start the table with four times this number, or 188 beamformer cores.

Table II: FPGA Resource Requirements (Arria-10 GX 1150)

| Beamformer <br> Cores | 18x19 Mult. <br> Hardmacros | CORDIC <br> Config. | ALMs | \% of Chip | M20K Mem. <br> Blocks | \% of Chip |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 188 | 3,008 | A | $2,526,720$ | 591 | 3,760 | 139 |
| 188 | 3,008 | B | $1,106,944$ | 259 | 3,760 | 139 |
| 188 | 3,008 | C | 613,632 | 143 | 3,760 | 139 |
| 188 | 3,008 | D | 354,944 | 83 | 3,760 | 139 |
| 94 | 1,504 | C | 306,816 | 72 | 1,880 | 69 |
| 47 | 752 | B | 276,736 | 65 | 940 | 35 |

The bottom row in Table II represents the same performance as the traditional approach, however, with a potentially reduced accuracy. It might still be of interest if power consumption of the FPGA can be lowered using this kind of circuitry. This can be subject of further studies for deliverable D8.16, (,,green measures").
The configuration using 94 beamformer cores is of higher interest since it would double the performance (see sections 5.3 ff ), again at a somewhat reduced precision. All other configurations cannot be implemented. So, the optimistic goal of pairing each multiplier hardmacro with a CORDIC unit cannot be reached without architectural changes.

For the Stratix-10 GX 2800 FPGA we start the table with the theoretical maximum number of beamformer cores, that is $11,520 / 16=720$. Since the number of ALMs only scales by a factor of 2.2 for the Stratix device, this method appears to be of little use on this plattform.

Table III: FPGA Resource Requirements (Stratix-10 GX 2800)

| Beamformer <br> Cores | 18x19 Mult. <br> Hardmacros | CORDIC <br> Config. | ALMs | \% of Chip | M20K Mem. <br> Blocks | \% of Chip |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| 720 | 11,520 | A | $9,676,800$ | 1037 | 14,400 | 123 |
| 720 | 11,520 | B | $4,239,360$ | 454 | 14,400 | 123 |
| 720 | 11,520 | C | $2,350,080$ | 252 | 14,400 | 123 |
| 720 | 11,520 | D | $1,359,360$ | 146 | 14,400 | 123 |
| 360 | 5,760 | C | $1,175,040$ | 126 | 7,200 | 61 |
| 180 | 2,880 | C | 587,520 | 63 | 3,600 | 31 |

### 5.1.2 Conclusion

Performing the complex multiply in polar coordinates suffers from a high resource consumption and low precision. Still there might be configurations and scenarios where this method can be used advantageously. This depends on application, chip architecture and downstream processing requirements and is therefore simply an engineering decision. The problems largely stem from the inefficient adder-logic on FPGAs. Therefore, especially for lowpower ASIC-designs it might be worth to re-evaluate the method.

### 5.2 Alternative FPGAs

As advertised by Altera, it was planned from the start of the Uniboard ${ }^{2}$ project to initially use the low-cost Arria-10 device and later upgrade to a more powerful, pin-compatible Stratix-10 device. As of now, however, there is no such device on the market [9]. Accordingly, there is very little information about performance (maximum core or DSP clock frequency) available. We therefore assume that the device in question will be a Stratix-10 GX 2800 [10] and conservatively estimate that the core clock can be as high as 500 MHz . As listed in Table I, there are 11,520 multiplier hardmacros and 11,721 M20K memory blocks on the chip. We further assume that all multipliers are used to construct beamformer cores and that other arithmetic units such as format converters are made from fabric logic. Then we will have 180 beamformer cores.

### 5.3 Increasing the Number of Beams

### 5.3.1 Circuit-Level

In the following we assume that the number of beamformer cores is doubled to 94 (Arria-10 using CORDIC, trading in accuracy) or approximately quadrupled to 180 (Stratix-10, at much higher chip costs).
In this section we assume further that the increase in processing power shall solely be used to double or quadruple the number of computed beams, while keeping the observing bandwidth ( 384 MHz ), number of channnels (384) and number of antennas (512) constant. The obvious consequences are:

- the input data rate (per clock) remains the same,
- both the Input Fan-Out Register Tree (see Figure 6) and the Weightfactor Fan-Out Register Tree (see section 4.7) need to service twice or four times the number of beamformer cores and might need to be adapted,
- the size of the Weightfactor Memory (see Figure 15) increases in size,
- the output data rate per clock increases by a factor of two or four.

While for the traditional Stratix-10 architecture all units scale linearly, we can reduce precision at several places in the Arria-10 CORDIC system to the precision that the CORDIC units provide, in this case 12 bits. Each of the eight weightfactor memory banks therefore has a width of $2 * 12 * 16=$ 384 bits and a depth of 4096 words. It can be constructed from 80 M 20 K memory blocks, for a total of 640 M 20 K memory blocks.
Since the number of antennas has not changed each beamformer core still produces one beam sample every 32 clocks. A beam sample consists of two SP-FP numbers of 32 bits each. Combined output data rate of 8 beamformer cores is therefore $5.76 \mathrm{~Gb} / \mathrm{s}$ (Arria-10) or $8 \mathrm{~Gb} / \mathrm{s}(\mathrm{Stratix}-10)$. Again we use one Output Unit (see Figure 16) for 7 or 8 beamformer cores.
Hardware requirements are summarized in Table IV.

## Table IV: Increasing the Number of Beams

| Architecture | Observing <br> Bandwidth | \# of <br> Antennas | \# of <br> Beams | \# of <br> $18 \times 19$ <br> Multipliers | \# of <br> M20K | \# of <br> Output <br> Units | Output Data <br> Rate (Gb/s) |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Arria-10 CORDIC | 384 MHz | 512 | 128 | 1,504 | 2,656 | 12 | 67.68 |
| Stratix-10 | 384 MHz | 512 | 256 | 11,520 | 5,808 | 24 | 180.0 |

Looking at Table IV it appears to be possible to compute twice or four times the number of beams without changes to the architecture if we accept reduced accuracy or higher chip costs.

### 5.3.2 System-Level

By simply replicating the entire system of 16 Uniboards (and the midplane), the number of beams can also be multiplied provided each system receives the proper set of weightfactors [11]. The Uniboards performing the polyphase filterbanks are redundant, though.

### 5.4 Increasing the Observing Bandwidth

### 5.4.1 Circuit-Level

For this study we assume that the observing bandwidth shall be doubled ( 768 MHz , Arria-10 CORDIC) or quadrupled (1536MHz, Stratix-10). Channel bandwidth shall remain at 1 MHz . That is, the increase in beamformer cores shall be used to compute the same number of beams from the same number of antennas, but on twice or four times the number of channels.
Since the filterbank design is not part of this workpackage we simply assume that it can be implemented on the respective plattform. However, we have to examine the bitrates at the input and outputs, i. e., at the optical interfaces and on the midplane.
If we assume 8 bits per sample from the ADC , the input data rate for each filterbank-FPGA is $1,536 \mathrm{e} 6 * 8 * 32=393 \mathrm{~Gb} / \mathrm{s}(\mathrm{Arria}-10)$ or $786 \mathrm{~Gb} / \mathrm{s}$ (Stratix-10). If we restrict ourselves to using current 40 GbE technology, these kinds of bandwidth are not available at the optical connectors of the fil-terbank-FPGAs alone. Thus, antenna connections must be distributed across the entire system, and routed via the midplane to the Uniboards that implement the filterbanks. This is shown for the Stratix-10 case. We assume, however, a collection of external switches that aggregate the data streams from the ADCs. An example configuration is shown in Figure 25.


Figure 25: Aggregation Switches and Uniboards

As can be seen, each filterbank-FPGA receives 8 antenna signals directly via its own optical interfaces. The remaining 24 signals must be routed through the midplane from the beamformer-FPGAs. As we recall from Figure 3, there is exactly one FD lane between any given pair of beamformerFPGA and filterbank-FPGA. One direction is used to distribute the ADC samples, as shown in Figure 26.


Figure 26: ADC Sample Distribution ( $1 / 2$ Uniboard shown, other Half identical for remaining Antenna Signals)
As can be seen, the 512 antenna signals can nicely be routed to the filterbank-FPGAs provided that the link speed across the midplane can be set to a net data rate of $12.288 \mathrm{~Gb} / \mathrm{s}$.
In reverse direction, depending on precision, the required data rate can exceed physical capabilities of the transceivers and wires. Each filterbankFPGA needs to send 16 or 32 channels from 32 antennas to a given beamformer-FPGA. Required data rates are shown in Table V .

Table V: Required Data Rate from Filterbank-FPGA to Beamformer-FPGA

| Architecture | \# of Channels | Data Rate @ 16;16 | Data Rate @ 12;12 | Data Rate @ 8;8 |
| :---: | :---: | :---: | :---: | :---: |
| Arria-10 CORDIC | 16 | $16.384 \mathrm{~Gb} / \mathrm{s}$ | $12.288 \mathrm{~Gb} / \mathrm{s}$ | $8.192 \mathrm{~Gb} / \mathrm{s}$ |
| Stratix-10 | 32 | $32.768 \mathrm{~Gb} / \mathrm{s}$ | $24.576 \mathrm{~Gb} / \mathrm{s}$ | $16.384 \mathrm{~Gb} / \mathrm{s}$ |

Thus, one can be optimistic about using 12-bit components for Arria-10 and 8-bit components for Stratix-10. Better accuracy for Stratix-10 will not be feasible for a midplane of this size.
The Input Stage on each beamformer-FPGA architecturally remains the same, however, the rate at which complex samples arrive is increased to $512 \mathrm{MS} / \mathrm{s}$ or $1 \mathrm{GS} / \mathrm{s}$. In the current architecture, consecutive samples in the input FIFOs (see Figure 5) go to the same memory block in the Memory and CXM Stage (see Figure 10). Thus, one would need to increase the transfer rate (write cycles per second) beyond the set limits.
A solution to this problem could be to rearrange the chip resources. In particular, the beamformer core would need to be extended to include 32 instead of 16 Memory and CXM Units. Obviously, the number of beamformer cores on each FPGA would then be halved. Data distribution for a given input vector would then be as shown in Figure 27.


Figure 27: Data Distribution in Modified Beamformer Core

In this way two complex samples are written per clock, bringing bus and memory clock rates back into the legal range. However, there are a number of system implications that need to be considered:

- Depending on sample precision, the width of the input FIFOs increases. Also, register consumption of the Input Fan-Out Register Tree increases to 41,472 flipflops ( $2.4 \%$ on Arria-10).
- The memory in a beamformer core should be consolidated. For 12 -bit components the total width of the memory is 768 bits. Thus the memory can be constructed using 20 M20K blocks. For the Arria-10 version, the beamformer unit would then consume a total of 1,128 M20K blocks including accumulators. For the Stratix-10 version, M20K blocks are not a critical resource.
- Width, depth and capacity of the Weightfactor Memory change significantly. The Weightfactor Memory has to provide 32 weights on 16 or 32 channels in parallel. This amounts to a data bus of 12,288 bits (Arria-10) or 16,384 bits (Stratix-10). The depth is 1,024 such words. The Weightfactor Memory can be built using 616 (Arria-10) or 820 (Stratix-10) M20K blocks.
- The Weightfactor Fan-Out Register Tree consumes 663,552 flipflops on the Arria-10 FPGA, or $39 \%$ of available resources. The CORDIC-pipelines consume approximately 650,000 flipflops. At such a high usage percentage, routing might become a problem. For Stratix-10 this is not an issue because of the HyperFlex-feature (registers embedded in the routing channels).
- Each beamformer core now produces one beam sample every 16 clocks. This amounts to a bit rate of $1.44 \mathrm{~Gb} / \mathrm{s}$ (Arria-10) or $2 \mathrm{~Gb} / \mathrm{s}$ (Stratix-10) per beamformer core. Thus we use one Output Unit (see Figure 16) per four beamformer cores, for a total of 12 (Arria-10) or 24 (Stratix-10) interfaces.

Table VI shows a summary of performance and resource consumption of this approach.
Table VI: Increasing the Observing Bandwidth

| Architecture | Observing <br> Bandwidth | \# of <br> Antennas | \# of <br> Beams | \# of <br> Beamformer <br> Cores | \# of <br> $18 \times 19$ <br> Multipliers | \# of <br> M20K | \# of <br> Output <br> Units | Output Data <br> Rate (Gb/s) |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Arria-10 CORDIC | 768 MHz | 512 | 64 | 47 | 1,504 | 2,052 | 12 | 67.68 |
| Stratix-10 | $1,536 \mathrm{MHz}$ | 512 | 64 | 90 | 11,520 | 4,060 | 24 | 180.0 |

### 5.4.2 System-Level

Provided that

- suitable external switches for the purpose as shown in Figure 25 are available,
- the digitized antenna signals are fed into the system as shown in Figure 26, and
- the necessary filterbanks can be implemented on the FPGAs
a collection of the original beamformer systems can be operated in parallel for processing larger observing bandwidths. Each system would then process / discard a disjoint set of frequency channels. Again all but one of the Uniboards that are used for filtering are redundant.

In case implementing these large-bandwidth filterbanks is a problem a low-cost compromise could be to use Stratix-10 for the filterbank-FPGAs and Arria-10 otherwise.

### 5.5 Increasing the Number of Antennas

### 5.5.1 Circuit-Level

In this section we examine if and how the number of antennas can be increased to 1,024 and 2,048 , respectively. That is, the increased processing power shall be used to process more antennas, but over the same bandwidth and for the same number of beams.

The obvious consequence is that the external aggregation switches need more ports, most of them with lower bandwidth. Per ADC we have $768 \mathrm{e} 6 * 8$ $=6.144 \mathrm{~Gb} / \mathrm{s}$. For 2,048 antennas, the switches in Figure 25 should be replaced by switches that provide 3210 GbE ports and 640 GbE ports, or equivalent. Data distribution is then identical to Figure 26 except that the data streams across the midplane carry signals from multiple antennas.

On the filterbank-FPGAs twice or four times the number of polyphase filterbanks need to be implemented. Again we simply assume that this can be done. For the data rate between any given pair of filterbank-FPGA and beamformer-FPGA the same considerations apply as listed in Table V, in this case, however, depending on the number of antennas.

Table VII: Required Data Rate from Filterbank-FPGA to Beamformer-FPGA

| Architecture | \# of Antennas | Data Rate @ 16;16 | Data Rate @ 12;12 | Data Rate @ 8;8 |
| :---: | :---: | :---: | :---: | :---: |
| Arria-10 CORDIC | 1,024 | $16.384 \mathrm{~Gb} / \mathrm{s}$ | $12.288 \mathrm{~Gb} / \mathrm{s}$ | $8.192 \mathrm{~Gb} / \mathrm{s}$ |
| Stratix-10 | 2,048 | $32.768 \mathrm{~Gb} / \mathrm{s}$ | $24.576 \mathrm{~Gb} / \mathrm{s}$ | $16.384 \mathrm{~Gb} / \mathrm{s}$ |

For the Input Stage the sample rate is increased similar to section 5.4.1. Again a solution could be to build beamformer cores from 32 Memory and CXM Units. The layout of one input vector from 2,048 antennas across 32 memory systems is shown in Figure 28. As in section 5.4.1, a beamformer core memory can be built from 20 M20K blocks, for a total of 1,128 M20K blocks (Arria-10).


Figure 28: Data Distribution for 2,048 Antennas

The Weightfactor Memory again has to provide 32 weightfactors in parallel, however, in this configuration only on 8 channels. Thus, for the Arria-10 CORDIC system this bus is 6,144 bits wide, for the Stratix-10 system 4,096 bits. The depth is $64 * 32$ (Arria-10) or $64 * 64$ (Stratix-10) such words. The Weightfactor Memory can be built using 616 (Arria-10) or 824 (Stratix-10) M20K blocks.
The Weightfactor Fan-Out Register Tree consumes 331,776 flipflops on the Arria-10 FPGA, or $19 \%$ of available resources.
Each beamformer core produces one beam sample every 32 (Arria-10) or 64 (Stratix-10) clocks. This amounts to an average bit rate of $720 \mathrm{Mb} / \mathrm{s}$ (Arria-10) or $500 \mathrm{Mb} / \mathrm{s}$ (Stratix-10). Thus we can use one Output Unit (see Figure 16) per 8 (Arria-10) or per 16 (Stratix-10) beamformer cores.
Table VIII shows a summary of performance and resource consumption of this approach.
Table VIII: Increasing the Number of Antennas

| Architecture | Observing <br> Bandwidth | \# of <br> Antennas | \# of <br> Beams | \# of <br> Beamformer <br> Cores | \# of <br> $18 \times 19$ <br> Multipliers | \# of <br> M20K | \# of <br> Output <br> Units | Output Data <br> Rate $(\mathrm{Gb} / \mathrm{s})$ |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Arria-10 CORDIC | 384 MHz | 1,024 | 64 | 47 | 1,504 | 2,010 | 6 | 33.84 |
| Stratix-10 | 384 MHz | 2,048 | 64 | 90 | 11,520 | 4,024 | 6 | 45.0 |

### 5.5.2 System-Level

Multiple of the original beamformer systems can be operated in parallel to process more antennas. Provided, however, that there is a downstream device that performs the final accumulation. As was detailled in section 4.8 there are six output stages on each beamformer-FPGA, each one connecting to one 10 GbE -line. Net data rate per line is $5.76 \mathrm{~Gb} / \mathrm{s}$.
An example configuration could be four systems processing 2,048 antennas. The accumulation device shall also be built from Uniboards.
For this configuration, the outputs of four corresponding beamformer-FPGAs need to be summed up and sent off to the central compute facility. Thus, the accumulating device needs 24 input-channels for each set of four beamformer-FPGAs. The output data rate is of course equal to that of a single beamformer-FPGA, and requires again six 10 GbE -channels.
As can be seen in Figure 3, each FPGA exposes 24 FD lines towards the optical transceivers, and 48 FD lines towards the midplane. By using break-out-boards that simply connect 24 FD lines from the midplane to QSFP-cages, one Uniboard can accumulate the outputs of 3 beamformer-Uniboards. Since a beamformer system contains 12 beamformer-Uniboards, we would need 4 accumulator-Uniboards, 8 breakout-boards and a special midplane for this kind of post-processing.
Processing requirements are moderate. Each beamformer-FPGA outputs SP-FP complex beam samples at a rate of $360 \mathrm{MHz} / 32 * 47=528.75 \mathrm{e} 6$ numbers per second. Each summation requires 6 FP operations, corresponding to 3.1725 GFlops , or roughly 10GFlops for each accumulator-FPGA. This is well within reach.
Figure 29 shows a 3D sketch of a breakout board and the accumulation unit. The midplane has the same dimensions as in Figure 21.


Figure 29: Sketch of Breakout Board and Accumulation Unit

### 5.6 Scalability Summary

On the microarchitectural level we have explored several points in the design space, as shown in Figure 30. The far-out points require the more expensive Stratix- 10 FPGAs. It stands to reason that this configuration can also implement other points in the spanned design space. We have also shown how multiple beamformer systems can be operated in parallel to be suitable for certain scenarios in the SKA environment.


Figure 30: Design Space

### 5.7 Alternative Architectures

One class of beamformers, called FFT-based beamformers, is not considered here since their use is mostly in radar mapping. Apart from the presented architecture, which is commonly called "subband beamformer", the most prominent other architecture is called "ring beamformer". The different architectures are shown in Figure 31, which was adopted from [13].


Figure 31: Subband and Ring Beamformers

The obvious advantage of the ring architecture is its simplicity (the cross-connect is avoided) and the fact that the number of antennas scales easily. However, the bandwidth of the ring connect (which also defines the maximum output rate) can quickly become a limiting factor when the number of beams increases. If $B$ beams are to be generated, then for each input sample $B$ output samples need to be put on the ring. Thus, this architecture is most suitable for single-beam beamformers. In case of the Uniboard ${ }^{2}$, the ring is a relatively low-bandwidth channel, so this architecture is probably not optimal.

### 6.1 Introduction

In this work the term "Green Measures" solely refers to methods for reducing the power consumption. Other aspects such as design for environmentfriendly manufacturing, safe disposal and recycling are not considered here.
Power consumption of computing machinery is affected by the following items:

1. Choice and design of the platform, chip technology,
2. required computing performance,
3. efficiency of the algorithms,
4. quality of implementation of the hardware, gateware, firmware and software,
5. presence or absence of power saving provisions.

### 6.1.1 Choice and Design of the Platform, Chip Technology

This has been done in another subproject, so it is not subject of this work. No recommendations regarding circuit desing or choice of components will be given.

### 6.1.2 Required Computing Performance

In case of a large and expensive installation such as an observing station with many antennas the required computing performance is fixed and must be provided under all circumstances. From a power consumption point of view it is best to operate a minimum set of systems at peak performance, rather than operating a larger number of systems at lower computing performance. This implies that any design effort should be spent on achieving or approaching peak performance of the individual systems.

### 6.1.3 Efficiency of Algorithms

An efficient algorithm computes the desired result with a minimum number of operations, or with the least expensive set of operations. This implies that algorithm development should take machine capabilities into account. For example, DSP blocks are very power efficient and can operate at higher frequencies than their fabric counterparts. Other quantities to minimize are data transfers, and storage space.
A complex sum of products leaves very little room for algorithmic optimizations. We have tried to use a different coordinate system (polar coordinates) in order to simplify the complex multiplication (see section 5.1). However, due to the peculiarities of FPGAs this was not very successful (see section 6.6). On other platforms such as full-custom ASICs this might be fundamentally different.

### 6.1.4 Quality of Implementation of the Hardware, Gateware, Firmware, and Software

An efficient implementation must avoid losses. Losses occur when all data is available, a processing element is free and still no activity takes place. Losses include scheduling, communication and synchronization losses. From a power consumption point of view this represents a power waste since any clock cycle that does not advance computation still consumes energy.
We have developed an architecture that avoids any losses and performs useful operations in every single clock. This was achieved by constantly streaming the weightfactors along with their indices through the circuitry (see section 4.5.1). Bubble-free operation is also supported by special circuitry such as a double-buffered accumulator (see Figure 14).
The price to pay are relatively high hardware expenses, and with it power consumption. A way to reduce the hardware and power costs by architectural means is outlined in section 6.5 , and used to the maximum extent in the final design.
On digital CMOS circuits the major goal of a power-efficient implementation is to reduce the number of signal transitions. Methods include lowpower coding, clock gating, selective activation and more. Several of these methods are discussed in section 6.4.
On FPGAs the actual (micro-) implementation is to a large extent beyond the influence of the circuit designer. However, one can loosely steer operation by means of a large number of compiler switches. Effectiveness of such compiler-oriented power optimizations is detailled in section 6.7.

### 6.1.5 Presence or Absence of Power Saving Provisions

Saving power by switching off complete systems such as a server computer is relatively simple from a technical viewpoint, although overall power down and power up procedures can be tricky, and increased wear of the components can be expected. Powering down individual components within such a system is far more complicated and requires comprehensive design support in the power supply system, interconnect structure, firmware and operating system. Applications must be aware of such platform changes and must be able (maybe even on-the-fly) to migrate processing tasks and data streams from a powered-down component to an alive one. In this work we do not consider such scenarios, e.g. powering down individual Uniboards in a rack or individual FPGAs on a given Uniboard. However, we examine the effects of using only parts of a given FPGA, and by that the ability of the tools and the hardware to power down unused tiles (see section 6.8).

### 6.2 Tools and Procedures for Power Consumption Measurements

In the absence of a Uniboard ${ }^{2}$ hardware system all power consumption measurements of the gateware are estimates obtained from Altera design tools, in particular from Altera PowerPlay. The tool flow, and methods to circumvent certain limitations of the Altera tools, are described in section 6.3.

### 6.3 Tool Flow

It seems that Quartus II 15.0 (Q15) does not support gate-level timing simulation for Arria 10 designs. For an accurate power consumption estimation using PowerPlay, a precise description of toggle activities per node is required. These are normally recorded during a gate-level timing simulation.
This short application note describes how to obtain a .vcd-file (Value Change Dump) from gate-level functional simulation. This simulation method generally assumes a propagation delay of 0 and might therefore not capture all transitions. The hope is that this is still more aligned with the physical behavior of the chip than a default toggle rate.
Despite the reduced complexity, gate-level functional simulation can still be too time consuming to be practical. For complex designs one has to resort to using PowerPlay with less accurate default toggle rates.

### 6.3.1 Procedure

The steps are outlined using a small test design. Assumed we have set up a Q15 project for an Arria 10 10AX115U4F45I3SGES part with just one VHDL-file:

```
library IEEE;
use IEEE.STD LOGIC 1164.ALL;
use IEEE.STD LOGIC-
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity tripleadd is
    port ( clk : in std_logic;
            a : in std_logic_vector (23 downto 0);
            b : in std_logic_vector (23 downto 0);
            C : in std_logic_vector (23 downto 0);
            d : out std_logic_vector (23 downto 0)':= x"000000");
end entity
architecture rtl of tripleadd is
    signal areg, breg, creg : std_logic_vector(23 downto 0) := x"000000";
begin
process (clk)
begin
    if (rising_edge(clk)) then
        areg <= a;
        breg <= b;
        creg <= c;
        d <= areg + breg + creg;
    end if;
end process;
end rtl;
```

Using Processing $\rightarrow$ Start $\rightarrow$ Start Test Bench Template Writer we generate a test bench skeleton and fill it arbitrarily:

```
LIBRARY IEEE;
USE IEEE.STD LOGIC 1164.ALL;
USE IEEE.STD LOGIC ARITH.ALL;
USE IEEE.STD__LOGIC_UNSIGNED.ALL;
ENTITY tripleadd vhd tst IS
END tripleadd_vh\overline{d_tst;}
ARCHITECTURE tripleadd_arch OF tripleadd_vhd_tst IS
SIGNAL Clk : STD LOGIC := '0';
SIGNAL a, b, c : STD_LOGIC_VECTOR(23 DOWNTO 0) := x"000000";
SIGNAL d : STD_LOGIC_VECTOR}(23 DOWNTO 0)
COMPONENT tripleadd
PORT ( clk : IN STD_LOGIC;
        a : IN ST̈D_LOGIC_VECTOR (23 DOWNTO 0);
        b : IN STD_LOGIC_VECTOR (23 DOWNTO 0);
        c : IN STD_LOGIC_VECTOR (23 DOWNTO 0);
END COMPONENT; OUT STD_LOGIC_VECTOR(23 DOWNTO 0));
BEGIN
clk <= NOT clk AFTER 25 ns;
instl : tripleadd PORT MAP ( }\textrm{M}=>\textrm{a},\textrm{b}=>\textrm{b},\textrm{c}=>\textrm{c},\textrm{clk}=>clk, d => d )
always : PROCESS (clk)
BEGIN
    IF (FALLING_EDGE(clk)) THEN
        a <= a + 1;
        b <= b + 3;
        c}<=C+
    END IF;
END PROCESS always;
END tripleadd_arch
```

For NativeLink we need to fill in some forms. Afterwards the "Settings"-form should look like shown below. Note that we are using the built-in Altera-ModelSim Starter Edition.


Figure 32: Settings for Simulation

Clicking on the "Test Benches..."-Button will open the following dialog box (shown after making changes):


Figure 33: Test Bench Settings

Clicking on the "New.."-Button will open the final dialog box for this step:


Figure 34: New Test Bench Settings
We could now perform an RTL simulation.
However, our goal is to run gate-level simulations so we need to make further changes. These relate to the EDA Netlist Writer. First we need to instruct it to generate a tcl-script which in turn instructs the simulator to record toggle activities of all nodes. Further we instruct it to generate a doscript for a gate-level functional simulation. After all the changes the dialog boxes look like shown in Figure 35 through Figure 37.


Figure 35: EDA Netlist Writer Settings

When clicking on "Script Settings", a dialog box should pop up:


Figure 36: VCD File Script Settings
After clicking on "More EDA Netlist Writer Settings", change the option "Generate netlist for functional simulation only" from Off to On.


Figure 37: More EDA Netlist Writer Settings

Now the Q15 tools "Analysis \& Synthesis", "Fitter (Place and Route)" and "EDA Netlist Writer" must be executed. Before we do this, we should set up a minimal constraints file defining the clock and a few delays (so that the Timing Analyzer won't complain):
\# Create a simple 50ns clock
create_clock -period 50 -waveform $\left\{\begin{array}{lll}0 & 25\end{array}\right\}$-name clk [get_ports clk]
set_input_delay -clock \{ clk \} 10 [get_ports \{a[*]\}]
set_input_delay -clock \{ clk \} 10 [get_ports \{b[*]\}]
set_input_delay -clock \{clk \} 10 [get_ports \{c[*]\}]
set_output_delay -clock \{ clk \} 10 [get_ports \{d[*]\}]
Additionally we might then want to make all I/O assignments. Now we can step through the tool flow above.
After quitting ModelSim we will find a .vcd-file at simulation/modelsim/tripleadd.vcd.

Using this .vcd-file we can now configure PowerPlay. The "Settings" dialog box should then look like in Figure 38.


Figure 38: PowerPlay Settings
While running PowerPlay it will report:
Info (222002): Starting scan of VCD file simulation/modelsim/tripleadd.vcd (o ns to End of File) for signal static probabilities and transition densities
Info (222003): Finished scan of VCD file simulation/modelsim/tripleadd.vcd (0 ns to End of File) for signal static probabilities and transition densities
When PowerPlay has finished execution it will report a high "Power Estimation Confidence", which was the goal of this exercise (see Figure 39a). When using a default toggle rate instead, the confidence is low: "user provided insufficient toggle rate data" (see Figure 39b). Note also the fairly large dynamic power estimate differences.

| 4 PowerPlay Power Analyzer To | $\times$ |
| :---: | :---: |
| PowerPlay Power Analyzer Summary |  |
| PowerPlay Power Analyzer Status | Successful - Thu Sep 10 17:01:59 2015 |
| Quartus II 64-Bit Version | 15.0.0 Build 145 04/22/2015 SJ Full Version |
| Revision Name | tripleadd |
| Top-level Entity Name | tripleadd |
| Family | Arria 10 |
| Device | 10AX115U4F45I3SGES |
| Power Models | Preliminary |
| Total Thermal Power Dissipation | 2205.12 mW |
| Transceiver Standby Thermal Power Dissipation | 0.00 mW |
| Transceiver Dynamic Thermal Power Dissipation | 0.00 mW |
| I/O Standby Thermal Power Dissipation | 4.25 mW |
| I/O Dynamic Thermal Power Dissipation | 1.14 mW |
| Core Dynamic Thermal Power Dissipation | 1.62 mW |
| Device Static Thermal Power Dissipation | 2198.11 mW |
| Power Estimation Confidence | High: user provided sufficient toggle rate data |

a.) Using the .vcd-file

| PowerPlay Power Analyzer Tool |  |
| :--- | :--- |
| PowerPlay Power Analyzer Summary |  |
| PowerPlay Power Analyzer Status | Successful - Thu Sep 10 17:29:58 2015 |
| Quartus II 64-Bit Version | 15.0 .0 Build 145 04/22/2015 SJ Full Version |
| Revision Name | tripleadd |
| Top-level Entity Name | tripleadd |
| Family | Arria 10 |
| Device | 10AX115U4F45I3SGES |
| Power Models | Preliminary |
| Total Thermal Power Dissipation | 2206.32 mW |
| Transceiver Standby Thermal Power Dissipation | 0.00 mW |
| Transceiver Dynamic Thermal Power Dissipation | 0.00 mW |
| I/O Standby Thermal Power Dissipation | 4.21 mW |
| I/O Dynamic Thermal Power Dissipation | 1.97 mW |
| Core Dynamic Thermal Power Dissipation | 1.98 mW |
| Device Static Thermal Power Dissipation | 2198.15 mW |
| Power Estimation Confidence | Low: user provided insufficient toggle rate data |

b.) Using a default toggle rate of $12.5 \%$

Figure 39: PowerPlay Power Analyzer Summary

### 6.4 Power-Efficient Gateware Implementation

Power consumption of a digital CMOS circuit can be divided into static and dynamic power consumption [14]. Static power consumption occurs without signal activity and is mostly due to leakage currents. Dynamic power consumption occurs during a signal level change at the output of a logic circuit. Static power consumption can hardly be influenced by a gateware designer except when given the possibility to selectively power down parts of the chip. In order to reduce dynamic power dissipation, the goal is to reduce overall switching activity within the design while maintaining functionality and performance. Several techniques have been developed over time [15], and a few will be discussed in this section.

### 6.4.1 Traveling Enable-Signal on Pipeline Registers

In the simplest case operands are continuously clocked through pipeline stages consisting of registers and combinational logic as shown in Figure 40 a. However, operation semantics might require that patterns that do not represent valid data are flagged. Such flags can also be used to prevent the registers from changing state, as shown in Figure 40b.


Figure 40: Traveling Enable Signal

Power savings can come from two sources:

- the register clock is gated, which keeps internal nodes and output signals from changing state,
- there are no signal changes througout the combinational logic (if present), nor at the inputs of the receiving register.

These savings are hard to quantify, since they depend strongly on the functionality of the design, and on the ratio of valid to invalid data slots. The method also presents costs to the design, since the clock-enable signals can have a large fan-out (depending on the width of the registers) and can present routing and/or timing problems. In the extreme case the clock-enable signals must be duplicated using additional registers.
In the design presented here there are several large pipeline structures which theoretically are subject to this optimization. This includes the fan-out trees for the samples and the weightfactors, and the adder trees (see Figure 12). However, both the weightfactor stream as well as the sequence of complex products do not have bubbles, so the method does not apply here.
This is different for the input fan-out register tree (see Figure 6 and Figure 45) since any given input vector has only one destination and so only one path through the tree is used. Thus, for a binary tree only $\log _{2} \mathrm{~N}$ instead of $2 * \mathrm{~N}-1$ registers are actually used, N being the number of leaves (ports). Two test designs have been made, each including one instance of the input fan-out tree. TreeTest includes clock-enable signals as shown in Figure 45. For the design named TreeTest_NoEn clock-enable signals are absent and so all registers are clocked all the time. Both designs were run through the tools as described in section $6 . \overline{3}$ to obtain accurate toggle rates. The results are shown in Table IX.

Table IX: Input Fan-Out Register Tree Power Consumption

| Enable- <br> Signals | Logic Utilization | Registers <br> used | Average Toggle <br> Rate | Core Dynamic <br> Power Consumption | Device Static <br> Power Consumption | Total <br> Power Consumption |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Present | 12,897 ALMs | 34,719 | $1.366 \mathrm{MT} / \mathrm{s}$ | 76.98 mW | 2202.51 mW | 2280.67 mW |
| Absent | $12,775 \mathrm{ALMs}$ | 34,252 | $37.050 \mathrm{MT} / \mathrm{s}$ | 529.78 mW | 2230.12 mW | 2761.08 mW |

Significant power savings can be obtained despite the fact that the pipeline stages do not contain any combinational logic. Thus, from a power consumption perspective, any FPGA-design should be examined carefully, and clock-enable signals should be included where applicable.

### 6.4.2 Organization of on-chip Memory Systems

FPGAs typically offer memory blocks implemented as hard-macros. Sizes can vary but are often in the order of 16 to 20 kBit . Typically these blocks are true dual-port, independent-clock RAMs with a memory cell array of $512 \times[32|36| 40]$ bits. In order to increase data rate (at the expenses of access latency) the blocks include selectable registers at the address and data inputs as well as at the data outputs. The blocks further include data multiplexers so that from an outside view the organization is configurable, i.e. from $512 \times 32$ to $1024 \times 16,2048 \times 8$ and so forth to finally $16384 \times 1$. A simplified diagram of a memory block is shown in Figure 41a.

As long as the desired memory depth does not exceed that of the $\times 1$-configuration large memory systems can be built by horizontal concatenation of memory blocks up to the chip capacity (see Figure 41b).


Figure 41: Memory Block and Memory System
On the VHDL-level, the gateware designer has two choices for including hard-macros: inference, or explicit instantiation. The former is often preferred because of its convenience and portability, leaving the concrete implementation details to the synthesis tools.
For example, the weightfactor memory bank in Figure 15 can easily be described as follows (only the read-port is shown):

```
type wfmemtype is array(0 to 2047) of std_logic_vector(607 downto 0);
signal wfmem_bank : wfmemtype;
signal intdata, outdata: std_logic_vector(607 downto 0);
signal addr : std_logic_vector(10 downto 0);
signal rden : std_logic;
process (clk)
begin
    if (rising_edge(clk)) then
        if (rden = '1') then
            intdata <= wfmem_bank(conv_integer(addr));
        end if;
        outdata <= intdata;
    end if;
end process;
```

This example uses both the address and the data output registers of each block. The general synthesis rules inside an EDA tool such as Altera Quartus for generating a memory system from the behavioral description above are of course unknown to the average designer, but it has been observed that the tools tend to use the internal data multiplexers and construct a system as in Figure 41 b from 61 memory blocks organized as $2048 \times 10$ bits. This appears to be the case irrespective of compiler settings (optimization mode performance vs. power).
From a power consumption point of view, the problem with this architecture is that for any access, all blocks are activated and therefore consume power. In contrast, consider the alternative implementation in Figure 42 . Here, only 16 memory blocks are active for a given access by means of the Enable-signals from the address decoder. However, several implications need to be considered:

- a separate MUX-stage in the FPGA-fabric has to be implemented, consuming logic resources and reducing the power savings from the selective activation,
- the latency will be increased by one for the same clock frequency, so it is a prerequisite that the overall design can tolerate this,
- data bus granularity is now 40 bits, leading to increased consumption of memory blocks ( 64 vs .61 ),
- the design effort is increased since the structure must be coded by hand and the memory blocks should be instantiated instead of inferred.

In order to evaluate the power consumption two test designs have been made: MemTest and MemTest_IP. Both are initiated with the same data. MemTest uses the behavioral description and is synthesized using memory blocks in $2048 \times 10$-configuration. MemTest_IP is implemented as shown in Figure 42 using M20K IP-blocks generated by QSys in a $512 \times 40$-configuration. PowerPlay was run over both designs as described in section 6.3. The results are shown in Table X. MemTest_IP uses only about $75 \%$ of the dynamic core power that MemTest consumes, despite the added multiplexer stage.

Table X: Power Consumption of Different Memory Architectures

| Design | ALMs | Registers | M20K <br> Blocks | Average Toggle <br> Rate | Core Dynamic <br> Power Consumption | Device Static <br> Power Consumption | Total <br> Power Consumption |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| MemTest | 16 | 56 | 4 | $8.736 \mathrm{MT} / \mathrm{s}$ | 2.09 mW | 2198.2 mW | 2207.4 mW |
| MemTest_IP | 48 | 98 | 4 | $5.974 \mathrm{MT} / \mathrm{s}$ | 1.57 mW | 2198.2 mW | 2206.8 mW |



Figure 42: Alternative Memory System Implementation

### 6.4.3 Low-Power Coding on Address Buses

In case addresses are incremented across large ranges the numeric coding can be important. The Gray code toggles only one bit from any number to its successor. In our case, for example, the weightfactor memory is read out sequentially over and over again. Thus, addressing the weightfactor memory using a Gray code counter might reduce the toggle rate. Without knowledge of the internal structure of the memory blocks, however, gate-level simulations should be carried out in order to quantify the effects.
The Gray code can be obtained by using a standard binary counter and an XOR-stage directly behind it. For an 11-bit counter, the operation is as follows:
gray_code [10:0] = bincode[10:0] xor 0_bincode [10:1]
In case the additional latency can be tolerated this logic can be placed in an extra pipeline stage. Then the maximum clock frequency is most likely not affected.
A test design that includes the weightfactor memory as described in section 4.6 with the modifications outlined in section 7 has been made (WFMemTest). It is parametrizable and allows binary and Gray code address counters to be used. PowerPlay was run over all design variants as described in section 6.3. In order to reduce simulation time, only one memory bank was included. The results are summarized in Table XI.

Table XI: Weightfactor Memory Bank Power Consumption

| Address <br> Format | Memory Generation <br> Method | M20K Blocks <br> used | Average Toggle <br> Rate | Core Dynamic <br> Power Consumption | Device Static <br> Power Consumption | Total <br> Power Consumption |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Binary | Instantiation | 64 | $29.693 \mathrm{M} / \mathrm{s}$ | 240.36 mW | 2212.98 mW | 2453.96 mW |
| Gray Code | Instantiation | 64 | $29.043 \mathrm{M} / \mathrm{s}$ | 238.60 mW | 2213.08 mW | 2452.31 mW |

As can be seen, the effects are not significant. Dynamic power consumption is dominated by the memory arrays and data buses. Although the fan-out is relatively large, the narrow address bus does not influence the toggle rate much. The use of Gray code will probably pay off only for very wide buses.
Memory systems design recommendations: From a power dissipation perspective memory systems should be implemented as shown in Figure 42. If applicable Gray coding can be included in the design since hardware expenses are very low.

### 6.4.4 Low-Power Coding on Parallel Single-Endend (off-chip) Buses

Although ultimately not used in the design this method should be discussed for completeness. Two problems exist with wide parallel single-ended buses, especially if they drive external connections: power consumption, and simultaneous switching noise (SSN). Both are caused by signal level transitions. SSN occurs when a large number or all of the signals change in the same direction at the same time. Then, the common ground potential can rise (ground bounce) or the power supply can drop in voltage.
Single-ended signaling has largely been replaced by differential (current-mode) transmission, wide parallel single-ended buses have been replaced by multi-gigabit serial connections. This has all but eliminated the SSN problem. However, some application domains still exist for single-ended buses, such as SDRAM channels or legacy buses such as VME or PCI.

Bus Invert Coding (BIC) addresses both problems at the expenses of one additional signal line [16]. For any two consecutive bit patterns to be placed on a parallel bus, the method counts the number of transitions. If this number is greater than $\mathrm{N} / 2, \mathrm{~N}$ being the bus width, the data item is inverted and the additional signal line is activated. The receiver can easily reconstruct the original data values. This limits the number of simultaneous transitions to $\mathrm{N} / 2+1$. The decrease in average power consumption depends on the bus width and on the statistics of the data, and is specified by the inventors to be in the range of $15 \%$ for 16 -bit buses.

### 6.5 Architectural Changes for Increased Power Efficiency

The distinguishing feature of the presented architecture is the constant stream of weightfactors along with their indices for all frequency channels through the circuitry. This allows any input vector to be assigned to any beamformer core for processing, which in turn allows the most complete usage of hardware resources. However, the price to pay is a very wide bus, and a very wide 8-to-1 MUX within each beamformer core (see Figure 8). This structure has become problematic during routing, causing the tools to terminate because of routing congestion.
The problem can be alleviated by splitting the beamformer cores in two groups, one for the lower frequency channels $(0-3)$ and one for the upper frequency channels (4-7). However, the total number of beamformer cores must then be divisible by two. In our case we can use 46 out of 47 cores. The performance target can still be reached. As derived in section 2 , the total number of complex multiplications per FPGA per second is $2.62144^{*} 10^{11}$. At a clock frequency of 360 MHz , the performance of the modified design is $2.6496 * 10^{11}$ complex multiplies per second.
The benefit we get is a significantly reduced broadcast structure which poses far less problems during routing, and also consumes less power.
Power savings: 3W.
In theory the splitting into groups can be continued, however, the number of beamformer cores that can be used must then be divisible by 4,8 and so forth. A maximum of 47 beamformer cores is not the most fortunate outcome in this regard. Using only 44 cores out of 47 already misses the performance targets.

### 6.6 CORDIC Arithmetic

When looking at Table II and Table XVIII, any attempt to use CORDIC arithmetic as discussed in section 5.1 is hopeless. Consumption of fabric logic is already far too high when using standard complex multipliers. Nevertheless a test design has been made to more precisely determine actual resource consumption. A stripped-down version of the beamformer system with only 16 beamformer cores or 8 per group was assembled.
For this design we assume that the complex samples arrive as 16 -bit magnitude (positive fixed-point number) along with a 16 -bit phase. The phase has been normalized to a range of $\pm 45^{\circ}$. Two bits are used to define the quadrant of origin. Thus, the angle part is a 14 -bit two's complement fixed-point number.
For simplicity we assume that the weightfactors are of the exact same format. The necessary scaling factor for the correction of the CORDIC pipeline has been pre-multiplied into the weightfactor magnitudes. Magnitudes are multplied by a $16 \times 16$ into 14 bits multiplier hardmacro. Angles are added into a 15 -bit two's complement number spanning the range of $\pm 90^{\circ}$.
Using Altera QSys, a CORDIC unit has been generated that outputs 24 -bit real and imaginary components. This is compatible with the standard version (see section 7.1). A target clock frequency of 360 MHz was specified which resulted in 27 pipeline stages. The unit consumes 4292 LUTs.
A two-stage de-normalization unit that uses the original quadrant bits transforms the vector into the final position.
Note that the accuracy of the CORDIC-approach is below that of the standard version. The width of the operands has been chosen such that they are compatible with the QSys parameters. The results are listed in Table XII (the Q15 fitter terminated prematurely).

## Table XII: Test Design Using CORDIC Arithmetic

| Design | ALMs | Registers | M20K Blocks <br> used | DSP Blocks <br> used | Core Dynamic <br> Power Consumption | Device Static <br> Power Consumption | Total <br> Power Consumption |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| CORDIC | 905,485 <br> $(212 \%$ of Chip $)$ | $1,083,748$ | No Data | 256 | - | - | - |

The already stripped-down version by far exceeds the available resources. No attempt was made to compile even smaller beamformer systems.
In order to obtain at least a vague idea about power consumption two test designs have been made that include just a single CXM-unit (see Figure 10). CXMTest_Standard uses complex multipliers as usual, CXMTest_CORDIC performs the multiplication in polar coordinates and uses the COR-DIC-pipeline as described above to transform the results back to Cartesian coordinates. The tool chain was run for the two designs as described in section 6.3. The results are listed in Table XIII.

Table XIII: Standard vs CORDIC Arithmetic

| Design | ALMs | Registers | M20K Blocks <br> used | DSP Blocks <br> used | Core Dynamic <br> Power Consumption | Device Static <br> Power Consumption | Total <br> Power Consumption |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Standard | 101 | 321 | 1 | 2 | 41.8 mW | 2200.5 mW | 2245.2 mW |
| CORDIC | 1,953 | 3,459 | 1 | $1(0.5)$ | 311.5 mW | 2220.6 mW | 2535.1 mW |

This puts the final nail in the CORDIC coffin. However, this outcome must clearly be attributed to the dismal adder performance on FPGAs, both regarding performance and area. For full-custom or gate-array designs this can be fundamentally different.

### 6.7 Compiler Options for Increased Power Efficiency

The Q15 tools offer 6 general optimization modes:

- Balanced (Normal flow)
- Performance (High effort - increases runtime)
- Performance (Aggressive - increases runtime and area)
- Power (High effort - increases runtime)
- Power (Aggressive - increases runtime, reduces performance)
- Area (Aggressive - reduces performance)

Normally the tool chain has been used with "aggressive performance" optimization to meet the clock target. The results are presented in section 7.4. For this test run the "high effort power" optimization method has been chosen, and numerous other compiler switches have been set for maximum power optimization. The results are shown in Table XIV, together with results from the performance-optimized design (cf. Table XX).

## Table XIV: Performance vs. Power Optimization

| Optimization <br> Target | ALMs | Registers | M2OK Blocks <br> used | DSP Blocks <br> used | Core Dynamic <br> Power Consumption | Device Static <br> Power Consumption | Total <br> Power Consumption |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| Performance | 250,546 | 941,022 | 1,678 | 1,472 | 22.876 W | 7.921 W | 33.924 W |
| Power | 251,363 | 940,951 | 1,678 | 1,472 | 22.854 W | 6.010 W | 31.992 W |

Thus, the power savings come mostly from a reduction of static power consumption and amount to $1.932 \mathrm{~W}(5.7 \%)$. However, we can observe a significant drop in clock frequency, which is shown in Table XV (cf. Table XIX).

Table XV: Timing Analysis for Power-Optimized Design

| Variant | Setup-Slack [ns] | TNS [ns] | Maximum System <br> Clock [MHz] |
| :---: | :---: | :---: | :---: |
| Slow $100^{\circ} \mathrm{C}$ | -4.155 | $-4,496.6$ | 144 |
| Slow $0^{\circ} \mathrm{C}$ | -4.723 | $-10,079.5$ | 133 |
| Fast $100^{\circ} \mathrm{C}$ | -2.418 | -151.1 | 192 |
| Fast $0^{\circ} \mathrm{C}$ | -1.369 | -59.7 | 241 |

As can be seen, the target clock frequency is missed by a wide margin. Considering the relatively small power savings this method appears to be of little use, at least for this design.

### 6.8 Scaling Performance vs. Power Consumption

In harsh environments such as around desert stations it might become necessary at times to reduce the heat dissipation because of cooling problems. Still it might be desirable to conduct observations in these situations. Performance must then be reduced, for example by using only a subset of antennas, or by processing only a subset of frequency channels. Not always might the lucky case occur that entire Uniboards can be powered down. We examine the question: is it better to reduce clock frequency, or is it better to reduce the active chip area? Several test designs have been made, and run through PowerPlay with a default toggle rate of $12.5 \%$. The results are shown in Table XVI.

## Table XVI: Scaling Frequency vs. Area

| Design | ALMs | Registers | M20K <br> Blocks <br> used | DSP <br> Blocks <br> used | Core Dynamic <br> Power <br> Consumption | Device Static <br> Power <br> Consumption | TxF Dynamic <br> Power <br> Consumption | TxF Standby <br> Power <br> Consumption | Total <br> Power <br> Consumption |
| :---: | ---: | ---: | ---: | ---: | ---: | ---: | ---: | ---: | ---: |
| $188 \mathrm{MHz}, 46$ Cores | 238,589 | 893,033 | 2,372 | 1,472 | 13.188 W | 4.290 W | 2.241 W | 0.885 W | 20.605 W |
| $94 \mathrm{MHz}, 46$ Cores | 238,747 | 892,909 | 2,372 | 1,472 | 7.294 W | 3.470 W | 2.241 W | 0.885 W | 13.891 W |
| $360 \mathrm{MHz}, 24$ Cores | 153,293 | 524,232 | 1,036 | 768 | 12.763 W | 4.116 W | 2.241 W | 0.885 W | 20.007 W |
| $360 \mathrm{MHz}, 16$ Cores | 103,752 | 351,412 | 876 | 512 | 9.166 W | 3.571 W | 2.241 W | 0.885 W | 15.864 W |
| $360 \mathrm{MHz}, 8$ Cores | 55,066 | 181,906 | 716 | 256 | 4.998 W | 3.089 W | 2.241 W | 0.885 W | 11.214 W |

The results are to a certain degree inconclusive, which might be due to the inaccurate power estimation when using PowerPlay with default toggle rates. One could have expected that when using less cores, device static power consumption should decrease more strongly because more unused tiles can be powered down. The recommendation leans towards scaling the clock frequency, which has the advantage that it can be done without FPGA reconfiguration.

## 7 IMPLEMENTATION

The predominant goal of the implementation is to meet the performance targets. This requires the design to be implemented such that the tools (synthesizer, mapper, router etc.) can

- fit all necessary logical elements on the chip,
- route all connections, and
- meet all target clock frequencies.

This must be achieved with reasonable margin to avoid tool failure after (minor) modifications.
To a second degree power consumption must be taken into account. The methods discussed in section 6 have been consequently applied wherever they would not reduce performance, or pose serious problems to the tools.

### 7.1 Implementation Restrictions

For a high performance FPGA design, one has to realize that signals typically spend much more time on the wire or interconnect structure than through logical elements. This has two implications:

- whenever the algorithm can tolerate it, deep pipelining should be used.
- The fan-out of signals should be kept small, ideally it should not be larger than 2 .

Together with the architectural changes discussed in section 6.5 this results in a number of modifications to the general architecture as presented throughout section 4.
Two further concessions to the limited hardware resources and/or tool capabilities had to be made:

- the width of the weightfactors had to be reduced from 19 bits to 17 bits per component, otherwise the IP generator consumed 4 DSP blocks per complex multiply. Thus, the outputs of each weightfactor memory bank are 555 bits wide instead of 619 bits.
- The operand width at the input of the adder tree had to be reduced from 33 to 24 bits to achieve both routing and timing closure.

Lastly it should be noted that no infrastructure for loading the weightfactor memories is present. This is most reasonably done when the surrounding environment and all interfaces are known. Currently these memories are implemented as ROMs, and the contents are specified via RAM-Init-files.
It has proven difficult to meet timing requirements for all four timing models (slow silicon @ $100^{\circ} \mathrm{C} / 0^{\circ} \mathrm{C}$, fast silicon @ $100^{\circ} \mathrm{C} / 0^{\circ} \mathrm{C}$ ). However, this can at least partially be attributed to the fact that the chips are still engineering samples, and that timing characteristics are preliminary. This view is supported by the large deviations in performance of the different models.

Thus, attempts to increase performance by architectural means (adding pipeline stages etc.) have been stopped when the "fast silicon" was within limits (with respect to setup-times). Final work on meeting timing is most reasonably postponed to the point in time when actual hardware is available, or when timing characteristics have settled.

### 7.2 Actual Architecture

Most importantly, the beamformer cores are arranged in two groups for frequency channels 0-3 and 4-7. Consequently there are two separate input and weightfactor fan-out register trees. The input FIFOs are 64 bits wide to service the two fan-out trees in parallel, in return however, they can operate on half the clock frequency. Finally the input data format must be changed so that for each antenna a sample from channel $n$ is followed by the sample from channel $n+4$. In essence two beamformer systems as described in section 4, but half the size, have been placed on one FPGA, as shown in Figure 43.


Figure 43: Actual Architecture

The modified input stage is shown in Figure 44. Note that this modification does not increase hardware consumption, instead it reduces routing efforts by means of the reduced clock rate. The new data format should not pose any problems to the filterbank FPGAs.


Figure 44: Modified Input Stage

The input fan-out register tree, as opposed to Figure 6, is a full binary tree to better meet timing. It has 32 leaves (ports). Unused logic due to unconnected ports will be removed by the synthesizer. The scheduler reads all input FIFOs, hands out VALID-flags, keeps track of the channel number and generates a 5-bit destination that represents a round-robin assignment for the 23 beamformer cores. For the operational principle see Figure 45 .


Figure 45: Input FIFO Read Ports, Scheduler, and Fan-Out Tree (One of Two)

For the weightfactor fan-out structure, instead of a tree a pipeline was chosen since it produced less problems during routing. In order to bridge the distances on the chip without flight times getting too long three stages per tap were chosen. To account for the geometric arrangement of memory blocks on the chip, the weightfactors are inserted in a staggered way. This is shown in Figure 46 . Note that two such structures are present, and so the flipflopcount is roughly 302,000 .


Figure 46: Weightfactor Fan-Out Pipeline

### 7.3 VHDL and QSys Source Files

The design includes two kinds of sources: VHDL-files, and IP-blocks generated using Altera QSys. The hierarchy of the source files is shown in Table XVII. QSys-files are shown in red. Note that modules synthesized by the tool chain during compilation are not included.

Table XVII: Source File Hierarchy

| Top Level | Level 1 | Level 2 | Level 3 | Level 4 | Level 5 | See |
| :---: | :---: | :---: | :---: | :---: | :---: | :---: |
| beamformer |  |  |  |  |  | Figure 43 |
|  | input_stage |  |  |  |  | Figure 44 |
|  |  | input_ethchan |  |  |  | Figure 44 |
|  |  |  | input_fifo |  |  | Figure 44 |
|  |  |  | input_phy |  |  | Figure 44 |
|  |  |  | input_txfttrl |  |  | - |
|  |  | input_phyres |  |  |  | - |
|  | csfan |  |  |  |  | Figure 45 |
|  | scheduler |  |  |  |  | Figure 45 |
|  | bf_unit |  |  |  |  | Figure 43 |
|  |  | bf_core |  |  |  | Figure 7 |
|  |  |  | wimux |  |  | Figure 8 |
|  |  |  |  | channel_fifo |  | Figure 8 |
|  |  |  |  | wimux_ctrl |  | Figure 9 |
|  |  |  |  | mux_wi_2_1 |  | - |
|  |  |  | cxm_stage |  |  | Figure 11 |
|  |  |  |  | cxm_unit |  | Figure 10 |
|  |  |  |  |  | cxm | Figure 10 |
|  |  |  | adder_tree |  |  | Figure 12 |
|  |  |  |  | par_add |  | Figure 12 |
|  |  |  | acc_stage |  |  | Figure 14 |
|  | wffan |  |  |  |  | Figure 46 |
|  | wfmem |  |  |  |  | Figure 15 |
|  |  | wfmem_bank |  |  |  | Figure 15 |
|  |  |  | wfmem_slice |  |  | Figure 42 |
|  | output_stage |  |  |  |  | Figure 20 |
|  |  | output_unit |  |  |  | Figure 16 |
|  |  |  | output_ctrl |  |  | Figure 17 |
|  |  |  | fxp_to_sp |  |  | Figure 16 |
|  |  |  | mux_80_4_1 |  |  | - |
|  |  |  | mux_80_2_1 |  |  | - |
|  |  | output_ethint |  |  |  | Figure 18 |
|  |  |  | output_ethchan |  |  | Figure 18 |
|  |  |  |  | output_fifo |  | Figure 18 |
|  |  |  |  | output_phy |  | Figure 18 |
|  |  |  |  | output_txfctrl |  | Figure 19 |
|  |  |  | output_atxpll |  |  | Figure 18 |
|  |  |  | output_phyres |  |  | Figure 18 |

The design constraints (mostly clock definitions) are in beamformer.sdc, the package file $b f$ defs.vhd includes some constant definitions.

### 7.4 Results and Performance

The majority of compiler options have been set for maximum performance, at the expense of runtime and chip area. The consumed chip resources are listed in Table XVIII. Note the almost complete usage of DSP blocks.

Table XVIII: Occupied Chip Resources

| Type | Total \# on Chip | Used | $\%$ |
| :---: | :---: | :---: | :---: |
| ALM | 427,200 | 250,546 | 59 |
| Register | $1,708,800$ | 941,022 | 55 |
| RAM Block | 2,713 | 1,678 | 62 |
| DSP Block | 1,518 | 1,472 | 97 |
| HSSI Rx Channel | 96 | 16 | 17 |
| HSSI Tx Channel | 96 | 6 | 6 |
| PLL | 176 | 17 | 10 |
| Pin | 928 | 55 | 6 |

The multi-corner timing analysis gave the results as shown in Table XIX. TNS is the sum of all occuring negative setup slack within a given clock domain. One can use this value to gauge the amount of design effort that will be necessary to meet timing. The reported TNS values in this case can be considered very small. Note also that the "slow $100^{\circ} \mathrm{C}$ "-model is slightly faster than the "slow $0^{\circ} \mathrm{C}$ "-model, which is unexpected and potentially an indication that timing characteristics will change.

Table XIX: Timing Analysis

| Variant | Setup-Slack [ns] | TNS [ns] | Maximum System <br> Clock [MHz] | Maximum Input <br> Clock [MHz] |
| :---: | :---: | :---: | :---: | :---: |
| Slow $100^{\circ} \mathrm{C}$ | -0.628 | -227.9 | 294 | 156.25 |
| Slow $0^{\circ} \mathrm{C}$ | -0.659 | -227.7 | 291 | 156.25 |
| Fast $100^{\circ} \mathrm{C}$ | 0.3 | - | 360 | 156.25 |
| Fast $0^{\circ} \mathrm{C}$ | 0.7 | - | 360 | 156.25 |

PowerPlay was run using a default toggle rate of $12.5 \%$. The results are shown in Table XX. Thus, the 48 beamformer FPGAs in the complete system have a power consumption of $1,628 \mathrm{~W}$.

TABLE XX: PowerPlay Results

| Average Toggle <br> Rate | Core Dynamic <br> Power Consumption | Device Static <br> Power Consumption | TxF Dynamic <br> Power <br> Consumption | TxF Standby <br> Power <br> Consumption | Power Consumption |
| :---: | :---: | :---: | :---: | :---: | :---: |
| $47.312 \mathrm{MT} / \mathrm{s}$ | 22.876 W | 7.921 W | 2.241 W | 0.885 W | 33.924 W |

## 8 SUMMARY

We have presented a beamformer design for the Uniboard ${ }^{2}$, in the first instance optimized for Arria 10 FPGAs. The distinguishing feature of the architecture is a constant stream of weightfactors along with their indices through the circuitry. This allows operation at a very high efficiency. The complete system contains 16 Uniboards including a filterbank. According to first implementation results the system should be able to generate 64 beams from 512 antennas over an observing bandwidth of 384 MHz . Several methods for scaling the performance have been presented. Green measures for reducing the power consumption have been included in the design.

