Semantic programming model-based design
Defining a hierarchical tiled multi-processor architecture

Kenneth C. Rovers, Jan Kuper and Gerard J.M. Smit

Computer Architecture for Embedded Systems group
CTIT, Department of EEMCS, University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands
Email: K.C.Rovers@utwente.nl
http://caes.cs.utwente.nl/Research/?project=BeamForce

Abstract—For a generic flexible efficient array antenna receiver platform a hierarchical tiled architecture has been proposed, giving a heterogeneous multi-processor system-on-chip (MPSoC), multiple chips on a board (MCoB) and multiple boards in a system (MBiS). A wide range of MPSoCs are predicted to be used in the near future but how to efficiently apply these designs remains an issue. We will advocate a model-based design approach and propose a single semantic (programming) model for representing the specification, design and implementation and allowing for verification, simulation, architecture definition and design space exploration.

A single model for specification, (formal or functional) verification, simulation and programming an MPSoC has obvious as well as some less obvious advantages. It allows for model-based design down to the implementation, especially for hierarchical MPSoC architectures. Partitioning and mapping of the functionality to an architecture is commonly done manually. Using the proposed approach the feasibility of (partly) automated design space exploration is discussed for determining either a partitioning and mapping for a given architecture or an optimal architecture based on set constraints.

The proposed hierarchical tiled architecture provides a flexible reconfigurable solution, however partitioning, mapping, modeling and programming such systems remains an issue. The proposed approach tackles these problems at a higher conceptual level, thereby exploiting the inherent composability and parallelism available in the formalism. Design space explorations is facilitated by allowing transformations between different partitionings and mappings. However, the generic applicability and limitations of this approach will need to be researched further.

Index Terms—Phased array, beamforming, hierarchical tiled architecture, MPSoC, semantic programming model, model-based design

I. INTRODUCTION

System design is greatly aided by the use of models. The models provide an abstraction at different levels of detail or functionality. They can also complement each other by providing different views of the system. In hardware, model-based design uses building blocks to define functional characteristics of the system at various degrees of sophistication, allowing simulation, testing and verification of systems. In software, this approach is called the model-driven architecture approach. In order to decouple the system design from an architecture, a high level model can be architecture independent and a model transformation can be applied to create an architecture dependent model.

A wide range of MPSoCs are predicted to be used in the near future [1] but how to efficiently apply these designs remains an issue. We will advocate a model-based design approach and propose a single semantic (programming) model for representing the specification, design and implementation (including programming the design) and allowing for verification, simulation, architecture definition and design space exploration.

After an introduction to the application domain and the used platform for our case study, the commonly used design approach for such systems is presented. The benefits of a model-based design are clear, but there are some draw-backs of the current approach. A new "semantic (programming) model" is proposed to help eliminating these draw-backs and develop this to allow design space exploration, architecture definitions and simulations with the same "semantic programming model".

A. Application domain

To illustrate the model-based design approach, we use a phased array receiver platform as an example of a high performance digital signal processing application. The design of these systems is mainly driven by functional requirements (e.g., resolution, sensitivity, response time) where non-functional requirements (e.g., costs, power consumption) are of secondary concern [2]. For that reason, no low-cost, low-power phased array systems are available yet. However, in areas like radio astronomy and for satellite receivers, phased array antennas show great promise but their large scale introduction has been obstructed by the high costs involved (for production). The goal is thus to develop a low-cost, low-power phased array receiver system.

The system blocks of a basic phased array system are shown in figure 1. In a phased array receiver, signals are received at multiple antennas with different time delays (or phase shifts) because of path length differences. After the RF (radio frequency) front end for each antenna, antenna processing (AP) may be applied for calibration or equalization purposes (to correct for electrical or mechanical distortions of the front-end). The signals are then combined by the beamforming processing (beamformer) to create a resulting signal with for example a maximum sensitivity in a direction of interest or a minimum sensitivity (a null) in the direction of an interfering
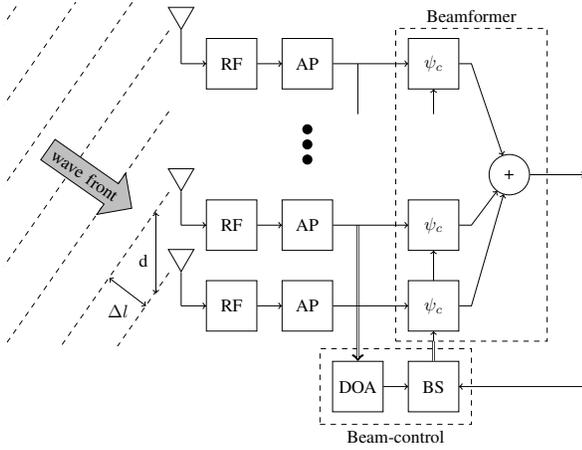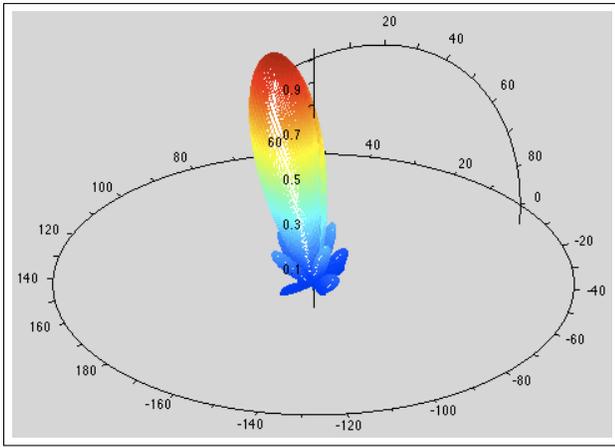
Fig. 1. Phased array receiver

Fig. 2. Phased array angular sensitivity

Fig. 3. Phased array angular sensitivity

signal. Beamsteering (BS) refers to changing the shape and direction of the formed beam by changing the gain and delay of the antenna signals to create a certain angular sensitivity or radiation pattern as shown in figure 2. Note that multiple beams in different directions can be formed by duplicating the antennas signals and apply the beamforming for each beam with different correction parameters.

To calculate the parameters, the beamsteerer needs to know in which angle (direction) to point the beam. This information is provided by the beam control process. It can be set by the user, based on an algorithm (for example for tracking a source) or based on an estimation of the angles of the strongest sources available. This latter estimation is provided by the direction of arrival (DOA) estimation process.

B. Reconfigurable tiled architectures

Phased array processing can be characterized as a streaming application with high data rates and processing requirements, but a regular processing structure. Because of costs, complexity, dependability and scalability reasons a design with mostly identical components is preferred, but because of func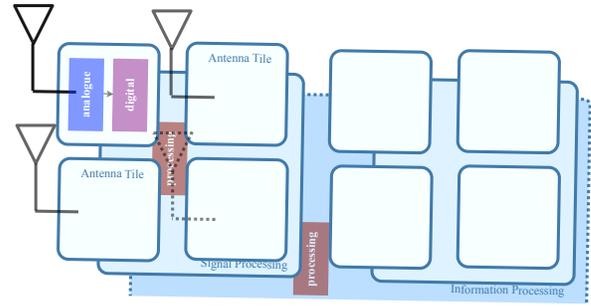tionality with different requirements and use it will be heterogeneous. We would like to limit the data rate as soon as possible through beamforming, because I/O is expensive. This implies that the processing is moved closer to the antennas. However, combined data cannot be separated later on, so we lose flexibility. Furthermore, the distributed processing must be synchronised. Because a scalable and dependable solution is needed, a tiled architecture is proposed with reconfigurable hardware to regain flexibility. Processing tiles are combined on multiple hierarchical levels. A multi-processor system-on-chip (MPSoC) can be extended to multiple chips on a board (MCoB) and multiple boards in a system (MBiS) giving a heterogeneous hierarchical tiled architecture (as in figure 3). We aim at a processing architecture which is flexible enough to support multiple methods of beamforming, as well as beamsteering and beam-control. [3]

A reconfigurable hierarchical processing array can provide flexibility and has a number of advantages. We can use only part of the array or create multiple sub-arrays to save energy or increase the lifetime. Reconfigurability (also in I/O routing) supports graceful degradation if tiles break down. Reconfigurability inherently leads to having an adaptable system, that adapts to changing environments while maintaining the quality of service.

II. MODEL BASED/DRIVEN DESIGN

Traditional design has followed the waterfall model, which has the disadvantage that the next design phase has to wait for the previous to finish and therefore making changes late in the design cycle very costly. Incremental and iterative design addresses these short comings by integrating part of the design as soon as possible instead of using one large integration step at the end of the design and extending the design with small steps going through the design cycle instead of developing the complete design at once.

The design steps (or cycle) consist of setting goals, doing research and doing development (i.e. why, what and how) followed by an evaluation. The requirements support the goals with more details which can be verified. During the research phase the goals and requirements are analysed and different options to satisfy the requirements are explored (diverge). This results in a formal specification of the selected solution (converge). The development phase uses this formal specification to synthesize a system in several steps. During evaluation

the synthesized system is verified, by tests or simulation to conform to the requirements and specifications. Validation can be used to confirm the goals are met. Depending on the results the design can be refined, starting the next design cycle.

A. Common design approach

A typical design approach uses systems engineering [4] with the analysis (goals and research), synthesis (development and implementation) and evaluation (verification and validation) steps. Systems engineering is a design approach which aims at using a holistic view with a *life-cycle* orientation that addresses all phases of the system design. Throughout the design it attempts to unify all involved contributors into an *interdisciplinary* effort. It uses well defined and specified *system requirements* which can be verified and validated down to a detailed implementation. Designing is performed using a *top-down* approach to decompose a larger system into smaller blocks, to make sure these blocks effectively fit together and to manage complexity.

Setting the design goals and defining the requirements is supported by using diagrams of a modeling language such as SysML or UML. A mathematical model can be used to define system inputs and outputs. During analysis the design is decomposed into smaller blocks. The diagrams can be extended by supplying more detail. When a complete mathematical model of the blocks is given, one provides a formal specification which can be checked for correctness. Often this model is platform (hardware/software) independent. Dataflow models provide a graphical representation of the flow of data through the processes of a system. The data forms the input and output of the mathematical models, while the processes are an abstraction of the equations. In the development phase a simulation model can be used to provide a functional specification for all the (sub-) blocks that are not (yet) implemented. This facilitates interdisciplinary work, such as when performing hardware/software co-design, at which point the hardware to use is not defined yet or when different options are evaluated. The implementation itself can consist of for example block schematics, hardware or software. The simulation model can be used during the design to evaluate design decisions and implementations by testing.

For the modeling language SysML or UML is commonly used. For the mathematical model a tool such as MATLAB can be used and together with Simulink can provide a simulation model. The leading formal specification language for software systems on the other hand is Z. The implementation is very application dependent, but in this paper we consider languages such as SystemC or VHDL for hardware and C for software.

B. Hardware/Software co-design

Processing systems often have a trade-off between what to do in hardware and what in software. Hardware refers to specific functionality with limited flexibility, but high efficiency (area, power, performance, cost), while software refers to some kind of processor which can be programmed and is therefore much more flexible. Of course this programmability

comes at a cost of efficiency. Hardware/software co-design refers to defining an architecture, and mapping functionality to hardware and software for this architecture.

During the analysis, a mathematical or dataflow model is defined to indicate the decomposition and intended functionality to achieve the set goals. An architecture is defined to implement this functionality by performing hardware/software co-design, balancing the trade-off between flexibility and efficiency. During synthesis the mathematical or dataflow model is mapped to the architecture, i.e. functionality is assigned to specific hardware and the assigned blocks are connected together. Thus a Y-chart approach of combining an architecture definition and a dataflow model is used to achieve an implementation by specifying a mapping between the two. Nowadays it is no exception to have a number of processors as well as specific hardware in an (heterogeneous) MPSoC as an architecture. The resulting implementation often has the hardware specified by a language such as SystemC or VHDL and the software in C.

C. Evaluation

Apparent from the previous section is that different tools and languages are used for the specification, mathematical model, simulation model and implementation, although of course they overlap to some degree. One of the larger problems with this is that a specific implementation (in for example C and for an ARM processor) can not easily be integrated into the simulation model. And although there are options to execute UML or to generate code from UML, this is mostly focussed on software and the language itself is not proven to be Turing complete.

Simulink is an environment for multi-domain simulation and Model-Based Design for dynamic and embedded systems. It can work with hierarchical models. It allows for code generation in C or VHDL and for hand-written algorithms in MATLAB, C, ADA or Fortran. MATLAB and Simulink use numerical algorithms to compute the dynamic behaviour. One problem with this approach is that for each block a sample-rate (simulation rate) is determined and the equations are evaluated for each sample time. Although Simulink supports multi-rate models, this is problematic in case of very different sample rates, such as for example down-conversion in a RF front-end. The lower sample rate blocks need to be evaluated with a much higher sample rate than otherwise needed, making the simulation slow. Another problem are variable time delays, for example when modeling physical systems. Note that MATLAB can be used to circumvent these problems, but then the Simulink model becomes less intuitive. Finally, although support exists for handwritten algorithms and embedded systems integration, it is harder to integrate code in for example newly designed hardware, such as an exotic processor with a special instruction set.

Hardware/software co-design is performed by mapping functionality to an architecture. This architecture can be a given, however as said, there are trade-offs involved in defining the architecture, depending on the functionality. We can define

the architecture by partitioning or clustering the functionality and determining the required performance figures such as throughput, latency and "cost", thereby providing the architecture and mapping. The result of which can be implemented. Of course we would also like to have the architecture flexible enough to support different functionalities by reconfiguring, otherwise a complete hardware implementation would always be best as the flexibility is unused. The factors involved in defining the architecture make it likely that different alternatives need to be evaluated. This definition of the architecture and/or the mapping is normally done by hand. Furthermore, the tools lack support for some kind of automated design space exploration. This means each new architecture requires a new or adapted model or design space exploration must be explicitly modelled in.

A major problem for multi-core and multi-processor systems, is how to program them efficiently. Of course, if each task can be assigned to a processor, the problem reduces to how to connect them efficiently. However, what if we want to distribute some functionality over different processors? Often the parallelisability of an algorithm is limited by the implementation. This is because imperative programming languages such as C are inherently sequential and it is difficult to determine that a used variable (memory location) does not change when being used somewhere else. The memory is the state of the algorithm and each statement changes this state. As the algorithm can or can not depend on this variable change, it correct operation when run in parallel can not be guaranteed. Therefore, the amount of parallelism that can be extracted is limited to about four [5].

D. Semantic (programming) model

We believe a single model can be used to provide the formal and functional specification of a design, as well as allowing one to develop this model into an implementation. A model-based design approach can then be used with a single model for specification, verification, simulation and implementation. We dubbed this a "semantic model" as the model itself can be the specification, an abstraction as well as an implementation, all with the same intended meaning. It is also a programming model as it can be used as a programming language to program a single processor, an MPSoC, a custom architecture and even hardware.

This believe is based on the one hand that a mathematical model can describe the system, while on the other hand the notion of the code is the design. MATLAB and Simulink are languages for mathematical computation, analysis and modeling. However, this is not extended down to being a full-featured programming language. Imperative programming languages on the other hand are not very suitable as a mathematical model, because it provides a sequence of statements, while a mathematical model describes equation, which are a set of relations. A variable in C is not the same thing as a variable in an equation. There are programming languages that describe a set of functions instead of a sequence of statements. These are called functional languages. The notion that the

code also provides a specification of a design can be better defended as the mathematical model can directly be described in a functional language giving the specification of the design, but being a programming language it can also be used as the implementation of (parts of) the design.

The functions of a functional program can model the component or (sub-) blocks of a design. By using abstract data types (ADT), we can provide all kinds of meta-data about the data such its representation, requirements, constraints etc. By using higher order functions, functions themselves can be used as arguments of other functions. Functions can also be used in an ADT, allowing one to annotate functions or components of a model with meta-data. A graphical representation can be provided easily as well as a tool to create models. A block thus represents a function, a connection an argument or result of a function, a chain or pipeline of blocks represents a function composition or the other way around for example. Higher order functions also allow one to explicitly model time as a parameter instead of implicit time modeling in a tool such as Simulink.

An advantage of a text-based representation besides a graphical model, is that is easy to define transformations to perform on the model. By using ADTs and higher order functions, the performance of the models can be calculated before and after the transformation with the use of an appropriate cost function. This supports relatively easy design space exploration, possibly (partly) automated. A mathematical model can thus be transformed to an equivalent, which represents a mapping to an architecture or the transformation maps the model to a predefined architecture. A dataflow model can be represented by a graph defined by ADTs. This graph can be partitioned or clustered with a transformation to kernels represented by functions. These kernels can have a number of implementations, for example a hardware or software implementation, all within the functional language and possibly annotated with performance figures. A mapping function can determine the optimal solution for a given architecture or the optimal architecture for the given performance figures according to some cost function.

As a programming language, a functional language can be used to program parallel or distributed systems. The inherent parallelism in the formalism is retained because of the use of higher order functions. Another important requisite is referential transparency or the use of immutable variables. This ensures a function has no side-effects that effect the program somewhere else, eliminating data dependency between functions besides the arguments and automatically allowing them to execute in parallel without problems such as deadlock or race conditions (i.e. thread-safe). This in turn makes a functional language very suitable for programming MPSoCs or distributed systems without the need for a communication model or middleware layer.

III. Case study

The design of a cheap generic flexible efficient array receiver platform is used as a case study. After the design goals,

a mathematical model of a phased array beamforming system is presented followed by a functional model in Simulink and a semantical model.

A. Goals

As said a generic solution allow the production volume to go up, therefore the price go down and making the platform a viable option for satellite receivers or radio astronomy. However, this implies that the platform must be flexible and scalable to support the different requirements of the applications. To reduce complexity largely identical components are preferred, but because of functionality with different requirements and use it will be heterogeneous. Thus, we propose a tiled architecture with a number of identical tiles which are reconfigurable to perform different functionalities. This allows the platform to be flexible by reconfiguration, but reduces complexity by using identical tiles. It also makes the platform scalable and dependable, which we would like to extend to multiple hierarchical levels.

The platform must support antenna processing, beamforming, beam-steering and beam-control. We will start with a basic system providing a simulation framework and beam-forming. Future work will extend this model.

B. Mathematical model

Phased array systems are based on the principle of interference using multiple antennas in an array to make a transceiver directional (figure 1). Based on the radar equation [6], the resulting signal after beam-forming can be represented by the source signal $S(t)$, an element factor depending on the sensitivity or gain of each antenna element S_e, an array factor depending on the element positions S_a, a correction (steering) factor S_c and a combining sum.

$$S = \sum a \cdot e^{j(\omega t \pm \psi_e \pm kl \pm \psi_c)}$$
$$= \sum S(t) \cdot S_e(\theta, \varphi) \cdot S_a(l) \cdot S_c(\theta_0, \varphi_0) \quad (1)$$

$$\Delta l = \vec{r} \cdot \vec{R} = dx \cdot u + dy \cdot v + dz \cdot w \quad (2)$$

$$\psi_c(\theta, \varphi) = k \cdot (-\Delta l(\theta_0, \varphi_0)) = \omega \cdot (-\Delta t(\theta_0, \varphi_0)) \quad (3)$$

with ψ the phase, $\vec{r}$ the element position, $\vec{R}$ the plane wave direction, u, v, w the direction cosines and $-\Delta t(\theta_0, \varphi_0)$ the time delay correction [2], [6], [7].

C. Functional model

The radar equation itself is based on a model of the system as a source, a transmitter, a channel and a receiver followed by a beamformer. This results a functional model implemented in Simulink and shown in figure 4. Note that for a phased array system, we have multiple receivers, each having its own channel from the source with a different path length. A lot of thought goes into the correct data structure to use for the data between each block, which is not directly evident from the model.

The results of simulation are as expected. However, for the time delay caused by different path lengths between the source
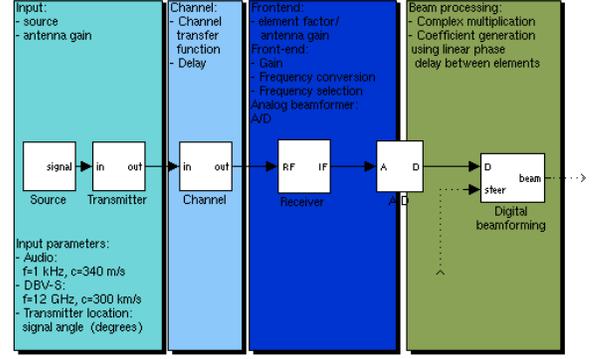

Fig. 4. Simulink phased array functional model

```
system :: (sig) -> (num)
system = beamformer . frontend doa elements
frontend :: (doa) -> [(pos)] -> (sig) -> [(num)]
frontend d ps = fmap (map (chain d) ps)
chain :: (doa) -> (pos) -> (sig) -> (num)
chain d p s = (adc . receiver d p . channel d p .
    transmitter d p) s

input t = S sin f a g t
simulation = map system (map input ts)
```

Listing 1. Phased array semantic model

and the antenna, a variable time delay block is used. This block buffers values for each simulation time step until the delay. If the delay is not exactly at a simulation step, the value is interpolated linearly between two point, thus resulting in inaccuracies.

D. Semantic model

The semantic model consists of a functional program, which models the same blocks connected by function composition and allowing composability by calling functions within functions. Equation 1 can be implemented straight forward (see listing 1), with a single source going to a separate `transmitter`, `channel` and `receiver` chain for each element. A direction of arrival (`d::doa`), a position (`p::pos`) and the source signal are arguments. Each position in the list is mapped to a separate `chain`. Together they form the `frontend`, which is input to the `beamformer`.

The listing can be run, thereby performing a simulation with results as expexted. Note that a function for the signal source is passed from block to block by the semantic model, until it is explicitly evaluated by the `ADC` block to a value at a specific time (specified by the list of time values `ts`). The variable time delay is thus just a change of time argument `t` of the source signal and is therefore exact.

IV. CONCLUSION

In this paper we have shown that a functional (programming) language is a suitable alternative to a (simulation) modeling language as well as being useful as a programming language (model). This "semantic (programming) model", besides being an alternative, adds a number of advantages. It has

the large advantage that it combines the (software) implementation with the simulations model, allowing one to simulate and verify the design and implementation continuously during the design process. Furthermore, because a functional program is effectively a set of equations (functions), the mathematical model can be implemented directly by the language and formal verification can be applied. Because of referential transparency and higher order functions, the evaluation can be delayed until it the value is really needed, for example allowing one to model an "ideal" front-end up to a certain sample time of the ADC at which point the sample value is calculated. It is therefore very easy to mix functional models with implementations and different levels of detail. This in turn supports hardware/software co-design with a functional specification of the selected mapping, which can be verified and simulated. The "semantic model" is text based, although it can be represented and used graphically very easily. Furthermore, the model can be annotated with performance figures and constraints, which can be calculated for the complete design. This and referential transparency allows one to perform transformation on the design, making (partly) automated design space exploration feasible. Furthermore, the transformation can be verified to be correct. This (partly) automated design space exploration on the basis of performance figures and architecture constraints support (partly) automated hardware software partitioning and mapping and architecture definition. Last but not least, because of referential transparency and higher order functions, one functional program can be used as a programming model to program multiple, possibly heterogeneous, parallel processors in a MPSoC and higher hierarchical levels.

References

[1] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, "The landscape of parallel computing research: A view from berkeley," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2006-183, Dec 2006. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html

[2] H. J. Visser, *Array and phased array antenna basics*. Chichester: Wiley, 2005.

[3] K. C. Rovers, M. D. van de Burgwal, A. B. J. Kokkeler, and G. J. M. Smit, "Rationale for and design of a generic tiled hierarchical phased array beamforming architecture," in *18th Annual Workshop on Circuits Systems and Signal Processing (ProRISC), Veldhoven, the Netherlands*. Utrecht: Technology Foundation, Nov 2007, pp. 160–168.

[4] B. S. Blanchard and W. J. Fabrycky, *Systems Engineering and Analysis*, 3rd ed. Upper Saddle River, NJ, USA: Prentice Hall, 1998.

[5] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 4th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2006.

[6] M. I. Skolnik, *Introduction to Radar Systems*, 3rd ed. New York, NY, USA: McGraw-Hill, 2001.

[7] H. L. van Trees, *Optimum array processing*. New York: Wiley-Interscience, 2002, vol. Detection, estimation and modulation theory.