Modeling of Reconfigurable Medical Ultrasonic Applications in BIP

Stefanos Skalistis and Alena Simalatsar

Rigorous System Design (RiSD) Laboratory
École Polytechnique Féréral de Lausanne (EPFL), 1015 Lausanne, Switzerland
{stefanos.skalistis,alena.simalatsar}@epfl.ch

Abstract

Medical ultrasonic imaging applications require high quality of images produced in real-time often with limited resources available. Deadlock-freedom and confluency must be guaranteed to ensure the correctness of the applications, while feasibility and optimality properties are required to provide the best Quality of Service (QoS) within available resources. In this paper we introduce BIP (Behavior-Interaction-Priority) framework components as main building blocks to model such applications in a correct-by-construction manner. Based on those components we model a reconfigurable multi-mode processing pipeline for ultrasonic imaging that supports QoS management by topology reconfiguration. Finally, as a proof of concept, we present a simple quality controller as a well-triggered component, which when combined with the processing pipeline can manipulate the quality of image processing.

1998 ACM Subject Classification D.1.3 Concurrent Programming, D.2.11 Software Architectures, F.1.1 Models of Computation, F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases Reconfigurable Pipelines, Quality of Service, Medical Ultrasonic Applications, Component-based System Design, Behavior-Interaction-Priority Modal Flow Graphs

Digital Object Identifier 10.4230/OASIcs.MCPS.2014.66

1 Introduction

Ultrasonic imaging is widely used in medicine [7] as a diagnostic technique to provide static images (e.g., B-mode) and dynamic changes (e.g., based on Doppler effect). Static imaging provides visualization of muscles and internal organs, to capture their size, structure and any pathological lesions. Ultrasonic imaging based on Doppler effect [6] is widely used to visualize motion, in particular blood flow for diagnosis, such as blood clots, heart valve defects, aneurysms and many others. All these applications require high quality of images produced in real-time. Often ultrasonic devices are used in trauma and first aid cases as well as for remote diagnosis. This drastically limits the available resources for ultrasound computation algorithms, which requires Quality of Service (QoS) management. Moreover, deadlock-freedom and confluency must be guaranteed to ensure correctness of the computational and controlling algorithms.

B-mode ultrasonic imaging, chosen as a case-study in this paper, can be performed in different ways, also called modes or processing pipelines, which may achieve the output image with different quality characteristics and resource requirements. Thus, quality of final images depends first of all on the chosen processing mode. Moreover, the components of a chosen processing pipeline may perform the computation with different quality outcome. Components processing quality levels can be controlled by certain parameters set, e.g. the cutoff quality of a low-pass filter by adjusting the order of the filter. Based on that, we

Medical Cyber Physical Systems – Medical Device Interoperability, Safety, and Security Assurance (MCPS'14).
Editors: Volker Turau, Marta Kwiatkowska, Rahul Mangharam, and Christoph Weyer; pp. 66–79

OpenAccess Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

distinguish two approaches for QoS management driven by i) topology, and ii) components quality levels.

BIP (Behavior-Interaction-Priority) framework [1] provides essential means for rigorous system design. We employ a particular branch of BIP framework, namely well-triggered modal flows [3], which ensures correctness by construction and encompasses a synchronous computation model. Well-triggered modal flows requires no additional coordination with the BIP engine compared to the classical BIP. This is an important advantage, since additional coordination implies potential computational overhead, which may be critical for a system with limited resources. A well-triggered modal flow is composed of synchronized components, which successively perform computation steps. It defines the behavior of the system. Well-triggered modal flows are considered to be most fitting to model real-time multimedia systems.

In the frame of development of a scalable low-power, high-performance and trusted ultrasonic platform, feasibility, optimality and quality control become of utmost importance. In terms of timing, feasibility implies that no processing task must miss its deadline. While operating under this constraint, the system must make optimal use of its resources and time budget and at the same time provide the maximum possible quality for the produced images. Existing work [5] formulates and addresses the problem of QoS control of real-time multimedia systems in a feasible and optimal manner.

In this paper we present the model of a reconfigurable multi-mode ultrasonic application. The application is modeled as a modal flow graph composed of well-triggered components, that guarantees deadlock-freedom and confluence by construction, with basic QoS management. We define four essential types of the components required to build a reconfigurable multimedia application, namely *processing*, *buffer*, *accumulator* and *mode-selection* components. We also reason about the composition of such components showing that it results in a well-triggered composite component, which is essential for overall deadlock-freedom and confluence. Each component of this modal-flow comprises a configuration port that allows external component reconfiguration by a specific set of parameters sent from a separate controller component. The configuration ports are used for both structure reconfiguration by managing mode-selection components and buffer sizes as well as specific quality level control of processing elements.

The rest of the paper is structured as follows; Section 2 provides background information regarding BIP, modal flow graphs, and QoS. In Section 3 the problem is presented through our case study of a real-life ultrasonic application. Following, Section 4 describes our generic approach for modeling image processing applications, with the use of modal flow graphs, that provides QoS management and guarantees deadlock-freedom and confluence. Section 5 illustrates our approach on a case study, emphasizing the QoS management by application structure. Finally, in Section 6 we conclude and present ideas for future work.

2 Background

In this section we first present the essential background information, which starts with a general description of the BIP (Behavior-Interaction-Priority) framework. Then we talk about a specific part of this framework, namely modal flow components and graphs. We conclude the section with the description of QoS management technique for multimedia systems.

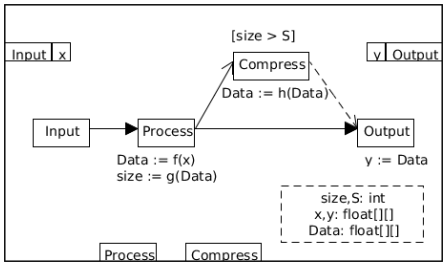

Figure 1 Example of Modal flow component.

2.1 BIP Framework

BIP [1] is a framework that provides essential means for rigorous design and modeling of heterogeneous systems. The BIP framework allows the modeling of systems as composition of atomic components by encompassing three layers: Behavior, Interaction and Priority. The behavior of each atomic component is described as 1-safe priority Petri-net extended with data and ports. The composition of these components is supported by the Interaction and Priority layers. Interactions between components, which are specified by connectors [2], are used to define the way systems are composed of components. Priorities are used to eliminate conflicts between interactions and thus restrict non-determinism.

In BIP the execution is driven by the BIP engine which has all the necessary information about the components, their connectors and the associated priorities. At every execution cycle, the engine receives information about the set of active ports for each of the atomic components. It then computes the set of interactions that have maximal progress and if there are more than one, picks one of them non-deterministically. The engine notifies the components of the chosen interaction and computes the associated data transfers. Each of the notified components then performs the associated transition and updates its state.

2.2 Modal Flow Graphs and Well-triggered Components

Modal flow components and modal flow graphs [3] are part of synchronous BIP and are used to model systems that are composed of synchronous components. Modal flow components are a particular class of 1-safe priority Petri-nets, extended with data and ports. As a result, modal flow components can directly be translated into 1-safe Petri-nets, following the semantics defined in [3]. These Petri-nets define the behavior of each component.

In modal flow components the dependency relations between events/actions are expressed using three kind of causal dependencies [3]:

- **Strong:** An event q strongly depends on p if the occurrence of p must always be followed by q. That is p and q can not happen independently.
- **Weak:** An event q weakly depends on p if the occurrence of p may be followed by q. That is either p happens or the sequence pq happens.
- **Conditional:** An event q conditionally depends on p if both p and q occur, then p must be followed by q. Otherwise p and q can occur independently.

In Figure 1, an example of a modal flow component is depicted. In the figure, solid arrows with filled arrowheads depict strong dependencies, solid arrows with normal arrowheads depict weak dependencies and dashed arrows depict causal dependencies. The rectangles represent ports and their associated data. This notation will be used in the rest of the paper.

In this example, a component is presented that can receive an input, process it and depending on the size decide to compress it, or not, before delivering the output. In Figure 1, it is depicted that *Output* strongly depends on *Process*, which, in turn, strongly depends on *Input*. This means that the component provides an output after processing, which, in turn, may occurs only after the component has received an input. It must be noted that this implies that the component, once the input is received, will obligatory process it and consequently produce the output.

Furthermore, *Compress* weakly depends on *Process*, which means that compression may occur after the processing. This depends whether the associated guard of that port, enclosed in square brackets, validates to true. Also, *Output* conditionally depends on *Compress*, that is if both occur then compression must happen before delivering the output. Finally, underneath each port the update functions are placed, which describe the associated computations.

Well-triggered components are modal flow components for which deadlock-freedom and confluence are guaranteed by construction iff the following constraints are met [3]:

- The causal dependency graph has no cycles.
- Each port has either strong or weak causes, but not both.
- Each port has at most a minimal strong cause.
- Each port that has strong causes, must have its guard true.

The example presented in Figure 1 is a well-triggered modal flow component. Interestingly, these constraints, e.g. graph acyclicity, are easy to check, either manually or automatically.

Well-triggered components can be composed based on interactions among their ports. The result of the composition is the modal flow graph that defines the behavior of the whole system. It must be noted that composition is a partial operation. This means that the composition of well-triggered components does not guarantee that the resulting modal flow graph will be well-triggered as well. Thus, in order to guarantee deadlock-freedom and confluence the constraints must be validated on the final modal flow graph.

As a result of confuency of well-triggered modal flow graphs, the existence of the BIP engine, that is present in classic BIP, is redundant and unnecessary. The engine is considered redundant since the confluent behavior of synchronous systems results in a deterministic execution of interactions. The engine is unnecessary as it introduces an extra processing overhead. This overhead originates from the fact that the BIP engine at each execution cycle computes the set of maximal interactions. Based on the confluent behavior of synchronous systems, this can be replaced by a single predefined scheduling of interactions out of all the possible ones. Finally, as a consequence of confluency in BIP components, code generation is possible.

2.3 QoS

There exist different approaches to systems design that address different levels of systems criticality. Currently, these systems are classified as safety-critical or best-effort systems. Safety-critical systems require high level of correctness, meaning no violation of critical constraints, e.g. timing constrains, when all the deadlines must be met. Engineering of such systems uses a conservative approach based on the worst-case execution time, which is often largely over-estimated and, therefore, implies not optimal or even redundant use of available resources. The best-effort systems are more relaxed in terms of critical constrains, where occasional miss of deadlines will not cause any hazardous outcomes. The design of such systems is mainly targeting efficient and optimal use of available resources.

Design of medical ultrasonic systems require meeting both critical and best effort properties. Such engineering approach is addressed in [5]. The authors proposed a method for fine grain QoS management of real-time applications, which allows the run-time adaptation of overall system behavior. The proposed approach provides control over three main properties:

- **Feasibility**, that is no deadline is missed;
- **Optimality**, that maximizes the use of available resources, e.g. provide the best QoS within specified resource constraints;
- **Smoothness** of quality levels, that is of particular importance to the multimedia applications.

Such QoS management considers a single-threaded process network application, which cyclically performs data transformation. Possible QoS levels and platform-dependent timing information of processing components must be provided as an input. The coordination of components execution is then controlled by a *controller* that monitors the progress of the computation within each cycle.

3 Reconfigurable Multimedia Systems

Generally, image processing and its applications follow the *input-process-output* paradigm. More specifically, an image processing application can be analyzed in several stages of computation, each of which receives the result of the previous computation stage as input, processes the input and delivers the output to the next computation stage.

This paradigm inherently enforces the components of an application to form processing pipelines. It is important to note that different pipelines may achieve their outputs with different quality characteristics, resource requirements and/or implementation. For example, in Section 3.1, different ultrasound imaging pipelines for B-mode are presented. Another typical example of that are pipelines that perform the required processing on raw images and then compress them, compared to pipelines that first compress the images and then perform the required processing [9]. The former, usually requires more resources but achieves better quality, while the latter requires less resources but results in degraded quality.

It is apparent that even if both pipelines deliver same outputs, their implementation may differ substantially since required processing actions performed on images have different nature.

To this end, we distinguish two different approaches for quality management:

- **QoS by structure**: In this approach, different quality outcomes are achieved based on the mode of processing pipeline (i.e. processing first, compression first, etc).
- **QoS by precision**: In this approach, having a concrete pipeline, different quality outcomes are achieved based on the parameters of the processing components (i.e. low compression-rate).

In this work we focus on providing a framework to model image processing pipelines that can be reconfigured by combining different modes in order to encompass QoS by structure.

3.1 Case Study

There exist several types of imaging applications that are based on ultrasound waves, including A-mode, B-mode (or 2D-mode), Doppler mode, Harmonic mode, and many others [4]. B-mode (brightness mode) ultrasound application is the most-known imaging technique due to its vast applicability in several diagnostic domains.

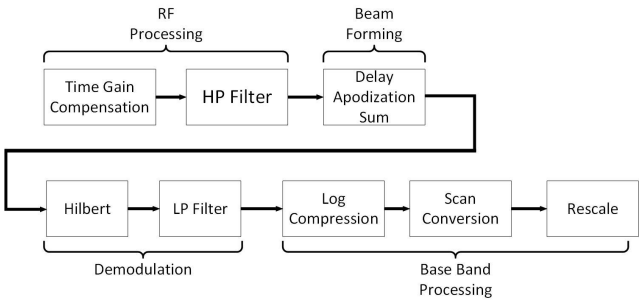

Figure 2 Baseline B-mode Processing.

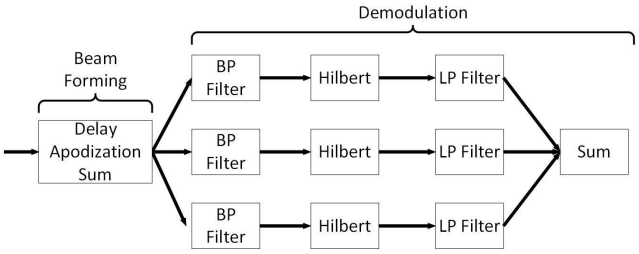

Figure 3 Frequency compounding.

In B-mode an array of transducers, called the probe, emits a beam of ultrasound waves and scans a plane through the body, which is then transformed into a two-dimensional image on a screen. There are variations of this technique that affect not only the way the retrieved signal is processed, but also the quality of the output image. Such variations include, but are not limited to:

- the mode of processing (baseline, in-phase and quadrature, frequency compounding, spatial compounding);
- the shape of the probe (linear, convex, phased array, etc);
- the type of the beam wave (planar, curved, etc);
- the angle of steering of the beam.

In this paper we focus on the algorithmic part of the variations of ultrasonic techniques, namely modes of processing. Generally, B-mode imaging consist of four processing stages; i) RF processing, ii) Beamforming, iii) Demodulation, iv) Baseband processing. These stages are comprised of several components, the ordering of which may slightly vary depending on the processing mode. More specifically, we consider the following modes:

1. *Baseline B-mode Processing*: This is the typical processing pipeline as depicted in Figure 2.
2. *Frequency Compounding*: In this mode multiple bands are separately demodulated and then summed. As a result, the output image has less high-frequency noise but also lower resolution. This mode differs from the *Baseline B-mode Processing* only in the Demodulation stage. The modified Demodulation is depicted in Figure 3.
3. *Spatial Compounding*: In this mode, instead of a single beam, several beams are emitted with different types (e.g. steered, curved, etc). Since multiple firings are used to reconstruct a single image, this procedure increases resolution. This mode differs from the *Baseline B-mode Processing* only in the Baseband stage. The modified Baseband processing is depicted in Figure 4.

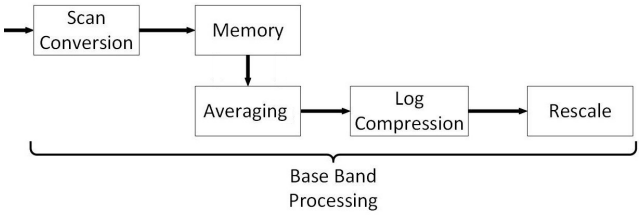

Figure 4 Spatial compounding.

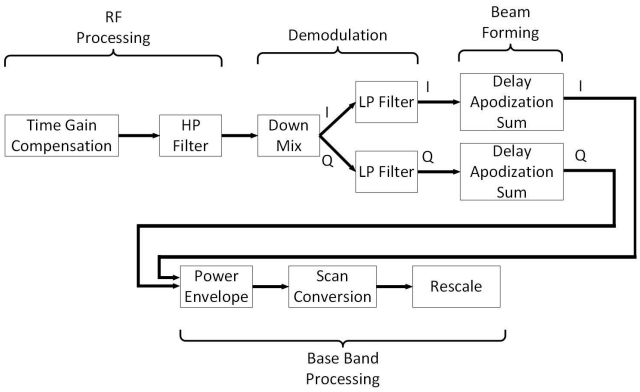

Figure 5 I/Q B-mode compounding.

4. *I/Q mode*: In this mode, in contrast with the aforementioned modes, the Demodulation stage occurs before Beamforming. The output quality depends on the subject under study (e.g. normal tissue, abdomen, etc.) as well as the configuration parameters of the components. The whole pipeline for this mode is depicted in Figure 5.

The quality of the resulting image depends on various aspects. It may depend on parameters set of each component. For example, a low-pass filter with higher filter order, i.e. cut-off quality, will result in a signal less contaminated with frequencies higher than cut-off frequency. Alternatively, the quality of the final image is also determined by the choice of the B-mode pipeline.

In this paper we exploiting the possibility to achieve optimal quality outcomes by reconfiguring the mode of operation. In this case study, we also focus on minimizing the program memory, that is to avoid redundant components where possible. The reason is that such over-provisioning may have a direct impact on implementation cost (i.e. in a FPGA implementation).

4 Components framework

In this section we present the framework of the classified components required to model ultrasound image processing that allow to provide QoS not only by structure, but also support QoS by precision.

We also show that their composition results in well-triggered components. This formal specification guarantees that the final pipeline, which is composed of these components, is deadlock-free and confluent.

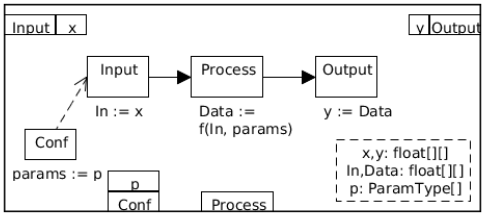

Figure 6 Processing Component.

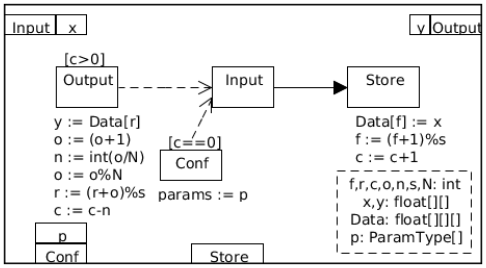

Figure 7 N-read Buffer Component of size s.

In order to model reconfigurable multimedia systems, we consider the following components:

- *Processing Components*: These components are the building blocks that follow the *input-process-output*. They are responsible for applying the necessary transformation to the input in order to get the desired output. They may have multiple inputs and/or outputs but to simplify we will refer them as input and output, respectively.
- *Memory Components*: These components are necessary for storing images between stages that produce multiple outputs which have to be processed separately. For this reason they must support multiple reads of the same value. They may have multiple inputs, but only a single output. There are several possible different types of memory components that can be modeled, such as FIFO, LIFO, etc. Following in this section, an N-read buffer is formally defined.
- *Accumulating Components*: These components are necessary for combining multiple images into a single one (e.g. different color channels). They may have multiple inputs, but only a single output. Following in this section, an N-write accumulator with a single input and a single-output is defined.
- *Mode-selection Components*: These components allow the reconfiguration of the pipeline. It has a single input and multiple outputs, one of which is active at any time, based on the mode.

Based on this components, it is possible to model complex pipelines that can be reconfigured and provide QoS by structure. In order to support QoS by precision, each of the aforementioned components must have a configuration port that will allow modification of the processing parameters.

4.1 Processing Component

In Figure 6, the main building block of a pipeline, the processing component, is presented as a well-triggered component. This component is responsible to receive an input, process it and finally output the result in one computation step. At the beginning of each computation step the component can receive new configuration parameters from the *Conf* port. It is assumed that the component is initialized with default parameters.

In ultrasound applications the processing component, receives at the *Input* port a 2-dimensional array that represents the image. Similarly, the component exposes the image at the *Output* port. The *Process* port is needed to signal that the processing has been performed. Finally, the *Conf* port is used to configure the component with appropriate parameters.

The strong dependencies between these ports enforce the *input-process-output* paradigm. The conditional dependency between the *Conf* port and the *Input* port implies that if both

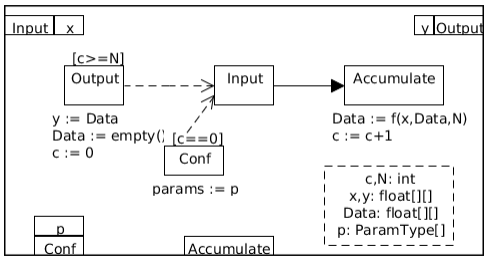

Figure 8 N-write Accumulator.

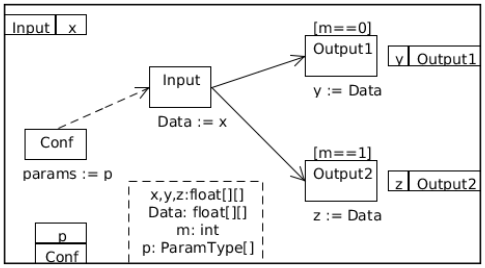

Figure 9 Mode-selector Component.

happen at same computation step, the configuration of the component must occur before the component receives the input. That means that a) the component can receive new configuration if there is no input to process; b) the component can receive the input and process it if there are no new configuration parameters; c) the component will receive first the new configuration parameters and then process the input based on these new parameters.

4.2 N-read Buffer Component

Figure 7 presents a buffering element, of size s as a well-triggered component. Following the standard notation, this component receives an image from the *Input* port and stores it internally, by copying the image to the memory and adjusting the front pointer f and the element counter c. The *Output* port exposes the oldest image stored internally, and when the image is read N times, the rear pointer r and the element counter c are adjusted. To achieve that behavior the output counter o and the next index n are used. The *Output* port is active, i.e. can be executed, only when the element counter is greater than zero, that is there is at least one image stored. Similarly, the *Conf* port is active only when there is no image in the buffer.

Storing and retrieving an element from a buffer can occur in arbitrary orderings. It is assumed that the buffer can execute both in a single computation step. Based on this assumption, *Input* and *Output* can occur independently, but if they occur in the same computation step *Output* precedes *Input*.

Finally, based on this well-triggered component, an unbounded buffer can be modeled as well by simplifying the update functions for the position pointers f and r, *f:=f+1* and *r:=r+1* respectively. A typical 1-read buffer can also be modeled by eliminating the update functions for o, n and replace them with the value of 1.

4.3 N-write Accumulator Component

Similar in logic to the buffer component is the accumulator component presented in Figure 8. This component when it receives a new input, it accumulates (e.g sum, average, select min/max, etc.) the new input with all the previously received inputs. When it has received N inputs, it outputs the result and empties the data in order to receive new inputs.

4.4 Mode-selector Component

Another important component required to model multi-mode processing pipelines is that of the mode-selector. The role of this component is to direct the received input to the correct output and thus change the processing pipeline.

In Figure 9, a selector component that supports two modes is presented. In this component the *Output1* and *Output2* ports weakly depend on the *Input* port. This means that the input can be received without producing any output, in the case when the selected mode is not valid. As in all previous cases the *Conf* precedes the *Input* port for the aforementioned reasons.

As with the processing component, the mode-selector component can be extended in multiple outputs, in a similar manner.

4.5 Composition of components

The framework components presented earlier, namely processing, buffer, accumulator and mode-selector components, are well-triggered. In order to construct deadlock-free processing pipelines from such components we have to reason if their composition results in well-triggered components as well. Composition is performed by merging the interacting ports and inheriting the dependencies from all the interacting ports. To check that the result is still well-triggered the constraints presented in Section 2.2 should hold.

Following, a descriptive reasoning is presented, regarding that the result of the composition of the framework components is well-triggered. A more formal proof can be found in [8].

Processing – Processing: Combining two processing components results in a well-triggered component if these are connected in series, that is the output of the former becomes the input of the latter. The resulting component actually performs the two processing steps sequentially, and is well-triggered.

Processing – Mode-Selector: Combining a processing component with a mode-selector component, in terms of connecting the respective output and input, results in a well-triggered component, since the mode-selector has no strong causes and no cycle is created.

Processing – Buffer: The same holds for combining the output port of the processing component with the input port of a buffer. Their composition is well-triggered, as the processing component has only strong causes and the input port of the component has only one strong cause and no weak causes.

Processing – Accumulator: Similarly, combining the output of a processing component with the input of an accumulator produces a well-triggered component. The resulting graph has no cycles and the accumulator component has only one strong cause and no weak causes.

Accumulator – Mode-Selector: The composition of an accumulator component with a mode-selector component, by connecting their respective output and input ports, results in a well-triggered component. Although, the output port of the accumulator has a guard, the mode-selector has no strong causes and the resulting graph has no cycles.

Accumulator – Processing: Combining, on the other hand, the output port of the accumulator component with the input port of a processing component is not straightforward. The output port of the accumulator has a guard, but this does not violates the constraints, since when the two ports are merged, the resulting port has no strong causes. Thus, the resulting component is a well-triggered component.

Buffer – Processing: Similarly, combining the output port of the buffer component with the input port of a processing component results in well-triggered component, as after the combination the resulting port has no strong causes.

Mode-Selector – Processing: In the same manner, combining the output port of a mode-selector component with the input port of a processing component results in a well-triggered component despite the guards.

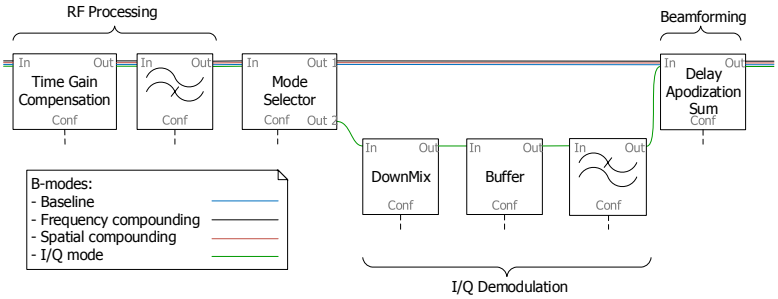

Figure 10 RF Processing, Demodulation(I/Q mode), Beamforming.

Mode-Selector – Buffer: Finally, the same holds for combining the output port of a mode-selector component with the input port of a buffer. Their composition is well triggered, as no cycles are created and the guards belong to ports with no strong causes.

It must be noted that there are several possible ways to combine these framework components. Nevertheless, the most interesting combinations, that can be used in processing pipelines, are those presented above.

5 QoS by Topology Reconfiguration

QoS by topology reconfiguration concerns the management of quality solely through the reconfiguration of the pipeline structure and not the configuration of its components. Figures 10-12 depict three consecutive parts of the reconfigurable pipline that consolidates the B-mode pipelines presented in Section 3.1.

This consolidated pipline enables quality management by merely altering only the topology of the pipeline based on the mode of operation, that is without changing the parameters of the processing components.

Every component of this pipeline is represented by one of the framework components, namely mode-selector, buffer, accumulator and processing component, that were defined in Section 4.

Each component has an In (resp. Out) port that corresponds to *Input* port (resp. *Output*) as defined for the processing components. Interactions between components are depicted with solid lines connecting participating ports. Each component has a *Conf* port that can be used for QoS managment by precision by altering the processing parameters of the components. Apparently, parameters adjusted through the *Conf* port for the mode-selector, accumulator and buffering components do not affect the quality per-se, as they do not perform any kind of processing.

In Figure 10, the first two components perform the RF processing, which is the same for all modes. After that, for all modes except from I/Q, the Beamforming is computed (by the delay-apodization-sum component). As stated in Section 3.1, in I/Q mode the Demodulation occurs before Beamforming. This is depicted in the lower branch (green line) in Figure 10 where the mode-selector component is used to switch among the modes. Further, that branch rejoins the normal pipeline in order for Beamforming to be performed.

Figure 11 depicts the Demodulation for Baseline, Frequency and Spatial compounding modes. In Baseline (blue line) and Spatial compounding (red line) the demodulation is performed through a hilbert-transformation followed by a low-pass filter. On the other hand,

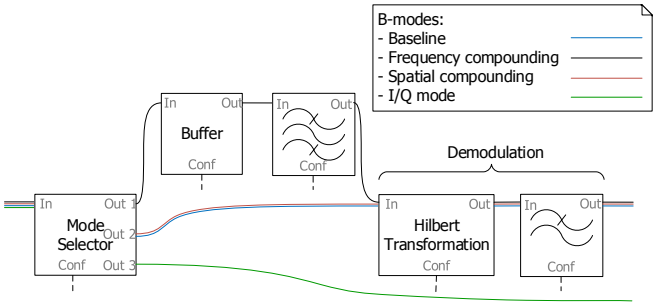

Figure 11 Demodulation (for all modes except I/Q).

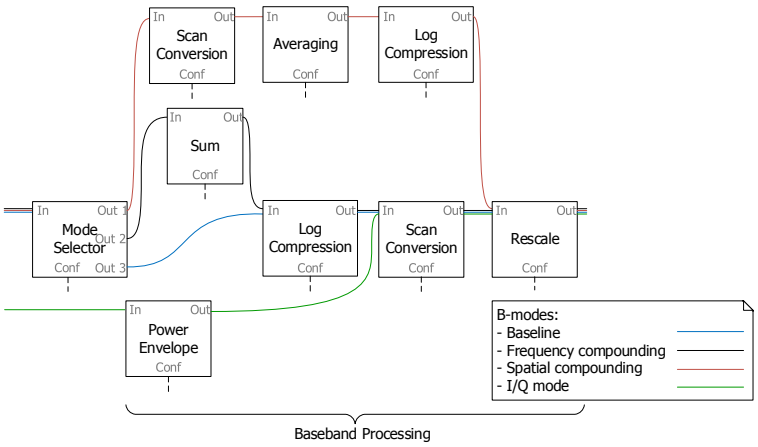

Figure 12 Baseband Processing (for all modes).

in Frequency compounding (black line) the image is passed multiple times through different band-pass filters, i.e spliting the image into several new images of different frequency-bands. The splitting is performed by using a buffer and reconfiguring the cut-off frequencies of the band-pass filter, rather than using multiple fixed filters, in order to reduce the number of components and thus reduce program memory. These images are then demodulated in the same manner as Baseline and Spatial compounding modes.

Figure 12 depicts the Baseband processing for all modes. In Spatial compounding, which is depicted on the top branch (red line), the multiple firings, which are required in this mode, are averaged after the scan-conversion. After averaging several images, the processing continues with log-compression followed by the rescaling so as to produce the final output. Similarly, in Frequency compounding (black line), the different images produced previously, by the band-pass filter, have to be summed-up. Then the same processing as with Baseline B-mode (blue line) is following. Finally, in the bottom branch, the I/Q signals are combined in the power-envelope and in the similar manner with the Baseline B-mode produces the final output.

It must be noted that the components Sum, Averaging and Power Envelope can not be modeled as processing components. Instead, they are modeled as accumulators. Similarly, the buffers present in this pipeline should be unbounded N-read buffers, where each them has an appropriate value of N.

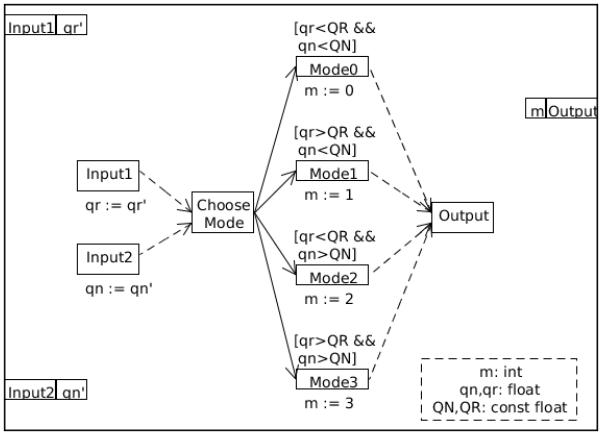

Figure 13 Simple QoS controller as well-triggered component.

This consolidated pipeline supports both QoS by structure based on the choice of a particular pipeline mode, and QoS by precision, where each component can be reconfigured separately. It also consists of well-triggered components, which can ensure deadlock-freedom and confluence. In Figures 10-12 there are some components with input ports belonging to more than one interaction, which is not allowed in modal flows in general. This can be resolved by manually implementing that part of the system. Of course, this may result in a non deadlock-free system and need to be further studied.

Finally, in Figure 13, a simple QoS controller is presented as a proof of concept that such a controller can be designed following the well-triggered paradigm. As such, the controller can be combined with the aforementioned pipeline and thus have a fully deadlock-free and confluent system that supports QoS management. This controller has two input ports, through which it receives the values *qr* (quality with respect to resolution) and *qn* (quality with respect to noise). The controller chooses the appropriate mode by comparing these values with the thresholds *QR, QN*. To make this more clear, the thresholds *QR, QN* distinguish the "*high*" and "*low*" quality for noise and resolution, respectively, while the input values are the desired quality levels. For example, if the requirements for quality with respect to resolution is "*high*" and the requirements for quality with respect to noise is "*low*", then the controller chooses mode *m=1*, which is then trasmitted to the mode-selector components that perform the choice of the porcessing mode.

6 Conclusion & Future Work

Ultrasonic imaging applications require high quality of images produced in real-time with limited resources available. In this context, feasibility, optimality and quality control are of significant importance, but the safety-critical nature of such applications requires guarantees that the system will be deadlock-free and confluent. We present an approach to model such applications using a synchronous computation model. Our approach is based on Modal Flow Graphs, which is a formalism that encompasses a synchronous computation model and guarantees by-construction deadlock-freedom and confluence provided the system satisfies some easy-to-check structural constraints.

There are two aspects of QoS management; *QoS by precision* is based on adjusting the parameters of some components of the computation chain, whereas *QoS by structure* is based

on changing the topology of the computation chain. We have presented a model of the pipeline that consolidates four modes of ultrasound B-mode processing and provides quality control by structure through pipeline reconfiguration, as well as supports quality control by precision through the adjustment of computational parameters at the component level.

We have introduced framework components, which are well-triggered modal-flow components, that can be used to build reconfigurable multimedia pipelines. We have identified the conditions that must be satisfied by the interconnection structure among the components in order to preserve deadlock-freedom and confluence. With the case study we have demonstrated how the processing pipline of the ultrasoic application can be composed out of these framework components.

Finally, we have presented a simple QoS controller as a well-triggered component which when combined with a reconfigurable pipeline results in a fully deadlock-free and confluent system that supports QoS management by topology reconfiguration.

As part of on-going and future work, in the context of our case study, we are investigating parameters (e.g. variable cut-off frequencies of the filters, levels of saprcity of the computation matrixes, etc.) and constraints (power, time) that affect the QoS management. Based on that, we are planning to extend the quality controller to take into account parameters and constraints of the underlying platform and provide optimal use of resources.

Acknowledgements. The work described in this paper is part of the UltrasoundToGo project. The aim of this project is to develop a scalable low-power, high-performance, trusted platform for 3D portable ultrasound imaging systems.

References

1 Ananda Basu, Saddek Bensalem, Marius Bozga, Jacques Combaz, Mohamad Jaber, Thanh-Hung Nguyen, Joseph Sifakis, et al. Rigorous component-based system design using the BIP framework. *IEEE Software*, 28(3):41–48, 2011.

2 Simon Bliudze and Joseph Sifakis. The algebra of connectors: Structuring interaction in BIP. In *Proceedings of the 7th ACM/IEEE International Conference on Embedded Software*, EMSOFT'07, pages 11–20, New York, NY, USA, 2007. ACM.

3 Marius Dorel Bozga, Vassiliki Sfyrla, and Joseph Sifakis. Modeling synchronous systems in BIP. In *Proceedings of the Seventh ACM International Conference on Embedded Software*, EMSOFT'09, pages 77–86, New York, NY, USA, 2009. ACM.

4 Richard SC Cobbold. *Foundations of biomedical ultrasound*. Oxford University Press, USA, 2007.

5 Jacques Combaz, Jean-Claude Fernandez, Thierry Lepley, and Joseph Sifakis. Qos control for optimality and safety. In *Proceedings of the 5th ACM International Conference on Embedded software*, pages 90–99. ACM, 2005.

6 Zahra Keshavarz-Motamed, Julio Garcia, Emmanuel Gaillard, Romain Capoulade, Florent Le Ven, Guy Cloutier, Lyes Kadem, and Philippe Pibarot. Non-invasive determination of left ventricular workload in patients with aortic stenosis using magnetic resonance imaging and doppler echocardiography. *PLoS One*, 9(1), 2014.

7 Sonia H. Contreras Ortiz, Tsuicheng Chiu, and Martin D. Fox. Ultrasound image enhancement: A review. *Biomedical Signal Processing and Control*, 7(5):419 – 428, 2012.

8 Stefanos Skalistis and Alena Simalatsar. Modeling of Reconfigurable Medical Ultrasonic Applications in BIP. Technical report, EPFL IC IIF RiSD, 2014.

9 Xiang Xie, GuoLin Li, ZhiHua Wang, Chun Zhang, DongMei Li, and XiaoWen Li. A novel method of lossy image compression for digital image sensors with bayer color filter arrays. In *Circuits and Systems, 2005. ISCAS 2005. IEEE International Symposium on*, pages 4995–4998. IEEE, 2005.