# Analyzing Synchronous Dataflow Scenarios for Dynamic Software-defined Radio Applications

Firew Siyoum[1], Marc Geilen[1], Orlando Moreira[2], Rick Nas[2], Henk Corporaal[1]

[1]Eindhoven University of Technology
[2]ST-Ericsson Eindhoven

***Abstract*—Contemporary embedded systems for wireless communications support various radios. A software-defined radio (SDR) is a radio implemented as concurrent software processes that typically run on a multiprocessor system-on-chip (MPSoC). SDRs are real-time streaming applications with throughput requirements. One efficient approach for timing analysis of concurrent real-time applications is the dataflow model of computation (MoC). Nonetheless, the dataflow modeling of SDRs is challenging due to their dynamically changing data processing workload. A dataflow MoC that is not expressive enough to capture this dynamism gives pessimistic throughput results. On the other hand, if it is too expressive and detailed, it may not be analyzable at all. In this paper, we address the challenge of dataflow modeling of SDRs such that their timing behavior can be accurately analyzed to guarantee real-time requirements without unnecessarily over-allocating MPSoC resources.**

**The basis of our modeling approach is splitting the dynamic data processing behavior of a SDR into a group of static modes of operation. Each static mode of operation is then modeled by a Synchronous Dataflow (SDF), which we refer to as *scenario*. This paper has two main contributions: 1) a scenario-based dataflow model of Long Term Evolution (LTE), which is the latest standard in cellular communication, and 2) investigation of existing throughput analysis techniques of SDF scenarios for our LTE model. Our results show that scenario-based worst-case throughput computation is $2$ to $3.4$ times more accurate than a state-of-the-art SDF analysis technique. Our investigation also shows that existing timing analysis techniques of SDF scenarios have very low run-time that scales very well with increase in graph size. This makes SDF scenarios suitable in practice for modeling and analyzing SDRs as well as similar dynamic applications.**

***Index Terms*—Synchronous Dataflow, Long Term Evolution, Software-defined Radio, Throughput, Scenario-aware Dataflow**

## I. Introduction

Present-day embedded systems for cellular, home and automotive communications support various wireless communication standards. We refer to these communication standards as *radios*. Smartphones, for instance, include different radios such as WCDMA, LTE and IEEE 802.11x. Modern radios are naturally dynamic in the sense that their transmission resource allocation change with channel conditions. As a result, radios have variable workload that is driven by control channels and signals. In addition, radios are real-time streaming applications with latency and throughput requirements. Therefore, radio designs must be *predictable* to ensure that temporal requirements are satisfied in all operating conditions.

The current trend in radio design shows that the implementation of physical layer functionalities of radios is shifting from dedicated hardware architectures to software processes for better flexibility and efficiency [1]. In *software-defined radio (SDR)* [2], some or all of the physical layer functionalities of a radio are implemented as concurrent software processes. These software processes typically run on a multiprocessor system-on-chip (MPSoC) for power and performance reasons. MPSoC architectures of SDR combine homogeneous and heterogeneous multiprocessing, including general purpose processors, vector processors and weakly programmable accelerators.

The mapping of the software processes onto the MPSoC and allocation of resources, such as memories and interconnects, is decided at the early stages of the SDR design. After such design decisions, the system must be analyzed to check if temporal requirements are met. This is not a trivial task due to the vastness of the design space and the complexity of SDRs. As a result, system-level modeling techniques are needed to efficiently analyze SDRs and ensure predictability. One established approach in the embedded domain for modeling and analyzing real-time streaming applications, such as SDRs, is the *dataflow* model of computation (MoC) [3].

Dataflow MoCs allow modeling of concurrent streaming applications as directed task graphs. Modeling and analyzing SDRs using dataflow MoCs has mainly two complementary challenges: the *dynamism of radios* and the *scarcity of MPSoC resources* [4]. On one hand, if the selected dataflow MoC is highly expressive in order to capture the dynamic behavior in detail, temporal analysis may not be possible at all. On the other hand, if it is made simplistic for the sake of analyzability, it may give pessimistic, if not invalid, results. Pessimistic results lead to unnecessary over-allocation of scarce MPSoC resources to satisfy temporal requirements. In this paper, we address the challenge of dataflow modeling of SDRs such that their timing behavior can be accurately analyzed to guarantee real-time requirements without unnecessarily over-allocating MPSoC resources.

The basis of our approach is splitting the dynamic data processing behavior of a SDR into a group of static modes of operation. Each static mode of operation is then modeled by a Synchronous Dataflow (SDF) [5] graph, which we refer to as *scenario* [6]. The possible orders of executions of these scenarios are specified by a finite-state machine (FSM) [7] [8]. In this work, we show the applicability of this MoC for SDRs by modeling and analyzing the baseband (physical layer) processing of Long Term Evolution (LTE) [9], which is the latest standard in cellular communication. This paper has two main contributions: (1) a scenario-based dataflow model of the baseband processing of LTE that accurately models dynamic behavior, and (2) investigation of existing worst-case throughput analysis techniques of SDF scenarios for our LTE model.

This scenario-based dataflow modeling is not new [6] [7]. This work extends previous works by presenting a technique for modeling control information exchanged between scenarios. In SDRs, the transfer of configurations and data from one mode of operation to the next is quite common. In LTE, for instance, a mode detection scenario broadcasts the type of a received frame to subsequent scenarios for configuration. We call such type data dependency between scenarios *scenario dependency*. Scenario dependencies that are not properly modeled result in an early start of execution of processes in subsequent scenarios. This ultimately leads to invalid worst-case temporal analysis. In this work, we present a technique to model scenario dependencies between different SDF graphs, and then show its applicability in our LTE model.

Our investigation on LTE's baseband processing shows that the scenario-based worst-case throughput computation is at least two times more accurate than a state-of-the-art SDF analysis technique. Our results also show that existing timing analysis techniques of SDF scenarios have very low run-time that scales very well with increase in graph size. This makes SDF scenarios suitable in practice for modeling and analyzing SDRs as well as similar dynamic applications.

The remaining part of this paper is organized in six sections. First, Section II recaps dataflow modeling. Then, Section III presents a scenario-based dataflow model for LTE. Section IV introduces a new technique for modeling scenario dependencies. Accuracy of throughput analysis techniques for our LTE model is investigated in Section V. We postpone the review of related works to Section VI, as preceding sections help to better present the literature study. Finally, the paper concludes in Section VII, summarizing this work.

## II. Preliminaries

This section recaps the basics of dataflow modeling. It discusses basic SDF concepts that are important for a complete understanding of this paper. In this paper, we use $\mathbb{N}$ to denote natural numbers and $\mathbb{N}^0$ natural numbers including zero.

### A. Synchronous Dataflow Graph

A Synchronous Dataflow Graph (SDFG) [5] is a directed graph that can model concurrent tasks. It can capture cyclic data dependencies between tasks. It also has efficient analysis techniques to compute throughput and buffer sizes [10] [11]. Due to its analyzability, it is widely used in system-level design flows for modeling and analyzing real-time streaming applications, running on MPSoCs [12] [13].

A SDFG consists of *actors* that are connected through *channels*. A channel represents a FIFO buffer through which actors communicate by sending *tokens*. A channel may have some *initial tokens* at the start. Every channel is exactly connected to one source actor and one destination actor. The connection point between an actor and a channel is referred to as a *port*. Each port of an actor is annotated with a fixed number, called the *port rate*.

An actor models a given system task, for example a software process. The duration of one complete execution of an actor is termed as its *execution time*, measured in any preferred *time-unit*. When an actor *fires*, i.e. starts execution, it reads tokens from all of its input ports. At the end of the execution, it produces tokens in all of its output ports. For each port of an actor, the number of tokens consumed or produced in every single execution of an actor is fixed, and it is equal to the port rate. A formal definition of SDFG is given in Definition 1.

**Definition 1** (SDFG)**.** *A SDFG is a tuple* $\mathbf{G} = (A, C, \mathcal{X}, \mathcal{I})$*, comprising a set of actors* $A$*, a set of channels* $C$*, execution times of actors* $\mathcal{X} : A \rightarrow \mathbb{N}^0$ *and initial tokens of channels* $\mathcal{I} : C \rightarrow \mathbb{N}^0$*. Given the set of all ports* $P$*, port rate of* $\mathbf{G}$ *is denoted as* $\mathcal{R} : P \rightarrow \mathbb{N}$*.*

An example SDFG consisting of three actors ($x$,$y$, $z$) and four channels is shown in Figure 1. In SDFG schematics, a black dot represents the number of initial tokens in a channel.

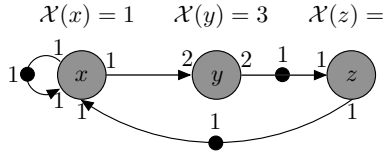



Fig. 1. Example SDFG: Graph **A**

The execution of a SDFG is a timed simulation of the executions of its actors. A property that ensures a deadlock-free execution of a SDFG is *consistency*. A SDFG is called consistent if the initial tokens configuration can be restored after a finite numbers of firings of its actors. For example, for Figure 1, the numbers of firings of actors $x$, $y$ and $z$ that bring the graph back to its original state are 2, 1 and 2, respectively. This can be conveniently written in vector form as $[2, 1, 2]$. This vector is termed as the *repetition vector* of the SDFG. The repetition vector determines one complete execution of the graph, referred to as *iteration*, as defined in Definition 2.

**Definition 2** (Iteration)**.** *An* iteration *is defined as an execution of a SDFG where each actor fires exactly as many times as its entry in the repetition vector.*

### B. The Time-stamp Vector

The purpose of this section is to show how the total number of initial tokens in a SDFG affects the performance of throughput analysis techniques, discussed later in Section V.

In temporal analysis of SDFGs, we are interested in completion times of iterations. This is because the number of iterations that can be completed per given time interval is a measure of the throughput of the graph. The collection of tokens that exist after each iteration of the graph is the same as the initial tokens configuration. Define a vector $\gamma$ that has exactly one entry for every initial token in the graph, to record the production times of initial tokens after each iteration. Vector $\gamma$ is referred to as a *time-stamp vector*. The length of the time stamp vector $|\gamma|$ is equal to the total number of initial tokens of the graph. However, the entries in this vector change between iterations.

A useful mathematical tool to analyze the timing evolution of the time-stamp vector of a SDFG is the $(max, +)$ algebra [14]. In *self-timed execution* of a given SDFG, an actor fires as soon as it has sufficient input tokens. Hence, the start time of an actor's firing is determined by the tardiest input token, i.e. the **maximum** of the production times of all input tokens. The end time of an actor's firing can be obtained by **adding** its execution time on its start time. As a result, the entire timing behavior of a self-timed execution of a SDFG can be analyzed using $(max, +)$ expressions.

The time-stamp vector of the $k^{th}$ iteration of the graph is denoted $\gamma_k$. The relationship between any two consecutive iterations is expressed by a matrix multiplication in $(max, +)$ algebra, i.e. $\gamma_{k+1} = \mathbf{M} \cdot \gamma_k$ where $k \in \mathbb{N}^0$. $\mathbf{M}$ is referred to as the *matrix* of the graph and has a size of $|\gamma| \times |\gamma|$. An algorithm to compute the matrix of a SDFG is provided in [15]. The matrix of SDFG $\mathbf{A}$, shown in Figure 1, and its time-stamp vectors for the first three iterations are as follows.

$$\mathbf{M} = \begin{bmatrix} 2 & 3 & 2 \\ 5 & 6 & 5 \\ 7 & 8 & 7 \end{bmatrix} \text{ and } \gamma_0, \gamma_1, \gamma_2, \gamma_3 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 3 \\ 6 \\ 8 \end{bmatrix}, \begin{bmatrix} 10 \\ 13 \\ 15 \end{bmatrix}, \begin{bmatrix} 17 \\ 20 \\ 22 \end{bmatrix}$$

The self-timed execution of a SDFG reaches a periodic phase after a finite number of transient iterations [10]. For the SDFG $\mathbf{A}$ of Figure 1, the periodic phase starts after the first iteration (after $\gamma_1$). The *period* of this periodic phase is 7 time-units (the difference between consecutive time-stamp vectors), and hence, the graph has a maximum throughput of $\frac{1}{7}$ iterations/time-unit.

For exact computation of time-stamp vectors, we start from a zero vector $\gamma_0$. However, approximations (upper-bounds to $\gamma$) are also possible. This is done by starting from a selected $\gamma_0$, called a *reference schedule* or a *delay* [15]. This technique

is referred to as *delay-period* approximation. As shown later in Section V, the technique allows faster throughput computation, since it makes the entire timing behavior periodic.

The time-stamp vector is a timing interface that separates iterations of a SDFG. It can also analyze a sequence of iterations of different SDFGs. To achieve this, $\gamma$ has to be extended first to cover all initial tokens that are common between these graphs. This approach is used to compute the throughput of dataflow MoCs that model dynamic applications through a set of SDFGs. One such dataflow model is discussed next in Section II-C.

### C. Synchronous Dataflow Scenarios

Real-life streaming applications, such as multimedia codecs and SDRs, go through different operating modes, depending on the processed data. A *scenario* refers to a single mode of operation of the application [6]. When an application is executing at a given scenario, its computation and communication characteristics mostly remain invariable. Hence, each scenario can be modeled by a static dataflow model, such as a SDFG. We refer to a SDFG that models a single scenario of an application a *scenario graph*.

A sequence of operating modes of an application is modeled by a sequence of executions of the corresponding scenario graphs. These executions can also be pipelined in time. All possible scenario sequences of an application are represented by a finite-state machine (FSM), as defined in Definition 3.

**Definition 3** (Finite-state machine (FSM))**.** *Given a set $S$ of scenario graphs, a finite state machine $\mathbf{f}$ on $S$ is a tuple $\mathbf{f} = (Q, q_0, \delta, \Sigma)$. $Q$ is a set of states, $q_0 \in Q$ is an initial state, $\delta$ is a transition relation between two states, $\delta \subseteq Q \times Q$, and $\Sigma$ is scenario labeling, $\Sigma : Q \rightarrow S$.*

Scenario graphs along with a FSM can be used to capture the dynamic behavior of an application [15] [8]. For example, let two SDFGs $\mathbf{A}$ and $\mathbf{B}$ represent two different scenarios of an application. Assume also that the application switches arbitrarily between these two scenarios. Figure 2 shows a FSM that captures this dynamic behavior. This modeling approach is referred to as *FSM-based Scenario-aware Dataflow (FSM-SADF)* [15] [8], as defined in Definition 4.

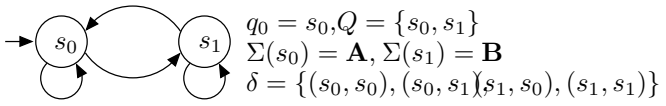



Fig. 2. Example of a FSM

**Definition 4** (FSM-SADF)**.** *An FSM-SADF model is a tuple $\mathbf{F} = (S, \mathbf{f})$, consisting of a set of scenario graphs $S$ and a finite-state machine $\mathbf{f}$ on $S$.*

The FSM-SADF is a class of Scenario-aware Dataflow (SADF) model [6]. In FSM-SADF, scenario transitions are non-deterministically specified by a finite-state machine, instead of the stochastic model proposed in [6]. The FSM allows a worst-case timing analysis of FSM-SADF through the timing evolution of the time-stamp vector, as discussed in Section II-B. Efficient timing analysis techniques for FSM-SADF are presented in [15] and [7].

Next, in Section III, we show the applicability of FSM-SADF for modeling and analyzing a dynamic SDR application. The section shows how FSM-SADF captures the dynamism of the SDR application, which a single SDFG cannot do without introducing pessimistic assumptions.

## III. Long Term Evolution (LTE)

Long Term Evolution (LTE) is a recent standard in cellular wireless communication technologies. It aims at high bit rates: a downlink peak rate of up to 300 Mbit/s and an uplink of 150 Mbit/s [9]. Due to the high bit rates, and the resulting high workload, the complexity of LTE receivers is enormous. The complexity is further increased by dynamism (data-dependent variations) of frames. In this section, we focus on the dynamism of LTE's physical layer frames, as SDR deals with a software implementation of the baseband processing (physical layer processing) of radios.

LTE uses *adaptive modulation and coding (AMC)* that dynamically adjusts modulation schemes and transport block sizes to adapt to varying channel conditions [16]. Consequently, the workload of LTE's baseband processing changes dynamically. In this section, we present a variation-aware dataflow model for LTE baseband processing that captures this dynamic workload. We first start by discussing the source of dynamism in the physical layer processing of LTE in Section III-A. Then, we show how we model this dynamism using FSM-SADF dataflow in Section III-B.

### A. Dynamism in LTE baseband processing

There are multiple sources of dynamism in LTE baseband processing that contribute to variable computation and communication requirements. These include variations in channel allocation of frames, modulation schemes and transport block sizes of upper layers. For the discussions of this paper, we limit ourselves to dynamism due to variations in channel allocations of frames. Nonetheless, the modeling concept can be equally applied to any complexity level of dynamism.

To show variations in channel allocations, we consider the downlink communication, which refers to the communication link from the base station (eNodeB) to the User Equipment (UE). Depending on the type of duplexing, there are two types of LTE physical layer frame structures. The downlink frame structure for Frequency Division Duplexing (FDD) is illustrated in Figure 3. A single frame is 10 milliseconds (ms) long. It consists of 10 *sub-frames* (1ms each) and each sub-frame consists of 2 *slots* (0.5ms each).

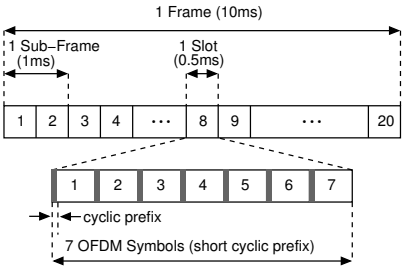



Fig. 3. LTE frame structure for FDD

LTE employs Orthogonal Frequency Division Multiplexing (OFDM) for downlink data transmission. The transmission resource within a sub-frame is organized by a *resource grid*, as shown in Figure 4. The width of the resource grid (in time domain) equals twice the number of symbols per slot, $N_{symb}^{DL}$. The height of the grid (in frequency domain) equals the number of OFDM sub-carriers per resource block, $N_{sc}^{RB}$, multiplied by the number of resource blocks per sub-frame, $N_{RB}^{DL}$. $N_{RB}^{DL}$ is determined by the downlink transmission bandwidth, while $N_{symb}^{DL}$ and $N_{sc}^{RB}$ are determined by the OFDM subcarrier spacing and the type of OFDM cyclic prefix used.

In practice, $N_{RB}^{DL}$, $N_{sc}^{RB}$ and $N_{symb}^{DL}$ are fixed once the system is configured. In the rest of this paper, we consider a bandwidth of 20MHz, a subcarrier spacing of 15KHz and normal cyclic prefix. Hence, $N_{RB}^{DL} = 100$, $N_{symb}^{DL} = 7$ and $N_{sc}^{RB} = 12$, as shown in Figure 4.

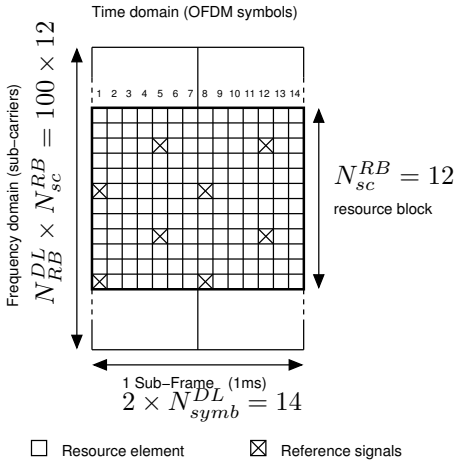



Fig. 4. Resource grid of a sub-frame

The time-frequency unit for resource allocation of the resource grid is a *resource element*. Resource elements of the resource grid are allocated to different data and control channels. Resource elements of the first OFDM symbol (the first column of the grid) are allocated to the *Physical Control Format Indicator Channel* (PCFICH) and partly to the *Physical Downlink Control Channel* (PDCCH). PCFICH contains information regarding the resource allocation of PDCCH. PDCCH can be allocated resource elements upto the third column of the resource grid. PDCCH, in turn, tells the locations of data channels, such as the *Physical Downlink Shared Channel* (PDSCH). PDSCH can be located between the second and the fourteenth columns of the resource grid.

Decoding a sub-frame consists of a number of tasks whose data dependency is captured by a directed graph, as shown in Figure 5. Some major tasks of the graph include *OFDM demodulation (dem)*, *channel estimation (est)*, *multiple-input and multiple-output summation (mimo)*, *OFDM demapping (dmp)* and *channel decoding (dec)*. The input-output data granularity of these tasks is an OFDM symbol (a column of the resource grid), that is about 4800 bytes. Hence, these tasks have to be carried out for each of the 14 symbols that constitute a sub-frame.

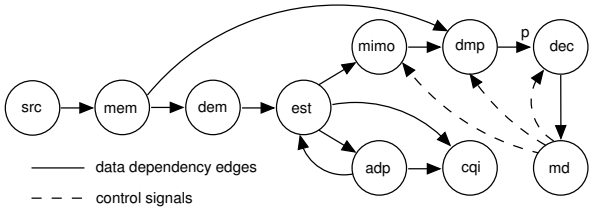



Fig. 5. A directed task graph of LTE's baseband processing

However, depending on the type of channel a symbol is allocated to, the properties of these tasks, such as functionality, execution time and communication rates, vary. For instance, task *dec* has a different execution time for a symbol that carry a control channel (192 time-units) than a data channel (an average of 75 time-units). In addition, its input data rate can vary between 11, 12 or 13 symbols while decoding a data channel. This is because the control channel is always between the first and the third symbols, leaving the remaining symbols for data channels.

Consequently, the execution time and the input-output data rates of tasks may change every symbol. This gives rise to the dynamic behavior of LTE's baseband processing. The challenge is now how to capture this dynamism in dataflow models for temporal analysis. Dataflow modeling and analysis of LTE, and other radios, that abstract from variable execution times and communication rates result in inefficient, if not invalid, implementations. This is because static worst-case conditions have to be considered for all operating conditions. Let us see, for instance, what a static SDFG model for the LTE baseband processing looks like.

All tasks of Figure 5, except *dec*, fortunately have fixed execution times and input-output token rates. Thus, the modeling effort simplifies to finding a fixed execution time for task *dec* and its input port rate $p$. The requirement for the selection of these two parameters is that the production time of tokens by actor *dec* must be conservative to (not earlier than) the actual production time of data by task *dec*. Symbols that carry control channels have to be decoded as soon as they arrive. This requires $\mathcal{R}(p) = 1$ and $\mathcal{X}(dec) = 192$. This configuration also ensures that the decoding of a data channel, which is carried out in a chunk of 11, 12 or 13 symbols, is also conservative at a sub-frame level. This SDFG of LTE's baseband processing is shown in Figure 6, where all port rates equal to one and execution times are shown by numbers written inside the actors.

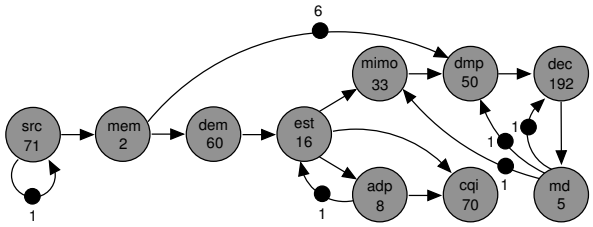


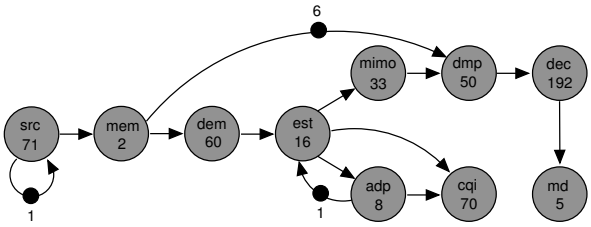

Fig. 6. A SDFG of LTE's baseband processing

Due to the static nature of the SDFG, the execution time of *dec* is fixed to 192 time-units for all symbols, though it is on average 75 time-units for data channels. In addition, actor *md* (the mode detection task) is executed for every symbol, even though it is only needed for the first symbol of the resouce grid. As a result, the timing analysis of this SDFG gives a pessimistic throughput result, as shown later in Section V. This fact necessitates a variation-aware dataflow model that gives more accurate throughput results. In the next section, we present a dataflow model for LTE that captures variable execution times and input-ouput data rates through a set of scenario graphs.

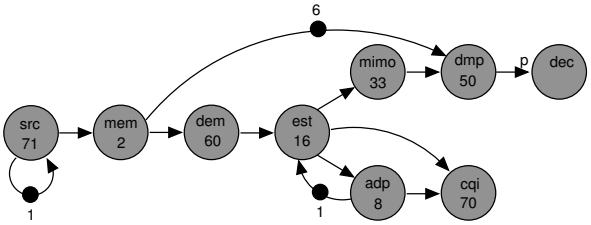
### *B. Dataflow Model of LTE baseband processing*

For the LTE's baseband processing, we identify five different modes of operation, depending on the type of symbol it is processing. When operating at a given mode of operation, the execution times and input-output rates of the tasks remain static. Therefore, each mode of operation can be modeled by a SDFG. In addition, the possible transitions between these five modes of operation are also known at design time. Hence, the transitions can be described by a finite-state machine (FSM), making FSM-SADF a suitable tool to model this application.

We present the FSM-SADF model for LTE's baseband processing in two steps: first in this section, without modeling the control signals (shown in Figure 5), and later in Section IV, with the modeling of these controls signals.

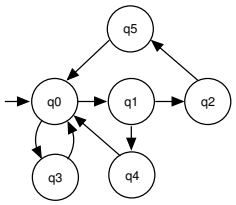



(a) Scenario graph 1: $\mathbf{S_1}$




$\mathbf{S_2} : \mathcal{X}(dec) = 192, \mathcal{R}(p) = 1$
$\mathbf{S_3} : \mathcal{X}(dec) = 970, \mathcal{R}(p) = 13$
$\mathbf{S_4} : \mathcal{X}(dec) = 895, \mathcal{R}(p) = 12$
$\mathbf{S_5} : \mathcal{X}(dec) = 820, \mathcal{R}(p) = 11$

(b) Scenario graph 2 to Scenario graph 5: $\mathbf{S_2} - \mathbf{S_5}$




$\Sigma(\text{q0}) = \mathbf{S_1}$
$\Sigma(\text{q1}) = \mathbf{S_2}$
$\Sigma(\text{q2}) = \mathbf{S_2}$
$\Sigma(\text{q3}) = \mathbf{S_3}$
$\Sigma(\text{q4}) = \mathbf{S_4}$
$\Sigma(\text{q5}) = \mathbf{S_5}$

sequences for sub–frame decoding

$\bar{q}_1 =< \text{q0, q3} >$
$\bar{q}_2 =< \text{q0, q1, q4} >$
$\bar{q}_3 =< \text{q0, q1, q2, q5} >$

(c) FSM

Fig. 7. FSM-SADF model of LTE receiver

A compact representation of the five scenario graphs that correspond to each mode of operation is shown in Figure 7(a) and Figure 7(b). Except for those explicitly shown in Figure 7(b), all port rates are 1 and execution times are indicated by numbers written inside the actors.

The first scenario graph, $\mathbf{S_1}$, models the decoding of the first symbol, which has the control format channel (PCFICH) and part of the control channel (PDCCH). At the end of the execution of $\mathbf{S_1}$, the *mode detection (md)* actor determines the scenario sequence to decode the remaining 13 symbols. The three possible sequences are: 1) executing $\mathbf{S_3}$ to decode all the 13 symbols for the data channel (PDSCH), 2) executing $\mathbf{S_2}$ to decode the second symbol for the control channel (PDCCH), followed by $\mathbf{S_4}$ to decode the remaining 12 symbols for the data channel (PDSCH), and 3) executing $\mathbf{S_2}$ twice to decode the second and third symbols for the control channel (PDCCH), followed by $\mathbf{S_5}$ to decode the remaining 11 symbols for the data channel (PDSCH). The FSM in Figure 7(c) shows the three scenario sequences $\bar{q}_1, \bar{q}_2$ and $\bar{q}_3$ to decode one complete sub-frame.

The timing analysis of the FSM-SADF model can be carried out by executing these scenario sequences. However, before we present the timing analysis, we first need to discuss an important modeling aspect that is not captured by Figure 7. It is mentioned earlier that the type of scenario sequence for a given sub-frame is determined by actor *md* of $\mathbf{S_1}$. Hence, actor *md* should run to completion before scenario graphs $\mathbf{S_2}$ and $\mathbf{S_3}$ start execution. Scenario-independent actors in these graphs can, in fact, start ahead of the completion of actor *md*. The dashed edges in the directed task graph of Figure 5 show actors that have dependency with actor *md*. The dashed edges represent data dependencies that exist across scenarios: from actor *md* of $\mathbf{S_1}$ to actors *mimo, dmp* and *dec* of $\mathbf{S_2}$ and $\mathbf{S_3}$. We refer to such types of data dependencies that exist between scenarios as *scenario dependencies*.

The presence of these scenario dependencies in the transitions $\delta_{q_0q_1} = (\text{q0}, \text{q1})$ and $\delta_{q_0q_3} = (\text{q0}, \text{q3})$ are not modeled by the FSM-SADF, shown in Figure 7. This has a serious consequence, as it may lead to invalid timing analysis. Section IV next presents a modeling technique for data dependencies that exist across scenario graphs.

## IV. Modeling Scenario Dependencies

The temporal behavior of an FSM-SADF model is analyzed by executing the possible scenario sequences specified by the FSM. For a given sequence, the scenario graph of each state is executed for one complete iteration. Figure 8 illustrates an example of a sequence of iterations $\langle \cdots, k, k+1, \cdots \rangle$ of two scenarios. As shown in the figure, these iterations are possibly pipelined in time.

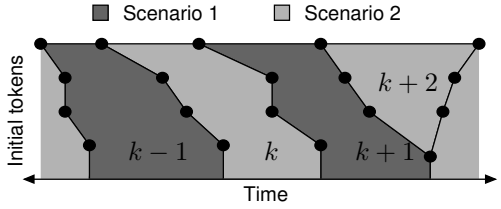



Fig. 8. Example of sequence of iterations

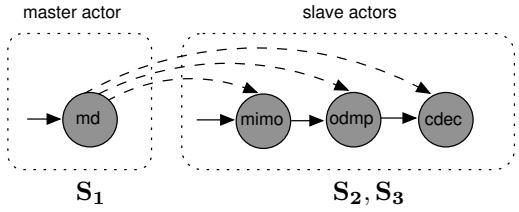
The end of iterations is marked by the production times of initial tokens, represented by the black dots in Figure 8. These initial tokens are the set of all initial tokens of the scenario graphs. Part of the initial tokens that are common between two scenario graphs (iterations) represent their data dependencies. This is because the starting times of actors that consume these common initial tokens is determined by the production times of the initial tokens in the previous iteration. Hence, to avoid earlier starting of actors, all data dependencies between iterations should be captured through common initial tokens between scenario graphs.

However, it is not possible to model data dependencies using common initial tokens unless both the source and destination actors of channels that carry the initial tokens exist in both scenario graphs. It requires, otherwise, channels that extend across the two scenario graphs, which FSM-SADF does not allow. This results in scenario dependencies, as discussed in Section III-B for the FSM-SADF model of Figure 7.

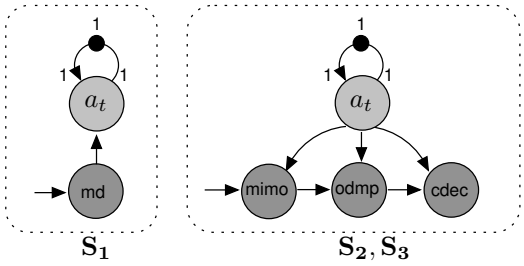
A scenario dependency is a data dependency from a source actor of a given scenario graph to a non-empty set of destination actors of a set of scenario graphs. We refer to the source actor as the *master actor* and the destination actors as the *slave actors*. A master actor and its slave actors can possibly belong to the same scenario graph. Slave actors must not fire before the master actor completes all of its firings of an iteration, i.e. as many firings as its entry in the repetition vector. Figure 9(a) illustrates the scenario dependency of the FSM-SADF model shown in Figure 7. Actors $mimo$, $dmp$ and $dec$ must receive configuration information from actor $md$ before they start execution.

Therefore, to determine the correct firing times of slave actors in subsequent iterations, the completion time of the master actor must be recorded. This can be achieved by introducing a new actor for every master actor. We refer to this new actor as the *time-stamp* actor. The time-stamp actor

is a SDFG actor with a single self-edge, as illustrated in Figure 9(b). The self-edge has exactly one initial token that carries the time-stamp of the completion of the master actor. The same time-stamp actor is introduced in all parent scenario graphs of the slave actors, as shown in Figure 9(b), so that the firing times of the slave actors cannot be earlier than the completion of the master actor in preceding iterations.




(a) Scenario dependency between master and slave actors




(b) time-stamp actor $a_t$ where $\mathcal{X}(a_t) = 0$

Fig. 9. Modeling scenario dependency

Any number of scenario dependencies can be modeled with this technique. All sorts of scenario dependencies are also possible. For instance, a master actor of a scenario dependency can be a slave actor in another scenario dependency. An actor can also be a slave for multiple scenario dependencies. In addition, some of the slave actors of a scenario dependency can be on the same scenario graph as the master actor. In the last case, a channel that connects the time-stamp actor with a slave actor needs initial tokens to avoid deadlock.

A generalized formal presentation of the scenario dependency modeling technique is presented next, in Algorithm 1. The algorithm also covers the assignment of port rates ($\mathcal{R}$) and initial tokens ($\mathcal{I}$) of channels that connect time-stamp actors with master and slave actors.

Table I defines notation for a given FSM-SADF model $\mathbf{F} = (S, \mathbf{f})$. We use $\mathbb{P}$ to denote powerset.

TABLE I
NOTATION FOR AN FSM-SADF MODEL $\mathbf{F} = (S, \mathbf{f})$

| Notation | Description |
|---|---|
| $\mathcal{A}(\mathbf{s})$ | the set of actors of scenario graph $\mathbf{s} \in S$ |
| $repVector(a)$ | the repetition vector of an actor $a \in \mathcal{A}(\mathbf{s})$ |
| $\mathcal{C}(\mathbf{s})$ | the set of channels of scenario graph $\mathbf{s} \in S$ |
| $srcPort(c)$ | the source port of channel $c \in \mathcal{C}(\mathbf{s})$ |
| $dstPort(c)$ | the destination port of channel $c \in \mathcal{C}(\mathbf{s})$ |
| $M_\mathbf{F}$ | the set of all master actors in $\mathbf{F}$ |
| $S_\mathbf{F}$ | the set of all slave actors in $\mathbf{F}$ |
| $T_\mathbf{F}$ | the set of all time-stamp actors in $\mathbf{F}$ |
| $O_\mathbf{F}$ | the set of all ordinary actors in $\mathbf{F}$ |
| $A_\mathbf{F}$ | the set of all actors in $\mathbf{F}$, or equivalently $A_\mathbf{F} = M_\mathbf{F} \cup S_\mathbf{F} \cup T_\mathbf{F} \cup O_\mathbf{F}$ |
| $\mathcal{G}(a_m)$ | parent scenario graph of $a_m \in M_\mathbf{F}$ |
| $\mathcal{S}(a_m)$ | slave actors of $a_m$, $\mathcal{S} : M_\mathbf{F} \rightarrow \mathbb{P}(A_\mathbf{F})$ |
| $\mathcal{SG}(a_m)$ | the set of parent scenario graphs of slave actors of $a_m \in M_\mathbf{F}$, $\mathcal{SG} : M_\mathbf{F} \rightarrow \mathbb{P}(S)$ |

For every master actor $a_m \in M_\mathbf{F}$, there exists one unique time-stamp actor as defined in Definition 5.

**Definition 5** (Time-stamp mapping)**.** *Time-stamp mapping is a bijective function, denoted as* $\mathcal{T} : M_\mathbf{F} \leftrightarrow T_\mathbf{F}$*. The corresponding master actor of* $a_t \in T_\mathbf{F}$ *is given as* $\mathcal{T}^{-1}(a_t)$*.*

Each time-stamp actor $a_t \in T_\mathbf{F}$ represents a uni-directional data dependency from one scenario to a non-empty set of scenarios. The set of all such data dependencies constitutes the scenario dependency of the FSM-SADF model, as defined in Definition 6.

**Definition 6** (Scenario dependency)**.** *Scenario dependency of* $\mathbf{F}$ *is defined as a function* $\mathcal{D} : T_\mathbf{F} \rightarrow S \times \mathbb{P}(S)$ *such that for any* $a_t \in T_\mathbf{F}, \mathcal{D}(a_t) = (\mathcal{G}(a_m), \mathcal{SG}(a_m))$*, where* $a_m = \mathcal{T}^{-1}(a_t)$*.*

The Scenario dependency $\mathcal{D}$ defines all data dependencies between scenarios that cannot be modeled with existing common initial tokens between scenario graphs. To model these data dependencies, the original FSM-SADF model $\mathbf{F}$ is transformed into a new dependency-aware model, as given in Algorithm 1. The transformation involves two main steps. For every time-stamp actor $a_t \in T_\mathbf{F}$, (1) add the time-stamp actor in the parent scenario graph of the master actor ($\mathcal{G}(a_m)$), and (2) add the same time-stamp actor to all scenario graphs of slave actors of the master actor ($\mathcal{SG}(a_m)$).

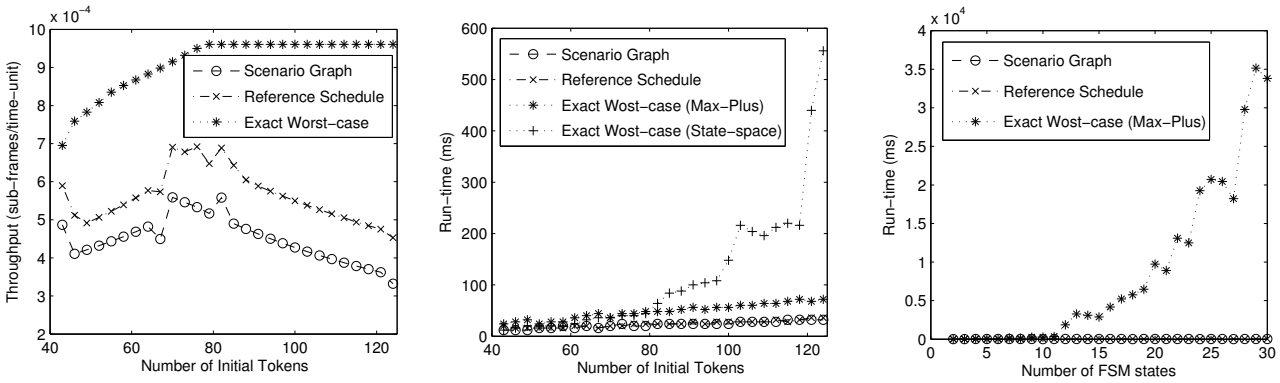
**Algorithm 1** Model scenario dependencies of an FSM-SADF model $\mathbf{F} = (S, \mathbf{f})$ from a given scenario dependency $\mathcal{D}$

```
ModelScenarionDependency(D)
T_F := domain(D) //set of time-stamp actors
for all a_t ∈ T_F do
    a_m := T^{-1}(a_t) //get master actor
    (s, S_s) := (G(a_m), SG(a_m))
    A(s) := A(s) ∪ {a_t} //add time-stamp actor to graph
    c := new channel a_m → a_t where I(c) := 0
    R(srcPort(c)) := 1
    R(dstPort(c)) := repVector(a_m)
    C(s) := C(s) ∪ {c}
    for all s_s ∈ S_s do
        A(s_s) := A(s_s) ∪ {a_t}
        for all a_s ∈ S(a_m) ∧ a_s ∈ A(s_s) do
            c := new channel a_t → a_s where I(c) := 0
            R(srcPort(c)) := repVector(a_s)
            R(dstPort(c)) := 1
            if s_s = s then
                I(c) := repVector(a_s)
            end if
            C(s_s) := C(s_s) ∪ {c}
        end for
    end for
end for
```

Timing analysis of FSM-SADF whose scenario dependencies are not properly modeled, may give erroneous throughput results. This is shown next, in Section V, that discusses the throughput analysis of the LTE dataflow model.

## V. THROUGHPUT ANALYSIS

Throughput requirements of SDRs, such as frame arrival rate, come from standards. We regard SDRs as hard real-time applications, as they must comply with standards and process frames at least at the rate of their throughput requirement. Thus, it is essential to analyze the worst-case throughput (WCT) of a SDR to ensure that the throughput requirement is satisfied in all operating conditions. The focus of this




(a) Comparing conservativeness (b) Comparing run-time (c) Comparing scalability

Fig. 10. Conservativeness, run-time and scalability of various throughput analysis techniques

section is the WCT computation of FSM-SADF models. First we introduce four terminologies for our discussion: *validity*, *conservativeness*, *run-time* and *scalability*.

Throughput of an FSM-SADF is expressed in *number of iterations per time-unit*. The actual WCT of an FSM-SADF is the minimum throughput that may appear in the execution of the model. Any computed WCT is said to be *valid* if and only if it is less than or equal to the actual WCT. *Conservativeness* refers to how close a valid computed WCT is to the actual WCT. To obtain a less conservative (less pessimistic) result, a detailed analysis of the FSM-SADF model has to be carried out. However, this may cost us in terms of *run-time*, which is the length of time the computation takes to run to completion. Another important aspect is how the run-time of a given technique scales with increase in graph size, which we refer to as *scalability*. Run-time and scalability are important properties since design-space exploration (DSE) in scenario-based design flows [8] is a long iterative process that involves large graph sizes.

The throughput analysis of the FSM-SADF model of LTE's baseband processing, shown in Figure 7, is presented in this section. The analysis is carried out using the publicly available dataflow tool, SDF3 [17]. Table II shows computed WCTs according to five different computation techniques that are presented in detail in [10], [15] and [7]. Figure 10 shows the relative conservativeness, run-time and scalability of the scenario-based techniques (methods 2 to 5).

TABLE II
WCT COMPUTATION OF THE FSM-SADF MODEL OF FIGURE 7 WITH AND WITHOUT SCENARIO DEPENDENCY MODELING (SDM)

($\times 10^{-4}$ sub-frames/time-unit)

| Method | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Name | Static SDFG | Scenario graph | Reference schedule | State-space | MaxPlus |
| Without SDM | 2.6 | 9.7 | 10.4 | 14.5 | 14.5 |
| With SDM | 2.6 | 5.2 | 6.6 | 8.9 | 8.9 |

The scenario-based WCT computation techniques (methods 2 to 5) are based on analyzing the timing evolution of the time-stamp vectors of the constituent scenario graphs, as discussed in Section II-B. In addition, the scenario transitions specified by the FSM is also considered by some of them. Our main observations from Table II and Figure 10 are presented next.

### A. Conservativeness

Methods 4 and 5 are based on computing the exact finishing time of iterations (time-stamp vectors). As a result, they give the exact WCT of the FSM-SADF model. Therefore, they can be used as a reference for comparing the relative conservativeness of the throughput computation methods. Methods 2 and 3 have similarity as they are based on approximations on the finishing time of iterations [15]. The approximations are delay-period linear upper bounds on the time-stamp vectors, as discussed in Section II-B. However, as shown in Figure 10(a), method 3 is less conservative than method 2. This is because method 2 does not consider the scenario transitions specified by the FSM [15].

### B. Validity

According to Table II, the actual WCT is $8.9 \times 10^{-4}$ sub-frames/time-unit, since this result is based on the exact WCT computation techniques that also consider scenario dependencies. Table II, as expected, shows that timing analysis without modeling scenario dependencies may give invalid WCT, as the results of methods 2 to 5 are all greater than the actual WCT.

### C. Pessimism of static dataflow

Method 1 is based on a single SDFG that has fixed execution times and port rates, assuming worst-case conditions. Such a static SDFG for the LTE baseband processing is given in Figure 6. The throughput of this SDFG is computed using the state-space exploration technique discussed in [10]. Table II shows that the static SDFG analysis gives a very pessimistic result. The other four scenario-based techniques improve this result by 2 to 3.4 times more (from 2.6 to 5.2 for method 2 upto 8.9 for methods 4 and 5).

### D. Run-time and scalability

Methods 2 and 3 are based on delay-period approximations of the the time-stamp vectors. Hence, they are not significantly affected by neither lengthy time-stamp vectors nor large FSM states. As a result, they have a very low run-time that scale very well with increase in initial tokens and FSM state sizes, as shown in Figure 10(b) and 10(c).

Method 4, on the other hand, is based on a state-space exploration technique that is applied directly on the time-stamp vectors, considering all scenario sequences specified by the FSM. This enables it to give an exact WCT. However, its run-time exponentially grows with increasing initial tokens, as shown in Figure 10(b). As a result, it is also omitted from Figure 10(c).

Method 5 employs a compact representation of initial tokens using *throughput graphs* [7]. The throughput graph is a

directed graph that has $V = |Q| \cdot |\gamma|$ vertices and $|V|^2$ edges, where $|Q|$ is the number of FSM states and $|\gamma|$ is the number of initial tokens in the FSM-SADF. The throughput computation on this graph gives an exact WCT but has an order of complexity $O(V^3)$. However, as shown in Figure 10(c), its run-time is in the order of tens of seconds that makes it practical for real-life applications, such as SDRs.

### E. Summary

This section shows that there is a trade-off in conservativeness, run-time and scalability between different WCT computation techniques of FSM-SADF. Method 2 and 3 trade accuracy for lower run-time that make them useful for long and iterative DSE algorithms. Method 3 can be preferred to method 2, as it is less conservative and has an equivalent run-time and scalability. Method 4 and 5 give the exact WCT, at the cost of run-time. Method 4, however, is barely scalable and could be cumbersome to use it in scenario-based design flows. On other hand, the run-time of method 5 is in the order of tens of seconds that makes it practical for analyzing dynamic SDR applications.

## VI. Related work

Synchronous Dataflow (SDF) [5] is the first dataflow-based model of computation to gain broad acceptance in DSP design tools due to its analyzablity as compared to other directed graph techniques such as computational graphs, petri nets and synchronous languages [3]. With SDF, it is possible to obtain a periodic schedule that can be implemented with bounded buffer size. However, the expressiveness of SDF is limited. and hence it cannot express applications' dynamism without over-allocation. For instance, it is shown in [8] that an SDF-based design-flow may lead to upto 66% in resource over-allocation, as compared to scenario-based techniques.

There are various proposed extensions of SDF to improve its expressiveness. Dynamic dataflow models such as Dynamic dataflow (DDF) [18], Boolean dataflow (BDF) [19],Integer dataflow (IDF) [20] and Core function dataflow (CFDF) [21] are expressively Turing-complete. However formal properties such as deadlock-freedom is an undecidable property for these dataflow models.

[22] and [23] suggest some dynamic dataflow models for software-defined radio (SDR) applications. However, they do not discuss the timing analysis of these dynamic dataflow models for predictable SDR design. The dataflow models proposed in these works include Scalable SDF (SSDF) [24], Parameterized Cyclo-static dataflows (PCSDF) [25], Cyclo-dynamic dataflow (CDDF) [26] and Mixed-mode vector-based dataflow (MMVBDF).These dataflow models, in one way or the other, can model applications' dynamism. However, one dataflow can be more expressive than the other, while still being analyzable. For example, Scenario-aware dataflow (SADF) [6] is more expressive than SSDF, PCSDF and MMVBDFs. SSDF allows integer multiples of token rates for an actor. SADF can model each rate with a separate scenario. PSDF and MMVBD are less expressive than SADF, since they require the parameterized consumption and production rates for a channel to be equal and does not support rates of 0 [6].

FSM-SADF is a version of SADF, where the transitions between scenarios are specified by a finite-state-machine (FSM). It has been shown in [8] that FSM-SADF can express dynamism in multimedia codecs. Efficient performance analysis techniques for FSM-SADF graphs are also presented in [15] and [7]. This paper extends these works by showing the applicability of FSM-SADF for SDRs.

## VII. Conclusions

Software-defined radios (SDRs) are real-time streaming applications with throughput requirements. Dataflow modeling of SDRs for timing analysis is challenging due to their dynamically changing data processing workload. In this paper, we address the challenge of dataflow modeling of dynamic SDRs such that their timing behavior can be accurately analyzed to guarantee real-time requirements. The basis of our modeling approach is splitting the dynamic data processing behavior of a SDR into a group of static mode of operations. Each static mode of operation is then modeled by a Synchronous Dataflow (SDF), which we refer to as *scenario*. This work shows the applicability of this approach by modeling Long Term Evolution (LTE), which is a recent standard in cellular communications. Our results show that the worst-case throughput computation by scenario-based analysis is at least two times more accurate than a state-of-the-art SDF analysis technique. Our investigation also shows that existing timing analysis techniques of SDF scenarios have very low run-time that scales very well with increase in graph size. This makes SDF scenarios suitable in practice for modeling and analyzing SDRs as well as similar dynamic applications.

## References


[1] O. Gustafsson *et. al*, "Architectures for cognitive radio testbeds and demonstrators an overview," in *CROWNCOM*, 2010.
[2] F. Jondral, "Software-defined radio: basics and evolution to cognitive radio," *EURASIP Journal on Wireless Comm. and Netw.*, 2005.
[3] S. Sriram and S. S. Bhattacharyya, *Embedded Multiprocessors: Scheduling and Synchronization*, 2009.
[4] C. van Berkel, "Multi-core for Mobile Phones," in *DATE*, 2009.
[5] E. Lee and D. Messerschmitt, "Synchronous dataflow," *IEEE Proceedings*, 1987.
[6] B. Theelen *et. al*, "A scenario-aware data flow model for combined long-run average and worst-case performance analysis," in *MEMOCODE*, 2006.
[7] M. Geilen and S. Stuijk, "Worst-case performance analysis of synchronous dataflow scenarios," in *CODES/ISSS*, 2010.
[8] S. Stuijk *et al.*, "A predictable multiprocessor design flow for streaming applications with dynamic behaviour," in *DSD*, 2010.
[9] D. Martn-Sacristn, "On the way towards fourth-generation mobile: 3gpp lte and lte-advanced," *EURASIP Journal on Wireless Comm. and Netw.*, 2009.
[10] A. Ghamarian *et. al*, "Throughput analysis of synchronous data flow graphs," in *ACSD*, 2006.
[11] S. Stuijk, M. Geilen, and T. Basten, "Throughput-buffering trade-off exploration for cyclo-static and synchronous dataflow graphs," in *Computers, IEEE Transactions on*. USA: IEEE, 2008.
[12] C. Lee *et. al*, "A systematic design space exploration of MPSoC based on synchronous data flow specification," *Journal of Signal Processing Systems, Springer*, 2010.
[13] A. Bonfietti *et. al*, "An efficient and complete approach for throughput-maximal SDF allocation and scheduling on multi-core platforms," in *DATE*, 2010.
[14] F. Baccelli *et. al*, *Synchronization and Linearity: An Algebra for Discrete Event Systems*. John Wiley Sons, 1993.
[15] M. Geilen, "Synchronous dataflow scenarios," *ACM Transactions on Embedded Computing Systems*, 2011.
[16] "3GPP TS 36.211 V8.6.0: Physical Channels and Modulation," 2009.
[17] "SDF3 - Synchronous Dataflow for Free," http://www.es.ele.tue.nl/sdf3/.
[18] J. Buck, "A dynamic dataflow model suitable for efficient mixed hardware and software implementations of dsp applications," in *CODES*, 1994.
[19] E. Lee, "Consistency in dataflow graphs," *IEEE Transactions on Parallel and Distributed Systems*, 1991.
[20] J. Buck, "Static scheduling and code generation from dynamic dataflow graphs with integer-valued control streams," in *Asilomar*, 1994.
[21] W. Plishker *et. al*, "Functional dif for rapid prototyping," in *IEEE/IFIP*, 2008.
[22] H. Berg, C. Brunelli, and U. Lucking, "Analyzing models of computation for software defined radio applications," in *SOC*, 2008.
[23] C. Hsu *et. al*, "A mixed-mode vector-based dataflow approach for modeling and simulating lte physical layer," in *DAC*, 2010.
[24] S. Ritz *et. al*, "High level software synthesis for signal processing systems," in *ASAP*, 1992.
[25] B. Bhattacharya and S. Bhattacharyya, "Parameterized dataflow modeling for dsp systems," *IEEE Transactions on Signal Processing*, 2001.
[26] P. Wauters *et. al*, "Cyclo-dynamic dataflow," *IEEE Transactions on Signal Processing*, 1996.