Abstract

Current multi-functional embedded systems such as smartphones and tablets support multiple 2G/3G/4G radio standards including Long Term Evolution (LTE) and LTE-Advanced. LTE-Advanced is the latest industry standard that improves upon LTE by introducing several feature-rich and complex improvements. Moreover, both LTE and LTE-Advanced have real-time requirements. LTE and LTE-Advanced receivers are typically scheduled on a heterogeneous multi-core processor to satisfy real-time and low-power requirements.

Manual simulation-based real-time analysis of such applications is infeasible. Dataflow can be used for real-time analysis. Static dataflow allows a rich set of analysis techniques to support real-time scheduling, but is too restrictive to accurately and conveniently model the dynamic data-dependent behavior for many practical applications, including LTE-Advanced. Dynamic dataflow allows modeling of such applications, but in general does not support rigorous real-time analysis.

Mode-controlled Dataflow (MCDF) is a restricted form of dynamic dataflow that captures realistic features in such applications and allows rigorous timing analysis. We stepwise refine and develop complete and fine-grained MCDF models of an LTE-Advanced receiver that include two key features: 1) carrier aggregation and 2) enhanced physical downlink control channel processing. We schedule the MCDF models on an industrial platform to benchmark them against (static) Single-rate Dataflow (SRDF) using existing buffer allocation techniques to demonstrate that these models are analyzable and practically applicable. Moreover, we also develop latency analysis techniques for single-rate and mode-controlled dataflow. For our LTE-Advanced receiver, relative to SRDF models, MCDF models offer 1) up to 15% smaller memory consumption, and 2) up to 1.6% smaller LTE-Advanced sub-frame processing latency.

Keywords: LTE, LTE-Advanced, Dataflow modeling.

1. Introduction

Current multi-functional embedded systems such as smartphones and tablets support multiple radio standards including Long Term Evolution (LTE) [1] and LTE-Advanced [2] running simultaneously. LTE-Advanced is the latest industry standard that improves upon LTE by introducing several compute intensive, feature-rich and complex improvements. These applications are deployed on a single platform to enable high-volume markets (low costs). They have real-time requirements, including end-to-end throughput and latency constraints [7]. They are scheduled on a heterogeneous multi-core processor [3] to satisfy these real-time requirements and also for competing battery life (low power).

LTE-Advanced provides higher data rates of up to 1 Gigabits per second (Gbps) [2] than the existing LTE standard. Manual, including simulation-based, real-time analysis of such complex applications is infeasible. Dataflow graphs provide a convenient way to model and analyze such applications [4]. Nodes in a dataflow graph are called actors and they typically model computations. Actors communicate by consuming or producing tokens on edges that connect actors together. Static dataflow offers a popular programming and analysis model for streaming applications [4]. However, static dataflow is too restrictive to model the dynamic data-dependent behavior in many practical applications, including an LTE-Advanced receiver [5]. For an LTE-Advanced receiver, such dependencies follow from the standard. Hiding these dependencies by choosing a coarser, more abstract granularity is possible, but it incurs a loss of precision in the analysis that cannot be afforded due to the stringent timing requirements. Moreover, for a coarser model, the resource allocation combined with the worst-case (hard real-time requirements) behavior often results into resource overallocation [5]. However, resource availability in embedded systems is tight. This necessitates a less-restrictive form of modeling for such systems.

Dynamic dataflow [4], e.g. Boolean dataflow [6], can relatively easily capture dynamic behavior present in an LTE-Advanced receiver. However, the price for this convenience is high: dynamic dataflow cannot be subjected to real-time temporal analysis. Mode-controlled Dataflow (MCDF) [7] is a restricted form of boolean dataflow that supports run-time mode switching with the goal of enabling design-time temporal analysis. In an MCDF graph, a specific subgraph is chosen to execute depending on a mode selected by a so-called mode-controller. In combination with a pre-specified set of so-called mode sequences, rigorous temporal analysis of MCDF models is possible.

In this paper, we address following key questions: can a
realistic LTE-Advanced receiver be modeled accurately as an MCDF graph(s)? Moreover, are the resulting graphs manageable by the analysis tools? The LTE modeling and analysis using mode-controlled dataflow has originally been proposed by us in [8]. In this paper, we extend the mode-controlled dataflow-based modeling and analysis for the LTE-Advanced receiver that includes advanced features such as Enhanced Physical Downlink Control Channel (EPDCCH) processing and carrier aggregation. The modeling of an LTE-Advanced receiver is not only guided by its functional behavior but also by certain features of the hardware platform such as mapping, parallelism, and heterogeneity. Whereas, other hardware features such as timing characteristics do not influence modeling but guide the real-time scheduling analysis stage. We benchmark the resulting MCDF models against (static) Single-rate Dataflow (SRDF) by employing an existing buffer allocation technique provided in [9]. Moreover, we develop latency analysis techniques for single-rate and mode-controlled dataflow models; they are briefly described in Appendix A. For our LTE-Advanced receiver, compared to SRDF models, MCDF models offer 1) up to 15% smaller memory consumption, and 2) up to 1.6% smaller LTE-Advanced sub-frame processing latency.

We introduce an LTE-Advanced receiver and mode-controlled dataflow in Section 2. In Section 3, we list a set of dataflow modeling challenges, mostly related to the dynamic behavior implied by the LTE-Advanced standard. In Section 4 and 5, we address these challenges one by one, and construct complete graph(s) for an LTE-Advanced receiver. In Section 6, we benchmark the resulting MCDF models against static dataflow. The related work is described in Section 7 and the conclusion is provided in Section 8.

2. Preliminaries

We model LTE and LTE-Advanced receivers using Mode-controlled dataflow. In this section, we introduce the relevant features of LTE and LTE-Advanced receivers. We also briefly introduce single-rate and mode-controlled dataflow.

2.1. LTE and LTE-Advanced receivers

LTE provides significantly higher data rates of 100 Mbps (downlink) and 50 Mbps (uplink) compared to the 3G cellular technology. In an LTE receiver (downlink on a user equipment), data is transmitted in terms of radio frames (10 msec in duration) [1] that are divided into 10 sub-frames, each sub-frame being 1 msec long (Figure 1a). Each sub-frame is further divided into two slots, each of 0.5 msec. A slot may contain 6 or 7 LTE symbols, depending on the normal or extended cyclic prefix used.

The transmitted downlink signal in LTE consists of a varying number of sub-carriers in a resource grid (Figure 1b). A sub-carrier has a fixed 15kHz bandwidth; the total number of sub-carriers depends on the system bandwidth which can range from 1.4MHz up to 20MHz. Each location present in the resource grid is called a resource element. To facilitate channel estimation and timing synchronization, LTE employs special reference signals in each physical resource block; LTE uses physical resource block (Figure 1b) consisting 12 sub-carriers for one slot duration as a basic scheduling unit for a user equipment. As shown in Figure 1b, the reference signals shown in gray are used for single antenna case, whereas the additional reference signals shown in black are used for multiple antenna case. LTE consists of three main physical channels: 1) Physical Control Format Indicator Channel (PCFICH) provides the location of the main control channel, 2) Physical Downlink Control Channel (PDCCH) provides the location and more information of the user data in the sub-frame, and 3) Physical Downlink Shared Channel (PDSCH) is the data channel that carries actual user data. This also gives rise to a fixed order in the decoding of the channels: PCFICH → PDCCH → PDSCH.

LTE-Advanced is a major enhancement of the LTE standard [2]. In LTE-Advanced, we consider two major features from document release 10 and 11: 1) Carrier Aggregation and 2) Enhanced Physical Downlink Control Channel (EPDCCH). Carrier aggregation allows aggregating up to 5 carriers of up to 20 MHz bandwidth each to support overall downlink peak rates up to 1 Gbps. Each such carrier is equivalent to a carrier of a single LTE receiver. The EPDCCH (E) channel is an additional control channel that increases the control signaling capacity of LTE-Advanced to accommodate signaling required for multiple carriers. The EPDCCH uses PDSCH resources in a sub-frame for control information transmission. It also changes the channel decoding dependencies: PCFICH → PDCCH → EPDCCH → PDSCH.

An LTE receiver is scheduled on an industrial modem platform [3], it is shown in Figure 2.

Figure 1: LTE (a) Radio frame consisting 10 sub-frames (i.e. 20 slots) (b) Resource grid with a single sub-frame (2 slots)

Figure 2: An industrial modem platform: with general purpose processors and weakly programmable accelerators

All the processing elements are connected through a ring-
bus to facilitate synchronization using control packets. Processing elements include an Embedded Vector Processor (EVP) [3], a general purpose processor such as ARM, and weakly programmable accelerators. Weakly programmable accelerators include demodulator and demapper for LTE symbols, and decoders for LTE control and data channels. Each processing element, upon receiving a control packet addressed to itself, 1) fetches the input data from the shared memory and processes it; 2) writes the output data to the shared memory; 3) creates one or more control packets for the next processing elements and so on. We consider a similar hardware platform for an LTE-Advanced receiver.

2.2. Mode-controlled dataflow

In static dataflow, e.g. (Multi-rate or) Synchronous Dataflow (SDF) [10], for each firing, an actor consumes (and produces) a fixed amount of tokens from its input (and to its output) edges. These amounts are called consumption (and production) rates. For Single-rate Dataflow (SRDF), these rates for each firing of an actor are 1. An example SRDF graph is shown in Figure 3a. The initial state of an edge is specified by initial tokens (or delays), shown as a dot(s) in Figure 3a.

![Figure 3: (a) An SRDF graph with its (b) self-timed schedule showing multiple overlapping iterations](image)

Dataflow graphs run continuously processing a virtually infinite input sequence. Figure 3b shows a self-timed schedule of the SRDF graph illustrated in Figure 3a, assuming equal execution times for S, A and B actors. In a self-timed schedule, an actor fires as soon as it has a sufficient amount of tokens present on each of its inputs. For each actor, the number in a box denotes the iteration number of the actor firing. An iteration of an SRDF graph is a sequence of actor firings where each actor fires only once, i.e., for an actor, its firing count is equal to its iteration number. The execution of a dataflow graph often overlaps multiple successive iterations as shown in Figure 3b.

Mode-controlled Dataflow (MCDF) [7] is a restricted form of Boolean Dataflow (BDF) [6] that supports static temporal analysis. In an MCDF graph, actors do not need to fire in every iteration of its execution. Instead, each iteration of the graph is associated with a mode such that only the pre-defined subgraph associated with that mode executes in that iteration. After all actors in the chosen subgraph have fired, the graph returns to the initial token distribution.

This modal execution of an MCDF graph is achieved using three special types of actors, namely, switch, select and mode-controller (MC). A switch actor fires in every iteration such that, depending on the mode associated with the current iteration, produces a token on only one specific outgoing edge. Likewise, a select actor fires in every iteration such that, depending on the mode associated with the current iteration, consumes a token from only a specific input edge. Lastly, the mode-controller also fires in every iteration and decides the mode of the current iteration. The mode-controller is connected via outgoing edges to each switch and select in the graph such that each token, produced on these edges is annotated with the mode of the current iteration.

A typical MCDF graph consists of one or more regular SRDF actors, one or more switches and selects, and a single MC. An example MCDF graph is shown in Figure 4a, it has three modes: 1, 2 and 3. The execution behavior of the graph for an iteration associated with mode 1 is described as follows. Actor S starts its firing by consuming the initial token from its self-edge. At the end of the firing, it produces a single token on each of its output edges. MC fires by consuming a token from S. At the end of the firing, it produces a single token, referred to as a control token, on each of its output edges. All these control tokens have the same value that indicates the mode value for this iteration; in this case, it is 1. Now, Switch starts its firing by consuming a token from S and a control token from MC. At the end of its firing, it produces a token on the output edge that is associated with mode 1 since this mode value is indicated by the control token. This enables A to fire producing a token on both output edges. Select consumes a control token from MC. Based on the value of the control token (i.e. 1), Select consumes a token from the modal input edge that is associated with mode 1. At the end of its firing, it produces a token on its output edge. Z consumes this token and finishes its firing. In this iteration, a subgraph comprising S, MC, Switch, A, Select and Z is executed.

![Figure 4: (a) An MCDF graph (b) Tunnel (inter-modal communication): 1 → 3](image)

For any given MCDF graph, there is a fixed number of modes, M. The MC is a special actor in MCDF; at each firing, the MC produces a single control token, with a mode value, on all its output edges. Note that all data-dependent actors such as switch and select actors are connected to MC through delay-less edges; these edges are termed as control edges and are shown in Figure 4a as dotted edges. A switch actor has one control input edge, one data input edge, and M modal output edges. Each modal output edge is associated with a mode. The actor fires by consuming a control token from the control input edge and a token from the data input edge. At the end of the firing, it produces a token on the modal output associated with the mode indicated by the control token. A select actor has one control input edge, M modal input edges and one data output edge. The actor fires by consuming a control token from the control input edge, and the token from the modal input edge that is associated
with the mode which is indicated by the control token. At the end of the firing, it produces a token on the data output. Note that a modal port of a switch/select can be unconnected, in that case, it fires without producing/consuming any token.

A tunnel actor \( T \) is a shorthand used to model inter-modal communication. It is comprised of MCDF actors as shown in Figure 4b. The tunnel \( T \) passes a token from the \( k^{th} \) (for \( k > 0 \)) iteration where mode 1 is selected to the \( l^{th} \) (such that \( l > k \)) iteration where mode 3 is selected first time after the \( k^{th} \) iteration. A tunnel actor can have more than one delay present on the edge from select to switch. For instance, let us assume the tunnel shown in Figure 4b with 2 delays instead of 1. In this case, the tunnel will pass a token from the \( k^{th} \) iteration where mode 1 is selected to the \( l^{th} \) iteration where mode 3 is selected the 2nd time after the \( k^{th} \) iteration. If a tunnel has more than one delay, we denote such tunnel using the number of delays present inside itself; for instance, a tunnel having 6 delays is denoted as \( T_{6} \).

An actor can be connected, directly or through one or more actors, to a modal edge of a switch or select. In this case, such an actor fires only in those iterations where the mode associated with the modal edge is chosen. We term such actors as modal actors. Contrarily, an actor can be connected, directly or through one or more actors, to an amodal (i.e. not associated with any mode) edge of a switch or select. In this case, it will fire in every iteration. We term such actor as an amodal actor. In Figure 4a, actors \( A, B \) and \( C \) are modal actors; while actors \( S \) and \( Z \) are amodal actors.

An MCDF graph executes in an iterative manner such that each iteration executes in a specific mode. A mode sequence is an order of modes, chosen by the MC, that describes the execution of a graph. Each element in a mode sequence uniquely indicates the mode of execution for a particular iteration. For a mode sequence \( c, c(k) \) denotes the \( k^{th} \) element in the sequence, that is the mode of iteration \( k \) in the execution of an MCDF graph. We specify mode sequences using regular expression. For instance \( c = 2 \cdot 1 \cdot 3 \) is a mode sequence of 3 iterations such that mode of each iteration is expressed as \( c(1) = 1, c(2) = 2 \) and \( c(3) = 3 \). We say that a mode sequence is periodic if it can be expressed as a Kleene star over a smaller mode sequence. For instance a mode sequence \( c = 1 \cdot 2 \cdot 1 \cdot 2 \) can be expressed as an instance of \( (1 \cdot 2)^{*} \). Note that the Kleene star over a mode sequence defines a set of periodic mode sequences. We say that a mode sequence is quasi-periodic if it can be expressed as a Kleene star over a choice of smaller mode sequences. For instance the sequence \( c = 1 \cdot 2 \cdot 1 \cdot 3 \) can be expressed as an instance of \( (1 \cdot 2) \cdot 1 \cdot 3 \). Note that the Kleene star over a choice of mode sequences defines a set of Quasi-periodic Mode Sequences (QMS). Also, observe that the Kleene star is a super-set of multiple periodic mode sequences. In this work, LTE and LTE-Advanced receivers are modeled using the mode sequences of equal length.

### 3. Problem description

We describe dataflow modeling challenges for LTE and LTE-Advanced receivers separately. We first list the challenges and then describe each challenge separately for both LTE and LTE-Advanced receivers.

<table>
<thead>
<tr>
<th>Model</th>
<th>Challenges</th>
</tr>
</thead>
<tbody>
<tr>
<td>LTE receiver</td>
<td>Variable sub-frame formats</td>
</tr>
<tr>
<td>LTE receiver</td>
<td>Channel Estimator (ChEst) Behavior</td>
</tr>
<tr>
<td>LTE receiver</td>
<td>Channel estimation and decoding stage dependency</td>
</tr>
<tr>
<td>LTE receiver</td>
<td>Sub-frame symbol assembly for decoders</td>
</tr>
<tr>
<td>LTE receiver</td>
<td>Varying resource block allocation</td>
</tr>
<tr>
<td>LTE-Advanced receiver</td>
<td>Varying resource block allocation</td>
</tr>
<tr>
<td>LTE-Advanced receiver</td>
<td>EPDCCH channel processing</td>
</tr>
<tr>
<td>LTE-Advanced receiver</td>
<td>Carrier aggregation</td>
</tr>
</tbody>
</table>

Table 1: Dataflow modeling challenges

Table 1 shows the challenges involved in modeling LTE and LTE-Advanced receivers. The resulting models that capture the mentioned challenges in Table 1 must respect the following constraints: 1) each individual actor can be assigned to an individual, possibly special purpose, processor, and 2) the running times of the MCDF-based analysis tools for such graphs must be affordable. Each challenge present in Table 1 is explained in one paragraph below.

#### 3.1. LTE receiver

LTE defines multiple physical channels (Section 2.1) whose mapping onto symbols vary per sub-frame. This variable mapping cannot be accurately modeled in static dataflow. This also gives rise to other challenges that are described in this section.

In each sub-frame, a channel estimator consumes the LTE symbols that have reference signals in it. The locations of reference signals in every sub-frame are fixed (see Figure 1b). This behavior can be conveniently modeled by static dataflow such as Cyco-static Dataflow (CSDF, [6]). The variation in cyclic prefixes (Figure 1a) changes the number of LTE symbols across sub-frames. This behavior cannot be modeled by CSDF conveniently. MCDF provides an efficient way to model the variation in cyclic prefixes.

A channel decoder requires a corresponding channel estimate to decode the current symbol. The channel estimator’s behavior delays the decoding of a symbol that creates a distance (referred to as “decoding distance”) between the channel estimation and the decoding stage. The variable sub-frame formats lead to different decoding distances for different channel symbols. Consequently, static dataflow cannot model these varying decoding distances conveniently.

Each channel decoder collects (i.e., consumes) its symbols present in a sub-frame to enable decoding. Since a channel occupies a varying number of symbols per sub-frame, its decoder will have a varying input consumption rate across sub-frames. Static dataflow cannot model such behavior accurately.

Static-order scheduling enables a fixed order of execution among the actors that are mapped on the same processing element. In an LTE receiver, because of the variable sub-frame formats, some actors cannot be scheduled in static-order as the order and the number of actors scheduled on a single processing element may vary across sub-frames. Consequently, a static-
order schedule cannot be applied to the actors scheduled on the same processing element.

3.2. LTE-Advanced receiver

Similar to the existing physical channels in LTE, the mapping of the EPDCCH (Section 2.1) onto symbols changes across subframes. This cannot be modeled accurately in static dataflow. Moreover, the EPDCCH channel adds another channel processing dependency: PCFICH $\rightarrow$ PDCCH $\rightarrow$ EPDCCH $\rightarrow$ PDSCH. This changes the fundamental channel processing dependencies in LTE-Advanced and postpones the decoding of the PDSCH data channel.

Carrier aggregation in LTE-Advanced enables multiple single-LTE carriers to run in parallel. In static dataflow (such as SRDF), the complexity in modeling these carriers together increases significantly. Moreover, the decoding of every physical channel is sequential among the carriers on our platform; therefore, the static dataflow-based modeling further postpones decoding of the EPDCCH channel. This cannot be modeled accurately in static dataflow. Instead, static dataflow models a single-LTE carrier to run in parallel. In static dataflow, the complexity in modeling these carriers increases significantly.

4. LTE receiver modeling

In this section, we address the LTE receiver modeling challenges shown in Table 1 in a step-wise and incremental manner.

4.1. Variable sub-frame formats

On the receiver side of LTE, different (control and data) types of information are mapped onto different physical channels (a subset of the resource elements in the resource grid) in a sub-frame. Recall Section 2.1, we consider three physical channels: 1) PCFICH, 2) PDCCH and 3) PDSCH. The PCFICH ($C_1$ channel) is a control channel that carries a control format indicator message which contains information about the structure and size of PDCCH. It is mapped on the $1^{st}$ symbol in a sub-frame. A PDCCH ($C_2$ channel) carries a downlink control information message which includes resource assignments and other control information for one or more user equipments. It occupies up to the $3^{rd}$ symbol in a sub-frame. The PDSCH ($D$ channel) is the main shared (among all users) data channel which carries all the user data. Depending on the $C_2$ channel mapping, $D$ channel occupies the remaining symbols in a sub-frame from the $2^{nd}$ or $3^{rd}$ or $4^{th}$ to the last ($14^{th}$) symbol in a sub-frame. Consequently, $C_2$ and $D$ channel occupy a varying number of symbols depending on the current sub-frame format indicator message.

The variable sub-frame format behavior cannot be modeled accurately in static dataflow. Instead, static dataflow models a case where a sub-frame has 3 $C$ and 13 $D$ symbols. Consequently, the SRDF-based modeling captures all the sub-frame formats together. This is because any sub-frame can have at most 3 $C$ or 13 $D$ channel symbols. This case does not exist in practice but in static dataflow it assumes this case for every sub-frame, leading to a pessimistic schedule [5].

MCDF, however, can model the variation in the number of $C$ and $D$ symbols in a sub-frame accurately, as shown in Figure 5. To reduce the graph complexity, we do not show the (dotted) control edges. We devise 3 modes i.e. $C_1$, $C_2$ and $D$ to process the respective channels. The modal actors connected to the modal edges of $W_{DEC}$ process the respective channel symbols. $SRC$ stands for the source, it receives an LTE symbol over antennas. Then the signal is demodulated by the OFDM (Orthogonal Frequency Division Multiplex) demodulator of the two receiver antennas ($DMOD_{R1}$ and $DMOD_{R2}$). They are mapped on a single accelerator on the hardware platform (Figure 2). For the actors mapped on the same processing element, such as for $DMOD_{R1}$ and $DMOD_{R2}$, we use static-order scheduling; it is enforced by connecting the actors in a cyclic fashion with an initial token on the input of the first actor in the cycle. This way, a static-order guarantees that only one of the actors can fire at a time. ChEst computes channel estimates. $MIMO$ computes the combined response for the two antennas. They are mapped on $EVP$ and hence a static-order is enforced between them. OFDM demapper ($DMAP$) demaps the symbols to softbits with the help of $MIMO$. $MC$ selects a mode based on the current symbol’s channel to forward the symbol data to the appropriate decoder. $DCID$ is mapped on $ARM$ and it extracts $D$ channel information from the $C_2$ channel. $MAC$ is a higher layer interface (a synthetic sink actor). All the other actors are mapped on separate accelerators.

Figure 5: 3 Channels $\rightarrow$ 3 modes: $C_1$, $C_2$ and $D$ modes

Every single execution of the graph (or firing of $MC$) corresponds to processing a single symbol in a sub-frame; hence to process an entire sub-frame, the graph will execute 14 times i.e. with a mode sequence of length 14. E.g. a mode sequence $(C_1 \cdot C_2 \cdot D^{11})$ represents the sub-frame having 1 $C_1$, 2 $C_2$ and 11 $D$ channel symbols. For any sub-frame, $MC$ is fired with the mode $C_1$ for the first symbol. After decoding the $C_1$ channel, $MC$ determines the mode sequence for the next 13 symbols/iterations for that sub-frame. Therefore, it does not need input from the other modes.

In SRDF, we have to assume for every sub-frame that each channel occupies the maximum number of symbols that are allotted to that channel. This creates extraneous firings for some actors that can delay the processing of a sub-frame. Contrarily, the MCDF-based model does not create extraneous firings, and hence, can provide more accurate timing behavior.

4.2. Channel Estimator Behavior

LTE uses special reference signals in each physical resource block as shown in Figure 1b to facilitate channel estimation and timing synchronization [1]. A Channel Estimator (ChEst) consumes only those symbols that have reference signals.
MCDF can model this behavior by dividing the consumption and non-consumption of reference signals by ChEst into different modes. Now, the ChEst is modeled using MCDF constructs as shown in Figure 6. We devise two new modes named C_{2R} and D_R to handle C_2 and D channel symbols having reference signals respectively. The R in the subscript of these modes denote the scenario where the symbol containing reference signals is used to compute the channel estimates to decode the respective channel symbol. Often a modal actor fires for multiple modes, for instance, ChEst_{2R} fires for C_1, C_{2R} and D_R modes. We merge the modes together in such a case to reduce graph complexity, for instance, the merging is denoted as C_1/C_{2R}/D_R. Now, ChEst_{2R} is associated with C_1, C_{2R} and D_R modes; and ChEst is associated with C_2 and D modes. If the current symbol contains reference signals then depending on the channel mapping, an appropriate mode associated to ChEst_{2R} is selected. Otherwise, an appropriate mode associated to ChEst is selected. The tunnel (T) allows communication between any mode pair (estimate sharing).

Figure 6: Presence or absence of reference signals → 2 modes

Since the selection of a mode occurs at switch and select actors, we assign a mode set for each modal edge of a switch and select. Mode sets of any two edges of a switch (and select) are disjoint. Union of mode sets of all the edges of a switch (and select) is the mode set of the input graph. All input and output edges of a modal actor are associated with the same mode set. A conversion of such a multi-modal MCDF graph into the canonical MCDF graph is trivial. For each multi-modal actor i.e. modal actors with multiple modes, its copies are created for each mode in the canonical graph. If the multi-modal actor has a self-cycle with an initial token, then that token will be modeled by a tunnel which will connect all the copies of the multi-modal actor together.

The ChEst in Figure 5 will be replaced by the MCDF constructs shown in Figure 6. The channel estimator takes different times to process symbols with and without reference signal in it. The MCDF-based modeling allows ChEst to have such behavior where it has different execution times based on its mode. Moreover, we can also model the multiple cyclic prefix lengths using MCDF, which is not possible using SRDF or CSDF. Note that in this paper, we limit our analysis to normal cyclic prefix length where a sub-frame has 14 symbols.

4.3. Channel estimation and decoding stage dependency

The reference signals are only transmitted for a few resource elements in the LTE resource grid (see Figure 1b). Hence, an interpolation needs to be performed along frequency and then along time direction to obtain channel estimates for every resource element. This forces ChEst to run 6 symbols ahead of the control (C) and data (D) decoding stages to decode the current symbol [5]. Moreover, a fast time filtering in ChEst allows decoding the C channel symbols (4 symbols ahead) faster than the D (6 symbols ahead) channel symbols. For each sub-frame, these decoding distances (i.e. 4 and 6 symbols distances for C and D channel symbol decoding respectively) need to be followed. Since the number of C and D channel symbols vary per sub-frame, these decoding distances have to be adjusted accordingly. However, as discussed in Section 4.1, static dataflow cannot model the varying number of C and D channel symbols per sub-frame accurately. Therefore, in static dataflow, these dependencies have to be forced for all possible C and D channel symbols, which may result in overestimation of resources [9].

We solve the issue of channel estimation and decoding dependency, as shown in Figure 7, by introducing 1) a new mode, called Drop (φ) and 2) six initial tokens on the (DMOD_R1, SW_{R1}) and (DMOD_R2, SW_{R2}) edges, and two initial tokens on (BRD, SW_{C}). BRD is a special actor that 1) broadcasts a token from its input to its outputs, and 2) does not consume time for its execution. This also forces DMAP and MIMO to become modal actors. Note that Drop mode does not have any associated modal actor in the graph. The newly added initial tokens i.e. delays are discarded using Drop mode that enables the channel estimation stage to run 4 and 6 symbols (tokens) ahead of the control and decoding stages respectively.

Figure 7: Control and data dependency → Drop mode (φ)

Initially, the first six consecutive firings of MC, fires with Drop mode. This only happens once at the start of the graph execution for the first six symbols of the first sub-frame, and hence, this behavior is termed as initialization behavior. For our model, the (finite and static) mode sequence consisting six drop modes models the initialization behavior. In this behavior,
$SW_{R1}$ and $SW_{R2}$ discard six initial tokens through drop mode. $SW_{D}$ discards the first 6 channel estimates, and $SW_{C}$ discards the two initial tokens along with the first 4 channel estimates. This ensures that for the $n^{th}$ symbol from $DMOD$ of both the receivers, $DMAP$ for control and data channels will receive the $(n + 4)^{th}$ and the $(n + 6)^{th}$ channel estimates respectively. These decoding distances are maintained for all possible sub-frame formats. After the first six drop modes, depending on the current sub-frame type, appropriate modes are selected to process the $C$ and $D$ channels.

These decoding distances imply that from the second sub-frame onwards, the channel estimates of the first six symbols of the current sub-frame are required to decode the last six $D$ channel symbols of the previous sub-frame. Consequently, the $D$ channel of the previous sub-frame can be decoded completely when the channel estimate of the $6^{th}$ symbol of the current sub-frame becomes available. After this, the decoding of the $C$ and $D$ channels of the current sub-frame can start. Now, the processing of the MCDF graph for a sub-frame having 1 $C_1$, 2 $C_2$ and 11 $D$ channel symbols can be modeled by the mode sequence $(D_2^2 \cdot D_2^2 \cdot D_2^2 \cdot D_1 \cdot C_1 \cdot C_2 \cdot 2^{17} \cdot D_2^2 \cdot D_2^2 \cdot D_2^2)$. Note that the first 6 ($D$ channel related) modes in the mode sequence denote the decoding of the last 6 $D$ channel symbols of the previous sub-frame; the decoding of the current sub-frame starts from mode $C_1$, and will continue until the channel estimate of the $6^{th}$ symbol of the next sub-frame becomes available. Note that if the current symbol contains reference signals then depending on the sub-frame type an appropriate mode having subscript “R” is selected.

The initialization behavior using drop modes provides a more convenient way of modeling the channel estimation and decoding stage dependency. Actor $MIMO$ does not fire during the initialization behavior; consequently, the channel estimates produced by $ChEst$ are discarded. This breaks the static-order present between $ChEst$ and $MIMO$ actors during the initialization behavior. However, this static-order is kept during the processing of different sub-frame formats. This implies that MCDF allows flexibility in enforcing static-order compared to static dataflow. Moreover, the use of drop mode reduces the extraneous actor firings which will reduce the delay in the sub-frame processing. Moreover, it can potentially reduce the number of token productions on some edges. This, in turn, will reduce the memory consumption since these tokens occupy memory space on an underlying hardware platform.

### 4.4. Variation in resource block allocation

The number of resource blocks allocated to a user equipment vary (from 6 to 100) across sub-frames depending on the channel conditions. These, in turn, vary execution time, token sizes and energy requirements of $DDEC$. Static dataflow cannot model this variation accurately. It can be modeled using MCDF. However, the number of modes (around 94) needed to model this variation make the graph complex. Moreover, these 94 modes form Cartesian product with the existing modes which further explodes the graph complexity, making it hard to take advantage of these fine-grained modeling. Moreover, modeling the variation in resource block does not offer any advantage for several aspects e.g. buffer allocation and voltage-frequency scaling. Therefore, currently, we do not model the varying resource block allocation.

### 4.5. Sub-frame symbol assembly for decoders

For every channel, all the demapped symbols are needed to be assembled to enable decoding. In static dataflow (Section 4.1), each channel decoder consumes the maximum number of symbols for that channel across all sub-frame formats, resulting in a pessimistic analysis. We solve this problem by introducing 1) several new modes that assemble demapped symbols, and 2) using a special type of tunnel called accumulator. The MCDF model that captures the sub-frame symbol assembly is shown in Figure 8. Since $C_2$ and $D$ channels need assembling of one or more demapped symbols (except $C_1$ channel), they need extra modes, namely $C_{12L}$, $C_{2L}$, and $D_L$. Each of these modes is selected to decode the last symbol occupied by its respective channel in that sub-frame. Mode $C_{12L}$ handles a scenario where only the first symbol in a sub-frame is occupied by $C_1$ and $C_2$ channels. Note that for simplicity, we do not show $C_{1DEC}$ actor and its connection to $MC$ for $C_{12L}$ mode. Also, we do not show unconnected ports of switches and selects to reduce the graph complexity.

![Figure 8: Channel symbol assembly](Image 315x242 to 549x472)

For the last symbol in a sub-frame, $DDEC$ is fired by selecting mode $D_L$. Except the last symbol, all the demapped data channel symbols in a sub-frame (at the most 12) are produced on the multi-modal edge $D/D_R$ of $SW_{Dec}$. Therefore, we use an accumulator to assemble all these 12 symbols from $D/D_R$ output of $SW_{Dec}$ and feed it to $DDec$. It is denoted as $A$ in a rectangular outline as tunnel actors in Figure 8. The accumulator [13] accumulates the tokens as many times as its input modes (i.e. $D/D_R$) appear in a mode sequence. The maximum number of symbols that can be accumulated using an accumulator can be computed for a given mode sequence using simulation.
Moreover, we add tunnels to pass demapped symbols from C channel modes to C_l.Dec. Also, we add the tunnel from DCID to DDEC that passes the information about the data channel in a sub-frame to D DEC.

Static dataflow cannot model the variable consumption rates of actors accurately. Accumulators provide an efficient way to model varying consumption rates of actors. This provides a different modeling approach that simplifies the modeling process and the resulting model.

4.6. Quasi-static-order scheduling

On our hardware platform (Figure 2), we employ a static-order scheduling among actors that are mapped on the same processing element. However, in MCDF, actors scheduled on the same processing element may have a varying order of firing per sub-frame; for instance, depending on the current sub-frame format, DMAP for C channel may fire 1, 2 or 3 times, which cannot be modeled using static-order in both static and mode-controlled dataflow. In static dataflow, DMAP is forced to fire 3 times to enable static-order schedule, which produces extraneous actor firings.

We use quasi-static-order scheduling to enable this varying static-order in MCDF. In such scheduling, a static-order among actors remains fixed for a given mode sequence (i.e. sub-frame), and it changes across mode sequences. The LTE receiver model with the quasi-static order is shown in Figure 9. This is achieved by using a pair of switch and select connected through an amodal edge with a single initial token along the modal edges that run between the switch and select pair. Likewise, in general, we can use a pair of switch and select connected through an amodal edge with a single initial token to enable quasi-static-order scheduling among actors mapped on the same processing element.

5. LTE-Advanced receiver modeling

LTE-Advanced is a major enhancement of the LTE standard [2]. In our modeling, we extend the LTE-receiver model with two major features (Recall Section 2.1) from LTE-Advanced document release 10 and 11: 1) Enhanced Physical Downlink Control Channel (EPDCH). 2) Carrier aggregation. The modeling of both the features is described below.

5.1. EPDCH downlink channel

An EPDCH (E) channel increases the control signaling capacity of LTE-Advanced by using the D channel resources for control information transmission i.e. it is also mapped on the last 11 to 13 symbols of a sub-frame. We consider a case where both D and E channels share the same number of symbols in a sub-frame. The E channel needs to be decoded as soon as possible to decode the D channel. This functional dependency forces the following order for channel decoding: C → E → D. Unlike C and D channels (Section 4.3), it is assumed that the decoding of the current symbol for E channel depends on the channel estimates of the same symbol i.e. the decoding distance for the E channel is 0. This not only changes the fundamental data dependencies among the channels but also the modeling of the decoding distances that are described in Section 4.3. The number of E channel symbols also vary per sub-frame. Consequently, the modeling of the E channel in static dataflow will yield pessimistic schedules.

The MCDF model shown in Figure 10 models the EPDCH channel processing. Actors shown in the yellow color process the E channel. In our hardware platform, E DEC is mapped on a separate accelerator. Note that we denote DMODE_R1 and DMODE_R2 using a single actor DMOD for simplicity. We model E channel processing more accurately, by 1) introducing (similar to D channel) three modes for E channel: E, E_R and E_l; 2) making SRC and DMOD actors modal that fire only
once for each symbol that is shared by E and D channels; 3) devising modes $E_{NS}$ and $D_{NS}$ for E and D channel symbols respectively that reuse the outputs from $DMOD$ and $ChEst$ of the other D and E channel modes (Note that the SR, DMOD and ChEst do not fire for $E_{NS}$ and $D_{NS}$ modes); and 4) adding more tunnels that enable the reuse of the $DMOD$ and $ChEst$ outputs by $C1, C2, C_{12L}, C_{2L}, E_{NS}$ and $D_{NS}$ modes. The SR, DMOD and ChEst actors do not fire for the C channel related modes, except $C_{15}$ mode. Moreover, we devise a skip ($\psi$) mode that is used, in case of multiple mode sequences, to make all the associated mode sequences of equal length. Note that the $\psi$ mode has no associated modal actors.

The execution of the MCDF model that processes a sub-frame having 1 C and 13 E (and D) channel symbols can be modeled by the mode sequence: $(D_R^7 \cdot D^2 \cdot D_R \cdot [C_{12L} \cdot E_{NS} \cdot E \cdot E_R \cdot E \cdot E_L \cdot D_{NS})$. The sub-frame processing by the MCDF model using the mode sequence is described as follows. The D channel of the previous sub-frame can be decoded completely when the channel estimate of the 6th symbol of the current sub-frame becomes available (see Section 4.3). After this, the decoding of the C, E and D channels of the current sub-frame can start. This means that the channel estimates of the first six symbols of the current sub-frame are used to decode 1) the last 6 D channel symbols of the previous sub-frame, and 2) the first 6 symbols that are occupied by C and E channels in the current sub-frame. In MCDF, we enable this reuse using tunnels (see Section 2.2) having 6 delays, denoted as $T6$ in Figure 10.

5.1. For the first six symbols (i.e., iterations), the D channel related modes from $D_R$, $D_L$, and $D_{\psi}$ are selected to decode the last 6 symbols of the previous sub-frame. The SR, DMOD and ChEst of the current sub-frame fire for these first six iterations. The symbols from these six iterations will be reused by the actors of the C and E channel modes of the current sub-frame. Consequently, $C_{12L}$ is fired for the first symbol, and $E_{NS}$ is fired for the next five symbols. For the remaining 8 symbols in the current sub-frame, depending on its type, appropriate E channel related modes (from $E_R$, $E_L$, and $E_{\psi}$) are selected as shown in the mode sequence. After processing the symbols of the current sub-frame for the E channel, these symbols are reused by the D channel. As shown in the mode sequence, the first 7 D channel symbols of the current sub-frame are processed using mode $D_{NS}$ which reuses the symbols from E channel related modes. We enable this reuse using the tunnels having 7 delays, denoted as $T7$ in Figure 10.

From the mode sequence, it can be inferred that the $1^{st}$, $2^{nd}$, ..., $14^{th}$ demodulated symbols of the current sub-frame will be produced on the output of $SL_{DP}$ when the $7^{th}$, $8^{th}$, ..., $20^{th}$ modes in the mode sequence will be fired respectively. The first D channel symbol (which is produced on the output of $SL_{DP}$ in the $8^{th}$ mode) present in the current sub-frame is decoded by the $D_{NS}$ mode (in the $21^{st}$ mode). The 13 delays present on the edge ($BRD, SW_{R12L}$) allow the $D_{NS}$ mode to consume this first D symbol. Likewise, the first 7 D channel symbols are decoded when the last 7 $D_{NS}$ modes from the mode sequence are fired. The remaining 6 D channel symbols of the current sub-frame will be decoded when the channel estimates from the first six symbols of the next sub-frame will become available.

Now, any mode sequence that processes a sub-frame having C, E and D channels has 27 modes. Table 2 provides the mode sequences to process three different sub-frame formats for the LTE-Advanced receiver model shown in Figure 10.

<table>
<thead>
<tr>
<th>Mode-sequences</th>
<th>$D_R^7 \cdot D^2 \cdot D_R \cdot [C_{12L} \cdot E_{NS} \cdot E \cdot E_R \cdot E \cdot E_L \cdot D_{NS})$</th>
<th>$D_R^7 \cdot D^2 \cdot D_R \cdot [C_{12L} \cdot E_{NS} \cdot E \cdot E_R \cdot E \cdot E_L \cdot E_{\psi} \cdot D_{NS})$</th>
<th>$D_R^7 \cdot D^2 \cdot D_R \cdot D_L \cdot [C_{15} \cdot C_{12L} \cdot E_{NS} \cdot E \cdot E_R \cdot E \cdot E_L \cdot \psi \cdot D_{NS})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E_{NS}$</td>
<td>$(D_R^7 \cdot D^2 \cdot D_R \cdot D_L \cdot [C_{15} \cdot C_{12L} \cdot E_{NS} \cdot E \cdot E_R \cdot E \cdot E_L \cdot \psi \cdot D_{NS})$</td>
<td>$(D_R^7 \cdot D^2 \cdot D_R \cdot D_L \cdot [C_{15} \cdot C_{12L} \cdot E_{NS} \cdot E \cdot E_R \cdot E \cdot E_L \cdot \psi \cdot D_{NS})$</td>
<td>$(D_R^7 \cdot D^2 \cdot D_R \cdot D_L \cdot [C_{15} \cdot C_{12L} \cdot E_{NS} \cdot E \cdot E_R \cdot E \cdot E_L \cdot \psi \cdot D_{NS})$</td>
</tr>
</tbody>
</table>

Table 2: Mode sequences for the MCDF model of an LTE-Advanced receiver shown in Figure 10

SRDF cannot model the varying number of C and D channel symbols in a sub-frame accurately (Section 4.1), which can result in pessimistic schedules or sub-frame processing times. The modeling of the E channel in SRDF will produce more spurious actor firings and pessimistic sub-frame processing times. Contrarily, MCDF allows modeling the E channel more accurately.

5.2. Carrier aggregation

Carrier aggregation allows aggregating up to 5 carriers to increase overall transmission bandwidth. Each such carrier has up to 20 MHz bandwidth, and its processing can be modeled by a single LTE receiver graph shown in Figure 10. In an LTE-Advanced receiver, there is always exactly one primary carrier.
and one or more secondary carriers. The maximum and minimum offsets in the arrival of symbols of the secondary carriers relative to the arrival of symbols of the primary carrier are bounded.

Figure 11 shows an MCDF model for carrier aggregation for two carriers; for simplicity and space reason, we show carrier aggregation for two carriers. We enable quasi-static-order scheduling on symbol level among carriers; on each processing element, within a sub-frame, for each symbol, actors of the first carrier are processed then the actors of the second carrier are processed. We term this scheduling of carriers as symbol-based scheduling. An offset actor, shown in Figure 11, is added to the secondary carrier which models the offset in the arrival of symbols of the secondary carrier relative to the primary carrier. For more than two carrier cases, an offset actor needs to be added before the source actor of every secondary carrier.

The MCDF subgraphs of primary and secondary carriers are demarcated by dashed edges in Figure 11. The actors of both the carriers that are mapped on the same processing element follow a quasi-static-order schedule. For instance, for DMAP actors, for each symbol in a sub-frame, the DMAP actors of the primary carrier are fired first; this is represented by the delay present on the input of $SW_{DP,SO}$ of the primary carrier. Once the DMAP of the primary carrier fires, $SL_{DP,SO}$ of the primary carrier produces a token to enable DMAP of the secondary carrier to fire and so on. Similarly, a static-order among DMOD actors of both the carriers is added. For the space reason, we do not show static-order among decoders of the carriers. Similarly, up to 4 secondary carriers can be added together by adding offset actors per secondary carrier, and by adding quasi-static-order among actors of the carriers that are mapped on the same processing element. The mode sequences present in Table 2 model the execution of the MCDF graph shown in Figure 11 for the three different sub-frame formats.

6. Experiments & Results

The LTE-Advanced receiver MCDF model is implemented in Heracles [7], a temporal analysis tool developed at Ericsson. In [7], the principles of MCDF analysis are explained. The SRDF model of an LTE-Advanced receiver is shown in Figure 12. For simplicity, we do not show actors of all the symbols of a sub-frame. This model shows the processing for a single LTE-Advanced carrier. The ODEM actor represents demodulation for two receiver antennas. Actors in blue, yellow and green color process $C$, $E$ and $D$ channels respectively. We use buffer allocation and latency analysis techniques to benchmark the MCDF models against the existing SRDF models of our LTE-Advanced receiver.

6.1. Buffer allocation

Actors communicate by producing and consuming tokens in First-In-First-Out (FIFO) manner on all the edges of a dataflow graph. Each such FIFO edge is mapped as a buffer on memory. Buffer allocation for a dataflow graph involves the computation of the minimum memory required by the buffers while reserving sufficient space for each produced token without overwriting any live tokens and guaranteeing the satisfaction of real-time constraints [17, 16]. In some hardware platforms, for instance, the one shown in Figure 2, do not support back-pressure
mechanism. When such mechanism is not supported, an actor executes as soon as it is enabled producing a token without checking for the availability of space on its output buffer. This may result in buffer overflow, where an actor overwrites a live token. Moreover, since there is no back pressure, buffer allocation needs to consider the variation in the execution times of actors i.e. both the best- and worst-case timing analysis. We compare the buffer sizes for our LTE-Advanced receiver using the two existing techniques: 1) the SRDF-based buffer allocation (SRDF) [16] and 2) the MCDF-based buffer allocation (MCDF) [18]. These two techniques can perform buffer allocation for hardware platforms with and without back-pressure. These are briefly described below.

6.1.1. SRDF-based buffer allocation

A sufficient buffer size for an edge is the maximum number of tokens alive (having overlapping lifetimes) on itself across all iterations. A token lifetime starts and ends when it is produced by its producer actor and consumed by its consumer actor respectively. A sufficient condition for bounded buffer sizes is to have a strictly periodic source (referred as dominant periodic source) that 1) must be the slowest among all the actors and 2) must have delay-less paths to all the actors of the graph [16]. Buffer sizes are computed using simulation. Simulation, in this context, means that we analyze a self-timed schedule (Section 2.2) of a graph and thus obtain the start time of actors. A self-timed schedule of a graph has a property that it reaches a periodic regime, after a finite number of iterations assuming constant execution time for actors across all iterations [7, 16]. Therefore, the simulation is performed until the end of the 2nd periodic iteration; this guarantees to cover all possible lifetime patterns of all the tokens. We use the best-case and worst-case self-timed schedules to obtain the earliest productions (start of a token lifetime) and latest consumptions (end of a token lifetime) of tokens. In the best- and worst-case self-timed schedules, each actor fires with its best- and worst-case execution time respectively in every iteration. Now, lifetime analysis is performed to compute the buffer size for each edge of a graph. For each edge, the total number of tokens whose lifetimes overlap with each other gives the buffer size for that edge. For more details on the SRDF-based buffer allocation, kindly refer to [16].

6.1.2. MCDF-based buffer allocation

In an MCDF graph, for each mode in a mode sequence, the mode-controller selects a (modal) subgraph associated with that mode to execute [7]. Each such modal subgraph is an SRDF graph. For a given mode sequence c, for each mode in c, its modal subgraph is extracted; these subgraphs are then connected in the sequence defined by the mode sequence c such that each inter-iteration dependency (edges with delays) from the MCDF graph is replicated across modal subgraphs. Such a (SRDF) graph is termed as mode sequence graph. A mode sequence graph extracted from the LTE-Advanced receiver MCDF graph shown in Figure 11 using one of the mode sequences from Table 2 will look similar to the LTE-Advanced receiver SRDF graph shown in Figure 12. For a mode sequence c of an MCDF graph G, its mode sequence graph G_c models the execution of G for c.

For an MCDF graph G having two mode sequences c_i and c_j, R = (c_i|c_j)^∗ denotes the set of all possible quasi-periodic mode sequences that consists of zero or more occurrences of c_i and c_j. We capture the execution behavior of the MCDF graph for R conservatively, by allowing the selection of both c_i and c_j simultaneously. Consequently, a new graph is created by merging the mode sequence graphs of c_i and c_j. In such a case, inter-mode sequence edges are added across all mode sequence graphs. The resulting graph is termed as the worst-case graph which captures all possible dependencies across all possible mode sequences. Similarly, the best-case graph is created where only the dependencies among the dominant periodic sources of all the mode sequences are added. These dependencies allow actors to fire as early as possible but never later than its actual best-case start time. These best- and worst-case graphs are SRDF graphs, and hence, the next steps required for buffer allocation are the same as the SRDF-based buffer allocation. There is only one difference: since we model multiple mode sequences running in parallel, an edge from the original MCDF graph may appear in all these mode sequences. In such case, buffer allocation technique chooses the maximum buffer size possible for that edge across these mode sequences as its size. For more details on the MCDF-based buffer allocation, kindly refer to [18].

6.1.3. Buffer allocation comparison: SRDF vs. MCDF

Let us consider a simplified SRDF model of an LTE receiver shown in Figure 13a, where we show the control (C) and data (D) channel processing for a single LTE symbol. In an LTE receiver, a symbol may carry the C or D channel. In SRDF, we must assume the worst-case behavior i.e. a symbol carries both the C and D channels. This leads to the overestimation of memory resources. For instance, in an iteration, a token produced by DMOD will be replicated on the buffers/edges from DMAP to MIMOs of C and D channels, resulting in larger buffer sizes. Similarly, in an iteration, Dec will consume tokens from DMAPs of both C and D channels.
Advanced receiver for multiple carriers. We apply our SRDF- and MCDF-based buffer allocation techniques on these MCDF and SRDF models of our LTE-Advanced receiver respectively. The resulting buffer sizes are shown in Figure 14.

The MCDF models provide up to 15% smaller memory consumption compared to the SRDF models. This is because of a more accurate modeling of the LTE-Advanced receiver by MCDF. In SRDF, we assume the presence of $C$, $E$, and $D$ channels in the first 3 symbols for each sub-frame. This (similar to the example shown in Figure 13a) produces several spurious actor firings and token productions that lead to the larger memory consumption.

6.2. Latency analysis

The start and finish times of actors in a self-timed schedule can be too optimistic and pessimistic respectively in SRDF-based modeling as shown in the previous section. This also affects several other factors, such as latency. Latency is typically the total amount of time from the start to end of a stimulus [15]. The MCDF-based modeling also offers a tighter latency analysis compared to SRDF-based modeling. Let us consider a simplified LTE receiver shown in Figure 13. A latency to decode a symbol is the maximum amount of time from the start time of the $\text{SRC}$ till the finish time of $\text{Dec}$. In the SRDF-based modeling for each LTE symbol, we have to consider the presence of both the $C$ and $D$ channels as shown in Figure 13a. Moreover, on our platform, $\text{DMAP}$ actors of $C$ and $D$ channels are mapped on the same processing element. In such a case, we have to enforce a static-order schedule between them (note, it is not shown in Figure 13a). This leads to a scenario where $\text{Dec}$ will be forced to fire later than its actual finish time, which increases the latency. Contrarily, the MCDF-based modeling allows to model different sub-frame formats using different mode sequences; consequently, $\text{DMAP}$ of only the chosen mode will fire which gives more accurate firing time for $\text{Dec}$. Therefore, MCDF-based modeling provides a tighter latency analysis compared to SRDF-based modeling.

6.2.1. Latency analysis comparison: SRDF vs. MCDF

We have developed SRDF- and MCDF-based latency analysis techniques to compare the SRDF and MCDF models of our LTE-Advanced receiver. These analysis techniques are briefly described in Appendix A. We compute the maximum latency in processing a sub-frame for each combination of carriers from 1 to 5.

The maximum latency is computed from the start of the first symbol source in a sub-frame to the finish of the $D$ $\text{DEC}$ of the same sub-frame. For our LTE-Advanced receiver, the MCDF models provide up to 1.6% smaller maximum latency compared to the SRDF models. In a sub-frame, the variation in the channel mappings occurs for the first three symbols. SRDF models produce spurious actor firings for these three symbols which explains the larger maximum latency compared to the MCDF models. Moreover, as the number of carriers in the SRDF and MCDF models increases, the difference between the maximum latencies provided by these models decreases. This is because,
as we add more carriers, the overall latency increases significantly compared to the reduction in the latency provided by the MCDF models.

Our LTE-Advanced receiver is scheduled on a hardware platform that is similar to the platform shown in Figure 2. We model several resource allocation and scheduling choices in our model. For instance, the decoder actor of each physical (e.g., C, E, and D) channel is mapped on a separate hardware accelerator. Moreover, we enforce quasi-static-order schedule among the actors that are mapped on the same processing element, such as DMAP actors of all the channels. These choices can influence the buffer sizes and latencies. For instance, if we map the decoders of all the channels of all the carriers on the same hardware element then it can increase the lifetimes of the tokens that are to be consumed by these decoders. This is because these decoders will now execute in sequential manner instead of parallel. Consequently, this will also increase the overall sub-frame processing latency. On the other hand, for instance, if we map DMAP actors of different channels on separate hardware accelerators then this will allow DMAP actors of these channels to run in parallel. This can potentially reduce the lifetimes of those tokens that are to be produced and consumed by these DMAP actors, which, in turn, can reduce the buffer sizes. Note that our modeling cannot handle the resource allocation case where an actor needs to be mapped on more than one processing elements.

7. Related Work

The modeling of an LTE receiver using dataflow have been studied earlier [22], [23], [19], [5]. FSM-SADF, MMVB and PSDF based dataflow approaches are employed in [5], [19] and [23] respectively. Table 3, shows different challenges involved in LTE-Advanced receiver modeling and compare other models with ours. Note that Table 3 shows more (fine-grained) modeling challenges than those provided in Table 1. Compared to the approaches provided in [5], [19] and [23], our modeling considers more number of practically relevant features of an LTE receiver. Moreover, we model the two main features of an LTE-Advanced receiver as shown in Table 1. Note that we limit our LTE-Advanced receiver modeling to a single cyclic prefix length in this paper; however, the varying cyclic prefix lengths feature can be modeled using MCDF.

### Table 3: Comparison of LTE-Advanced receiver models

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Separate C and D channels</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Separate C₁ and C₂ channels</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Data decoding dependency</td>
<td>✓</td>
<td>✓</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Control decoding dependency</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Channel estimator’s variable token consumption pattern</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>Variable cyclic prefix lengths</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>✓</td>
</tr>
<tr>
<td>Varying resource block allocation</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>EPDCCH channel processing</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Carrier aggregation</td>
<td>✓</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Several dataflow flavors, such as MMVB [19], PSDF [20], FSM-SADF [5] and SPDF [21], improve over static dataflow in terms of expressiveness. However, the majority of them, except FSM-SADF, are not as expressive as MCDF, MMVB, PSDF, and SPDF require parameterized and equal production/consumption rates for a channel. Moreover, they do not allow mode specific actor firings. For instance, the channel decoders in an LTE-Advanced receiver fire for a subset of modes which is not possible to model with these dataflow flavors. Moreover, the variable sub-frame formats that involve the variable consumption rate for the PDSCH channel decoder (Section 4.1 and 4.5) cannot be modeled by these dataflow flavors. FSM-SADF and MCDF are equivalent in terms of their expressiveness. FSM-SADF is an analysis model whereas MCDF is both a programming and analysis model with known properties and analysis techniques [7].

8. Conclusion

We have developed analyzable, accurate, and realistic LTE-Advanced receiver models in a stepwise manner using Mode-Controlled Dataflow (MCDF). We demonstrate that MCDF is more expressive and is capable of capturing the dynamic behavior of an LTE-Advanced receiver compared to Single-rate Dataflow (SRDF). The MCDF-based modeling offers several advantages over SRDF-based modeling such as the efficient memory resource allocation and tighter latency analysis. The MCDF-based modeling and analysis can also provide an improved energy-aware analysis and scheduling. The resulting MCDF models of an LTE-Advanced receiver are benchmarked against existing SRDF-based models using latency analysis and existing buffer allocation techniques. This shows that the models are manageable by the analysis tool. For our LTE-Advanced receiver, compared to SRDF models, MCDF models provide 1) up to 15% smaller memory consumption and 2) 1.6% smaller maximum latency in the LTE-Advanced sub-frame processing.

Carrier aggregation feature in LTE-Advanced allows an additional degree of freedom in scheduling symbol processing on our hardware platform. In this paper, we considered symbol-based scheduling among carriers of our LTE-Advanced receiver. Currently, we are working on the MCDF-based mod-
eling of two other scheduling policies \(^1\) for our LTE-Advanced reciever: 1) channel-based (within a sub-frame, for each channel, actors of the first carrier are processed then the actors of the second carrier are processed, and so on.), and 2) carrier-based (within a sub-frame, the actors of the first carrier are processed then the actors of the second carrier are processed, and so on.). The ability to model (and analyze) these scheduling policies demonstrate the feasibility and practical importance of our MCDF-based analysis tool.

Appendix A. Latency analysis

In this appendix, we describe our SRDF- and MCDF-based latency analysis techniques. We first provide formal definition of single-rate dataflow and relevant properties, and then we describe our latency analysis techniques.

Appendix A.1. Single-rate dataflow

An SRDF graph, extended with the Best-case Execution Time (BCET) and Worst-case Execution Time (WCET) for actors [16], is denoted by \( G = (V, E, d, \tau, \hat{\tau}) \), where \( V \) is the set of vertices and \( E \) is the set of edges. Valuation \( d: E \rightarrow \mathbb{N}_0 \) gives the number of delays for an edge. Valuations \( \tau, \hat{\tau}: V \rightarrow \mathbb{R}^*_+ \) give the BCET and WCET of an actor respectively. Valuations \( \hat{\tau}, \tau \) are met [7].

An iteration of an SRDF graph is a sequence of actor firings. Let \( \mathcal{A} \) denote a sequence of actor firings. The best- and worst-case self-timed schedules of an SRDF graph execution. For a source \( i \) and sink \( j \), we measure latency as the difference between the finish and start times of the specific firings of sink and source actors respectively:

\[
L(i, k, j, l, \tau) = f(j, l, \tau) - s(i, k, \tau),
\]

where \( L(i, k, j, l, \tau) \) denotes the latency between \( i \) in the \( k^{th} \) iteration and \( j \) in the \( l^{th} \) iteration for execution time \( \tau \). \( f(j, l, \tau) \) denotes the finish time of \( j \) in the \( l^{th} \) iteration with execution time \( \tau \). Note that this definition is adapted from the definition provided in [7], where the latency is computed as the difference between the start times of two specific actor firings.

Typically, real-time streaming applications execute processing potentially infinite sequence of input data. In such cases, we are often interested in cyclic latency analysis. We then can define latency between the \( k^{th} \) iteration of a source actor and the \( (k + n)^{th} \) iteration of a sink actor as:

\[
L(i, k, j, k + n, \tau) = f(j, k + n, \tau) - s(i, k, \tau),
\]

where \( n \) is a fixed iteration distance.

In practice, actors have variable execution times which result in self-timed schedules with variable execution times which makes latency analysis challenging. The maximum latency \( L_{\text{max}}(i, j, k) \) between any iteration \( k \) of source \( i \) and the \( (k + n)^{th} \) iteration of sink \( j \) is given as:

\[
L_{\text{max}}(i, j, k) = \max_{k \geq 0} L(i, k, j, k + n, \tau)
\]

\[
= \max_{k \geq 0} \max_{\tau \geq \hat{\tau}} (f(j, k + n, \tau) - s(i, k, \tau)).
\]

The best- and worst-case self-timed schedules of an SRDF graph provide an upper and lower bound to any valid self-timed schedule of \( G \) [16], and is shown as:

\[
\hat{s}(i, k) \leq s(i, k, \tau) \leq \hat{s}(i, k), \forall i \in V, \forall k \in \mathbb{N}_0, \hat{\tau} \leq \tau \leq \hat{\tau}.
\]

Consequently, by putting the upper and lower bounds from Equation A.7 in Equation A.6, we get:

\[
L_{\text{max}}(i, j, k) = \max_{k \geq 0} \max_{\tau \geq \hat{\tau}} (f(j, k + n, \tau) - s(i, k, \tau))
\]

\[
\leq \max_{k \geq 0} (\hat{f}(j, k + n) - \hat{s}(i, k)).
\]

Equation A.3 denotes the MCR of a worst-case self-timed schedule. In best-case (denoted as \( \hat{s}(i, k) \)) and worst-case (denoted as \( \hat{s}(i, k) \)) self-timed schedules, in each iteration, an actor fires with its best-case and worst-case execution times respectively. Similarly, the MCR \( \mu(G, \hat{\tau}) \) of the best-case self-timed schedule can be obtained by replacing \( \hat{\tau} \) in Equation A.3 by \( \hat{\tau} \). Apart from start times, we use \( \hat{s}(i, k) \) and \( f(i, k) \) to denote the finish times of \( i \) in the \( k^{th} \) iteration in the best- and worst-case self-timed schedules respectively.

Appendix A.2. SRDF-based latency analysis

Latency is typically the total amount of time from the start to end of a stimulus [15]. We consider a self-timed schedule for SRDF graph execution. For a source \( i \) and sink \( j \), we measure latency as the difference between the finish and start times of the specific firings of sink and source actors respectively:

\[
L(i, k, j, l, \tau) = f(j, l, \tau) - s(i, k, \tau),
\]

where \( L(i, k, j, l, \tau) \) denotes the latency between \( i \) in the \( k^{th} \) iteration and \( j \) in the \( l^{th} \) iteration for execution time \( \tau \). \( f(j, l, \tau) \) denotes the finish time of \( j \) in the \( l^{th} \) iteration with execution time \( \tau \). Note that this definition is adapted from the definition provided in [7], where the latency is computed as the difference between the start times of two specific actor firings.

Typically, real-time streaming applications execute processing potentially infinite sequence of input data. In such cases, we are often interested in cyclic latency analysis. We then can define latency between the \( k^{th} \) iteration of a source actor and the \( (k + n)^{th} \) iteration of a sink actor as:

\[
L(i, k, j, k + n, \tau) = f(j, k + n, \tau) - s(i, k, \tau),
\]

where \( n \) is a fixed iteration distance.

In practice, actors have variable execution times which result in self-timed schedules with variable execution times which makes latency analysis challenging. The maximum latency \( L_{\text{max}}(i, j, k) \) between any iteration \( k \) of source \( i \) and the \( (k + n)^{th} \) iteration of sink \( j \) is given as:

\[
L_{\text{max}}(i, j, k) = \max_{k \geq 0} L(i, k, j, k + n, \tau)
\]

\[
= \max_{k \geq 0} \max_{\tau \geq \hat{\tau}} (f(j, k + n, \tau) - s(i, k, \tau)).
\]

The best- and worst-case self-timed schedules of an SRDF graph provide an upper and lower bound to any valid self-timed schedule of \( G \) [16], and is shown as:

\[
\hat{s}(i, k) \leq s(i, k, \tau) \leq \hat{s}(i, k), \forall i \in V, \forall k \in \mathbb{N}_0, \hat{\tau} \leq \tau \leq \hat{\tau}.
\]

Consequently, by putting the upper and lower bounds from Equation A.7 in Equation A.6, we get:

\[
L_{\text{max}}(i, j, k) = \max_{k \geq 0} \max_{\tau \geq \hat{\tau}} (f(j, k + n, \tau) - s(i, k, \tau))
\]

\[
\leq \max_{k \geq 0} (\hat{f}(j, k + n) - \hat{s}(i, k)).
\]

\(^1\)This work will be included in the Author's Ph.D. thesis, to be submitted Fall 2016 at the University of Eindhoven.
Self-timed scheduling with variable execution times makes latency analysis challenging. This is because a graph can have non-converging throughputs i.e. $\mu(G, \hat{\tau}) \neq \mu(G, \tilde{\tau})$. In such cases the maximum latency may not be bounded i.e. the distance between $\hat{s}(i, k)$ and $\hat{f}(j, k + n)$ can be ever increasing.

We consider the latency analysis for dataflow graphs having dominant periodic source. Dominant periodic source is defined as follows:

**Definition 1.** (Dominant Periodic Source) We say that a graph has a dominant periodic source if:

1) It has one or more actors with constant execution times that are connected together to form a cycle having exactly one delay, and this cycle has the largest cycle ratio.

2) There must exist an actor in this cycle with a delay-less path to every other actor in the graph.

An LTE-Advanced receiver has a dominant periodic external source. The source actors $S_{rc1}, S_{rc2}, \ldots S_{rc14}$ shown in Figure 12, model the periodic arrival of symbols, and constitute the dominant periodic source. We call the cycle involving the dominant periodic source the source cycle. Such a graph has $\mu(G, \hat{\tau})$ and $\mu(G, \tilde{\tau})$ equal to the cycle mean $\mu(src)$ of the source cycle. This condition enables the graph to keep up with the rate of the source, which is required for a realistic application. In such a case, a graph has converging throughputs that is enforced by the dominant periodic source $\mu(src)$: $\mu(G, \hat{\tau}) = \mu(G, \tilde{\tau}) = \mu(src), \hat{\tau} \leq \tau \leq \tilde{\tau}$.

A self-timed schedule of a strongly-connected (a directed graph may not be strongly-connected graph. This property is extended for any valid self-timed schedule of such a non-strongly-connected graph $G$ without back-pressure having a dominant periodic source $src$ [16], and is depicted by Equation A.9.

\[ s(i, k + 1, \tau) = s(i, k, \tau) + \mu(G, \tau), k > K(G, \tau). \] (A.9)

$K(G, \tau)$ is the number of iterations in transient phase and can be computed by simulation. The period of this periodic regime is $\mu(G, \tau)$ which denotes the MCR of the self-timed schedule of $G$ with $\tau$.

The property depicted in Equation A.9 for SRDF graphs having a dominant periodic source $src$ implies the following

\[ \hat{s}(i, k + 1) = \hat{s}(i, k) + \mu(src), k > \hat{K}(G). \] (A.10)

\[ \hat{f}(j, k + 1) = \hat{f}(j, k) + \mu(src), k > \hat{K}(G). \] (A.11)

By subtracting Equation A.10 from A.11, we get:

\[ L_p = \hat{f}(j, k + 1) - \hat{s}(i, k + 1) = \hat{f}(j, k) - \hat{s}(i, k), \]

\[ k > \max(\hat{K}(G), \tilde{K}(G)). \] (A.12)

It can easily be inferred from Equation A.12 that $L_p$ is constant for $k > \max(\hat{K}(G), \tilde{K}(G))$. $p$ in the subscript of $L_p$ stands for periodic phase. Now, we can compute the bound $L_p(i, j, n)$ on the latency in the periodic phase as:

\[ L_p(i, j, n) = \hat{f}(j, k + n) - \hat{s}(i, k) = (\hat{f}(j, k) - \hat{s}(i, k)) + n\mu(src) = L_p + n\mu(src), k > \max(\hat{K}(G), \tilde{K}(G)). \] (A.13)

It can easily be inferred from Equation A.13 that $L_p(i, j, n)$ is constant for $k > \max(\hat{K}(G), \tilde{K}(G))$. This means that the maximum latency between $i$ and $j$ with iteration distance $n$ during the periodic phase of $G$ is bounded. Moreover, we can measure the maximum latency during the transient phase by simulating the self-timed execution of the graph $G$ till the execution enters into a periodic phase. We simulate $G$ with best- and worst-case self-timed schedule. The bound $L_t(i, j, n)$ on the maximum latency between $i$ and $j$ during the transient phase is given by Equation A.14:

\[ L_t(i, j, n) = \max_{k \leq \max(\hat{K}(G), \tilde{K}(G))} (\hat{f}(j, k + n) - \hat{s}(i, k)). \] (A.14)

The subscript $t$ in $L_t(i, j, n)$ denotes the transient phase. The bound on the maximum latency between $i$ and $j$ with the iteration distance $n$ is the maximum latency bound across the transient and the periodic phase; it is given as follows:

\[ L_{\text{max}}(i, j, n) = \max(L_t(i, j, n), L_p(i, j, n)). \] (A.15)

From Equation A.15, it is clear that we need to simulate the graph with best- and worst-case self-timed schedules separately to compute the latencies in transient phases and the bound during the periodic phase. We need to simulate the graph only till the end of the $n^{th}$ periodic iteration. We compute the bound on the maximum latency between $i$ and $j$ with the iteration distance $n$

**Appendix A.3. MCDF-based latency analysis**

An MCDF graph is associated with multiple mode sequences (Section 2.2). Similar to the MCDF-based buffer allocation described in Section 6.1.2, for latency analysis, we extract the best- and worst-case graphs from an MCDF graph. These graphs capture all possible best- and worst-case dependencies across all possible mode-sequences. We use these best- (denoted as $G$) and worst-case (denoted as $G$) SRDF graphs to perform latency analysis. Note that we verify dominant periodic source condition only for the best- and worst-case SRDF graphs that are extracted from the MCDF graph. The best- and worst-case SRDF graphs must have the same dominant periodic source $src$ such that $\mu(G, \hat{\tau}) = \mu(G, \tilde{\tau}) = \mu(src), \hat{\tau} \leq \tau \leq \tilde{\tau}$. Moreover, we verify that the throughputs of these graphs are same. This is a sufficient condition to guarantee the bounded maximum latency.
The latency analysis using the $\hat{G}$ and $\hat{G}$ SRDF graphs is similar to the SRDF-based latency analysis. We simulate the best-and worst-case graphs separately to obtain the best-case start times and worst-case finish times of actors. There is only one difference compared to the SRDF-based latency analysis: for given source $i$, sink $j$ and the iteration distance $n$, we compute the maximum latency between the copies of $i$ and $j$ from all possible mode sequences. For instance, we select the earliest $\bar{s}(i,k)$ and latest $\hat{f}(i,k+n)$ among all the mode sequences; the latency bound $L_{\text{max}}(i,j,k,k+n) = \hat{f}(i,k+n) - \bar{s}(i,k)$ gives the bound on the maximum latency between $i$ and $j$ in the $k^{th}$ iteration. The bound on the maximum latency between $i$ and $j$ for the iteration distance $n$ is given by the maximum of $L_{\text{max}}(i,j,k,k+n)$ across transient and periodic phases.

Acknowledgement

This work was funded by the CA104 COBRA-NL project, granted within the European Catrène program, duration 2012 till 2014. In addition, this work was in part carried out in Ericssons DSP Innovation Center, Eindhoven, The Netherlands, duration 2012 till 2014.

References

[23] H. Kee et al., Fpga-based design and implementation of the 3gpp-lte physical layer using parameterized synchronous dataflow techniques, in: ICASSP, 2010.