FP-Scheduling for Mode-controlled Dataflow: A case study

Alok Lele*, Orlando Moreira†, Kees van Berkel†*
*Eindhoven University of Technology, Eindhoven, The Netherlands
†Ericsson B.V., Eindhoven, The Netherlands.

***Abstract*—Dual-Radio Simultaneous Access (DRSA) is an emerging topic in Software Defined Radio (SDR) in which two SDRs are running simultaneously on a shared hardware, typically a heterogeneous Multi-Processor System-on-Chip (MPSoC). Each SDR has a independent hard latency and/or throughput requirement and needs rigorous timing analysis. Moreover, SDRs are often modeled in enriched variants of dataflow to accommodate the growing dynamic execution of SDRs, making it a challenge to perform timing analysis on them.**

This paper considers the preemptive Fixed Priority Scheduling (FPS) of SDRs modeled in *Mode-Controlled Dataflow*. To the best of our knowledge this is the first attempt on static timing analysis of FPS for a (semi-)dynamic variant of synchronous dataflow. We propose a two-phase algorithm to determine the worst-case response time of an actor. We demonstrate our analysis results for a DRSA case study of two 4G-LTE receivers.

I. Introduction

A Software Defined Radio (SDR) is constituted of an iterative pipeline of processes conditioned by complex data dependencies. Dual-Radio Simultaneous Access (DRSA) is a multi-radio modem [10] in which we consider two SDRs (4G-LTE receivers in our case) to run simultaneously on a heterogeneous Multi-Processor System-on-Chip (MPSoC). On one hand the individual SDRs have hard latency and/or throughput requirements. On the other hand the underlying platform needs to conform to strict power consumption and chip area constraints. To balance between the timing requirements and hardware constraints, we need rigorous timing analysis.

SDRs are often modeled using *dataflow graphs* [8], [5]. Conventionally, the nodes (called actors) model computation (ex. tasks), while edges model inter-actor dependencies. The actors communicate across edges by consuming and producing *tokens* from and onto edges connecting them to other actors in the graph. We use an initial placement of tokens on the edges to depict inter-iteration dependencies between actors. We observe that static dataflow proves insufficient in describing the dynamism in the execution of a 4G-LTE receiver [6]. Instead, we use an enriched variant of dataflow called *Mode-Controlled Dataflow* (MCDF) [5]. MCDF is a restricted variant of Boolean Data Flow (BDF) [1] which introduces certain special actors that conditionally consume (or produce) tokens from (or on) specific edges depending on the mode of its execution.

This paper addresses preemptive Fixed Priority Scheduling (FPS) of MCDF graphs assuming a fixed mapping of actors to processors. We focus on our DSRA case study where we have two 4G-LTE receivers running simultaneously on a shared MPSoC. We propose a two-phase FPS analysis approach consisting of an initial coarse-grained analysis and a subsequent fine-grained analysis.

FP-Scheduling: Static vs Mode-controlled dataflow Depending on the consumption and production rates of actors in a graph one can classify a static dataflow graph as either *single-rate* (consumption and production is always 1), *multi-rate* (consumption and production is constant), or *cyclo-static* (consumption and production change in a deterministic iterative manner) [5]. Note that, one can always convert a multi-rate or cyclo-static graph into an equivalent single-rate graph.

Several techniques exist to analyze FPS of static dataflow graphs since all actors execute a fixed (or deterministic) number of times per iteration. One may thus extract an execution model per actor as done in [9], [11] to perform response time analysis. Alternatively one may condition the dataflow graph such that for a particular actor, the *self-timed* execution (firing as soon as input is available on all incoming edges) of the graph thereafter demonstrates the worst-case interference that actor may cause to the execution of a lower priority actor [4].

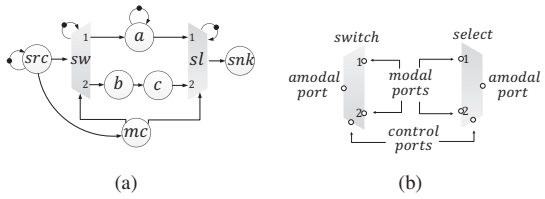


Fig. 1. FP-Scheduling for MCDF: A motivating example

However, in an MCDF graph the firings of actors are dependent on the mode in which an iteration of the graph is executing. For instance, consider the MCDF graph in Figure 1(a). Actors src, snk, mc, sw and sl execute in every mode. However actors sw and sl are special actors called *switch* and *select*, respectively, and will, respectively, produce and consume tokens on specific edges depending on the mode in which they fire. The mode of firing is conveyed to them within the tokens they receive from the special actor mc, known as the *mode controller*. Depending on the mode in which sw and sl fire, either a executes or b and c execute.

The MCDF graph is also associated with one or more *recurring mode sequences* that define the sequence of mode changes. While mode changes within a mode sequence are deterministic, at the end of the sequence the graph may observe a transition to any associated mode sequence or execute in the same mode sequence again. This semi-dynamic execution behavior makes it difficult to infer the maximum interference

978-3-9815370-4-8/DATE15/©2015 EDAA

the execution of actors in an MCDF graph cause to the execution of other actors running on the same processor. For instance, assume that actors a and c execute at a higher priority than an actor x of another graph. It is not trivial to compute the worst-case response time of x as it is unclear as to which actor between a and c (or perhaps both) interferes with the execution of x. On one hand we may exploit the information in a mode sequence to help determine the interference by a and c on the execution of x within that mode-sequence; on the other hand, the transition between mode sequences themselves is non-deterministic making it difficult to compute the interference to x across mode sequence transitions.

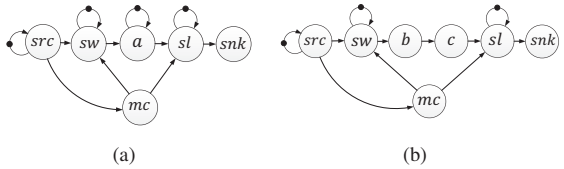
II. Mode-Controlled Dataflow

We now give an overview of Mode-Controlled Dataflow (MCDF), which is a restricted form of Boolean Dataflow (BDF) [1]. MCDF allows for mode switching without compromising the properties that make a graph amenable to temporal analysis, as is in Synchronous Dataflow (SDF) [3]. Scenario-Aware Dataflow (SADF) [7] is a similar dataflow variant, but instead of having special *mode-dependent* actors, they have a set of *scenarios* each with its own graph configuration, and neatly defined transitions from one scenario to another.

A dataflow graph is a directed graph constructed using *actors*, directed *edges*, and an initial placement of *tokens*. The edges connect two actor via incoming and outgoing *ports*[1]. In MCDF, we categorize ports as either *modal* or *amodal*. While amodal ports produce (or consume) tokens in every firing of its actor, modal ports produce (or consume) only when an actor is firing in a specific mode. The canonical form of MCDF [5] defines only two special MCDF actors, namely switch and select, that can have *modal* ports while all other actors have only amodal ports. A *switch* actor has one amodal incoming port and one or more modal outgoing ports. Conversely, a *select* actor has one amodal outgoing port and one or more modal incoming ports. Every switch and select in a graph also has a *control port* via which it connects to a *mode controller* actor. The mode controller produces a token per outgoing port in every iteration which contains information of the mode in which a switch or select actor will fire when consuming it.

An MCDF graph contains one mode controller and an arbitrary number of selects and switches. We say an actor belongs to a particular mode if it is connected to a modal port or another actor of that mode. We do not allow an actor to belong to more than one mode. Switches and selects do not belong to any mode themselves but serve as interfaces between actors belonging to some mode and those that do not belong to any mode. Figure 1(a) depicts an MCDF graph with a mode controller mc, one switch sw, and one select actor sl. Actor a belongs to mode 1, and actors b and c belong to mode 2. Actors src and snk do not belong to any mode.

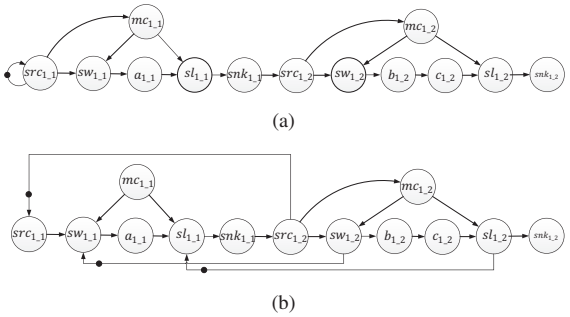
Mode sequence: For simplicity, assume that the consumption and production rates of all actors in Figure 1(a) is 1. Effectively, each iteration of the graph corresponds to a single mode of execution. A *modal sub-graph* represents the subset of the graph that executes in a specific mode as shown in Figure 2. A *mode sequence* defines an order of modes in which the switch and select actors will execute. For a given mode sequence we can *unroll* the graph such that it represents the execution of the graph across the entire mode sequence as shown in Figure 3(a). A graph can be also associated to a *recurring mode sequence* such that after executing an iteration corresponding to the last mode in the sequence, a graph continues to execute starting from the first mode in the mode sequence and iterating through the entire sequence thereafter, as depicted in Figure 3(b). Given a valid mode sequence for an MCDF graph one can always derive an equivalent unrolled static dataflow graph [12]. Typically a graph is associated with one or more recurring mode sequences. We can use a regular expression to represent recurring mode sequences and is equivalent to the finite state machine in SADF [7].



(a) (b)

Fig. 2. (a) Switch and select actor representation. (b) Modal sub-graph for mode 2 of the MCDF graph in Figure 1(a)

Input frames of a 4G-LTE receiver consist of exactly 14 sub-frames. Each sub-frame executes in some particular mode of the LTE-receiver. Thus we have multiple mode-sequences each of length 14 and representing the execution of a complete frame. If $s_1, s_2, ..., s_n$ represents the n possible mode sequences for a frame, we express this recurring set of mode sequences as $(s_1|s_2|...|s_n)^*$. In this paper we only consider MCDF graphs that are associated with recurring mode sequences of same length.



(a)

(b)

Fig. 3. (s) Unroll of graph in Figure 1(a) for mode sequence [1,2]; (b) Unroll of graph in Figure 1(a) for recurring mode sequence [1,2]*

Dominant periodic source: The MCDF graphs in this paper assume a dominant periodic source, i.e. there exists an actor such that there is a token-less path from it to all other actors in the graph and that the critical cycle (cycle with the largest cycle ratio [4]) contains this actor. A dominant periodic source defines an upper and lower bound to the execution of actors in the graph which is periodic. Moreira et al [5] shows that the lower bound on the firing of actors is given by their *self-timed schedule* (actors fire as soon as they have sufficient input) assuming best case execution times. Meanwhile, they show that the upper bound on the firing of actor is given by the *static-periodic schedule* (actors fire according to a strict period) assuming worst-case execution times.

[1]We do not show ports on the actors in a graph to unclutter the Figures. However, we have shown ports in Figure 1(b) for better understanding the special MCDF actors

III. Analyzing FPS of MCDF

We now propose our technique to analyze the response time of a lower priority actor sharing the processor with one or more higher priority actors of MCDF graphs. We first provide an overview of a recent static-analysis technique for FPS of static dataflow graphs [4]. We then extend this analysis to MCDF graphs using a two-phase analysis approach.

In our DRSA case-study we consider that only actors that represent computational units are mapped on to a processor of the underlying platform. Note that the mode controller, and the switch and select actors represent decision points of graph execution but do not represent any computation themselves. Each actor mapped on to a processor is assigned a unique priority on that processor and migration is not permitted.

A. FPS analysis for static dataflow

The response time analysis technique for FPS of static dataflow graphs in [4] characterizes the graph of a higher priority actor such that we: **(a)** Assume worst-case execution time for the high priority actor; **(b)** Assume best-case execution time for all other actors; **(c)** Block the first firing of the high priority actor until the upper bound on its firing time given by its *static periodic schedule*; **(d)** Assume the time of release of the blocked actor to coincide with the critical instant.

The self-timed execution of the graph of a higher priority actor after the above characterization demonstrates the worst-case interference the higher priority actor may cause to the execution of a lower priority actor. By obtaining the worst-case interference from each higher priority actor, we can compute the worst-case response time of the lower priority actor.

B. Challenges in extending FPS analysis to MCDF

The blocking time of a high priority actor proposed in [4] is determined by computing a *static-periodic schedule* [5] of its graph. One cannot directly compute the static-periodic schedule of an MCDF graph, but instead we need to obtain a static-dataflow equivalent of the MCDF graph that preserves the execution dependencies within individual mode-sequences and across all possible transitions between mode sequences.

The other challenge in FPS analysis for MCDF is that the transitions from one mode sequence to another mode sequence are not known *apriori*. Therefore one must explore all possible transitions across mode sequences to infer the maximum interference the execution of actors in an MCDF graph can cause to the execution of a low priority actor of another graph. Furthermore, the execution of the low priority actor may suffer interference from multiple iterations of the MCDF graph, thereby exponentially increasing the number of traces. We now discuss how we tackle each of these challenges.

1) Static periodic schedule for MCDF: To obtain a static-dataflow equivalent of an MCDF graph we first obtain modal sub-graph per mode where each modal sub-graph is a static dataflow graph. We then merge the modal sub-graphs such that any inter-iteration dependency (edge with initial tokens) in a sub-graph must be reflected across all other sub-graphs.

For instance, Figure 4 depicts the merge of the modal subgraphs of Figure 2. Note that, for instance, we have added edges $src1$ to $src2$ (and vice-versa) with one initial token to represent the self-edge on the src actor in the individual modal subgraphs in Figure 2. Similarly, we have added an edge with an initial token from $sw1$ to $sw2$ (and vice-versa) and from $sl1$ to $sl2$ (and vice-versa), respectively.

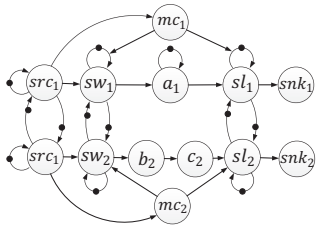


Fig. 4. Merging individual modal sub-graphs

Similarly, we can also merge the unrolled graphs of individual recurring mode sequences associated with an MCDF graph. When merging modal sub-graphs, we do not add dependencies between modal actors since they only exist in a particular sub-graph. However we must account dependencies across unrolled graphs for modal actors as a mode may exist in different mode sequences. For instance, note the dependencies between actors a_{1_1}, a_{2_1}, a_{2_2} for the merging the unrolled graphs for modes sequences $[1,1]^*$ and $[1,2]^*$ in Figure 5.

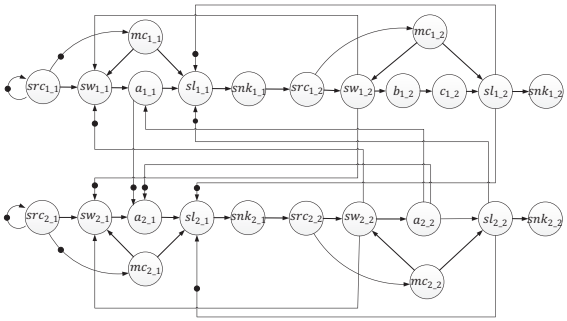


Fig. 5. Merging unrolled graphs of individual mode sequences

We now find a static periodic schedule for the final merged graph such that the start of firing of an actor in the static periodic schedule assuming worst-case execution gives an upper bound to the start of firing of the actor in any valid self-timed execution of the original MCDF graph. This upper-bound can be used as the blocking time for a high priority actor in the response time analysis proposed in [4] to obtain the maximum load of the high priority actor on its processor.

2) Obtaining the worst-case interference trace: Instead of exploring all possible traces to obtain the maximum load, we propose a two step analysis where we first perform a coarse-grained analysis, and then proceed to explore only a few traces to refine our response time results.

Coarse-grained analysis: If $n_{g,v}$ instances of an actor v with worst-case execution time $\hat{\tau}_v$ occur in an unrolled graph g with a *maximum cycle ratio* (MCR) μ_g, then we say that v imposes a load of $n_{g,v} \times \hat{\tau}_v$ per time period μ_g. Note that the MCR of unrolled graph is equal to the length of the mode sequence times the MCR of the original MCDF graph. Similar

to traditional periodic event models, we now attribute a coarse-grained load profile (c_v, T_v) to an actor v such that:

$$c_v = \max_{i \in U}(n_{i,v} \cdot \tau_v), \tag{1}$$

where U is the set of all unrolled graphs of the original MCDF graph of v, and $T_v = \mu_g$ where μ_g is the maximum cycle ratio of an unrolled graph $g \in U$ for any of the mode sequences.

Once we obtain the coarse-grained load profile of every actor in the set of all higher priority actors HP we can compute a pessimistic worst-case response time $\hat{r}(x)$ of a lower priority actor x using the fixed point computation for traditional response time analysis of periodic task systems:

$$\hat{r}(x) = \hat{\tau}_x + \sum_{v \in HP} \lceil \frac{\hat{r}(x)}{T_v} \rceil \cdot c_v \tag{2}$$

Fine-grained analysis: The coarse-grained response time analysis above assumes the entire load c_v executes as a single contiguous block within a period $T_v$and thus assumes the remainder of the period as a contiguous block of potentially idle time that the low priority task may execute in. However, as the period T_v may span across multiple iterations of the original MCDF graph in which instances of the high priority actor v may execute after irregular intervals. Consequently our coarse grained response time response time result may also be non-conservative. To obtain an accurate conservative bound on the worst-case response time, we switch to our fine grained analysis in which we employ the slot-filling based response time analysis proposed in [4]. Note, however, that we choose the blocking times as described in section III-B1. Also note that we only need to perform our fine grained analysis for the last iteration of the high priority graph.

C. Algorithm Complexity

Our coarse-grained analysis employs conventional fixed-point analysis for FP-scheduling which is NP-Hard [2]. Meanwhile, [4] does not address the complexity of its proposed FPS analysis as it is difficult to define sufficiently large simulation time that accommodates the entire execution of the low priority actor. Since DRSA case study considered in this paper we use slot filling only for the last iteration of the high-priority graph, the length of the simulation trace only need to cover one iteration of the graph. Unfortunately, we suspect that FPS of more than two MCDF graphs will significantly increase the complexity of the analysis as we may have to iteratively perform our fine-grained analysis until the result converges.

IV. Experimental results

In this section we present the experiments for our Dual-Radio Simultaneous-Access case study in which we consider two 4G-LTE receivers running in parallel with actors of one receiver having priority than the other. Figure 6 is taken from [6] and shows the MCDF graph of the 4G-LTE receiver. In our case study, while most components of a 4G-LTE receiver have dedicated resources, the $ChEF$ and $MIMO$ actors of both 4G-LTE receivers are mapped to the same resource.

Table I presents the computed response times using only our coarse-grained analysis vis-a-vis our complete response time analysis. We see a significant gain is performing our complete response time analysis.

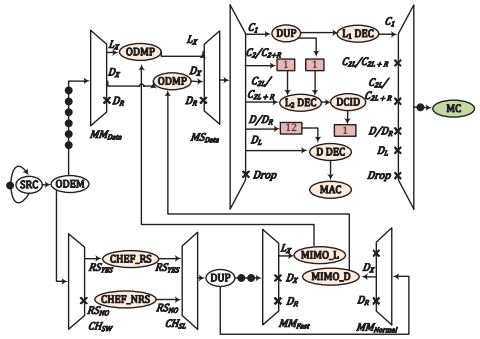


Fig. 6. 4G-LTE receiver

TABLE I. Response time results

Actor	Execution Time	Coarse-grained response times	Fine-grained response times
ChEF_RS	30	810	1080
ChEF2_NRS	15	795	1020
MIMO_L	30	810	1080
MIMO_D	30	810	1080

V. Conclusion

In this paper we consider a Dual-Radio Simultaneous Access (DRSA) case study where the MCDF graphs of two 4G-LTE receivers are scheduled on a heterogeneous MPSoC under preemptive fixed priority scheduling. We show why it is not trivial to determine the worst-case interference an actor of an MCDF graph can cause to the execution of a lower priority actor of another graph. We propose a two-phase response time analysis technique consisting if a coarse-grained analysis followed by fine-grained refinement to obtain a conservative worst-case response time of an actor in our MCDF setting.

References

[1] J.T. Buck, *Static scheduling and code generation from dynamic dataflow graphs with integer-valued control streams*, SSC, 1994.

[2] F. Eisenbrand and T. Rothvoss, *Static-priority real-time scheduling: Response time computation is np-hard*, RTSS, 2008.

[3] E.A. Lee and D.G. Messerschmitt, *Synchronous data flow*, Proceedings IEEE **75** (1987), no. 9, 1235–1245.

[4] Alok Lele et al, *Analyzing preemptive fixed priority scheduling for data flow graphs*, In Proc. of ESTiMedia, 2014.

[5] Orlando Moreira and Henk Corporaal, *Scheduling real-time streaming applications onto an embedded multiprocessor*, Springer, 2014.

[6] H.L. Salunkhe et al, *Mode-controlled dataflow based modeling & analysis of a 4g-lte receiver*, DATE, 2013.

[7] F. Siyoum et al., *Analyzing synchronous dataflow scenarios for dynamic software-defined radio applications*, SoC, 2011.

[8] S. Sriram and S. S. Bhattacharyya, *Embedded multiprocessors: Scheduling and synchronization*, Marcel Dekker, Inc., 2000.

[9] D. Thiele and R. Ernst, *Optimizing performance analysis for synchronous dataflow graphs with shared resources*, DATE, 2012.

[10] K. van Berkel et al, *A Multi-Radio SDR Technology Demonstrator*, SDR Forum Conference, 2009.

[11] Ernesto Wandeler et al, Lothar Thiele, Marcel Verhoef, and Paul Lieverse, *System architecture evaluation using modular performance analysis: A case study*, Int. J. Softw. Tools Technol. Transf. (2006).

[12] Yu Yi, *Energy-aware scheduling for modal radio graphs*, Master's thesis, Eindhoven University, 2013.