Energy-Aware Scheduling for Modal Radio Graphs
Abstract

In most cases radio applications are running on battery-operated devices. Besides real-time requirements, lower power processing is also considered as an essential design objective. Voltage and Frequency Scaling (VFS) has already been proved as an efficient method to reduce energy consumption for real-time streaming applications. However, most previous work is depending on a static data flow model. Dynamic behavior of radio applications can be adequately expressed by Mode-Controlled Data Flow (MCDF). In this thesis, we address VFS policies for MCDF graphs to minimize energy consumption while still meeting the timing constraints.

On application level, both radio applications with or without time overlap between frames are studied. Deadline-constrained VFS is carried out for applications without overlaps between frames and throughput-constrained VFS are proposed for applications that allow overlapping between frames. On platform level, we use two types of platform: local VFS switches for each processor and a single global VFS switch. Both platform types are considered for deadline-constrained and throughput-constrained VFS policies.

Experiments carried out show that VFS policies can significantly reduce energy consumption for both applications with and without time overlaps between frames. The results show that a platform with one global VFS switch offers less usage of slack time. Thus local VFS is more efficient in energy saving. As an example, for the WLAN model, continuous local VFS gives an energy reduction up to 50.89%, while discrete local and discrete global VFD algorithms with 2 frequency levels reduce the energy consumption by at most 49.61% and 49.56%, respectively. Meanwhile all these dynamic VFS policies save more energy for MCDF graphs comparing to static VFS where each processor has a constant frequency while working. The conclusion above is proved again by the experiments for the LTE model. However, the discrete global VFS does not reduce the energy consumption for the LTE model.
# Table of contents

## 1 Introduction
1.1 Background .......................... 1
1.2 Radios ................................. 1
1.3 Multiprocessor Platform ............... 3
1.4 Problem Description .................. 4
1.5 Contribution .......................... 5
1.6 Organization .......................... 6

## 2 Preliminaries
2.1 Data Flow ............................. 7
2.2 Mode-Controlled Data Flow .............. 9
   2.2.1 MCDF Constructs ................. 10
   2.2.2 Schedules and Temporal Analysis 13
2.3 Dynamic Voltage and Frequency Scaling 14
   2.3.1 Continuous/Discrete VFS ......... 15
   2.3.2 Local/Global VFS ............... 16
2.4 Power Model .......................... 18
2.5 Optimization Problem ................. 20
   2.5.1 Linear Programming ............. 21
   2.5.2 Convex Programming ............. 21

## 3 Overall Approach
3.1 Mode Sequence Notations ............... 25
3.2 Static Sub-Sequences .................. 26
3.3 Frame Format Detection Point .......... 26
3.4 Unrolling of MCDF Graph ............... 27
3.5 Radio Application Classification ....... 32
   3.5.1 Radio Application without Time Overlap Between Static Sub-sequences 32
# Table of contents

<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Energy-Aware Scheduling for Modal Radio Graphs</td>
<td></td>
</tr>
<tr>
<td>3.5.2 Radio Application with Time Overlap Between Static Sub-sequence</td>
<td>33</td>
</tr>
<tr>
<td>4 Wireless LAN</td>
<td>37</td>
</tr>
<tr>
<td>4.1 Overview</td>
<td>37</td>
</tr>
<tr>
<td>4.2 Temporal Requirements</td>
<td>39</td>
</tr>
<tr>
<td>4.2.1 Synchronization</td>
<td>39</td>
</tr>
<tr>
<td>4.2.2 Header</td>
<td>40</td>
</tr>
<tr>
<td>4.2.3 Payload and CRC</td>
<td>42</td>
</tr>
<tr>
<td>4.3 DVFS</td>
<td>42</td>
</tr>
<tr>
<td>4.3.1 Continuous Local VFS</td>
<td>43</td>
</tr>
<tr>
<td>4.3.2 Discrete Local VFS</td>
<td>43</td>
</tr>
<tr>
<td>4.3.3 Discrete Global VFS</td>
<td>45</td>
</tr>
<tr>
<td>4.4 Experiments and Results</td>
<td>48</td>
</tr>
<tr>
<td>5 Long Term Evolution</td>
<td>53</td>
</tr>
<tr>
<td>5.1 Overview</td>
<td>53</td>
</tr>
<tr>
<td>5.2 Temporal Requirements</td>
<td>56</td>
</tr>
<tr>
<td>5.3 DVFS</td>
<td>57</td>
</tr>
<tr>
<td>5.3.1 Continuous Local VFS</td>
<td>58</td>
</tr>
<tr>
<td>5.3.2 Discrete Local VFS</td>
<td>58</td>
</tr>
<tr>
<td>5.3.3 Discrete Global VFS</td>
<td>59</td>
</tr>
<tr>
<td>5.4 Experiments and Results</td>
<td>62</td>
</tr>
<tr>
<td>6 Conclusion</td>
<td>67</td>
</tr>
</tbody>
</table>
CHAPTER 1. INTRODUCTION

1 Introduction

1.1 Background

Nowadays, computers are everywhere to help us in many areas. Besides the considerable amount of desktops and laptops, a large number of computers systems are embedded as part of a complete mechanical or electrical system with their own dedicated function. These computer systems are commonly referred to "embedded systems".

Embedded systems are embedded as part of larger systems that span from small portable devices such as cellular phones, video cameras and radio players to larger systems like cars and aircrafts. Some of the most used embedded systems in our life are "smart phones". Current cellular phones are capable of much more than simple voice communication, they should also support functionalities such as media playing, messaging, GPS positioning and Internet browsing. However, all their functionalities still require wireless-communication as long as the devices are portable[12] In this project, we focus on Embedded computer systems to handle wireless communications. This application domain is commonly referred to as radio baseband processing.

ST-Ericsson is dedicated to the development of software-defined implementations of digital radio modems on embedded multiprocessor systems. In these modems, there are huge varieties of radio standards that need to be supported like GSM, WCDMA, WLAN 802.11a/b/g/n, bluetooth, LTE-FDD and so on. Radios contain multiple functionalities with different computation types, therefore different types of hardware are also required. In this chapter, we will discuss the requirements of radio baseband applications, the hardware platform used in this thesis together with problem description and contribution of this thesis.

1.2 Radios

A traditional radio is a wireless communication system implemented in a number of stages. Figure 1.1 from [12] shows the general data flow of a radio transceiver through several basic functional blocks. There are three sub-stages in the baseband processing: Digital filtering, MoDem and CoDec. All these stages are processed in the digital domain and repeatedly activated[11]. They constitute a chain of functional dependencies over which a strict end-to-end real-time requirement is defined.

One of the main requirements of radios is timing. The correct behavior of radios depends not only on the value of the computation but also on the time at which the results are produced. In this way, radio baseband processing is said to be a Real-Time (RT)[4] appli-
CHAPTER 1. INTRODUCTION

Filters
Modem
Codec,
\((de)\)mux
RF/IF
D/A                  A/D
Application processing
Control
Digital baseband
Figure 1.1: Stages of a radio transceiver (from [12])

cation. For example, if a Wireless LAN receiver fails to receive an acknowledgement packet within the time interval prescribed by the standard, the base station will retransmit the same packet over and over again. A video player that only produces very few frames per second will give illusion of movement to the human eyes.

We consider two basic types of timing requirements: latency and throughput. By latency the maximum reaction time to an input is important. In other words, the time interval between the input and output data should fall strictly into a specific range. Throughput requirement asks for a certain number of outputs within a time interval. A sufficient rate of output production is required. For example, phone calling requires in time processing for data items, otherwise the unfinished data will be dropped or retransmitted. Meanwhile it is also important that audio arrives at a certain rate to ensure the continuity.

Beside being real-time applications, radios are also streaming applications because they are repeatedly activated by an endless sequence of input data. As shown in figure 1.1, the system receives input data items from Radio frequency stage. Each data item is sent to the digital baseband processing stage after conversion from the analogue domain to the digital domain. All the sub-stages in digital baseband processing are repeatedly activated. The result of processing will be sent out by the RF stage as an endless sequence of output data items.
CHAPTER 1. INTRODUCTION

1.3 Multiprocessor Platform

From the previous section we already discussed that radio baseband processing contains three sub stages: Digital filtering, MoDem and CoDec. These three stages have different computation and work load types[16]. For example the MoDem stage contains a much heavier processing of vectors and matrixes. When it comes to radio baseband processing, each stage needs a different type of processor for its specific requirements. In other words, radio applications should be mapped onto heterogeneous multiprocessor system platforms with multiple types of processors. By doing so, better performance can be achieved together with lower energy consumption.

Figure 1.2: Abstract multiprocessor system template (from [12])

Figure 1.2 from [12] shows the template of the heterogeneous multiprocessor system platform aiming at radio baseband processing[12]. Three different types of processors are defined. EVP is a vector processor that mainly deals with detection, synchronization and demodulation[16]. ARM is a general-purpose processor aiming at control and general functionalities. Software CoDec (SWC) is a hardware accelerator that is used for baseband coding and decoding. Notice that several instances of each might be employed on one system. Depending on the kind of radio, multiple vector processors, general-purpose processors and hardware accelerators are employed. Also, each processor has its own data and code memory. The processors are connected by a multi-layer Advanced High Performance bus.

In most cases, multiprocessor systems used for radio are battery-operated devices. A design objective is to minimize the energy consumption so that longer battery service time can be reached. In CMOS VLSI circuits, the energy consumption is quadratically propor-
tional to the supply voltage[5]. Therefore, reducing the voltage level can significantly save energy by switching the voltage switches on the multiprocessor system to lower levels when processors are not fully active. Normally, the hardware platform provides several supply voltages to choose. A processor can choose a specific voltage/frequency level for an application and keep running at the same voltage/frequency. On the other hand it can also switch to different voltage levels while running an application. The voltage choice can be made independently by each processor or by a global switch for the entire system. With voltage switch per processor, the system becomes more flexible such that more slack time can be used. However, it requires multiple independent power supplying for each processor, which is very expensive. Systems with one global switch to control the supply voltage is much cheaper, but it brings less usage of slack time to save power. In this thesis, voltage scaling is considered for both: local voltage switches and a single global voltage switch.

1.4 Problem Description

As discussed before, radio baseband processing is a hard real-time streaming application that runs on heterogeneous multiprocessor platform. Besides the real-time requirements, energy consumption must also be considered when they are running on battery-operated devices. In this thesis we are concerned with two types of requirements:

- Both latency requirement and throughput requirement must be guaranteed;
- The energy consumption should be as low as possible.

To analyze the temporal behavior of applications, several concepts and components are necessary, including: a model of computation, a scheduler with a specific strategy and an analyzer of the schedule. At ST-Ericsson in Eindhoven, the Heracles tool is designed to schedule radio applications on a target multiprocessor platform and do the timing analysis of the scheduling. In the Heracles tool, radio applications are modeled as data flow graphs. Data flow is a well-known programming model which is suitable for expressing streaming applications[2]. The well-defined firing rules of the data flow model offer powerful analytical properties.

The Heracles tool takes a data flow graph file (including graph information, actor mapping information) and a multiprocessor platform description file as inputs. As output, it allows feasibility checking of timing requirements and scheduling for data flow graphs on the multiprocessor systems. The Heracles tool first parses the input data flow graph into the system and test the correctness of this graph. It verifies if all actors are mapped onto processors of the system, if input/output rate is consistent and so on. After that it will convert the input graph into an Homogeneous Synchronous Data Flow (HSDF) graph. The temporal performance can be be found through graph analysis on self-timed analysis. The schedule result includes the actor mapping relations and execution order. The Heracles tool supports simulation on multiprocessor platforms.

However, due to the data dependencies in the data flow graph and the timing requirements, processors are not fully active inside the original schedule. This gives us the possi-
ability to adjust the frequency of processors into lower levels such that longer execution time is gained to fulfill the slack time while decreasing operating frequencies. The main reason to decrease the frequencies of processors is that frequency is increasing with supply voltage. With lower frequency, the system is running on lower voltage level. As discussed before, in CMOS circuit, reducing voltage level can significantly save energy. This is called dynamic voltage and frequency scaling (DVFS)\footnote{5}. The main idea is to perform DVFS such that the system provides "just-enough" speed to process the workload while still meeting the timing requirements.

In previous work, different VFS policies were introduced. The results already proved that DVFS can reduce the energy consumption for radio applications modeled as data flow graphs\footnote{9}\footnote{6}. However, most of them are using static models for real-time streaming applications. Static Data flow (StDF) is a popular model for streaming applications. StDF allows verification of RT requirements and deadlock-freedom. However, it can only express applications with fixed data-rate. When modeling the baseband processing of a radio transceiver, a change of states is more like a change of computation graphs. StDF does not support the expression of this kind of state changes. Dynamic Data flow (DDF) allows mode changing, but it is not analyzable for temporal behavior, it is impossible to verify the deadlock-freedom and the throughput constraints. The work\footnote{12} carried out by Orlando Moreira at ST -Ericsson shows that Mode-controlled data flow (MCDF) is sufficiently expressive to model radio applications such as Wireless LAN and LTE, while still keeping the ability to analyse the temporal behavior.

Based on the description above, our challenge is to adapt DVFS policies for MCDF graphs while still allowing temporal behavior analysis. Different voltage and frequency points sets are obtained per actor per mode according to variant mode sequences. First, a set of voltage-frequency points for each task in the application (in fact a schedule of voltage-frequency points for each VF-switch in the system) should be obtained. At the same time, changing the speed of each task makes the execution time scalable, hence a new schedule of the MCDF that still meets the time requirements is needed. By solving the DVFS problems, we want to find out an energy-aware schedule that meets both the energy and timing requirements for MCDF models.

1.5 Contribution

In this thesis, we proposed a technology to unroll a MCDF graph into an equivalent SRDF graph with respect of a sequence of control tokens that represent an input frame. By unrolling an MCDF graph into an equivalent SRDF graph, each iteration of the new SRDF graph indicates the processing of one frame.

After that, two types of radio application modeled as MCDF graph were studied: radio applications with and without time overlap between static sub-sequences. Deadline-constrained is used for radio applications without time overlap between static sub-sequences and throughput-constrained VFS is studied for radio applications that allow overlaps between static sub-sequences. For each type of VFS problem, a special instance is studied.
The Wireless LAN 11a receiver is used as an example of radio applications without time overlap between static sub-sequences. We enroll the MCDF graph into an equivalent SRDF graph due to the SSS. Because WLAN model does not allow overlap between SSSs, the only timing constraint we need to concern is the deadline constraint for a SSS. In this way, only one iteration of the equivalent SRDF graph is important without thinking about the predictability of the following iterations. Self-timed schedule is used for deadline-constrained VFS problem to support temporal behavior analysis.

The Long Term Evolution Downlink Transmission model (LTE DL) is a radio application with time overlap between static sub-sequences. The LTE model has an infinite input data sequence and overlap across frames is allowed here. In throughput-constrained VFS, static periodic scheduling is carried out to ensure the periodic appearance of outputs. By using static periodic scheduling we can reduce the problem from infinite to finite.

Timing requirements analysis is applied for both example applications. After that, three different dynamic VFS algorithms are applied respectively, namely: continuous local VFS, discrete local VFS and discrete global VFS.

1.6 Organization

The remainder of this thesis is organized as follows. In chapter 2, some preliminary knowledge is introduced to help understanding of data flow graphs, VFS strategies and other concepts in this thesis. Chapter 3 describes the general approaches for deadline-constrained VFS and throughput-constrained VFS for MCDF graphs. Chapter 4 and 5 discuss the WLAN receiver and LTE DL model respectively, and also propose the DVFS algorithms for both instances. In chapter 4 and 5 we also discuss the experiment results for the DVFS algorithms on both instances. Finally, chapter 6 gives our conclusion based on the previous chapters and gives suggestion for future work.
2 Preliminaries

2.1 Data Flow

The heracles tool developed by ST-Ericsson in Eindhoven takes data flow as the model of computation. In this section, the graph definitions will be introduced together with terminologies. We also discuss several variants data flow models used in this thesis.

Data flow graphs are directed graphs. A directed graph can be represented as \( G = (V, E) \) where \( V \) is a set of nodes and \( E \) is a set of edges. Every edge is an ordered pair \((i, j)\) where \( i, j \in V \). For an edge \( e(i, j) \), we say \( e \) is directed from source node \( i \) to destination node \( j \). Edges represent First-In-First-Out (FIFO) queues while nodes represent computation blocks called actors. In data flow graphs, a number of concurrent actors communicate through unidirectional FIFO channels. Data is transported in discrete chunks called tokens.

An actor in a data flow graph can fire when it is activated by data availability. Data flow model has well-defined firing rules, that is an actor can only fire when there are sufficient tokens on each of its input channels. When an actor is allowed to fire, it consumes tokens from its inputs according the consuming rate. After finishing computation, it produces tokens to all its outputs.

When all actors in a data flow graph consume/produce only one token on each of their inputs/outputs per firing, the graph is called a Single-Rate Data Flow (SRDF) (also called Homogeneous Synchronous Data Flow)[10]. A Single-Rate Data flow is a directed graph \( G = (V, E, d, t) \). A node \( i \in V \) is called an actor and represents an entity of the application that consumes inputs and produces outputs after a fixed period of time; \( t(i) \) is the execution time of actor \( i \). An edge \( e \in E \) represents the communication channel; \( d(i, j) \) is the number of initial tokens on edge \( e(i, j) \), which is also called the delay of edge \( e(i, j) \). Figure 2.1 shows an example of SRDF containing four actors \( A, B, C \) and \( D \). The numbers inside indicate the execution times of each actor. The black dot indicates one initial token on edge \((D, A)\). It is proved that an SRDF is deadlock free if and only if there is at least one initial token in every cycle[14]. By deadlocked graph we mean the situation where a set actors cannot fire because they requires others in this set to fire to gain enough tokens on their inputs.

SRDF graphs are commonly used because of its useful analytical properties. An iteration of the SRDF indicates all the actors in \( V \) have fired once. For all \( e(i, j) \) in \( E \), according to data dependencies, the start time of both actors should follow:

\[
s(j, k) \geq s(i, k - d(i, j)) + t(i)
\]  

(2.1)

where \( s(j, k) \) and \( s(i, k - d(i, j)) \) are the start time of actor \( j \) in the \( k^{th} \) iteration and the start time of \( i \) in the \( k - d(i, j)^{th} \) iteration. This rule denotes the data dependency constraint.
CHAPTER 2. PRELIMINARIES

Figure 2.1: An example of SRDF

It means for an edge, the start time of sink actor in iteration \( k \) cannot be earlier than the finishing time of source actor in iteration \( k - d(i, j) \).

The goal of timing analysis is to check if the real time requirements are met. The latency can always be calculated as the difference between the finishing time of sink and the start time of source. However, when it comes to throughput, we need the concept of **Maximum Cycle Mean (MCM)**. A directed cycle \( c \) in graph \( G \) is a path from an actor to itself that traverses each node in it once. The cycle mean of cycle \( c \) is calculated as:

\[
\mu_c = \frac{\sum_{i \in V(c)} t(i)}{\sum_{e \in E(c)} d(e)}
\]  

(2.2)

where \( V(c) \) and \( E(c) \) indicate the sets of actors and arcs belonging to the directed cycle \( c \). The MCM is defined as:

\[
\mu(G) = \max_{c \in \mathcal{C}(G)} \frac{\sum_{i \in c} t(i)}{\sum_{i \in c} d(i, j)}
\]  

(2.3)

where \( \mathcal{C}(G) \) is the set of cycles in \( G \). The reciprocal of the MCM provides the maximum attainable throughput of the SRDF.

A **Self-Timed Schedule** (STS) of an SRDF graph is a schedule where each actor starts to fire immediately when they have enough tokens for each of its input port. It is known as an as-soon-as possible schedule. In STS an actor starts to fire as soon as all precedence constraints are met. For an actor \( j \), the start time in iteration \( k \) is defined as:

\[
s(j, k) = \max_{(i, j) \in E} \left\{ \begin{array}{ll}
s(i, k - d(i, j)) + t(i, k - d(i, j)), & d(i, j) \leq k \\
0, & d(i, j) > k
\end{array} \right.
\]  

(2.4)

A **static Periodic Schedule** (SPS) of an SRDF is a schedule such that for all actors \( i \in V \) and all \( k > 0 \). Thus the start time of actor \( i \) in iteration \( k \) is known when we know the start time of the first iteration and the period \( T \).

\[
s(i, k) = s(i, 0) + T \times k
\]  

(2.5)

After applying SPS, the precedence constraint of edge \( e(i, j) \) becomes:

\[
s(j, k) \geq s(i, k) - T \times d(i, j) + t(i)
\]  

(2.6)
CHAPTER 2. PRELIMINARIES

Notice that for an SRDF graph, an SPS can be found if and only if \( T \geq \mu(G) \), otherwise no SPS exists.

When it is allowed that \( \text{proc}(i) \neq \text{cons}(j) \) for edge \( e(i, j) \), the graph becomes Multi-Rate Data Flow (MRDF). It is proved that a MRDF graph can be converted into an equivalent SRDF graph. However, actors in MRDF graphs still have consistent producing/consuming rate. Data flow graphs like SRDF and MRDF are called Static Data Flow (StDF). In StDF, the number of tokens produced and consumed by each actor is fixed per firing. If the worst-case execution time of each actor is known, the maximum guaranteed rate of production (throughput) can be computed. If the worst case execution time of each actor is defined, the maximum attainable throughput can be computed. StDF allows to verify real-time requirements, deadlock-freedom and so on. However, lacking of flexibility limits the expressiveness of StDF. As discussed before, radio baseband processing contains several stages/modes. StDF can be efficient to model one mode, however it is inconvenient to express the communication between different modes. Dynamic Data Flow (DDF) allows different graph changing across iterations[3]. Most DDF graphs separate an application into small applications such that a set of tasks are executed in each iteration with variants rate. It is suitable for mode transitions between iterations, but it does not provide analyzable properties.

2.2 Mode-Controlled Data Flow

In previous sections, we mentioned that StDF is more analyzable while lacking expressivity. Meanwhile DDF can express mode changing and mode transition but few analyzable properties are provided. In radio baseband processing, the existence of different frame formats and data types requires mode switching properties. Meanwhile a static schedule is also needed for temporal analysis. The choice of data flow model should be made considering the trade-off between expressiveness and analytical properties.

In [12], Mode-Controlled Data Flow (MCDF) is introduced to represent the dynamic behavior of streaming applications. An MCDF is a dataflow graph where for each firing of a designated actor, called the Mode Controller (MC), actors belonging to a specific, pre-defined subgraph are fired. Beside MC, there are also some other types of actors such as mode switch actors, mode select actors and tunnels. These actors are called data-dependent actors. They will be explained in next subsection. The output of the MC is an integer in the closed interval between 1 and \( M \), which determines which subgraph is firing in this iteration. In an MCDF graph \( G = (V, E, t, d, M, \text{mode}, \text{atype}) \), \( M \) denotes the set of modes in the graph, \( \text{mode} \) is the mode valuation of actors and \( \text{atype} \) is the actor type valuation for actors. MCDF actors are either modal or amodal. An actor belongs to a mode \( m \) if it is connected to another actor belonging to mode \( m \) or a data-dependent actor associated with mode \( m \). An actor is called amodal if it is not belonging to any mode, which is represented as \( \text{mode}(i) = \bot \). Modal actors only fire in iterations with corresponding modes while amodal actors always fire in every iteration. We define a sub-graph \( G_j = (V_j, E_j, t_j, d_j) \) of mode \( m_j \) where \( V_j \) indicates the set of all amodal actors together with modal actors with mode \( m_j \), \( E_j \) denotes edges connect actors in \( V_j \).
2.2.1 MCDF Constructs

Besides single rate actors, MCDF graphs also contain several special actors\[12\]. The Mode Controller (MC) has several control output ports. The output tokens of them are control tokens to decide which sub-graph is executed. At each firing, MC produces one control token on each of its output ports. After a sequence of firing of MC, a special mode sequence occurs. A mode sequence is a valid sequence of control signals produced by the mode controller. The MCDF sub-graphs will execute in the order of mode sequence.

Beside MC, MCDF also uses three types of data-dependent actors: Mode Switch, Mode Select and Mode Tunnel, as shown in Figure 2.2. Data dependent actors produce and consume tokens as normal, at the same time these data depended actors also have control input port. The token comes from control port tells it which of its other ports should consume/produce tokens.

A Mode Switch actor has one data input port, \( M \) data output ports and one control input port that determines on which data output port the output token should be produced. Each output port is associated with a modal actor. On each of its firing, it will copy the input data token and consume exactly the same token on the corresponding output port. In contrast, a Mode Select actor has \( M \) data input ports, one data output port and one control port that selects which input token should be consumed. Both mode select and mode switch will consume one control token and one data input token in each firing and produce one data output token. Notice that for mode select, if the control token indicates that the current mode is \( m_j \) while the corresponding data input port is not connected, it can still fire as normal and consume an output token with no data information contained. This also happens to mode switch actor: the data input ports and data output ports can be unconnected.

A Mode Tunnel actor has one data input port, one data output port and one control port. In Figure 2.2 (c), when control signal “1” occurs, it will read the token from the data input port and store it as the internal state. Every time when a new control signal “1” comes, the internal state will be refreshed. Until the control input port has value “2”, it copies the token stored to the data output port. In this way it delivers to its consumer the value produced by the last previous execution ordered by the MC\[12\]. In fact a mode tunnel can be treated as a block of a special construct containing one mode switch, one mode select actor and
two modal actors inside as shown in Figure 2.3. As a normal mode tunnel actor, whenever it receives a control token of mode 1, actor A will make a same output token and store it as the internal state. When it comes to mode 2, actor B will produce a token same with the internal token and clear the internal token. However, if a sequence of mode 1 occurs, mode tunnel can only store the data token from the latest mode 1. To solve this, another type of mode tunnel called Accumulator is defined. In accumulator, the inner token has the size large enough to store several data input tokens according to the application. It is regarded that the inner token has a stable number of positions to put a data token in. When mode 1 occurs, actor A will add this data input token to the first available position in the inner token. And when mode 2 occurs, it will fire the whole inner token out and make it empty again. In this way, more information from mode 1 is stored without only the latest one. Beside one mode to one mode situation, accumulator actor can also store input tokens from multiple modes.

An MCDF graph with $M$ modes is composed of a MC actor; an arbitrary number of mode select, mode switch and mode tunnel actors; an arbitrary number of static, single-rate actors. The MC and all data-dependent actors are amodal actors that need to fire in every iteration. Single-rate actors could be both modal or amodal. [12] already pointed out that the following rules must be respected for a well-constructed MCDF graph to ensure that it can always return to the initial state of token placement.

- There is only one mode controller.
- Modal actors can either be connected with other modal actors or to the ports associated with their mode on data-dependent actors.
- The MC output port is connected to control input ports of all data-dependent actors.
through edges with no initial token.

- There are no delay-less cycles in the graph.

Figure 2.4 shows a simple example of an MCDF graph containing one mode controller, actor A in mode 1, actor B in mode 2 and an amodal actor Source. Assuming that the first control token produced by the MC is "1", mode Switch will pass the token from Source to actor A. Actor A produces two tokens, one is sent to Select and the other one is stored in Tunnel. Select will send the token to MC again to inform the finishing of this iteration. After that the MC will produce another token. For example if control token 2 is given in the second iteration, Switch will send the token from Source to actor B. In this iteration, Tunnel will fire the inner token out as output, and clear the internal state. Then actor B fires. The results is sent to MC again through Select. If control token 3 appears, all data dependent actors will fire and consume the input tokens, but generate no output tokens.

![Diagram of MCDF example](image-url)
2.2.2 Schedules and Temporal Analysis

An iteration of the MCDF is corresponding to not the firing of all actors but the actors in one subgraph. Unlike in SRDF, the mode sequence is also considered in the temporal analysis. A mode sequence is a valid sequence of control signals produced by the mode controller. The MCDF graphs will execute in the order of the mode sequence. An element in the mode sequence indicates the value of the mode for this iteration. Only actors that have this mode or amodal actors will fire in this iteration. The start time $s(i, k)$ of an actor might not exist if the mode of iteration $k$ is different from the mode of actor $i$.

The start time of actor $i$ is defined as $s(i, k, c)$ where $i$ is an actor in the graph, and $k$ means the $k^{th}$ iteration of execution under the mode sequence of $c$. $s(i, k, c)$ is only defined when $mode(i)$ equals to the $k^{th}$ element of mode sequence $c$ or $mode(i) = \bot$. Otherwise we define $s(i, k, c) = \bot$.

Equation 2.1 shows the precedence constraints in SRDF. It indicates the relationship between the firing of actor $i$ and the firing of actor $j$ in edge $(i, j)$. For an edge with no initial tokens on it, in iteration $k$ the sink actor cannot fire until the source actor finished producing one token. That is the start time of actor $j$ in iteration $k$ should be later than the finishing time of actor $i$ in the same iteration. However, if the delay value $d(i, j)$ is not equal to zero, then the first $d(i, j)$ firings of actor $j$ are only depended on those initial tokens but not the firing of actor $i$. Since in each iteration actor $i$ only produces one token, only when it comes the $d(i, j) + 1^{st}$ iteration actor $j$ starts to consume tokens produced by $i$.

This rule can also applied to amodal edges in MCDF graph. By amodal edge we mean an edge that both its sink and source actors are amodal actors. In this situation, both actors on this edge will fire in every iteration, exactly one token is produced and one token is consumed. Thus the precedence constraint for amodal edge $(i, j)$ is:

$$s(j, k, c) \geq s(i, k - d(i, j), c) + t(i), \quad d(i, j) \leq k$$

[12] gives the definition of edge modality, that is:

$$emode(i, j) = \begin{cases} 
mode(i), & \text{if } mode(j) = \bot \\
mode(j), & \text{otherwise}
\end{cases}$$

If $emode(i, j) \neq \bot$ then we say edge $(i, j)$ is modal edge. In this situation at least one actor is modal actor, thus modal edges only produce or consume token on iterations where their mode has been selected by the mode controller. In this way, the token produced by $i$ in the first iteration where mode $emode(i, j)$ is choose will be consumed by $j$ in the $d(i, j)^{th}$ selection of $emode(i, j)$ in the mode sequence. Unlike amodal edges where the firing of $j$ in iteration $k$ is depended on the firing of $i$ in the $k - d(i, j)^{th}$ iteration, it is now depended on iteration $k'$ where the same mode occurs. The value of $k'$ is defined by $k - \delta(k, c, d(i, j))$ which is called Modal Delay. From iteration $k - \delta(k, c, d(i, j))$ to iteration $k$, $d(i, j)$ times of mode $emode(i, j)$ have been selected. Hence the precedence constraint for modal edge $(i, j)$ is:

$$s(j, k, c) \geq s(i, k - \delta(k, c, d(i, j)), c) + t(i), \quad \delta(k, c, d(i, j)) \leq k$$
when $\text{emode}(i, j) = c(k)$.

An MCDF graph also allows self-timed execution. Same as STS in SRDF graph, actors start to fire as soon as all precedence constraints are met. The difference is that the worst case STS is not unique but changing with each mode sequence. By updating the precedence constraints in 2.4 we have the new start time function as:

$$s(j, k, c) = \max_{(i, j) \in E} \begin{cases} s(i, k - \delta(k, c, d(i, j)), c) + t(i), & d(i, j) \leq k \land \text{emode}(i, j) = c(k) \\ s(i, k - d(i, j), c) + t(i), & d(i, j) \leq k \land \text{emode}(i, j) = \bot \\ 0, & \text{otherwise} \end{cases}$$

### 2.3 Dynamic Voltage and Frequency Scaling

To generate a required schedule, the scheduler tool needs to know following information: the data flow graph file (containing mapping relationship to the hardware platform and the timing requirements); the hardware platform description file ; the schedule strategy. The example of SRDF graph in figure 2.1 is mapped to a hardware platform with two processors. Task $A, B, D$ are mapped to processor $\pi_1$ and task $C$ is mapped to processor $\pi_2$.

As mentioned before, the numbers inside the data flow graph indicate the execution time of actors. Since each task has a constant number of cycles to be executed, the execution time depends linearly on the clock cycle period of the processor element. That is $t(i) = \frac{nc(i)}{f(\pi(i))}$ where $f(\pi(i))$ is the frequency of processor where task $i$ is mapping to. Because one processor can only execute one task at the same time, we only need to map the frequency function to the task name instead of the processor. If the number of cycles that an actor needs is a constant that does not change with processor type and clock cycle period, this relationship is specified as:

$$t(i) = \frac{nc(i)}{f(i)}$$

where $nc(i)$ is the cycle number of actor $i$ while $f(i)$ indicates the frequency of the processor when running task $i$.

Figure 2.5 shows an example static periodic schedule of SRDF graph in figure 2.1 with a period of $T = 24 \mu s$. Tasks $A, B, D$ are mapped on processor $\pi_1$ and task $C$ is mapped on $\pi_2$. As mentioned before, schedules for RT streaming applications have infinite length. Using a Static Periodic Schedule can reduce the problem from infinite to finite. Hence deadline-constrained VFS is applied to blocks with a period of $T$. The thick arrows indicate the deadline constraints for each period (block). The dashed arrows point to the end time of all tasks. Slack time exists in this schedule as shown in figure 2.5. This gives the possibility to adjust the supply voltages of processors to lower levels to slow the system down. Therefore both processors achieve higher utilizations. This is called **Voltage and Frequency Scaling (VFS)**. By performing VFS algorithms we want the system to provide "just-enough" speed to process the workload while still meeting the timing requirements. In this thesis we focus on Dynamic VFS (DVFS). Unlike Static VFS (SVFS) where each processor only has one
frequency solution and remains same speed when running an application, DVFS allows frequency changes during the execution of an application. Thus tasks mapping on the same processor can have different frequency and voltage solutions.

![Figure 2.5: Static Periodic Schedule for SRDF](image)

2.3.1 Continuous/Discrete VFS

Continuous VFS assumes that frequency changes in a continuous interval. Every task can run at any frequency value within this interval [13]. In this interval, each frequency has a one to one correspondence relationship with a execution time for an actor. No matter what kind of combinations of several frequency levels, we can use one frequency value to achieve the same execution time. In this way we assign a constant frequency value to each task.

The given SPS gives a 5 \( \mu s \) slack time after task \( D \) finishes. Also, processor \( \pi_1 \) is idling after finishing task \( B \) and before staring task \( D \) because it needs to wait the finishing of task \( C \) on processor \( \pi_2 \). A new SPS for the SRDF graph is shown in Figure 2.6 by applying the continuous VFS where actor \( A \) and \( C \) are still running on the original frequency. We treat this frequency as the base frequency, which is the highest frequency that can be reached in this system. Meanwhile task \( B \) and \( D \) are running at medium frequency. The new schedule has less slack time while still meeting the same deadline-constraint. By using lower frequency more, the hardware platform only need to provide lower supply energy, such that lower energy consumption is reached.

Figure 2.6 is shown for the situation where task \( B \) and \( D \) are running at the medium frequency. Processor \( \pi_1 \) remains at the same frequency when processing task \( B \) and \( D \).

Continuous VFS is not practical because real platforms only provide a limited number of discrete voltage-frequency points to choose. In this case, an actor can only run on frequencies within the frequency set. A sequence of frequency levels is applied to each actor. An actor is divided into small parts and each part has its own frequency. The utilization of each frequency level is the proportion occupied by the part of this actor running at this frequency. Figure 2.7 shows the discrete VFS for the same example where three different frequency...
levels are adapted to every actor: Idle, High and Low. For different actors, the utilization of each frequency level varies.

Because of the limited number of frequency levels, discrete VFS will lose optimization comparing to continuous VFS. The red arrows in figure 2.7 indicate that the slack time remains after applying discrete VFS. This is because the number of cycles for each frequency level should remain integer. For example, the execution time for task $B$ after discrete VFS is:

$$t_B = \frac{nc(B_H)}{f_H} + \frac{nc(B_L)}{f_L}$$

where $nc(B_H) + nc(B_L) = nc(B)$. $nc(B_H)$ and $nc(B_L)$ are both integer. This value might not able to reach the same execution time as in continuous VFS.

### 2.3.2 Local/Global VFS

The VFS problem can use both local VF-Switches and a single, global VF-Switch. In local VFS, each processor has its own VF-Switch interval[13]. Hence in the same time instance, different processors can execute at different speed. Tasks mapping on different processors can be executed at various frequencies. Figure 2.8 gives an example of local VFS. Task $B$ is extended to fulfil the slack time between task $B$ and $D$. Tasks $A$, $C$, $D$ run at the same frequency, which is the base frequency. Between time interval [8,15], two processors are working at different speed. Processor $\pi_1$ has to switch to a different VF level twice. Notice that we only consider the situation when both processors are active.

In many practical multiprocessor platforms, only one global VF-switch is available. Thus at any point of time, all the active tasks, no matter which processor they are mapped to, must run at the same frequency. Figure 2.9 shows an example where processor $\pi_1$ and processor
\( \pi_2 \) always execute at the same frequency. In the original schedule in figure 2.5, task \( B \) and \( C \) have a overlap of 5 time units, when task \( B \) is executing, task \( C \) is also executing. Thus when applying global VFS, the part of task \( C \) together with \( B \) should always apply the same frequency setting. After that, task \( C \) still has some unfinished part to execute. In this way, global VFS reaches lower utilization of processors.
2.4 Power Model

In CMOS VLSI circuits, the energy consumption is quadratically proportional to the supply voltage, which means that we need to have a supply voltage as low as possible. However, a lower voltage brings a slower execution speed. The goal for DVFS is to find an energy-aware scheduling with scaled execution times of actors using VFS, such that a minimum energy consumption is reached under the temporal constraints.

In this thesis only dynamic power is considered, that is[5]:

\[ P_{\text{dynamic}} = \alpha C v_{dd}^2 f \] (2.13)

where \( \alpha \) and \( C \) are some constant coefficients and \( v_{dd} \) and \( f \) are the voltage and frequency values. Frequency \( f \) determines the execution time of actors. When performing VFS, frequency \( f \) is actually scaling with supply voltage[5]. The relationship between voltage and frequency can be described as:

\[ f = K \frac{(v_{dd} - v_{th})^2}{v_{dd}} \] (2.14)

where \( v_{th} \) is the switching threshold voltage of a transistor, here we set it to be 0.55V. And \( K \) represents a circuit dependent constant.

Without applying VFS, all processors are running at the base frequency. We set the base frequency as the highest frequency of all processors when running at the voltage of \( v_{HH} = 1.1 \text{V} \), which is called \( HH \text{igh} \) mode. The base frequency running at this voltage is \( f_{HH} = 312 \).

Figure 2.9: Global VFS
CHAPTER 2. PRELIMINARIES

As mentioned before continuous VFS is not practical. In fact our hardware platforms only supports discrete VF-points for each processor. Beside \textit{HHigh} mode there are four modes named \textit{High}, \textit{Low}, \textit{LLow}, \textit{Elow} from our target multiprocessor platform. Each has a frequency half of the adjacent higher mode, which will bring twice the execution time. If a processor is not working, it is said to work in \textit{Sleep} mode. Table 2.1 shows the relationship between different modes.

Table 2.1: Frequency Modes

<table>
<thead>
<tr>
<th></th>
<th>HHigh</th>
<th>High</th>
<th>Low</th>
<th>LLow</th>
<th>Elow</th>
<th>Sleep</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>312</td>
<td>156</td>
<td>78</td>
<td>39</td>
<td>19.5</td>
<td>0</td>
</tr>
<tr>
<td>Voltage</td>
<td>1.1V</td>
<td>0.9022V</td>
<td>0.7818V</td>
<td>0.7058V</td>
<td>0.6562V</td>
<td>0</td>
</tr>
<tr>
<td>Time extension factor</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 2.2: Power Model

<table>
<thead>
<tr>
<th>Power(mw)</th>
<th>EVP</th>
<th>SWC</th>
<th>ARM</th>
<th>DC</th>
</tr>
</thead>
<tbody>
<tr>
<td>HHigh</td>
<td>confidential number</td>
<td>confidential number</td>
<td>confidential number</td>
<td>0.0</td>
</tr>
<tr>
<td>High</td>
<td>confidential number</td>
<td>confidential number</td>
<td>confidential number</td>
<td>0.0</td>
</tr>
<tr>
<td>Low</td>
<td>confidential number</td>
<td>confidential number</td>
<td>confidential number</td>
<td>0.0</td>
</tr>
<tr>
<td>LLow</td>
<td>confidential number</td>
<td>confidential number</td>
<td>confidential number</td>
<td>0.0</td>
</tr>
<tr>
<td>Elow</td>
<td>confidential number</td>
<td>confidential number</td>
<td>confidential number</td>
<td>0.0</td>
</tr>
<tr>
<td>Sleep</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

In our case, all scalable actors are running on three types of processors: \textit{EPV}, \textit{SWC} and \textit{ARM}. VFS is not available for source/sink/latency actors because they are just virtual tasks and they are not mapping on real processors. Hence we assumes that those actors are mapping on some unscalable virtual processors. Those unimportant processors are referred to \textbf{Don't Care} (DC). The power value \(p(\pi(i), \text{mode})\) is shown in Table 2.2. \(\pi(i)\) indicates the processor running task \(i\), and \(p(\pi(i), \text{mode})\) indicates the power of processor \(\pi(i)\) when running at a specific mode. The new execution time for actor \(i\) now becomes:

\[
t(i) = t_{\text{orig}}(i) \times (1 \times u_{\text{HHigh}}(i) + 2 \times u_{\text{High}}(i) + 4 \times u_{\text{Low}}(i) + 8 \times u_{\text{LLow}}(i) + 16 \times u_{\text{Elow}}(i)) \tag{2.15}
\]

where \(u\) indicates the utilization of an actor on each mode. The sum of them should be equal to 1. That is \(1 \times u_{\text{HHigh}}(i) + 2 \times u_{\text{High}}(i) + 4 \times u_{\text{Low}}(i) + 8 \times u_{\text{LLow}}(i) + 16 \times u_{\text{Elow}}(i) = 1\). And the new energy consumption function becomes:

\[
E(i) = t_{\text{orig}}(i)(1 \times u_{\text{HHigh}}(i) \times p(\pi(i), \text{HHigh}) + 2 \times u_{\text{High}}(i) \times p(\pi(i), \text{High}) + 4 \times u_{\text{Low}}(i) \times p(\pi(i), \text{Low}) + 8 \times u_{\text{LLow}}(i) \times p(\pi(i), \text{LLow}) + 16 \times u_{\text{Elow}}(i) \times p(\pi(i), \text{Elow})) \tag{2.16}
\]
CHAPTER 2. PRELIMINARIES

By multiplying the power and execution time we can get the energy consumption.

The power model above only shows the power consumption values of given levels. When applying continuous VFS, voltage and frequency level beyond the given set will be chosen. The energy consumption and scaled execution time need to be calculated by the frequency and voltage setting. Equation 2.13 and 2.14 already showed the relationship between power, frequency and voltage. At the same time, base frequency, base supply voltage and base power consumption are all given. Thus the power consumption is calculated as:

\[
p(i) = \frac{p(i)}{p_{HHigh}(\pi(i))} = \frac{\alpha C v^2(i) f(i)}{\alpha C v^2_{HHigh} f_{HHigh}}
\]

(2.17)

\[
p(i) = \frac{v^2(i) f(i)}{v^2_{HHigh} f_{HHigh}} \times p_{HHigh}
\]

(2.18)

The scaled execution time is also inversely proportional to frequency:

\[
t(i) = t_{orig}(i) \times \frac{f_{HHigh}}{f(i)}
\]

(2.19)

Thus the new energy consumption is:

\[
E(i) = p(i) t(i) = t_{orig}(i) \times p_{HHigh}(\pi(i)) \times \frac{v^2(i)}{v^2_{HHigh}}
\]

(2.20)

The power model above shows us the relations between energy consumption, execution time, voltage levels and frequencies. Using voltage and frequency levels as variables, we can verify the temporal behavior and calculate the corresponding energy consumption.

2.5 Optimization Problem

Optimization is the problem of finding the best solution from all feasible solutions under the requirement of an objective function. Applying DVFS for RT streaming applications on heterogeneous multiprocessors is a typical optimization problem. The problem is to minimize the total energy consumption while meeting temporal behavior constraints. With different VFS policies, it can be modeled as various optimization problems. An optimization problem can be described as:

\[
\begin{align*}
\text{minimize} & \quad f(x) \\
\text{subject to} & \quad g_i(x) \leq 0, \quad i = 1, \ldots, m \\
& \quad h_i(x) = 0, \quad i = 0, \ldots, p
\end{align*}
\]

where \( f(x) \) is the objective function to be minimized over the vector of variables \( X \). A minimization problem is defined as previous, a maximization function can be achieved by negating the objective function. \( g_i(x) \leq 0 \) are called inequality constraints and \( h_i(x) = 0 \) are called equality constraints. With different types of objective functions, equality constraints and inequality constraints, an optimization problem can be divided in to several categories. In this thesis only linear programming and convex programming are studied.
2.5.1 Linear Programming

When the objective function, equality constraints and the inequality constraints are all linear, the optimization problem is called a **linear programming** (LP)[7]. A **linear function** means a function $f(x)$ that follows the following properties:

\[
  f(x) + f(y) = f(x + y) \\
  f(ax) = af(x)
\]

Mapping relationship like $f(x) = ax + b$ is not linear if $b$ is not equal to 0. It is called an affine function.

A Linear programming is expressed as:

\[
  \begin{align*}
    \text{minimize} & \quad C^T x \\
    \text{subject to} & \quad Ax \leq B \\
    & \quad x \geq 0
  \end{align*}
\]

where $C$ and $B$ are already known vectors with the same size of $x$ and $A$ is a already known matrix of coefficients. $(.)^T$ is the matrix transpose. In linear programming, objective function $C^T x$ and constraint $x \geq 0$ are linear functions while the inequality constraints are using affine function $Ax - B$.

When it comes to real life, it is common that the solution of vector $x$ are required to be integers. For example, when solving the problem of numbers of trucks needed for a transportation problem, or how many steps needed to reach a specific position in a graph. Discrete VFS mentioned in section 2.3 allows one actor have a sequence of VF points, but it also requires that each VF level should execute a integer number of cycles. An optimization problems like:

\[
  \begin{align*}
    \text{minimize} & \quad C^T x \\
    \text{subject to} & \quad Ax \leq B \\
    & \quad x \geq 0 \\
    \text{and} & \quad x \in \mathbb{N}
  \end{align*}
\]

is a **Integer Linear Programming** (ILP). If only some of the variables are constrained to be integer, then the problem becomes a **Mixed Integer Linear Programming** (MILP).

With the concept of ILP, real life optimization problems become easier to express. However it also increases the difficulty of solving them. As with data flow modeling, a trade-off between expressiveness and solvability is needed. In the discrete VFS example, instead of solving the number of cycles for each frequency, we can solve the utilization of that frequency mode. By changing from modeling the problem as a ILP to a LP, it becomes easier to solve.

2.5.2 Convex Programming

**Convex programming** (CVX) is an instance of non-linear programming that contains Least Squares (LS), Linear Programming (LP) and Quadratic Programming (QP)[8]. It is the optimization problem of minimizing **convex functions** over a convex set. An object is said to be
convex if for every pair of points within the object, every point on the line between the pair of points is also within the object. A convex set means for any two points $X$ and $Y$ inside this set, every point on the straight line segment that joins them is also inside this set. If a function $f(x)$ is defined on a convex set, and for every two points on the graph of the function that the following relationship occurs:

$$\frac{f(x)+f(y)}{2} \geq f\left(\frac{x+y}{2}\right)$$

the function is called a convex function. Figure 2.10 shows an example of a convex function and a convex set.

Convex Programming follows the same formulation of standard optimization problem as:

$$\begin{align*}
\text{minimize} & \quad f(x) \\
\text{subject to} & \quad g_i(x) \leq 0, \quad i = 1, ..., m \\
& \quad h_i(x) = 0, \quad i = 0, ..., p
\end{align*}$$

Convex programming has a wide variety of applications and reliable numerical algorithms. However it requires high level of expertise in both convex analysis and numerical algorithms. An efficient modeling and constructing method of convex programming is Disciplined Convex Programming[8]. It imposes a set of conventions that must follow when constructing convex programming problems. The conventions contain a set of conditions to
guarantee convexity. However, it is still possible to model convex programming problems without obeying these rules, but thus they are not valid DCPs. There are several reasons to use DCP. Firstly it is hard to decide if an arbitrary mathematical program is convex or not, but it is easy to tell if it is a DCP. Also, DCPs can be conversed into solvable form automatically. As long as a problem is formed as a DCP, it can be solved. Furthermore, sophisticated solvers are developed for DCP. In this thesis, all optimization problems are modeled as DCPs.
3 Overall Approach

In previous work, different VFS policies are carried out obeying to either latency or throughput constraints\cite{9}\cite{6}. The results there show that DVFS is feasible in energy saving for radio applications modeled in SRDF\cite{9}\cite{6}. However, previous work using SRDF graph cannot model dynamic behavior of radio applications. MCDF graph is more practical to model the dynamic behavior of radio baseband processing. Therefore, we want to find solutions of the VFS problems for MCDF graphs.

In radio applications, valid source information always consists of a list of symbols. These symbols are organized in specific frame. For example, a WLAN packet contains two synchronization symbols, one header symbol and several payload symbols with a length varies from 1 to 256. The execution of different types of symbols requires multiple data flow graphs. As discussed in chapter 2, a typical MCDF graph for a radio application contains several subgraphs and each of them is responsible for one specific type of symbol. Whenever a symbol is received, the mode controller will generate a control token to choose the correct mode to execute. And one iteration of the MCDF graph will decode exactly one symbol. For a typical radio application, only a finite number of frame formats exists according to the standard. Each frame format corresponds to a finite well-defined symbol sequence. To guide the MCDF graph to execute in a correct order of modes for one frame, a mode sequence $c$ is needed. The $k^{th}$ element of this mode sequence $c(k)$ indicates which mode is needed in this iteration. In other words, each element in a mode sequence represents the execution of one symbol in a frame.

3.1 Mode Sequence Notations

As mentioned above, a typical radio application has a finite number of valid frame formats. Each of them is represent by a specific mode sequence. Thus the composition of a mode sequence indicates the structure of a frame. To formulate mode sequences, following notations are used:

1. $m$: mode selection.
2. $\cdot$: concatenation of two modes.
3. $m^n$: concatenation of $n$ mode selections on mode $m$.
4. $(m_1|m_2)$: one selection in sequence $c$ can be either mode $m_1$ or mode $m_2$. 

For example, a valid frame for the WLAN model with 2 payload symbols is represented by mode sequence:

\[ c_{WLAN,1} = S^2 \cdot H \cdot P^2 \cdot C \]

where \( S, H, P \) and \( C \) indicate the modes for "Synchronization", "Header", "Payload" and "CRC".

### 3.2 Static Sub-Sequences

The rule to construct a mode sequence \( c \) containing all mode control tokens with respect to a valid frame structure is the radio mode sequence pattern. A valid mode sequence \( c \) respected to the radio mode sequence pattern is called a Static Sub-Sequence (SSS). A static sub-sequence represents a list of symbols containing valid source information as an input data item for a radio application, such as a subframe in the LTE model or a packet in the WLAN receiver. For example, in the WLAN receiver model, a correct SSS always starts with two synchronization modes \( S(synchronize) \) following by a symbol of mode \( H(header) \). After that \( n \) times of mode \( P(payload) \) occurs and one mode \( C(CRC) \) ends the SSS. Equation 3.1 shows the mode sequence pattern for the SSSs of the WLAN model, each specific number of \( n \) brings an SSS.

\[ c_{WLAN,n} = S^2 \cdot H \cdot P^n \cdot C, n \in [1, 256] \]  

(3.1)

In the execution of a WLAN receiver, only mode sequences following the sequence pattern of SSS will occur. Without considering a single symbol as a complete input data item, we now treat a static sub-sequence \( c \) as a valid input item. In this way radio applications have an infinite long execution of static sub-sequences. A radio application can have more than one valid mode sequence pattern. In this way multiple SSSs are allowed to occur in the execution of the MCDF model of a radio application. If the number of SSSs is \( n \), then the application is execution a input items list as equation 3.2. Each time the application will process one frame, and the frame list can be infinitely long.

\[ (c_1|c_2|...|c_n)^\infty \]  

(3.2)

### 3.3 Frame Format Detection Point

A typical radio application have multiple frame formats. Sometimes a radio application will be informed about what kind of frame is coming before start processing it. However it is also possible that the structure of this SSS remains uncertain before successfully decoding the first few symbols. For example, in the mode sequence pattern shown in equation 3.1, the value of \( n \) remains unknown until finishing decoding the header symbol. The Frame Format Detection Point (FFDP) indicate the end of the firing of the Mode Controller on which the the application knows which SSS it is processing. Because each SSS has a specific VFS solution, it is important to know from which part of an SSS that we can apply the specific VFS solution. Before the coming of the FFDP, a general VFS setting is applied.
3.4 Unrolling of MCDF Graph

As mentioned before, what is repeatedly executed in the MCDF graph is a list of static sub-sequences. However, this kind of repetition is not representative by iterations of an MCDF graph because in MCDF graph one iteration indicates only the execution of one symbol. In a MCDF graph, the processing of one SSS is needs multiple iterations. The temporal behaviors of iterations are different such that it is hard to analyze. To include the processing of a static sub-sequence in one iteration of a graph, we will build an SRDF graph for the MCDF graph where one iteration of this SRDF graph represents the processing of the entire SSS. By unrolling an MCDF graph into an equivalent SRDF with respect to a specific mode sequence, it is possible to analyze the temporal behavior and schedule the graph in terms of static sub-sequences.

In a MCDF graph, each actor might fire more than once in the processing of one SSS. When unrolling the MCDF graph into an equivalent SRDF, actors are copied multiple times such that each copy of an actor indicates one firing of the original actor when processing an SSS in the MCDF graph. In other words, for an amodal actor \( i \), the copy number \( q_i \) is equal to the length of the mode sequence \( c \) because it fires in all iterations of the original MCDF graph; for a modal actor \( j \), the copy number \( q_j \) is equal to the number of times that \( \text{mode}(j) \) appears in sequence \( c \). Let \( G \) denotes the original MCDF graph. Let \( c \) denotes a static sub-sequence. The equivalent SRDF graph is represented as \( G'(G, c) \). We define that list \( i'[q_i] \) is the list of new actors in \( G'(G, c) \) come from actor \( i \) in \( G \).

In [15] an algorithm is proposed for conversion from Multi-Rate Data Flow graph \( G'_M \) to SRDF graph \( G'_S \). In the algorithm of [15], after finishing making copies of original actors, a function \( \text{add_new_edges}(e(i, j), i'[q_i], j'[q_j]) \) is designed to add edges to the new graph. In this function, \( e(i, j) \) denotes an edges in original graph, \( i'[q_i] \) denotes a list of actors copied from actor \( i \) and \( j'[q_j] \) is the same as \( i'[q_i] \). In [15], all actors in the original graph \( G'_M \) fires when they have enough input tokens. When making new edges from \( e(i, j) \in G'_M \), all copies of actor \( i \) and \( j \) are involved.

A similar approach can be applied to unroll an SRDF graph with a specific number of iterations of this SRDF graph. Unlike the MRDF to SRDF conversion where repetition vector is calculated according to the firing rate, repetition number \( q_i \) for actor \( i \) is determined by the firing times, which is equal to the number of iterations. Same function \( \text{add_new_edges}(e(i, j), i'[q_i], j'[q_j]) \) is used to construct the equivalent edges in the new unrolled graph. Figure 3.1.a shows an example of unrolling an SRDF graph with 2 iterations. Both actor \( i \) and actor \( j \) are copied twice. Edge \( e(i, j) \) is now converted into two new edges in the new graph and all copies of actor \( i \) ( \( i'[q_i] = \{i(1), i(2)\} \) ) and actor \( j \) ( \( j'[q_j] = \{j(1), j(2)\} \) ) are involved.

However this function cannot be applied in our case directly because of the edges connecting modal actors to amodal actors. For example, \( e(\text{switch}, j) \) is an edge from a mode switch actor to a modal actor \( j \). If actor \( j \) only fires in the second iteration in the mode sequence, then only the second copy of actor \( \text{switch} \) has connection with copies of actor \( j \) instead of whole list of \( \text{switch}'(q_{\text{switch}}). \) The edge construction function for an edge \( e(i, j) \)
is changed into add_new_edges\((e(i, j), i'[e(i, j)], j'[e(i, j)])\) where \(i'[e(i, j)]\) is a sub set of \(i[q_i]\) and it only contains copies of actor \(i\) that have connection with copies of actor \(j\). An example is shown as in figure 3.1.b. When unrolling the MCDF into a SRDF with a mode sequence "\(c = 1 \cdot 2\)". Only the second copy of actor switch is connected with the copy of actor \(j\). In this case, \(\text{switch}'[e(\text{switch}, j)] = \{\text{switch}(2)\}\) and \(j'[e(\text{switch}, j)] = \{j(2)\}\).

The algorithm for unrolling an MCDF graph \(G\) into a equivalent SRDF graph \(G(G', c)\) with respect of mode sequence \(c\) is shown as Algorithm 3.1. For each copy of actor \(i\), we use an index value to indicate which iteration it belongs to in the original MCDF graph. Without getting the correct repetition actors directly, all actors are copied \(|c|\) times, where \(|c|\) is the length of the mode sequence. In this way, step 1 make copies of each actors for all iterations in the SSS.
CHAPTER 3. OVERALL APPROACH

However, not all actors are firing in all iterations of the MCDF graph, step 2 tries to
detect and remove the redundant copies of actors. If actor \( i \) has a mode value different with
the \( k^{th} \) element of the SSS, which means that actor \( i \) is not firing in iteration \( k \) of the MCDF
graph, then we delete the \( k^{th} \) copy \( i(k) \) from list \( i'[q_i] \). After that, all equivalent actors are
successfully constructed.

As mentioned above, the new function \( add\_new\_edges(e(i, j), i'[e(i, j)], j'[e(i, j)]) \) is
not using \( i'[q_i] \) and \( j'[q_j] \) as the inputs but only the sunsets of them. This is because not
all copies of \( i \) will be connected to copies of \( j \). The new inputs \( i'[e(i, j)] \) and \( j'[e(i, j)] \) only
contains parts of the copies of these two actors such that each actor in \( i'[e(i, j)] \) has at least
one connection with actors in \( j'[e(i, j)] \), and vice versa. \( i'[e(i, j)] \) and \( j'[e(i, j)] \) are then
called the Connected Subsets of actor \( i \) and \( j \) for edge \( e(i, j) \). Step 3 helps to find the
connected subsets for all edges in the original MCDF graph.

Finally, step 4 constructs all the equivalent edges in the new unrolled SRDF graph.

\[
\text{Algorithm 3.1: Unrolling an MCDF graph to an SRDF graph}
\]

| Step 1 | Copy all actors \(|c|\) times |
|--------|------------------------------|
| \( \forall i \in V_G, \forall k \in [1, |c|] : \) |
| adding actor \( i(k) \) into \( i'[q_i] \) |

<table>
<thead>
<tr>
<th>Step 2</th>
<th>Construct equivalent actors</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \forall i \in V_G, \forall k \in [1,</td>
<td>c</td>
</tr>
<tr>
<td>if ( \text{mode}(i) \neq \bot ) and ( \text{mode}(i) \neq c(k) )</td>
<td></td>
</tr>
<tr>
<td>removing actor ( i(k) ) from ( i'[q_i] )</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step 3</th>
<th>Get Connected Subsets for all edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \forall e(i, j) \in E_G, \forall k \in [1,</td>
<td>c</td>
</tr>
<tr>
<td>if ( i(k) \in i'[q_i] ) and ( j(k) \in j'[q_j] )</td>
<td></td>
</tr>
<tr>
<td>adding actor ( i(k) ) into ( i'[e(i, j)] )</td>
<td></td>
</tr>
<tr>
<td>adding actor ( j(k) ) into ( j'[e(i, j)] )</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Step 4</th>
<th>Construct equivalent edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \forall e(i, j) \in E_G : )</td>
<td></td>
</tr>
<tr>
<td>( add_new_edges(e(i, j), i'[e(i, j)], j'[e(i, j)]) )</td>
<td></td>
</tr>
</tbody>
</table>

Considering the MCDF graph as in figure 3.2. Assume that this MCDF graph has a valid
stable static sub-sequence with four symbols inside, represented by a mode sequence of
"1 · 2 · 3 · 3"

This mode sequence contains four numbers, which means the processing of this frame
needs four iterations of the MCDF graph. When converting it into a SRDF graph for one
frame, a repetition vector is required to show how many copies are needed for each actor.
Each edge in the MCDF graph is now represented by edges between the copies of source
and sink. In this SRDF graph, each actor fires exactly once when processing one frame.
Figure 3.2: Example of MCDF graph for unrolling
CHAPTER 3. OVERALL APPROACH

Figure 3.3: Conversion from MCDF to SRDF
CHAPTER 3. OVERALL APPROACH

3.5 Radio Application Classification

As mentioned before, a radio application can have more than one valid mode sequence pattern. Each static sub-sequence $c_n$ for this application can be unrolled into an SRDF graph $G'(G, c_n)$. In an iteration of the equivalent SRDF graph, the entire static sub-sequence (SSS) is processed. The application keeps executing those SRDF graphs and in each iteration of an SSS one of these equivalent graphs is chosen. Radio applications have infinite input static sub-sequences and across iteration date dependencies. To deal with these problems, overlaps between static sub-sequences need to be considered. For radio application with and without overlaps between static sub-sequences, different approaches are carried out for the VFS problem in this thesis.

3.5.1 Radio Application without Time Overlap Between Static Sub-sequences

In some radio applications such as WLAN receiver, no overlap across Static Sub-sequences is possible. Considering the processing of one static sub-sequence, it is not overlapped and influenced by other SSSs as shown in figure 3.4. In the WLAN model, all the previous of a frame must be completed before a new frame is received. Across iteration data dependencies are guaranteed to be respected. In this situation, the timing requirement is focus on the latency constraint of a static sub-sequence.

A latency requirement exists when the maximum or minimum reaction time for processing a sequence of input data is important. For the SRDF graph $G'(G, c)$, latency can be treated as the difference between the start time of the first source actor with the finish time of the last sink actor. Having a latency small enough can guarantee a sufficient throughput of this system. Deadline-constrained VFS is carried out for these applications. If we consider the processing of a mode sequence as an iteration for the unrolled SRDF graph, only one iteration here needs to be taken into account. In this way, no periodic execution analysis is need. In this iteration of SRDF graph $G'(G, c)$, actors can fire as soon as possible following the rule of worst case self-timed schedule.

![Figure 3.4: Radios without overlap between SSs](image)

A deadline constrained VFS is a VFS problem that using energy consumption when
processing the whole static sub-sequence as the objective function, while subjecting to four types of constraints:

- precedence constraints of the MCDF graph;
- voltage-frequency relation equations;
- constraints on the supported voltages for each processor;
- deadline constraints.

The first three types of constraints are already given in the previous chapter when discussing the temporal analysis of MCDF graphs. As discussed above, only one time execution of a static sub-sequence is considered here with no overlap across SSSs is allowed. Self-time schedule is used for deadline-constrained VFS so that the precedence constraints are the same as shown in equation 2.10.

A radio application needs to handle multiple sequence patterns. Each type of static sub-sequence requires a specific VFS solution because of the different structure of the STS schedule. However, radio sequence format information is involved in the first few symbols of the source sequence such that the radio sequence pattern remains unknown before the Frame Format Detection Point. In this way a stable VFS solution is needed for symbols received before the FFDP without the knowledge of the sequence structure. In the WLAN model, the FFDP is the finishing of the header symbols. Thus we need to assign a stable partial deadline for header processing for all SSSs of the WLAN model beside the global deadline for an entire SSS.

In this category, an example of WLAN 802.11a receiver is studied as an example. The details will be introduced later in the following chapter.

### 3.5.2 Radio Application with Time Overlap Between Static Sub-sequence

Figure 3.5 shows another situation where the radio application keeps processing the infinite input sequences. The throughput constraints are important for streaming applications with infinite input data items to ensure the rate of outputs for the continuity. Because of the existence of overlaps between the processing of static sub-sequences, we need to analyze the across iteration data dependencies. Throughput cannot be treated easily as the reciprocal of latency now because of overlapped execution. In figure 3.5, we assume that each static sub-sequence has the same length and the same processing time. The latency for an SSS is $t_2$. Because of the overlapped execution, the throughput now is equal to $1/(t_2 - t_1)$, which is larger than $1/t_2$.

Throughput-constrained VFS is used for radio applications with overlaps between static sub-sequences. In previous work, when modeling radio applications in SRDF graphs, VFS problems are determined by generating static periodic scheduling (SPS)[9]. By using SPS we can reduce the problem from infinite to finite. Meanwhile the cross iteration data dependencies are naturally represented by the precedence constraints in SPS. In a static periodic
scheduling, all actors inside, including the sink actor, are firing with period $T$. Thus with a time interval of $T$, an output is produced. Output generation rate $1/T$ should be larger than the throughput constraint. Section 2.1 already showed that SPS of an SRDF graph $G$ only exits if the period $T$ is larger than $MCM(G)$. For an SRDF graph, there is always an SPS to reach the maximum attainable throughput.

The solution for the VFS problem is a static periodic sequence of voltage and frequency points for each actor. This leads to a set of start time $s(i)$ and a set of execution time $t(i)$ for all actors. If we know the start time of all actors in the first firing, the entire schedule is predictable. Figure 3.6 shows an example of throughput-constrained VFS (TC-VFS) for an SRDF[9]. The minimum throughput constraint is converted into a maximum period of $4\mu s$. In figure 3.6(a), an SPS with period of $4\mu s$ is generated. Arrows inside indicate data dependencies, thus red arrows indicate where slack time occurs. After applying TC-VFS, all actors start to fire immediately when all their precedence actors finish. In the new SPS schedule, knowing the start times of $S, A, B, C$ in the first iteration of the SRDF graph (white color) gives the structure of the entire schedule.

The technique in [9] cannot be applied to our MCDF model directly. Although by the unrolling step the radio application modeled by MCDF graph $G$ has already be converted into SRDF graphs, it still need to handle multiple sequence patterns. In each iteration, one of those static sub-sequences is chosen to fire. An example is shown in figure 3.7. When considering an edge $e(i, j)$ with delays, the token consumed by $j$ in the iteration where graph $G'(G, c_n)$ is firing might be produced by actor $i$ in iteration where another SRDF graph $G'(G, c_m)$ is firing.

Each valid static sub-sequence of this MCDF graph has its VFS solution using SPS. When solving the VFS problem for one static sub-sequence, the previous SSSs on which it is dependent are uncertain. A worst-case analysis is carried out to ensure that no matter what the dependent previous SSSs are, the data dependencies constraints are followed.
In this category, an example of LTE Downlink model is studied. The details will be introduced in chapter 5.
CHAPTER 3. OVERALL APPROACH

Figure 3.7: Cross SRDF graphs data dependency
4 Wireless LAN

4.1 Overview

In a WLAN receiver, a valid static sub-sequence is called a packet. Figure 4.1 shows the structure of a WLAN 802.11a packet[12]. When a packet is detected by the RF, a Wireless LAN receiver must perform synchronization until it is successful. This is done over two long symbols with the length of 4\(\mu s\). After that, the packet header needs to be demodulated and decoded to determine the size of payload. Then payload symbols, each having a length of 4\(\mu s\), are processed by a demode-decode sequence followed by a cyclic-redundancy check (CRC). The number of payload symbols varies per packet between 1 to 256. The number of payload symbols in a packet is indicated by its header. If CRC is successful, which means all payload symbols have been correctly received, an acknowledgement symbol must be sent within 16\(\mu s\). This time guard is called Short Intra-Frame Spacing (SIFS) and it is the deadline for processing this packet.

A WLAN packet contains three parts that need to be processed. These jobs cannot be represented by a static data flow because payload has variable size and synchronization might fail. A static data flow graph cannot handle these multiple static sub-sequences. Figure 4.2 shows the MCDF model for a Wireless LAN receiver application that contains four modes: synchronization (mode 1), header processing (mode 2), payload processing (mode 3) and CRC (mode 4). Input tokens from source are stored in the FIFO of arc (“source”, “switch”), which will be sent to the corresponding output port selected by “mc”. Each iteration of the MCDF graph will consume one input token produced by the RF source. The processing of a WLAN packet starts from mode 1. After two iterations of firing mode 1, the “sync” actor...
will inform the mode controller that it succeeded in synchronization. Mode 2 fires two modal actors: "hdem" and "hdec". Several demodulation parameters are also communicated to the "pdem" actor that does the demodulation of payload symbols in mode 3. The result sent by "hdec" to "mc" indicates the length of payload symbols and informs "mc" the start of mode 3. Notice that the decision of mode controller is depended on the last input data only in mode 1 and mode 2. In mode 3 and mode 4, decisions can be taken without receiving result of the previous iteration. This is reflected by the fact that "select" can immediately generate a token after receiving a control token from the MC. This leads to the result that the mode controller can consume all control tokens needed by modes 3 and mode 4 at the same time. Once a payload symbol is received, it will be processed by "pdem" and "pdec" actors. After all payloads have been finished, "mc" will choose mode 4 for execution. Meanwhile the control
token of mode 4 will also active the consumer port of Tunnel between mode 3 and mode 4. After CRC check is finished, a response message will be sent out to the base-station through RF. Also we should notice that the source token for mode 4 is discarded because the mode 4 is not processing part of the input WLAN packet.

4.2 Temporal Requirements

Considering the MCDF graph in figure 4.2, Valid static sub-sequences follows the pattern as:

\[ c_n = S^2 \cdot H \cdot P^n \cdot C \tag{4.1} \]

where \( n \) is integer number from 1 to 256. With a specific value of \( n \), a valid SSS occurs. In the WLAN receiver, no overlap is allowed between static sub-sequences. An SSS cannot start processing until the finishing of the previous one. In this way, across SSS data dependencies are guaranteed and we only focus on the executing of one SSS. As discussed in chapter 3, the execution of one SSS \( c_n \) of the MCDF graph \( G \) can be represented by an equivalent SRDF graph \( G'(G, c_n) \). Only one iteration of this equivalent SRDF graph is considered such that the timing requirement is focus on the latency constraint of this SSS. In this section, we will carry out analysis on the latency constraints of the WLAN receiver. If we assume the start time of the target SSS is \( s(SSS) = 0 \), then the latency constraints can be treated as deadline constraints.

As already showed in figure 4.1, the global deadline for the processing of one WLAN packet is exactly 16\(\mu s\) after receiving two synchronization symbols (2\(\mu s\)), one header symbol (4\(\mu s\)), \( n \) payload symbols (\(n \times 4\mu s\)) and one source symbol for mode 4 (4\(\mu s\)).

\[ \text{ deadline} = (2 \times 4\mu s + 4\mu s + n \times 4\mu s + 4\mu s) + 16\mu s \tag{4.2} \]

where \( n \) is the number of payload symbols.

There are 256 different types of valid SSSs for WLAN model. Each of them requires a unique VFS solution because they have different equivalent SRDF graphs. When running a specific SSS practical, the hardware multiprocessor platform will configured to the corresponding VFS setting to execute this SSS. However, as mentioned in section 3.5.1, which SSS is processed remains unknown before the frame format detection point (FFDP). In the WLAN model, the size information of payload is given by the header symbol, thus the value of \( n \) is indeterminate until the header has been decoded. When processing "Synchronization" and "Header" modes, it is unknown that the VFS setting for which static sub-sequences is processed now. In this way, we need to set a general VFS solution for these two modes independent from the structures of the SSSs. In other words, specific deadline constraints are constructed for different parts of an SSS. In the following subsections we will discuss how to analyze and construct those partial deadlines.

4.2.1 Synchronization

As mentioned in section 4.1, input tokens consumed by source are stored in the FIFO of arc ("source", "switch"). The first iteration of synchronization symbol starts once the correspond-
ing source symbol is received. While executing the first iteration of mode 1 in the MCDF graph, source actor keeps firing at the same time to receive the next symbol. As shown in figure 4.3, if $t_{sync}$ is larger than $4\mu s$, which is the execution time of actor "source", the next token consumed by source will be piled in the FIFO between source and switch until "sync" finishes firing. The delay time of this source symbol is $t_{sync} - t_{src}$. During this time period, this arc needs a buffer with the same size of one source token. We should notice that the start of the second synchronization has also been delayed by the iteration of the previous symbol. In this way, delays will be added up to the next iteration. Thus if all iterations have the same amount of delay, eventually the waiting time of some source tokens will become larger than $t_{src}$. In this way, more than one source tokens will be piled in arc (source, switch). This number will increase with the execution of the WLAN packet. With long mode sequence and execution time of synchronization, a large scale buffer is need on this arc. Meanwhile the huge, uncontrolled delay will also cause high possibility of deadline missing. Considering all the reasons, the execution time of synchronization should not be larger than source. The temporal requirement for mode 1 is:

$$t_{mode1} \leq t_{src}. \quad (4.3)$$

![Figure 4.3: delay analysis of synchronization](image)

#### 4.2.2 Header

The partial deadline for mode 1 guaranteed that the processing of header can start once the header is received. Without applying DVFS, a schedule as in figure 4.4.a is carried out. Notice that before the Synchronization modes already finished before finishing receiving the header symbol due to the deadline for mode 1 proposed in previous subsection. In this way
"hdem" actor can start firing once the header symbol is received. It is clearly shown that slack time exists in this schedule, and thus DVFS can be applied to slow down actors to fulfill the slack time. The problem is how to assign the slack time to all tasks.

Figure 4.4: slack time analysis

As already discussed, the deadline for the processing of one WLAN packet is exactly $16\mu s$ after the time when the last source symbol has been received, which gives a stable global slack for all different static sub-sequences. This global slack can be assigned to all tasks. Furthermore, there exist local slacks caused by the data dependencies in the MCDF. For example, the first firing of "pdem" should wait for the finishing of "source" and "hdec". However because "hdec" finishes faster than "source", the time interval between "hdec" and "pdem" becomes slack time that can be used. We now define the local slack time.

Local Slack is defined as follows. A MCDF graph $G = (V, E, t, d, M, mode, atype)$ is given where $V$ is the set of actors, $E$ is the set of arcs, and $f(n)$ is the finishing time actor $n$. For an actor $j$, let $A(j)$ be the set of precedence actors defined as:

$$A(j) = \{i \in V : (i, j) \in E\}. \tag{4.4}$$

The local slack caused by actor $j$ is:

$$local\_slack(j) = \sum_{i \in A} (\max_{i \in A}(f(i)) - f(i)). \tag{4.5}$$
Unlike global slack, local slack can only be used by the corresponding actors. In this way we can divide the timing extension into two steps. First filling the local slacks to get a compact schedule. Then assign the global slack to time blocks that contain more actors running than others. Thus the maximum conversion rate from time extension to energy saving can be reached.

Figure 4.4.b shows the result after implementing step one. It is shown that in header mode, only one actor is running at the same time. However in payload mode, "pdem" and "pdec" become a pipeline execution. In this way, the power consumption is bigger in mode 3 than that in mode 2, thus the global slack should be assigned to mode 3 rather than the header mode. Aiming at only using the local slack for mode 2, the temporal requirement for mode 2 is:

\[ t_{mode2} \leq t_{src}. \]  \hspace{1cm} (4.6)

### 4.2.3 Payload and CRC

In the previous discussion, we already came to the conclusion that payload processing can start immediately when the first payload symbol is received. After filling all the local slacks, a pipeline execution of "pdem" and "pdec" is reached. Hence besides local slack, global slack is also used for this part. The only temporal constraint that should be considered in this part is the global deadline.

### 4.3 DVFS

As mentioned in chapter 3, DVFS is applied to one time execution of specific static sub-sequence. Deadline-constrained VFS is carried out for the WLAN application. A deadline-constrained VFS is a VFS problem that uses the energy consumption as the objective function while subjecting the solution to four types of constraints:

-precedence constraints of the MCDF graph;

-voltage-frequency relation equations;

-constraints on the supported voltages for each processor;

-deadline constraints.

To construct the algorithms for the deadline-constrained VFS problems, those four types of constraints need to be transformed into detailed formulas according to the specific type of VFS.

In this thesis, we studied deadline-constrained VFS evaluating the impacts of continuous or discrete frequency levels, and of local VFS switches or a global VFS switch.
CHAPTER 4. WIRELESS LAN

4.3.1 Continuous Local VFS

In MCDF, an amodal actor fires in all iterations while modal actors only fire in the corresponding iteration of the same mode. The start time $s(i, k, c)$ and frequency $f(i, k, c)$ of actor $i$ in iteration $k$ of mode sequence $c$ is only defined for iterations where $c(k) = \text{mode}(i)$. This is also applied to the dynamic voltage level $v(i, k, c)$. Actor $i$ is mapped to processor $\pi(i)$. According to the power model proposed in section 2.4, $v_b$, $f_b$ indicate the base voltage level and the base frequency level, while $P_{HH}(\pi(i))$ indicates the power consumption of processor $\pi(i)$ when running at the base frequency. Thus the energy consumption is determined by:

$$E = \sum_{\forall k \in c} \sum_{\forall i \in k} t_{\text{orig}}(i) \times \frac{P_{HH}(\pi(i))}{v_b^2} \times v(i, k, c)^2.$$  \hspace{1cm} (4.7)

And the execution time of actor $i$ is represented as $t(i, k, c) = t_{\text{orig}}(i) \times \frac{f_b}{f(i, k, c)}$.

As discussed above, four types of constraints need to be subjected to here. Self-timed scheduling is applied here. Hence the optimization problem should follow the precedence constraints of equation 2.7 and 2.9. Meanwhile, the voltage-frequency relations, the constraints on the supported voltages for each processor on the system is following the equations proposed in section 2.4. The partial deadline constraints for WLAN packet are already analyzed in the previous section. We should notice that, for non-scalable actors $V_s$, such as "source", frequency level is equal to the base frequency. CLVFS can be modeled as algorithm 4.1.

However, we should notice that the voltage-frequency relation is a non-linear function. This equation makes the VFS problem not a DCP problem anymore such that it is not solvable by the DCP solver. To form it as a disciplined convex programming, parameters of voltage should be removed. Considering the equation:

$$f = K \times \frac{(v - v_{th})^2}{v}$$  \hspace{1cm} (4.8)

where $v_{th} = 0.55$ and $0.6 \leq v \leq 1.1$, $f$ is monotonically increasing with voltage level $v$. Hence the objective function of energy consumption can be replaced by:

$$E_{\text{solver}} = \sum_{\forall k \in c} \sum_{\forall i \in k} t_{\text{orig}}(i) \times P_{HH}(\pi(i)) \times \frac{f(i, k, c)^2}{f_b^2}.$$  \hspace{1cm} (4.9)

The voltage-frequency relations constraints can be removed. In this way, the correct frequency setting for all tasks and all iterations is obtained without getting the real energy consumption. The new optimization problem gives answer to the correct frequency setting, another step is need to calculate the corresponding voltage setting for each actor using equation 4.8. After that, the correct energy consumption is giving by the original equation as shown in 4.7.

4.3.2 Discrete Local VFS

In this case, we have a discrete frequency set $F$. Processors can only execute on frequency levels belonging to this set. Because an individual frequency may not be optimized, we allow
CHAPTER 4. WIRELESS LAN

CLVFS

\[
\text{minimize: } \sum_{k \in C} \sum_{i \in k} t_{\text{orig}}(i) \times P_{HH}(\pi(i)) \times \frac{v(i,k,c)^2}{v_b^2}
\]

subject to:

1. Precedence constraints
\[
\forall (i, j) \in E, \text{emode}(i, j) = \bot, \forall k \in [0, ||c|| - 1]: \\
s(j, k, c) - s(i, k - d(i, j), c) \geq t(i, k - d(i, j), c)
\]
\[
\forall (i, j) \in E, \text{emode}(i, j) \neq \bot, \forall k \in [0, ||c|| - 1]: \\
s(j, k, c) - s(i, k - \delta(k, c, d(i, j)), c) \geq t(i, k - \delta(k, c, d(i, j)), c)
\]

2&3. Voltage-frequency relations and supply voltage range
\[
\forall i \in V: \\
\frac{f(i,k,c)}{f_b} = \frac{(v(i,k,c) - v_d)^2/v(i,k,c)}{(v_b-v_d)^2/v_b} \\
v_{\text{min}}(\pi(i)) \leq v(i, k, c) \leq v_d \\
s(i, k, c) \geq 0
\]
\[
\forall i \in V_S: \\
s(i, k, c) \geq 0, \quad f(i, k, c) = f_b
\]

4. Deadline constraints
\[
\forall k \in [0, 2]: \\
t_{\text{end of iteration}_k} \leq t(\text{source}) \times (k + 2)
\]
\[
\forall k = ||c|| - 1: \\
t_{\text{end of iteration}_k} \leq t(\text{source}) \times (k + 1) + 16 \mu s
\]

Algorithm 4.1: Continuous local VFS

Each actor to switch between frequency levels in \( F \) per firing. Section 2.4 already gives the equation to calculate the energy consumption for a firing of an actor using a discrete frequency set:

\[
E(i, k, c) = \sum_{f \in F} (t_{\text{orig}}(i) \times P(\pi(i), f) \times \frac{f_b}{f} \times u(i, k, c, f)) \\
= t_{\text{orig}}(i) \times \sum_{f \in F} (P(\pi(i), f) \times \frac{f_b}{f} \times u(i, k, c, f)). \tag{4.10}
\]
Each firing of an actor is divided into several parts and each part has its own frequency setting, \( u(i, k, c, f) \) indicates the utilization of frequency level \( f \) for actor \( i \) in iteration \( k \) of mode sequence \( c \). Summing all parts of an actor running on different frequency levels makes an entire task. In this way, the sum of utilizations of all possible frequencies should be 1.

\[
\sum_{\forall f \in F} u(i, k, c, f) = 1. \tag{4.11}
\]

The execution time of actor \( i \) is represented as:

\[
t(i, k, c) = t_{\text{orig}}(i) \times \sum_{\forall f \in F} \left( \frac{f_b}{f} \times u(i, k, c, f) \right). \tag{4.12}
\]

The problem is once again subjected to the four types of constraints we mentioned before. Again, self-timed scheduling is applied here. Hence the optimization problem should follow the precedence constraints of equation 2.7 and 2.9. Meanwhile, the voltage-frequency relations, the constraints on the supported voltages for each processor on the system is following the equations proposed in section 2.4. The partial deadline constraints for WLAN packet are already analyzed in the previous section. For non-scalable actors belong to set \( V_s \), such as "source", only the base frequency can be used, which means that the utilization of base frequency \( f_b \) is 1 and utilizations of other frequency levels are 0. This is represented by constraint \( u(i, k, c, f_b) = 1 \). DLVFS can be modeled as in algorithm 4.2.

Since all constraints are linear, DLVFS is modeled as a linear programming problem. It can be optimally solved.

### 4.3.3 Discrete Global VFS

As mentioned in section 2.3, many practical multiprocessors only have one single global VF-switch. With a global VF-switch, tasks running at the same time on different processors must switch frequency simultaneously. Without voltage and frequency changes within the execution of one task, we only allow voltage and frequency changes when it comes to the end of one parallel block. A parallel block is the shortest time interval of a schedule within which the set of running task remains constant. By assigning exactly the same frequency setting for all tasks in one parallel block within the time interval of this block, we can ensure that all processors have the same speed at the same time. The DGVFS solution can be found in three steps:

(1) Find a compacted schedule for mode sequence \( c \).
(2) Identify all the parallel blocks in the schedule.
(3) Perform the VFS for all parallel blocks.

In the first step we want to gain a compact schedule with minimum local slack time. In section 4.2.2, we already discussed that by minimizing the local slacks, a compact schedule can be reached. In other words, the first step is aiming at minimizing local slacks in the original STS schedule. An example is shown in figure 4.5 where SSS \( c = SSHPP \) is executed.
**CHAPTER 4. WIRELESS LAN**

**DLVFS**

minimize: \( \sum_{k \in c} \sum_{i \in k} t_{orig}(i) \times \sum_{f \in F} (P(\pi(i), f) \times \frac{b}{F} \times u(i, k, c, f)) \)

subject to:

1. **Precedence constraints**

   \( \forall (i, j) \in E, \text{emode}(i, j) = \bot, \forall k \in [0, ||c|| - 1] : \)

   \[ s(j, k, c) - s(i, k - d(i, j), c) \geq t(i, k - d(i, j), c) \]

   \( \forall (i, j) \in E, \text{emode}(i, j) \neq \bot, \forall k \in [0, ||c|| - 1] : \)

   \[ s(j, k, c) - s(i, k - \delta(k, c, d(i, j)), c) \geq t(i, k - \delta(k, c, d(i, j)), c) \]

2&3. **Voltage-frequency relations and supply voltage range**

   \( \forall i \in V : \)

   \[ t(i, k, c) = t_{orig}(i) \times \sum_{f \in F} (\frac{b}{F} \times u(i, k, c, f)) \]

   \[ \sum_{f \in F} u(i, k, c, f) = 1 \]

   \[ s(i, k, c) \geq 0 \]

   \( \forall i \in V_S : \)

   \[ s(i, k, c) \geq 0, u(i, k, c, f_b) = 1 \]

4. **Deadline constraints**

   \( \forall k \in [0, 2] : \)

   \[ t_{\text{end of iteration } k} \leq t(\text{source}) \times (k + 2) \]

   \( \forall k = ||c|| - 1 : \)

   \[ t_{\text{end of iteration } k} \leq t(\text{source}) \times (k + 1) + 16 \mu s \]

---

Algorithm 4.2: Discrete local VFS

The black arrows indicate the deadlines in this graph. Figure 4.5 shows that local slacks exist in the original schedule. As we can see, local slack is mainly caused by the long execution time of the source actors comparing to other actors. In the WLAN model specially, the MCM of each sub-graph of the MCDF model is determined by the source actor. An easy method to fulfill the local slack caused by the source actor is to increase the execution time of other actors such that the MCM without calculating source actors can reach the same level with the original MCM with source actors. If we use \( G' \) to represent a copy of graph \( G \) but without
the source actors. Then execution time of actor $i$ is now refreshed as:

$$t(i) = t(i) \times \frac{MCM(G)}{MCM(G')}.$$  \hfill (4.13)

In an MCDF graph, each mode has a specific sub-graph with its unique MCM. Furthermore, since the existence of tunnel between different modes, overlap across different modes is not allowed. In this way, actors with different mode have different extension multiple factor.

Step 2 helps to identify all the parallel blocks in the updated schedule. After that, step 3 applies the discrete VFS problem on those PBs with respect to the deadline constraints. Figure 4.5 shows a example of applying these three steps on an MCDF graph with a static sub-sequence $c = SSH P P$. Step 1 is using the local slacks and step 3 is using the global slack. As discussed in section 4.2.3, global slack is only assigned to payload mode.

---

**DGVFS**

**step 1: find a compact schedule**

- Let $G_m$ denote the sub-graph of mode $m$
- Let $G'_m$ denote a copy of $G_m$, where $t(i) = 0, \forall i \in V_S \in G'_m$
- Let $M$ denote the set of all modes in $G$

  for each $m \in M$
  
  - if $\frac{MCM(G_m)}{MCM(G'_m)} \geq \max_{f \in F} \frac{f_0}{f}$
    
    $$mul(m) = \max_{f \in F} \frac{f_0}{f}$$
  
  - else if $\frac{MCM(G_m)}{MCM(G'_m)} \leq 1$
    
    $$mul(m) = 1$$
  
  - else
    
    $$mul(m) = \frac{MCM(G_m)}{MCM(G'_m)}$$
  
  for each $i \in G$ && mode$(i) = m$
  
  $$t(i) = t(i) \times mul(m)$$

  end for

  end for

  Compute a new schedule for the updated graph $G$

---

**step 2: identify parallel blocks PB**

  for any time $t \in Schedule$

  - if Parallelism at $t \neq$ parallelism at $t + \Delta t (\Delta t \rightarrow 0)$
    
    Register parallel block in PB

  end for

  remove unscalable tasks in all parallel blocks

  for each parallel block $j \in PB$

    for each frequency level $f \in F$

      $$p_{pb}(j, f) = \sum_{\forall i \in j} p(\pi(i), f)$$

    end for

  end for

---

47
4.4 Experiments and Results

We apply three DVFS algorithms to the MCDF graph of Wireless LAN baseband receiver shown in Figure 4.2. Experiments are conducted for five different mode sequences with variant length of payload symbols. We apply exactly same deadline constraints together with same MCDF model and hardware mapping relationship for each experiment. The solver for convex/linear programming is CVX, a Matlab-based modeling system for convex optimization.
CHAPTER 4. WIRELESS LAN

step 3: DGVFS algorithm

minimize: \( \sum_{j \in PB} \frac{t(j)}{mul(mode(j))} \times \sum_{f \in F} (P_{fb}(j, f) \times \frac{f_b}{T} \times u(j, f)) \)

subject to:

\( \forall j \in PB, \text{ finish time}(j) \leq t(source) \times 4 : \)
\( \sum_{f \in F} \left( \frac{f_b}{T} \times u(j, f) \right) = mul(mode(j)) \)

\( \forall j \in PB, \text{ start time}(j) \geq t(source) \times 4 : \)
\( t(j) = \frac{t^{(i)}}{mul(mode(j))} \times \sum_{f \in F} \left( \frac{f_b}{T} \times u(j, f) \right) \)
\( \sum_{j \in PB} t(j) \leq \text{global deadline} - t(source) \times 4 \)

Algorithm 4.3: Discrete global VFS

Table 4.1: Results for WLAN

<table>
<thead>
<tr>
<th>Mode sequence</th>
<th>Energy Reduction (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CLVFS</td>
</tr>
<tr>
<td>( S^2 H^1 P^1 C^1 )</td>
<td>48.79</td>
</tr>
<tr>
<td>( S^2 H^1 P^2 C^1 )</td>
<td>49.35</td>
</tr>
<tr>
<td>( S^2 H^1 P^{10} C^1 )</td>
<td>50.52</td>
</tr>
<tr>
<td>( S^2 H^1 P^{100} C^1 )</td>
<td>50.88</td>
</tr>
<tr>
<td>( S^2 H^1 P^{256} C^1 )</td>
<td>50.89</td>
</tr>
</tbody>
</table>

The results shown in Table 4.1 uses the power model carried out in section 2.4. For discrete VFS, five frequency/voltage levels are supported by the hardware platform, which gives the highest possibility of time extension of 16 times longer. However, we must consider the fact that in a real application, the multi-processor system might only contain two levels to choose. Besides the base frequency, we need to choose another suitable frequency level. This voltage/frequency level cannot be too high otherwise it gives a very low scaling possibility, which leads to a bad energy reduction result. Hence we choose modes \( H_{High} \) and \( Low \) in our experiment. Table 4.2 shows the result of energy reduction using two discrete levels \( H_{High} \) and \( Low \).

We also consider static VFS as a control group. By static VFS we mean each processor has a stable speed and this speed remains unchanged during executing. The value of this speed is the best case (slowest) speed to reach the deadline constraints. From figure 4.6 we can find that dynamic VFS gives a significant energy saving by around 50%. All DVFS algorithms gives better energy reduction than static VFS, this is because in static VFS, the speed for each processor will be limited by some actors with long execution time but small slack time. These actors decide the maximum expansion level. Thus, for actors with short execution time but long slack time, still much slack time remains in the new schedule.
### Table 4.2: Results for WLAN (Discrete level: High and Low)

<table>
<thead>
<tr>
<th>Mode sequence</th>
<th>Energy Reduction (%)</th>
<th>DLVFS-2 levels</th>
<th>DGVFS-2 levels</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S^2H^1P^1C^1$</td>
<td>41.15</td>
<td>38.83</td>
<td></td>
</tr>
<tr>
<td>$S^2H^1P^2C^1$</td>
<td>42.35</td>
<td>40.36</td>
<td></td>
</tr>
<tr>
<td>$S^2H^1P^{10}C^1$</td>
<td>46.28</td>
<td>45.33</td>
<td></td>
</tr>
<tr>
<td>$S^2H^1P^{100}C^1$</td>
<td>49.30</td>
<td>49.17</td>
<td></td>
</tr>
<tr>
<td>$S^2H^1P^{256}C^1$</td>
<td>49.61</td>
<td>49.56</td>
<td></td>
</tr>
</tbody>
</table>

The results also show that CLVFS brings the highest energy savings, followed by DLVFS and DGVFS. This difference is in line with the previous assumption that CLVFS offers an upper bound of energy saving. The reason why global VFS offers worse numbers is that in order to make sure all processors have same speed at the same time, it cannot fulfill all local slack in the compact schedule in the first step.

Another conclusion from the results is that with more payload symbols in one frame, better energy saving is given when using dynamic VFS algorithms, while worst energy saving is given when using static VFS. In static VFS, synchronization mode and header mode provide larger cycle mean with less local slack. The VFS solution is limited by the first two modes, thus when it comes to payload mode, a low energy reduction value occurs. With longer payload, the overall result becomes worse. However, in dynamic VFS, different iteration allows different VFS solution. Furthermore, global slack is also used in this part, which gives payload mode higher energy saving than other modes. With the increasing of payload length, dynamic VFS gives better energy savings.
Figure 4.6: energy reduction for results WLAN
5 Long Term Evolution

5.1 Overview

Long Term Evolution (LTE) is a standard of high-speed wireless communication for mobile phone and data terminals. The frame structure we studied in this thesis is based on Frequency Division Duplexing (FDD) mode. In FDD mode, the 10 ms radio frame is divided into 20 slots with a length of 0.5 ms. Two slots build up a subframe. Thus one frame contains 10 subframes with the length of 0.1 ms[1]. In this thesis, a valid input data item that is represented by an SSS is a subframe.

![Frame Structure of LTE](image)

Figure 5.1: Structure of LTE frame

In our model, one LTE downlink slot consists of 7 symbols. Figure 5.1 shows the structure of one subframe with 14 symbols. Different colored blocks indicate downlink physical channels to carry information blocks received from MAC and higher layers. The Physical Control Format Indicator Channel (PCFICH) appears in the first symbol. It carries the Control Frame Indicator (CFI), which indicates the number of symbols used for control channel transmission in each subframe. The Physical Downlink Control Channel (PDCCH) contains information of the Downlink Control Information (DCI) message, which helps assigning resources. The number of PDCCH symbols is determined by the CFI symbol. The total number \(N\) of both types of control symbols varies from 2 to 4. The Physical Downlink Shared Channel (PDSCH) is the main data channel in LTE. It appears in the last
(14 − N) symbols of a subframe.

Since that the DCI has variable size, an MCDF graph is modeled for the LTE DL instead of a static data flow model. Figure 5.2 is the MCDF model for LTE DL model[17]. An LTE downlink subframe contains three parts to process. However only three basic modes is not enough when modeling in MCDF graph. LTE uses reference symbols for channel estimation. For a specific type of symbols, it is uncertain whether reference symbol is needed. The other problem is that when executing the last symbol of control symbols or data symbols, extra functionality of sending message to coming mode is needed. Considering all situations, 7 modes is proposed to model LTE DL model in MCDF graph as shown in Table 5.1.

<table>
<thead>
<tr>
<th>Mode</th>
<th>Symbols</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>PCFICH+PDCCH</td>
<td>Reference</td>
</tr>
<tr>
<td>2</td>
<td>PDCCH (Not final)</td>
<td>No reference</td>
</tr>
<tr>
<td>3</td>
<td>PDCCH (Final)</td>
<td>No reference</td>
</tr>
<tr>
<td>4</td>
<td>PDCCH (Final)</td>
<td>Reference</td>
</tr>
<tr>
<td>5</td>
<td>PDSCH (Not final)</td>
<td>No reference</td>
</tr>
<tr>
<td>6</td>
<td>PDSCH (Not final)</td>
<td>Reference</td>
</tr>
<tr>
<td>7</td>
<td>PDSCH (Final)</td>
<td>No reference</td>
</tr>
</tbody>
</table>

A LTE DL subframe contains three parts, PCFICH, PDCCH and PDSCH and summing them together makes 14 symbols. As discussed before, each symbol needs a control token to guild the MCDF graph to choose the correct mode. An static sub-sequence (SSS) is sequence of control tokens for a specific subframe structure. A valid SSS in the LTE model can be expressed as:

\[ c_N = PCFICH \cdot PDCCH^{N-1} \cdot PDSCH^{14-N}, \]  

where \( N \) indicates the CFI value that varies from \( \{2, 3, 4\} \). Because of the different subframe formats, the LTE model has three valid SSSs. Considering the 7 modes above, all SSSs is transformed into the SSSs as shown in table 5.2.

<table>
<thead>
<tr>
<th>CFI</th>
<th>Static Sub-Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>( c_2 )</td>
<td>1 3 5 5 5 6 5 5 5 5 6 5 5 6 7</td>
</tr>
<tr>
<td>( c_3 )</td>
<td>1 2 3 5 5 6 5 5 5 5 6 5 5 6 7</td>
</tr>
<tr>
<td>( c_4 )</td>
<td>1 2 2 4 5 6 5 5 5 5 6 5 5 6 7</td>
</tr>
</tbody>
</table>
Figure 5.2: MCDF graph for LTE DL model
5.2 Temporal Requirements

The LTE Downlink is a radio application with infinite input subframes. In this model, overlap between subframes is allowed such that we need to analyze the cross subframes data dependency. Each subframe is represented by a specific SSS. As mentioned in chapter 3, the execution of one SSS $c_N$ of the MCDF graph $G$ can be unrolled into an equivalent SRDF graph $G'(G, c_N)$. One iteration of the equivalent SRDF graph indicates the entire processing of one SSS. Throughput-constrained VFS (TC-VFS) is used for radio application with overlaps between SSSs. In previous chapters, we already know that TC-VFS is determined by generating a static periodic scheduling (SPS). By using SPS, we can reduce the problem from infinite to finite. Meanwhile the cross SSSs data dependencies are naturally represented by the precedence constraints of SPS. Each valid SSS of this MCDF model has its own equivalent SRDF graph and VFS solutions using SPS. When solving the VFS problem for one SSS, the dependent previous SSSs are uncertain. A worst-case analysis is carried out to ensure no matter what the dependent previous SSSs are, the precedent constraints are followed. The detail will be discussed in the next section.

By using SPS, the rate of outputs for SSSs are associated with the period $T$. Although each equivalent SRDF graph $G'(G, c_N)$ has its own SPS solution, we assume that all of them have the same period $T$. This is because we want to guarantee that no matter what kind of SSS list occurs, the throughput of the LTE model is stable.

The LTE DL subframe has a length of 1 ms. Figure 5.3 shows an example of the periodic execution of the SSSs with the same period $T$. If the period $T$ is larger than the subframe length, the executions of the following subframes will be delayed. These delays will be accumulated and eventually become arbitrary large. To eliminate the influence of delays, the execution rate should catch up the rate of income SSSs. In this way, the maximum possible period is $T = 1ms$.

Figure 5.3: Latency analysis of LTE subframes
5.3 DVFS

In section 3.2.2 we already discussed the feasibility of solving the TC-VFS problem by generating static periodic scheduling. In the LTE MCDF graph $G$, what is repeating periodically is the subframes, each represented by a static sub-sequence. An SSS $c_N$ can be modeled by an equivalent SRDF graph $G'(G, c_N)$ by unrolling the MCDF graph. For each equivalent SRDF graph, an SPS following the throughput constraint can be assigned by the algorithms in [9]. The VFS solution for SSS $c_N$ containing information of start time $s_{G'(G,c_N)}(i)$ and execution time $t_{G'(G,c_N)}(i)$ for all actor $i$ in the SRDF graph $G'(G, c_N)$. By knowing the start time and execution time variables for actors in graph $G'(G, c_N)$, correct temporal behavior in the infinite SPS schedule for graph $G$ is predictable.

However, the solution above just considers the situation where only one type of subframe keeps executing without the cross iteration data dependencies between different SSSs. In the LTE model, three types of SSS exist, namely $c_2, c_3, c_4$ with respect to the CFI value. Thus three equivalent SRDF graphs are modeled for the LTE MCDF graph: $G'(G, c_2), G'(G, c_3)$ and $G'(G, c_4)$. In one iteration, one of these SRDF graph is chosen to fire. When analyzing an edge with delays in the SRDF graph $G'(G, c_N)$, it actually has data dependencies with the execution of the SRDF graph chosen in another iteration. An example is shown in figure 3.7. When considering an edge $e(i, j)$ with delays, the token consumed by $j$ in the iteration where graph $G'(G, c_n)$ is firing might be produced by actor $i$ in iteration where another SRDF graph $G'(G, c_m)$ is firing.

The cross SSS data dependencies are modeled by edges with delay as shown in equation 2.4. For edge $e(i, j)$, since SPS is applying, $s(i, k - d(i, j))$ is equal to $s(i, k) - T \times d(i, j)$. The precedence constraint for one SRDF graph only is represented as 2.6. Worst case analysis is carried out here to take all transition possibilities into account. In other words, no matter which SRDF graph $G'(G, c_n)$ is firing in iteration $k$, we consider all possibilities of SSSs in iteration $k - d(i, j)$ and take the worse case of start time and execution time of actor $j$. Hence, it is sure that no matter what kind of transition occurs, the VFS solution for actor $j$ always respects any the temporal requirements. This solution is quite similar with the normal DVFS using SPS schedule. For each equivalent SRDF graph $G'(G, c_N)$ indicate an SSS, all actors belonging to this graph have their start time $s_{G'(G,c_N)}(i)$ and execution time $t_{G'(G,c_N)}(i)$. In the overall situation, no matter which SSS is processing in iteration $k$, the temporal behavior of actors in this iteration can be calculated by the corresponding variables for this subframe graph.

For example, is we only consider the execution of graph $G'(G, c_2)$, the precedence constraint is shown as equation 5.2.

$$s_{G'(G,c_2)}(j) \geq s_{G'(G,c_2)}(i) + t_{G'(G,c_2)}(i) - T \times d(i, j).$$  (5.2)

As discussed before the dependent firing of actor $i$ might not happen in an iteration of
In discrete local VFS each actor runs with a discrete frequency set \( F \) instead of a stable frequency \( f(i) \) in one time firing. At each firing, actor \( i \) has been divided into parts with different frequencies. Similar with the function we proposed for WLAN model, the energy consumption for actor \( i \) in SRDF \( G \) is:

\[
\sum_{\forall G'(G,c_n) \in ME} \sum_{\forall i \in G'(G,c_n)} P_{HH}(\pi(i)) \times \frac{f_{G'(G,c_n)}(i)^2}{f_b^2}
\]
In section 4.3.3 three basic steps are proposed for global DVFS algorithm: the model in section 2.4. Constraints are not involved in TC-DLVFS. Power consumption values are already given in minimum energy consumption. The voltage-frequency relation constraints and voltage range are not involved in CLVFS. The goal is to optimally set workload for each task onto different frequency levels with the scaled execution time of actor $i$.

In the algorithm 5.2 we give a linear programming formulation for throughput-constrained DLVFS. Again, the sum of utilization of all possible frequencies for one actor should be equal to 1. The scaled execution time of actor $i$ is represented as:

\[
E_{G'(G,c_n)}(i) = \sum_{v \in F} (t_{orig}(i) \times P(\pi(i), f) \times \frac{f_b}{f} \times u_{G'(G,c_n)}(i, f))
\]

\[
= t_{orig}(i) \times \sum_{v \in F} (P(\pi(i), f) \times \frac{f_b}{f} \times u_{G'(G,c_n)}(i, f))
\]  \hspace{1cm} (5.6)

Again, the sum of utilization of all possible frequencies for one actor should be equal to 1. The scaled execution time of actor $i$ is represented as:

\[
t_{G'(G,c_n)}(i) = \sum_{v \in F} (\frac{f_b}{f} \times u_{G'(G,c_n)}(i, f))
\]  \hspace{1cm} (5.7)

In the algorithm 5.2 we give a linear programming formulation for throughput-constrained DLVFS. The goal is to optimally set workload for each task onto different frequency levels with minimum energy consumption. The voltage-frequency relation constraints and voltage range constraints are not involved in TC-DLVFS. Power consumption values are already given in the model in section 2.4.

### 5.3.3 Discrete Global VFS

In section 4.3.3 three basic steps are proposed for global DVFS algorithm:

**CHAPTER 5. LONG TERM EVOLUTION**

**CLVFS**

minimize: \[
\sum_{G' \in (G,c_n) \in ME} \sum_{i \in G'(G,c_n)} t_{orig}(i) \times P_H(\pi(i)) \times \frac{v_{G'(G,c_n)(i)}}{v_b^2}
\]

subject to:

\[\forall (i, j) \in E_{G'(G,c_n)} \land \forall G'(G,c_n) \in ME : \]

\[s_{G'(G,c_n)}(j) \geq \max_{\forall G'(G,c_n) \in ME \land (i,j) \in E_{G'(G,c_n)}} \{s_{G'(G,c_n)}(i) + t_{G'(G,c_n)}(i) - T \times d(i,j)\}\]

\[\forall G'(G,c_n) \in ME \land \forall i \in V_{G'(G,c_n)}, \]

\[f_{G'(G,c_n)}(i) = \frac{(v_{G'(G,c_n)}(i) - v_b)^2}{2/v_{G'(G,c_n)}(i)}\]

\[v_{min}(\pi(i)) \leq v_{G'(G,c_n)}(i) \leq v_d\]

\[s_{G'(G,c_n)}(i) \geq 0\]

\[\forall G'(G,c_n) \in ME \land \forall i \in V_{G'(G,c_n)} - S, \]

\[s_{G'(G,c_n)}(i) \geq 0, f_{G'(G,c_n)}(i) = f_b\]

Algorithm 5.1: Continuous local VFS

\[E_{G'(G,c_n)}(i) = \sum_{v \in F} (t_{orig}(i) \times P(\pi(i), f) \times \frac{f_b}{f} \times u_{G'(G,c_n)}(i, f))\]

\[= t_{orig}(i) \times \sum_{v \in F} (P(\pi(i), f) \times \frac{f_b}{f} \times u_{G'(G,c_n)}(i, f))\]  \hspace{1cm} (5.6)

In the algorithm 5.2 we give a linear programming formulation for throughput-constrained DLVFS. The goal is to optimally set workload for each task onto different frequency levels with minimum energy consumption. The voltage-frequency relation constraints and voltage range constraints are not involved in TC-DLVFS. Power consumption values are already given in the model in section 2.4.
CHAPTER 5. LONG TERM EVOLUTION

DLVFS

\[
\begin{align*}
\text{minimize: } & \sum_{G' \in ME} \sum_{i \in G'} t_{\text{orig}}(i) \times \sum_{f \in F} (P(\pi(i), f) \times \frac{p_f}{T} \times u_{G'}(i, f)) \\
\text{subject to: } & \forall (i, j) \in E_{G'}(c_n) \wedge \forall G' \in ME, \\
& s_G(j) \geq \max_{G' \in ME \setminus \{(i, j) \in E_{G'}(c_n)\}} \{s_{G'}(i) + t_{G'}(i) - T \times d(i, j)\} \\
& \forall G' \in ME \setminus \forall i \in V_{G'}(c_n), \\
& t_{G'}(i) = t_{\text{orig}}(i) \times \sum_{f \in F} (\frac{p_f}{T} \times u_{G'}(i, f)) \\
& \sum_{f \in F} u_{G'}(i, f) = 1 \\
& s_{G'}(i) \geq 0 \\
& \forall G' \in ME \setminus \forall i \in V_{G'}(c_n) \_S, \\
& s_{G'}(i) \geq 0, u_{G'}(i, f_b) = 1 \\
\end{align*}
\]

Algorithm 5.2: Discrete local VFS

1. Find a compacted schedule for an SSS \(c_n\).
2. Identify all the parallel blocks in the schedule.
3. Perform the VFS for all parallel blocks.

The main difference between WLAN instance and LTE is that deadline-constrained VFS is assigned for WLAN instance while for LTE model we are applying throughput-constrained VFS. Overlaps between SSSs are allowed here. In this way, it is not possible to apply the same algorithm as for the WLAN model because there will be multiple executions of SSSs in one block. The detail implementation of all three steps need to be adjusted for throughput-constrained VFS problem for the MCDF graph model.

In the first step we are aiming at gaining a compact schedule with minimum local slack time. The SPS is used for the throughput-constrained VFS problem. In each iteration, an equivalent SRDF graph \(G'(G', c_n)\) for the SSS \(c_n\) is chosen to fire. As discussed in section 4.2.2, minimize local slack time need to extent the execution time of some actors. The existence of overlap across SSSs limit the possibility of changing execution time of actors by modes, which is suitable in WLAN model. However, the sources actors are always executing and having the same temporal behavior for different SSSs. It is possible to shrink the source actors for all SSSs at the same rate such that in the resulting graph, source actors are not involved in the critical cycles which generate the MCM. By doing so, local slack time caused by source actors is reduced.
A compacted schedule with a shorter period appears after step one. The compact schedule is still following static periodic execution of SSSs thus it can be divided into minimum periodic regions. In the SPS of one SRDF graph, all periodic regions are exactly the same because every part of the schedule is executing in a same period. As shown in figure 5.4, the executing of one sequence is divided into three parts. If we are only processing one SSS infinitely, then it is possible to relocate parts of the execution of one SSS into one periodic region. This also means that all periodic regions contain the same information and by analyzing one of them we can built the correct VFS problem.

![Figure 5.4: Periodic region](image)

In fact the application need to handle multiple SSSs. In each iteration one SSS is executed, hence one periodic region contains execution of multiple SSSs. In the example of Figure 5.5, the periodic region contains execution of the first three SSSs. In fact if there are \( p \) SSSs involved in one periodic region, and there are \( m \) different SSS formats, the number of periodic regions is:

\[
n(pr) = m^p
\]

The number of SSS formats is determined by application standard, however the number
of parallel SSSs is depending on the specific schedule. Actually, the number of SSSs parallel-ling with the SSS in iteration $k$ is equaling to the number of iterations started within the execution time interval of iteration $k$. In the SPS in this chapter, all SSSs are executed with the same latency and period, thus the number of parallel SSSs is:

$$nr_{of\_parallel} = \left\lceil \frac{\text{latency}}{\text{period}} \right\rceil$$ (5.9)

Each type of periodic region is then divided into parallel blocks. Global discrete DVFS problem is then built for each type of periodic region with the throughput constraint. As discussed before, minimum throughput constraint equals to maximum period constraint in SPS schedule. Therefore, for each periodic region, a partial latency-constrained DVFS is built with the target period $T$. Algorithm 5.3 shows the global discrete VFS algorithm for LTE model. Again, $ME$ denotes the set of equivalent SRDF graphs for all SSSs. The first step is divided into two parts, firstly we shrink the source actors to gain a compact schedule. Secondly, we want to find out all possible periodic regions. To find out a periodic region with $p$ SSSs parallelism, we need to execute a list of $p$ SSSs. Between time interval $(p - 1) \times T$ and $p \times T$, there exists a region with all SSSs involved. The problem is now changing into finding out all possible list of SSSs with a length of $p$. This is shown in the second part of step one. Step 2 identifies all parallel blocks in periodic regions in the updated schedule. Finally, step 3 finds VF answers for each parallel block and repeat this procedure for all periodic regions. This is accomplished by a linear programming.

**DGVFS**

**step 1: find a compact schedule and periodic regions**

1. for each $G'(G, c_n) \in ME$

   Let $G'_ns(G, c_n)$ DENOTE a copy of graph $G'(G, c_n)$
   
   where $t(i) = 0, \forall i \in V_{G'(G, c_n)}^G(G, c_n)$
   
   if $MCM(G'(G, c_n)) \geq MCM(G'_ns(G, c_n))$

   $$mul = \frac{MCM(G'(G, c_n))}{MCM(G'_ns(G, c_n))}$$

   for each $i \in V_{G'(G, c_n)}^G$

   $$t(i) = t(i)/mul$$

   end for

end for

**5.4 Experiments and Results**

Worst case analysis is carried out for all transition possibilities in LTE model in local DVFS algorithms. However the aim is still reducing energy consumption for different subframes
2. \[ p = \max_{\forall (G, c_n) \in ME} \left[ \frac{\text{latency}_{G',c_n}(G,c_n)}{\text{MCM}(G',c_n)} \right] \]

\[ \text{find\_all\_transitions}(m, p) = \]

\[ \text{from (SSS } c_1 \text{) to (SSS } c_m \text{)} \]

\[ \text{print (current SSS) + find\_all\_transitions}(m, p - 1) \]

Computer a new schedule for all possible subframes sequences

Following steps are applied for all possible periodic regions

---

**step 2: identify parallel blocks PB**

**for** any time \( t \in \text{Schedule} \)

**if** Parallelism at \( t \neq \text{parallelism at } t + \Delta t(\Delta t \rightarrow 0) \)

Register parallel block in PB

**end for**

remove unscalable tasks in all parallel blocks

**for** each parallel block \( j \in PB \)

**for** each frequency level \( f \in F \)

\[ p_{pb}(j, f) = \sum_{\forall i \in j} p(\pi(i), f) \]

**end for**

**end for**

---

**step 3: DGVFS algorithm**

**minimize:**

\[ \sum_{\forall j \in PB} t(j) \times \sum_{\forall f \in F} (P_{pb}(j, f) \times \frac{6}{T} \times u(j, f)) \]

**subject to:**

\[ \forall j \in PB \]

\[ t(j) = t(j) \times \sum_{\forall f \in F} (\frac{6}{T} \times u(j, f)) \]

\[ \sum_{\forall j \in PB} t(j) \leq T \]

Algorithm 5.3: Discrete global VFS

considering the influence of its previous executing. No matter what subframe, exactly same throughput (period) constraints with the same hardware mapping relationship are applied.

Table 5.3 and Figure 5.6 show the results for LTE model using local VFS. For discrete VFS problem two categories are studied, one is with all five frequency/voltage levels and the other one is only using level \( HHigh \) and mode \( Low \). As the same in WLAN experiments, static VFS problem is also involved as control groups. From the Figure 5.6 we can find that dynamic local VFS can give a significant energy reduction from 10% to 20%. All dynamic local VFS algorithms give better energy reduction results then static ones. Notice that when
using two discrete frequency/voltage levels $H_{High}$ and $L_{ow}$, static DLVFS can only give an energy reduction of 0.7%. This is because in static VFS, frequencies for each processor have been limited by actors with long execution time. In our LTE model, the MCMC of the graph is already close to the throughput (period) constraint. Thus there is not so much available space for VFS scaling. At the same time, our experiment results again show that CLVFS give the upper bound of energy saving.

Table 5.3: Results for LTE with local DVFS

<table>
<thead>
<tr>
<th>CFI</th>
<th>Energy Reduction (%)</th>
<th>CLVFS</th>
<th>DLVFS-5levels</th>
<th>DLVFS-2levels</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>static</td>
<td>dynamic</td>
<td>static</td>
<td>dynamic</td>
</tr>
<tr>
<td>2</td>
<td>15.35</td>
<td>18.72</td>
<td>14.79</td>
<td>15.97</td>
</tr>
<tr>
<td>3</td>
<td>15.35</td>
<td>19.10</td>
<td>14.79</td>
<td>17.28</td>
</tr>
<tr>
<td>4</td>
<td>16.82</td>
<td>19.06</td>
<td>16.22</td>
<td>16.66</td>
</tr>
</tbody>
</table>

Figure 5.6: energy reduction for LTE using local VFS

For global VFS, experiments are conducted with all possible periodic regions, which come from different subframes sequences. In our LTE model, it is calculated that at most three subframes are involved in one periodic region. Meanwhile there are three types of subframes. Overall, $3^3 = 27$ situations need to be simulated. Table 5.4 shows the results from both static and dynamic DGVFS problems.

To explain this unsatisfactory result we need to go back to the algorithm introduced in the previous section. After applying the first step in Algorithm 5.3 periodic regions with smaller period appears. The throughput-constrained VFS problem is now turning into a partial latency-constrained VFS problem. The latency constraint here is the original period.
Table 5.4: Results for LTE with global DVFS

<table>
<thead>
<tr>
<th>subframe sequence</th>
<th>Energy Reduction (%)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>static-DGVFS</td>
<td>dynamic-DGVFS</td>
</tr>
<tr>
<td>CFI2 + CFI2 + CFI2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CFI2 + CFI2 + CFI3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CFI2 + CFI2 + CFI4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>CFI4 + CFI4 + CFI4</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

constraint $T = \frac{1}{\text{throughput}}$. However, because the source actors in our target LTE model is not so big comparing to other actors, the length of the periodic regions from step one is nearly the same with our period requirement, which brings a very small design space for frequency/voltage points.

Periodic regions appear in the updated SPS schedule after shrinking source actors. The DGVFS algorithm is actually using the difference between the length of periodic region and the period constraint $T$ as global slack for this region. However, source actors in the LTE model have short execution times, which brings a very short global slack for a periodic region. As we discussed before, step one in global discrete VFS helps to fulfill the local slack caused by long source actors. Therefore other local slacks still remain inside the updated schedule. To obtain a better result, we need to increase the usage of local slacks.

With the previous DGVFS algorithm, those unused local slacks distribute in a disorganized way. If there exists a time interval that all processors are idling, we can use it as a new global slack. A simple way to gain such empty time interval is relocating local slacks in the schedule such that block only contains slacks without actor executions appears. Because of the existence of local slacks, there must be some actors that have freedom to switch their position is the schedule without delaying others and it is possible for us to relocate actors and slacks inside the schedule. Actually when relocating local slacks to the same time interval we also get more parallelism of actors, which give higher energy saving at the same level of time scaling. However, relocating a already existing schedule in an optimized way can be NP-hard. In this thesis it will not be discussed.
6 Conclusion

Nowadays radio applications are everywhere in our life to handle wireless communication. Radio base band processing is a hard real-time streaming application that runs on multiprocessor platform[4]. Multiprocessor systems used for radio applications are battery oriented devices in most cases. A design objective is to minimize the energy consumption such that longer battery service time can be reached. In this thesis we focused on two types of requirements: timing requirements and low energy requirements. Mode controlled data flow (MCDF) graph is used to model the dynamic behavior of radio applications. It has been asserted that MCDF is sufficiently expressive to model radio applications while still keeping the ability to analysis temporal behavior[12]. Dynamic voltage and frequency scaling (DVFS) has been used as a common method to reduce energy consumption[5]. The main idea is to perform DVFS algorithms such that the system provides just enough speed to process the workload while still meeting the timing requirement.

In this thesis, we first proposed a technology to unroll a MCDF graph into an equivalent SRDF graph with respect of its valid static sub-sequence. By unrolling an MCDF graph into an equivalent SRDF graph, we successfully included the processing of one SSS into one iteration. In this way, it became possible to analyze the temporal behavior and to schedule the graph in terms of static sub-sequences.

After that, two types of radio application were studied: radio applications with and without time overlap between static sub-sequences. For radio application without time overlap between static sub-sequences, only one time execution of a specific SSS is important such that the timing requirement is focusing on the latency of this SSS. Latency requirements occur when the maximum reaction time for input data is important. Deadline-constrained VFS was carried out for this type of applications. A deadline-constrained VFS problem is subject to several important constraints such as precedence constraints of the MCDF graph; voltage-frequency relation equations and the specific deadline. Self-times schedule (STS) was used for deadline-constrained VFS problem such that all actors start to fire immediately when their precedence constraints are met. For radio application with time overlap between static sub-sequences, infinite input items need to be processed. The throughput constraints are important to ensure the rate of outputs for the continuity. Static periodic schedule (SPS) was used for throughput-constrained VFS problems. All actors are firing with period \( T \) such that with a time interval of \( T \) an output is produced. In MCDF graph, what is repeating periodically is frames which are represented by static sub-sequences (SSS). To generate a SPS for the SSSs, we converted the MCDF graph \( G \) to equivalent SRDF graphs \( G'(G, c_n) \) with respect to the SSSs.

For each category of radio application, a special instance was carried out as examples.
CHAPTER 6. CONCLUSION

Timing requirements is applied for both two example applications. VFS algorithms for these applications were divided into three types: Continuous local VFS (CLVFS), Discrete local VFS (DLVFS) and Discrete global VFS (DGVFS).

Experiments carried out in chapter 4 and chapter 5 indicated that dynamic VFS algorithms give significant energy reduction comparing with static VFS for both deadline-constrained VFS and throughput-constrained VFS. This is because that in static VFS the speed of each processor will be limited by actors with long execution time but small slack time. Meanwhile dynamic VFS allows more flexible changing on frequency and voltage levels to increase the usage of slack time. We have also established that local VFS is better then global VFS in terms of energy saving. The reason why global VFS offers worse number is that in order to make sure all processor have the same speed at the same time, it cannot fulfill all local slacks.

From our experiment results, global discrete throughput-constrained VFS gives a bad outcome for LTE model. This is because of the low usage of local slacks inside the SPS schedule. As future work, a new step can be added to global throughput-constrained VFS. That is relocating actor firings in the existing schedule without affecting the periodic executing. In this way we want to gain blocks containing only slacks inside with any actor executions. The new empty blocks can be used as global slack. Higher energy reduction appears because of the higher usage of local slacks. Another branch of future work is focusing on self-timed executing instead of replying on fully static schedules to gain more energy reduction.
Bibliography


