Dynamic Scheduling Implementation to Synchronous Data Flow Graph in DSP Networks

ENSC 833 Project Final Report

Zhenhua Xiao (Max)
zxiao@sfu.ca

April 22, 2001

Department of Engineering Science, Simon Fraser University

Contents

1. Abstract

Synchronous data flow graph was first introduced by E.A.Lee in 1987[1]. It can describe a large part of the DSP applications. Static Scheduling can be done at compilation time because of the nature of SDF, which makes application fast and transportation overhead minimized. As the DSP hardware evolves, more and more high-speed communication channels and nodes are added to one board without high cost. This makes speed and transportation a less concern while development time becomes critical for the market. This project investigates a dynamic scheduling method to implement the static scheduling algorithm, and using Opnet to do the simulation. The result shows that such an implementation will achieve the same performance under certain condition, and has some advantages over the Static Scheduling, such as code size, scalability and maintenance. The method will shorten the software development time with only minor performance degradation on real application.

2. Introduction

2.1 Motivation

A lot of papers are dealing with how to optimize the Synchronous Data Flow Graph on a given application. As a result of the optimized SDF Graph, a Static Scheduling Algorithm could be done at compilation time. Such a Static Scheduling Algorithm has a lot of advantages, such as minimized memory usage, high utilization of DSP, in order to achieve high system performance. As a software engineer, one question came out of my mind is that to implement Static Scheduling Algorithm is not easy from a software point of view. Since we need to program in every detail, while once the configuration changes, the old code is largely unusable.

The point here is after we are given the application, after we optimized the SDF, and get the Static Scheduling Algorithm, we know what is the best performance could be, can we use Dynamic Scheduling Algorithm to implement the software, and we still can achieve the same performance? One thing I want to make clear is: we first need to get the Static Scheduling Algorithm, to know what is the best, then we can use Dynamic Scheduling to achieve that “best”, as a goal.

The purpose of this project is: given the application, given the known best SDF and Static Scheduling, I try to use Dynamic Scheduling to meet the “best performance”, and implement it in software. If that could be done, then we not only get the best performance, but also gain some advantages coming with Dynamic Scheduling, such as code maintenance, scalability. Or we could say in other way, we get the best performance of Static Scheduling, but also overcome some disadvantages out of it.

For the relationship with the course, we know DSP is used in a device, which can be a surveillance unit in packet network, or receiver in Software Defined Radio. The goal of

Dynamic Scheduling Implementation is to achieve the high system through put in network devices, to increase the network performance. Since more and more DSPs are used in network devices, this is becoming more important.

2.2 Introduction

Synchronous Data Flow Graphs was first introduced by E.A.Lee in 1987[1]. As the definition, SDF contains two basic elements: Block and Arc. Block represents actions or tasks. A block is said to be Synchronous if we can specify a priori the number of input samples consumed on each input and the number of output samples produced on each output each time the block is invoked. Arc represents the flow of data, and the corresponding sequence of the firing of blocks. Arc could also be treated as the FIFO, since one block on one side of the arc put the samples into arc, and another block on other side of the arc consumes the sample. In the following report, we names sample as token, in the sense that a block could be invoked only when the number of tokens accumulated on the arc exceeds the thresh hold value.

A Synchronous Data Flow (SDF) graph is a network of synchronous blocks. Following is an example:

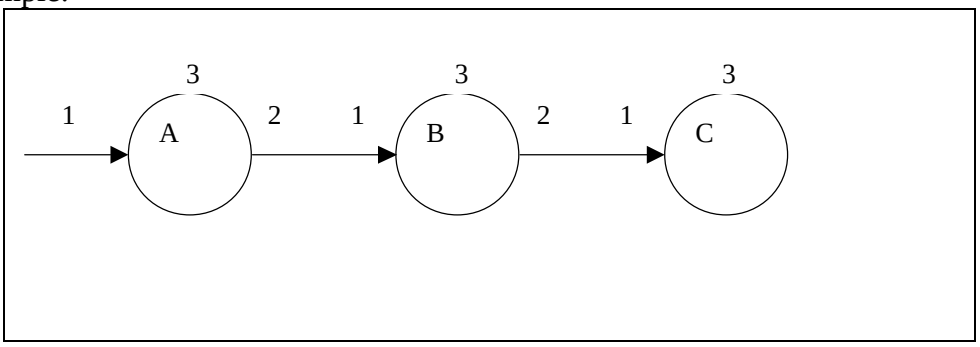

Figure 2-1 A Synchronous Data Flow Graph Example.

In this example (we will use the same example through out the paper), Task A can be fired at any time when input token number is bigger than or equal to 1, once it is fired, it will create 2 tokens on the output arc. Every time Task B is invoked, it will first consumes one token created by Task A, then produces 2 token at the time it finishes. Task C will consume 1 token when it invokes. All tasks take equally 3 time unit to finish as indicated above the circle.

Lets imagine in a situation that every packet (*generally speaking it is called signal, we say it packet because we use packet structure in Opnet to do simulation)* entering into the system create one token for Task A, Task A will have token to invoke one time for this packet, Task B will have enough token to invoke 2 times, and Task C will have tokens to invoke 4 times (because Task B will eventually produce 4 tokens for Task C). Once the Task C is invoked 4 times for the same packet, we assume that the packet has been completely processed and can leave the system. In this case, if there is only one DSP to perform all the tasks, the sequence of the Tasks executed could be one of the following schedules:

Static Scheduling **SS1**:

- ABBCCCC
- ABCBCCC
- ABCCBCC

The Service Time for each of the Scheduling Algorithm is 21 time units.

Although the 3 schedules have the same Service Time, the 3rd schedule is mostly optimized because it needs the lowest buffer memory. But if we are only concerning the Service Time of the system, then those 3 scheduling algorithm are equally good.

Apparently, this schedule will not be valid if there are 2 DSPs ready for executing, because the above schedule doesn't take into count the parallel computing power. The same problem will remain if we want to upgrade the system from 2 DSPs to 3 DSPs and later on. Following Figure show the best scheduling for 2 DSPs. (We want to get the shortest Service Time for every packet in and out). We will use this for a comparison later:

Static Scheduling **SS2**:
DSP 1: ABCC*ABCCC*BC*ABCCC*BC*ABCCC*BC
DSP 2: BCC*BC*ABCCC*BC*ABCCC*BC*ABCC

The Service Time for first packet is 15 time units, for second packet is 16.5 units, the third has 15 time units, forth has 16.5 time units and so on. The 1.5 time unit oscillation is due to 1.5 unit time waiting for *SPU* on the even numbered packet, because at the time the packet entering the system, the *SPU* is not ready.

As we have seen, SDF is good for static scheduling, since all the input, output and other relations between tasks are known. Static Scheduling has a lot of advantages: SDF graph can be optimized using technique such as retiming[5], transportation overhead can also be minimized. All of those can be done and tested in compilation time. If the situation or environment remains the same, the solution will remain the perfect one. However, once the environment is changed, e.g., the arriving packets speed increased, more DSP added into the board to increase the throughput, new scheduling algorithm should be applied, and that takes a lot of time to develop a new algorithm, code and load the program into the system: the old algorithm is obsolete at that time. From software perspective, it is not efficient if we have to redo every programming.

This project is to investigate the feasibility of Dynamic Scheduling Algorithm Implementation to SDF, that is to address the following question on the above example:

Can we design a system, using one Dynamic Scheduling Algorithm, achieve as good a performance as a Static Scheduling in 1 DSP environment? If the answer is yes, Can that Dynamic algorithm also achieve as good a performance in 2 DSP configuration without change in Dynamic Algorithm?

Since we already know the best algorithm using Static Scheduling techniques, i.e., we know SS1 (either one) is best for 1 DSP environment, SS2 is best for 2 DSP environment. Then our objective is to find a Dynamic Scheduling Algorithm, so that in a 1 DSP environment, it will have same Service Time as in "SS1" and in 2 DSPs environment, it will have same Service Time as "SS2". If we can achieve this, then we successfully find out a Dynamic Scheduling Implementation, in the sense that not only the implementation can achieve the same performance or have the same Service Time as Static Scheduling, or rather, match the performance of Static Scheduling Algorithm, but also, the Dynamic Scheduling Algorithm brings something which we are short of in Static Scheduling Algorithm: scalability, maintenance, modulation, flexibility in software. The performance is something we cannot sacrifice, the advantages coming with Dynamic Scheduling are what we want.

The contribution of this project includes:

- *Proposed a procedure to find the Dynamic Scheduling Algorithm matching the performance as good as Static Scheduling Algorithm. Details in 3.2*

- *Construct a new model in Opnet, the new model is acting as a simulation, it can do the simulation for different SDF Graphs. Details in 3.3.2*

- *Find a way to map the SDF into several simple matrixes, the matrixes are for DSP to have the knowledge of SDF and for decision making purpose. Details in 3.3.4*

- *Simulate the example to show that a Dynamic Scheduling Approach to the SDF with performance equal to Static Scheduling is feasible. Details in 4.*

The paper is organized as follows. First discuss the recent evolvement in DSP structure, which makes speed and memory usage a less concern. Then propose a procedure to find the best Dynamic Scheduling Algorithm. Based on the SDF and process, a model in Opnet is created. The SDF is mapped into the system using square matrix. Simulation results show that such implementation is possible. Finally, summarize and point out future research direction.

3. Dynamic Scheduling Implementation

3.1 DSP Network Evolvement

DSP applications were put a lot of attention on minimize the code size, memory usage to achieve high through in real time. Excessive transportation overhead was avoided as much as possible. In recent years, significant improvement has been done on the DSP board. A typical DSP board contains 4 – 8 DSPs. There are a lot of high speed channels between DSPs. Such a high channel could reach the speed of 150Mbyte/second. With the existing of such dual direction and dedicated channel, it becomes a less concern the time takes to transport the information between DSPs. The speed of DSP increases dramatically and the memory spaces are abundant also.

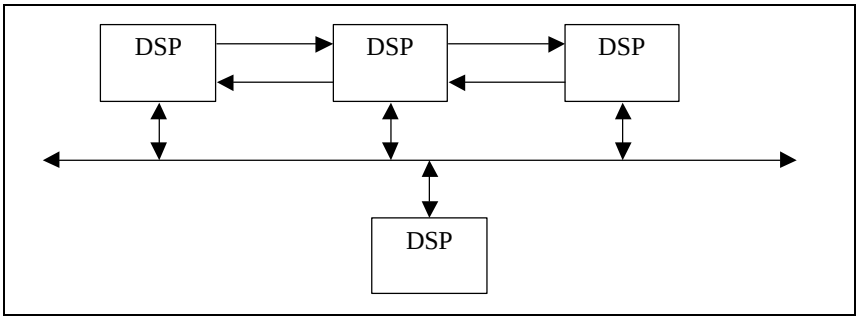

Figure 3-1 DSP Network Structure

Based on such a DSP structure, the time to market becomes a more critical issue. Whether the software put on the DSP is easy to maintain, easy to upgrade and ready to expand the scale is more and more important.

Another issue is that some applications need to vary the configuration in run time. For example, in the morning we may want 3 DSPs dedicated in on channel, in the afternoon, due to surge of traffic, we may want to allocate 5 DSPs into that channel. The change in the configuration will cause the change of algorithm.

3.2 Dynamic Scheduling Approach

First we make it precise what is the role of this Dynamic Scheduling Implementation. Following note details the some important properties, roles and objectives of this approach:

1. *The Dynamic Scheduling Implementation is not to replace the Synchronous Data Flow Graph and Static Scheduling Algorithm.*
   The purpose of the Dynamic Scheduling Algorithm is based on the Synchronous Data Flow Graph, a lot of optimization has to be done firstly to make the SDF fits the application best. Techniques such as retiming could be used at that time.

2. *The Static Scheduling Algorithm should be available before hand in order to verify that we have found the right Dynamic Scheduling Algorithm.*
   Rather than to replace the Static Scheduling Algorithm, we need the Static Scheduling result to verify that we have found the right Dynamic Scheduling Algorithm. Because the Dynamic one is always trying to meet the static algorithm, not pass. In the certain fixed condition, like in 1 DSP environment, or 2 DSP environment, the Dynamic Algorithm could meet the performance, but can not exceed the performance which the Static Scheduling can achieve.

3. *The Advantage of the Dynamic Scheduling Algorithm is we can have a single algorithm in several or all environments, which the Static Scheduling Algorithm can certainly not be able to achieve.*
   It is clear that the Static Scheduling Algorithm can fit for one environment in a case. For example, the best algorithm for 1 DSP won't fit for 2 DSPs. But in an environment the resources need to be allocated depending on the traffic, if we use

Static Scheduling Algorithm, we have to code for the different cases and load all these code into the memory. But with Dynamic Scheduling Algorithm, we can possibly using one and fits all. Of cause, the Dynamic Scheduling must be simulated and tested before it is loaded into the board.

4. *There is no guarantee whether there indeed exists a Dynamic Algorithm that satisfies all our objectives.*
   The procedure is a method to find the result, but not telling any thing about whether the Dynamic Algorithm is existing or not. There could be no such a Dynamic Scheduling Algorithm which can meet the best performance in all cases. Our chance is we have possibility to find a Dynamic Scheduling Algorithm for up to certain number of DSPs, which is sufficient for the application.

5. *An additional Scheduling Unit is needed to facilitate the Scheduling Task.*
   The *Scheduling Unit* (*SU*) is not a normal DSP, it is a dedicated DSP only to do the scheduling, rather than do the actual tasks. Because of the existing of such a supervisory DSP, all knowledge could be centralize to this DSP, and the other DSP since they have little knowledge, can be simply expanded. All the upgrade could also be done on *SU* only, this will reduce the software maintenance effort.

6. *We are considering the Implementation in an environment that the time to finish the task is much bigger than the time to transmit the packets.* The Dynamic Approach will certainly bring more transportation overhead. But because of the high speed channel between DSPs, those additional transportation overhead may not be excessive. In this project, we omit the transportation overhead brought from the new algorithm in order to focus on the topic. The trade off from the Dynamic Algorithm is mentioned as a future investigation in Chapter 4.

Following Figure shows the procedure to find the best Dynamic Scheduling Algorithm.

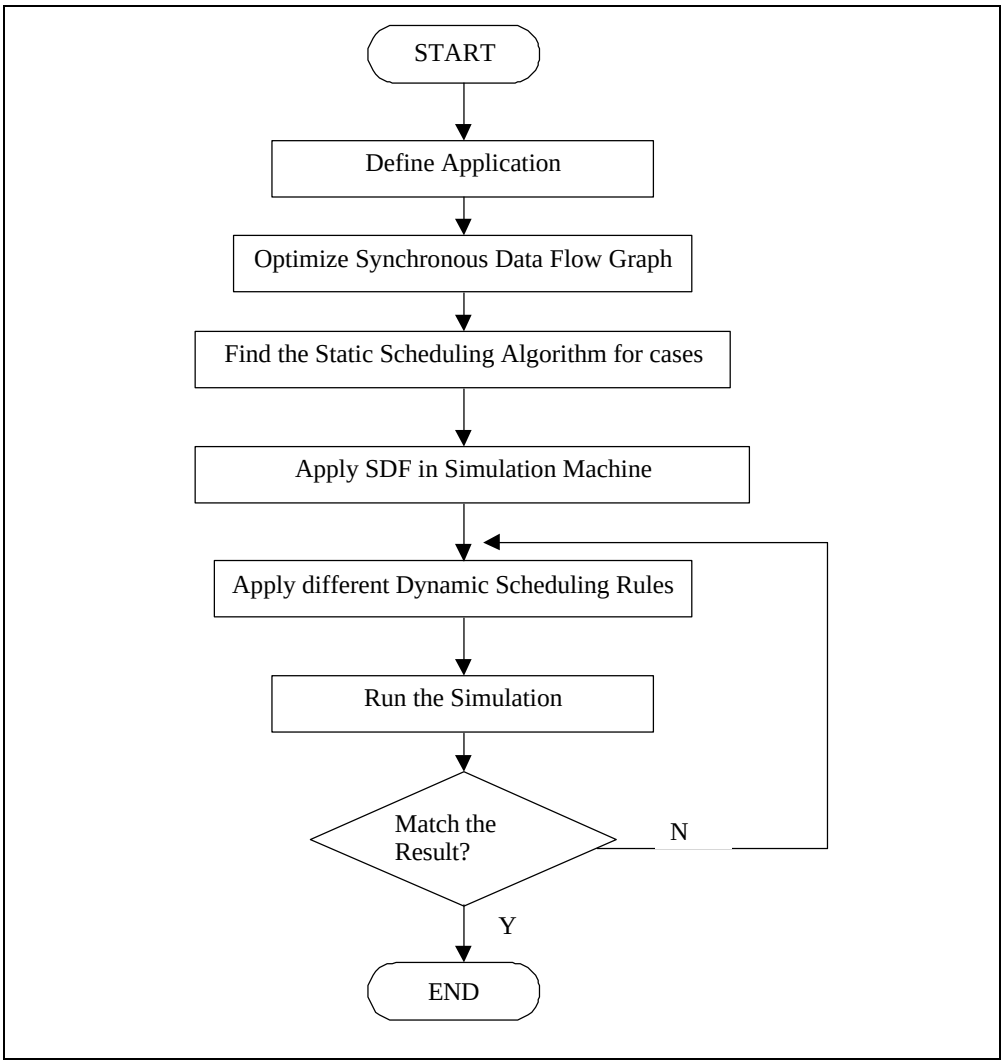

Figure 3-2 Procedure to Find The Best Dynamic Scheduling Algorithm.

In the Figure 3-2, we can also see some properties of the approach. We can see that we first obtain the Scheduling Algorithm using the Static method, then we try to match the Dynamic Result to the Static one.

The Dynamic Scheduling Implementation has several advantages over the Static Scheduling:

1. The Dynamic Scheduling rule can apply to several conditions, unlike the Static Scheduling, which need one algorithm for each condition. This will eventually largely reduce the code size. And the code is more efficient in the sense that they can be reused in other condition.

2. Another important property of Dynamic Scheduling is that it is rules oriented instead of detail oriented. Rule-oriented doesn't mean that the Dynamic Scheduling need not the knowledge of SDF, it does need that knowledge. But the

Scheduling Unit when perform the scheduling task, it merely uses the rules to determine which task to send to the *SPU*. While in Static Scheduling, the DSP should have all the knowledge, how many tasks to do, in what sequence.

3.3 Modeling DSP Structure Using Opnet

In this section, we will describe how we create the Simulation engine in Opnet to do the Dynamic Scheduling Simulation, and find the best fitting algorithm. The simulation engine should only have the knowledge of SDF, and based on the current status do the scheduling task. We will first discuss the logical lay out in a *Scheduling Unit*, which is the core unit in such a system. Then we discuss the Opnet model. We will further show the data structure and how to map the SDF in this engine. This engine can simulate different tasks and SDFs, please note that same engine can also be ported to DSP and used in the real world.

3.3.1 Scheduling Unit

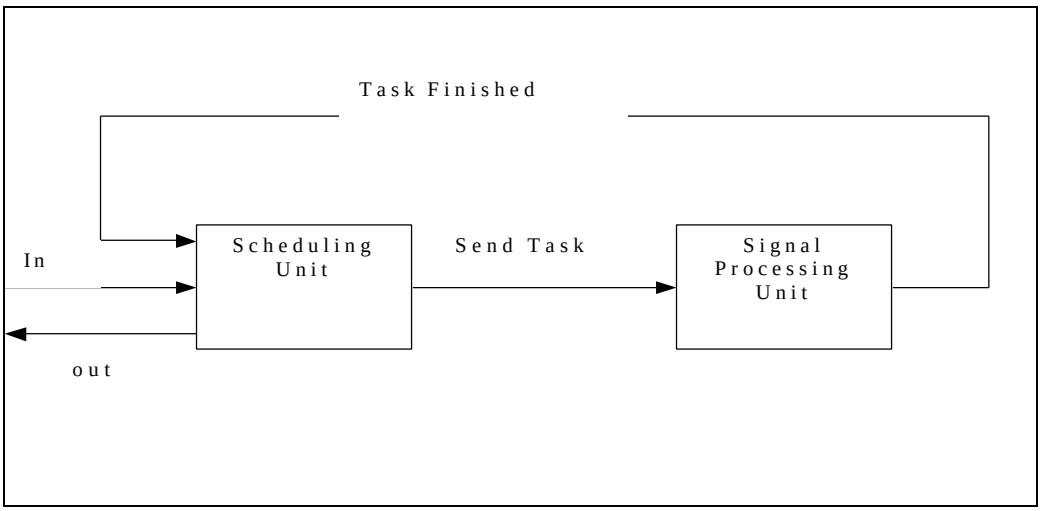

Figure 3-3 Scheduling Unit Logical Lay Out.

The upper graph shows the logical lay out of the DSP architecture. There are 2 basic roles here, *Scheduling Unit* and *Signal Processing Unit*.

Scheduling Unit (*SU*) is the core unit, it receives packets, arrange tokens, assign tasks to *Signal Processing Unit*, and receives the acknowledgement from *Signal Processing Unit* when the task is done[3]. *SU* has the knowledge of application, has the knowledge of SDF, but it doesn't has the knowledge of how scheduling is to be done. That has to be done in the run time according to certain rules set by the user. And the *SU* will schedule according to the rules. Since *SU* will not have the knowledge of what future will happen, e.g., it cannot know when the next packet is to come, the *SU* will always make the judgments based on what happen before and what is the current status. Following Figure shows the logical element in the *Scheduling Unit*.

Signal Processing Unit (*SPU*) is the unit to do the actual signal processing, which we can see from its name. *SU* usually only has one copy in the architecture, while *SPU* can have

multiple copies. Also, *SPU* number can be increased or decreased in the run time. This comes true when in a big application, resources are allocated dynamically, in such an environment, the Dynamic Scheduling in *SU* will be informed the changes in *SPU* numbers and change the scheduling thereafter.

3.3.2 Opnet Modeling Layout

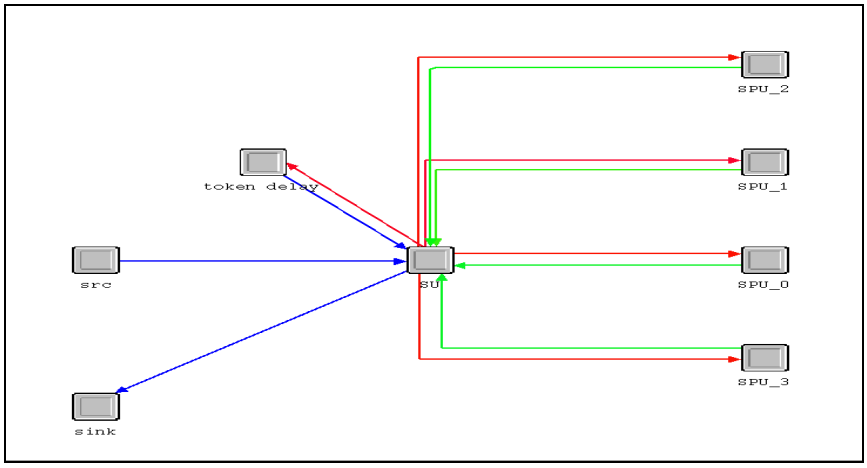

Figure 3-4 Opnet Modeling Of Scheduling Unit.

Scheduling Unit (SU) is the core of the network. Almost all the knowledge is put into the *SU*. *SU* has several responsibilities. First, it is an interface for the system to face the outside world: the *SU* is receiving incoming packets from SRC and send out packets out to SINK. Second, *SU* will do the actual scheduling. *SU* knows how many *SPU* is connected and available for computing, it knows the SDF, knows how many tasks need to be done for a packet, when to send the packet out when it is finished. *SU* holds all the task tokens in the queue, and based on the existing *SPU* ready/busy status, based on the current packet status, it will decide which packet which task to perform, then send the task to the best *SPU*.

Signal Processing Unit (SPU) is the DSP which does the actual task. It will receive the task assignment from *SU* and execute it according to instruction in the assignment packet. When task finished, it will resent it back to *SU*.

Links between SU and SPU represents the high speed communication channel in real board. The link with in the Opnet node model has no transmission delay and propagation delay. Of cause, the fastest link has above two delays, but since this project's purpose is to see the Dynamic Approach to SDF in such an environment that the time need to perform the task is much bigger than the time for transportation. So we use this kind of link.

Token_Delay is a special module. This should be within the *SU* in a real application environment. We have a separate module in Opnet to make the program more clean and clear. The purpose of this module is to delay those tokens that once they are created,

certain amount of time need to be delayed before they are available for the next task. In SDf, it is a bar on the arc. In Opnet model, we use token delay matrix to describe this.

Src and Sink are for external packet source and external destination after the packet has all tasks performed.

3.3.3 Data Structure and Elements

There are different type of packets, queues, variables in the engine, some of most importance are listed and described below:

General Packets is packets that come in and out of the DSP system. The represents the actual packets flowing in the network.

Token Packets. All tokens created after the task finishes or consumed when the task is invokes has type of *Token Packet.* The *Token Packets* format contains the information such as which *General Incoming Packet* it is for and the *Task Id*. *Token Packet* is only for *SU* internal usage, it never actual exits the *SU* module. The token is destroyed at the time the *Task Packet* created in scheduling procedure. We need *Token Packet* and *Task Packet* separately because one *Task Packet* may consume one or more different *Token Packets.*

Task Packets is the Packet that *SU* assign the task and send to *SPU*. *SPU* receive this type of packet and execute the task described in the *Task Packet.*

Queues exists for *General Incoming Packets, Token Packets.* We need to store them temporary and delete the structure in the queue when no longer needed. All queues are listed queue, the packets added to the queue is appended at the tail. But since we scheduling dynamically, the packets come in first may not be served first. So all the queues are not FIFO queues. In the queue elements, they not only contain the packets or tokens, but also have variables indicating the useful information such as the *last served time*. The queue structure is listed in the appendix.

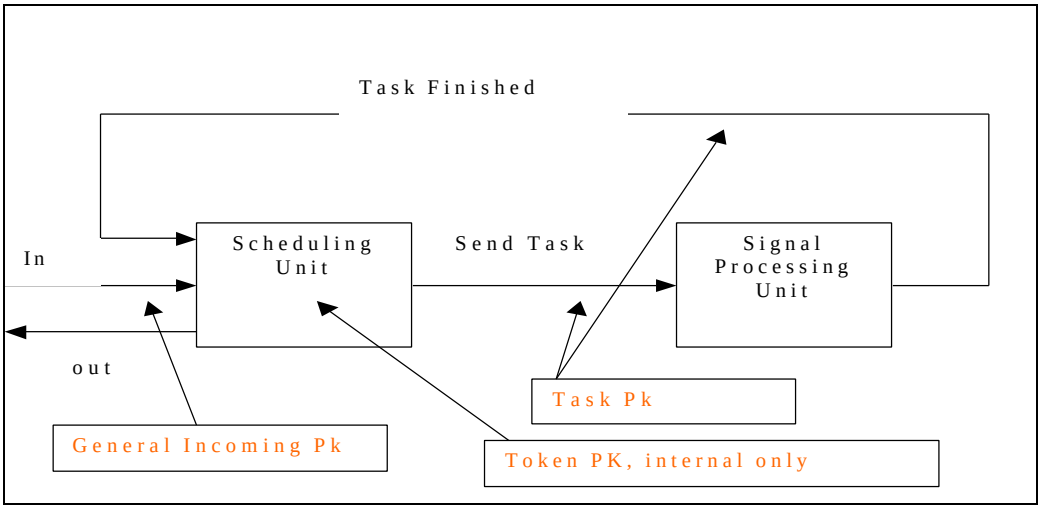

Figure 3-5 Effective Area Of Packet Variable.

The above figure illustrates the effective area in the Model.

3.3.4 Mapping the SDF into Opnet

In order for *SU* to have the knowledge of SDF, a mapping of SDF into Opnet is used. There are 2 ways to map the SDF into Opnet: we can use a list structure to represent the blocks, and the list structure can have several leaves, one leaf can have several parent leaves. The advantage of this mapping algorithm is its visibility in resembling the SDF structure, but the disadvantage is also obvious, the computer has to traverse the whole tree to allocate one element.

Another way of mapping is to use the matrix to map the SDF, this is the method we will use in the modeling, since this mapping is clear and easy to locate the resources, it is discussed in detail in the following:

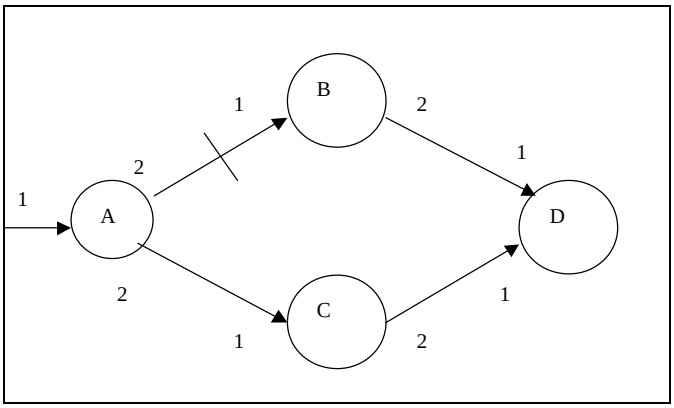

Figure 3-6 SDF For Matrixes Examples.

For the above topology, we can see that every incoming packet will allow Task A to execute once, Task B C D all execute once. After the Task D's execution, the packet will exit the system. We use one topology matrix to describe the topology of the Graph.

0	1	0	0	0
0	0	1	1	0
0	0	0	1	0
0	0	0	0	1
0	0	0	0	0

the topology matrix is 5 * 5 matrix, we denote the start of the program to be node 1, then task A be node 2, task B is node 3, and C node 4, D node 5. if there is a "1" in the matrix at i row and j column, e.g., element at [1,2] is value 1, it means there is a connect from node 1 (Task A) to node 2 (Task B). We specify that node 1 is the entry point, for other node, the node number assigned to each task is arbitrary. This matrix shows all the arcs in the topology, those arcs are Fifos containing the tokens.

Another matrix we create is the “produce token matrix” and “consume token matrix”. These matrix are of the same size of Topology matrix. We have seen that the “1” valued element in topology matrix represents an arc, in the same location in “produce token matrix”, the value means how many tokens will be created on that arc when node i finishes the task. And correspondingly the same for “consume token matrix”. So for the above topology, the “produce matrix” is:

0	1	0	0	0
0	0	2	2	0
0	0	0	2	0
0	0	0	0	2
0	0	0	0	0

the “consume token matrix” is:

0	1	0	0	0
0	0	1	1	0
0	0	0	1	0
0	0	0	0	1
0	0	0	0	0

The element in “Delay matrix” represents the time delay for the token between the time the token is created and the time the token is ready to use. The “Delay matrix” for the above example is as following:

0	0	0	0	0
0	0	1	0	0
0	0	0	0	0
0	0	0	0	0
0	0	0	0	0

Above matrixes are needed before the program runs. This means the *SU* should have the knowledge to begin. The following matrix are status matrix, they are for *SU* to determine the current status of all tasks, all tokens and all packets.

We need a “token number matrix”. The element on location [i, j] means how many tokens are existing on that arc.

We need a “token queue matrix”. We have said that each arc is acting like a FIFO, which contains tokens. This expression is not accurate. For example, on arc from node 2 (Task A) to node 3 (Task B), there can be multiple tokens, those tokens could be for different packets. But it is not necessary that the token created first be consumed first. Sometimes it is better for the token created later to execute for better overall system performance and the token created first is discarded. In this case, a list is needed instead of a FIFO. So the “Token queue matrix” is actually a matrix pointing to a series of lists or queues. But only

those locations that arc exists will have a queue, other elements will remain NULL and never be accessed since there is no arc for that element.

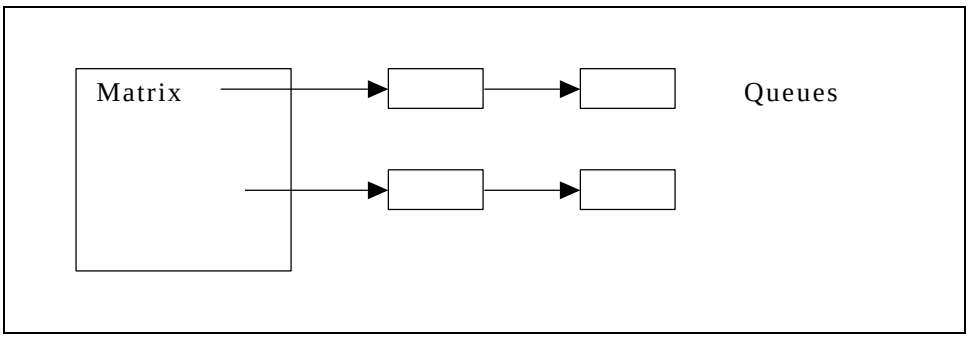

Figure 3-7 Matrixes and Queues in SU.

The advantages of having those matrixes are obvious. We can input the SDF in a simple way, and the simulation model can traverse the matrix to do the scheduling. Those matrixes are actually serving as an abstract layer of the real SDF and program.

3.3.5 Scheduling Unit Process Model Using Opnet

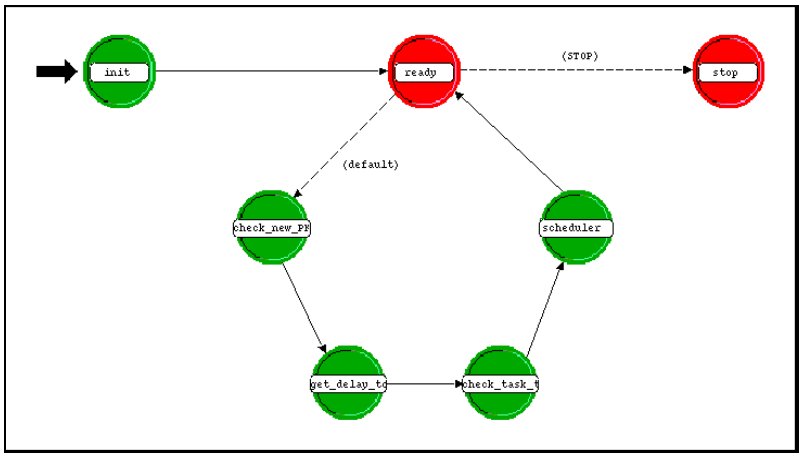

Figure 3-8 Scheduling Unit Process Model in Opnet

Figure 3-8 shows the detailed implementation of *Scheduling Unit* Process Model in Opnet. There are total 7 states in the process. We discuss the most important states in the following, the whole process shows how *SU* do the scheduling internally.

1. *Init* State
   The *Scheduling Unit* gets the knowledge of Synchronous Data Flow Graph in this state. Several matrix such as topology matrix, consume matrix, produce matrix, delay matrix, task duration matrix are set at this point. The *SU* should also know how many Signal Process Unit is ready. Although we have 4 *SPU* connected to *SU*, they are not necessary all ready for task. This state only be reached once at the beginning of the program.

2. *Ready* State
   The process will always be at this state between interrupts. In this state, the *SU* did nothing except waiting for incoming interrupts, the interrupts could be from incoming packets, or from *SPU*s if they have finished their assigned job. The interrupt can also be from the token_delay module. When the token is ready to add to the queue after the delay time expires, the token_delay module will send the token back to *SU*. In the real world, this can be done internally in *SU* instead of 2 modules.

3. *Check_new_PK* State
   SU check the link for incoming packets. If there are incoming packets, the *SU* will store the packet in a listed queue and generate one token or tokens according to SDF immediately. The token will be marked for which task and for which packet. The queue size will also be updated. The task will not be scheduled until step 6.

4. *Get_delay_token* State
   If a token need to be delayed a certain time before available for next task, the token will be send to token_delay module. After the delay time expires, the token_delay module will send back the token or tokens. If *SU* receives token from token_delay module, it assumes that the token(s) are available immediately, *SU* will store the token in the list queue according to its task type.

5. *Check_task_return* State
   In order for *SPU* to execute the assigned task, the *SU* will send a task assignment token to *SPU*, after the *SPU* finishes the task, *SPU* will send back the task assignment token. *SU* receives the token, check which task is finished and the task is for which packet, then *SU* will go to packet queue, mark the corresponding packet the this task has been done. *SU* will check whether all the necessary task has been done on this packet, if this is the case, *SU* send out the packet and delete this block in the queue. If not, *SU* will check whether this task will produce tokens on out arc, if yes, the *SU* will create tokens and append them. But if the token need some delay time to be available, the *SU* will send it to token_delay module immediately instead of pending them.

6. *Scheduler* State
   The scheduler is making decision based on the status of SDF. The status means how many packets is waiting in the queue and how many tokens are there for each individual task, are the *SPU*s busy or not. We know that one packet require several different tasks to be done. The status could be modified in Step 3, 4, 5. This is the core of the whole simulation machines. We need to choose different rules and expect different performance. Rules are like earliest deadline, task balance.

7. *Stop* State
   Stop is the final state when *SU* receives a simulate stop interrupt. It will then report some essential values indicating the system performance.

4. Simulation Results

First we present the 1 DSP Service Time graph using Dynamic Scheduling Implementation as a bottom line for comparison of the others. All simulations are based on the SDF Graph depicted in Figure 2-1. We set the incoming packet interval differently according to the following table, which is the maximum load for each configuration:

Simulation results are done with 2 different Dynamic Scheduling Algorithms: *Balanced Schedule* and *Earliest Deadline*. The detailed scheduling implementation logic is described in appendix. In brief:

- *Balanced Schedule* tries to spread the *SPU* resources on different *General Incoming Packets*.
- *Earliest Deadline* tries to schedule the *SPU* to the *General Incoming Packet* which comes into the system earlier than the others.

Following Table shows the Simulation Setting, all configurations tested using above 2 algorithms.

DSP Number	Packet interval (time units)	Simulation Duration (time units)
1	21	180
2	10.5	180
3	7	180
4	5.25	900

Table 4-1-1 Simulation Settings

In Simulation, 1 time unit is set to 1 second in Opnet.

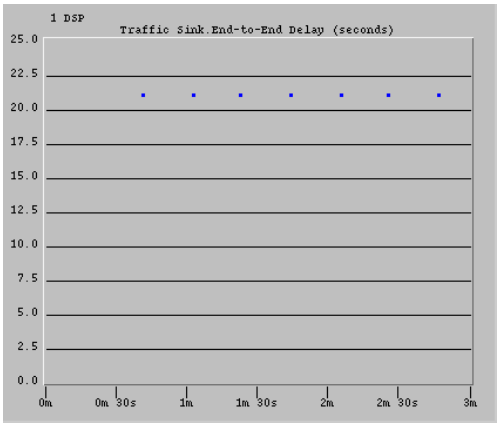

Figure 4-1 Service Time on One DSP configuration.

In the Graph, we use 1 DSP, *Balanced Scheduling* method, the Service Time are straight 21 time units, this is the same as if we use Static Scheduling Algorithm in SS1. Actually in 1 DSP environment, no matter we can use either *Earliest Deadline* or *Balanced Scheduling*, they have the same result as the *Static Scheduling*. When we continue to look at the result of 2 DSP, 3 DSP, 4 DSP environment, we can get the following interesting results:

- *Certain Algorithm can be chosen in Dynamic Scheduling to approach the best result.* In Figure 4-2, we use Balanced Schedule in a 2 DSP configuration, we can see that the performance matches the Static Scheduling SS2, which is optimized. This is great since we not only use Dynamic Scheduling to get as good a performance as Static Scheduling again, but also, the same Dynamic Scheduling Algorithm could be used in both 1 and 2 DSP environment. This is the objective we want at the very beginning. It also means that, we have found out a way to use Dynamic Scheduling to implement the SDF.

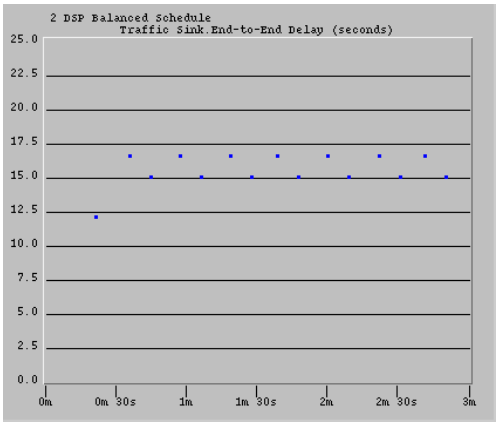

Figure 4-2 Service Time With Two DSP Balanced Schedule.

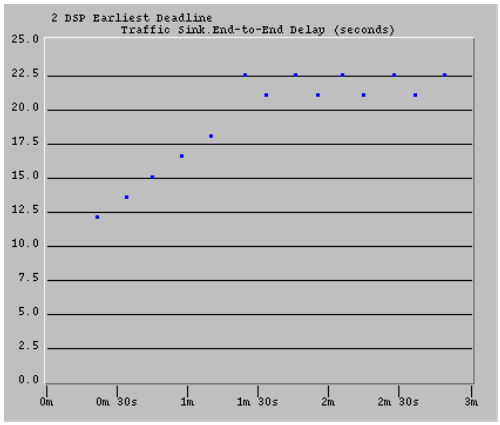

Figure 4-3 Service Time With Two DSP Earliest Deadline.

- *The algorithm need to be simulated to get the result, not from the name*
  The figure 4-3 is the result when we use the Earliest Deadline Scheduling Algorithm. We can see that the simulation result shows the Dynamic Algorithm is at 21 time units or 22.5 time units Service Time at the end, rather than 15 and 16.5 time units delay in the previous Algorithm. The result is far from what the name suggests.

- *The Dynamic Algorithm need some time to converge to the best performance, while the static algorithm has no such time, The time to converge could differ for different algorithms*
  From Figure 4-2 and 4-3, we can see that rather than achieve the Static Scheduling Performance immediately, the Dynamic Scheduling may need some time to converge to the same result. This seems unavoidable in Dynamic Scheduling Algorithm, since at the time of scheduling, we have no knowledge of

time of the next coming packet. The scheduling is based on only the current status. Although some of the SDF may result in immediate match.

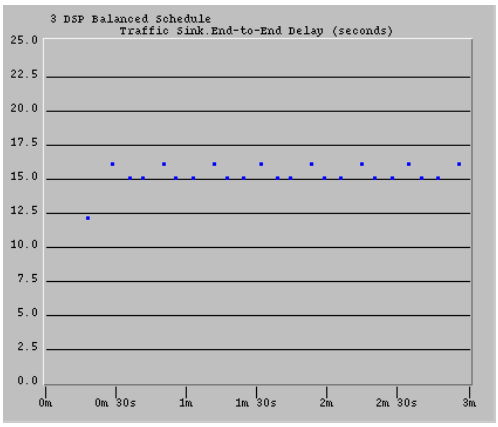

Figure 4-4 Service Time With Three DSP Balanced Schedule.

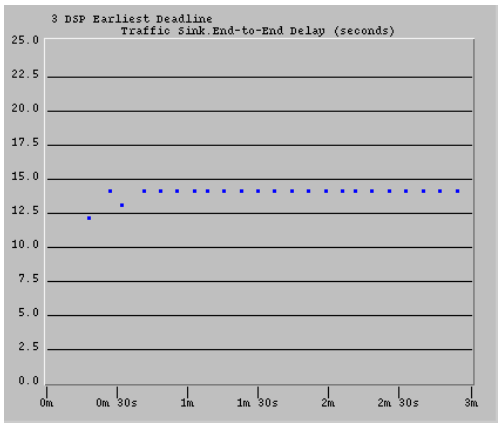

Figure 4-5 Service Time With Three DSP Earliest Deadline.

- *Different Configuration may result in different Dynamic Scheduling Algorithm performance.* Figure 4-4 and Figure 4-5 are Service Time results for 3 DSP configuration using *Balanced Schedule* and *Earliest Deadline* Algorithm separately. At this time, Earliest Deadline out performs the *Balanced Schedule.* Because *Balanced Schedule* tries to spread the *SPU* resource over different packets, thus enlarge the Service Time.

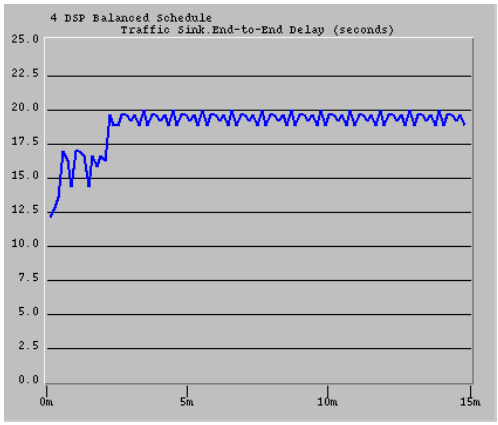

Figure 4-6 Service Time With Four DSP Balanced Schedule.

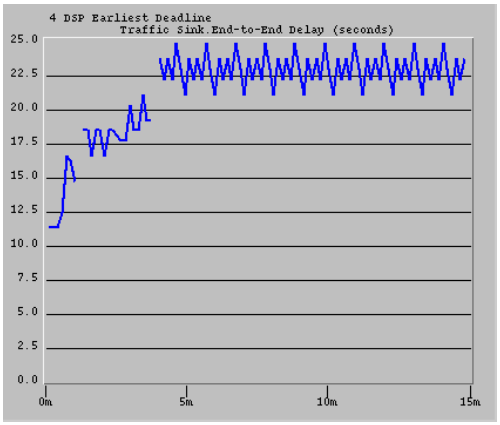

Figure 4-7 Service Time With Four DSP Earliest Deadline.

- *It could also happen that none Dynamic Scheduling Algorithm can match the Static Scheduling performance.* We can see in a 4 DSP environment, both scheduling algorithm results in a bad performance. We have known that for a 2 DSP configuration, the Service Time could be 15 – 16.5 time units, so for a 4 DSP configuration, it can NOT be worse than 2 DSP configuration, otherwise we can simply set the DSP into clusters and 2 DSP in one cluster, then we do the Scheduling according to the cluster.

- *The size of the DSP code is almost fixed, with little change in different DSP configuration or SDF modification.* Once we allocate the memory for the matrixes, the Dynamic Scheduling is done based on those matrixes, no change should be done on Scheduling Algorithm. Only the matrixes inputs should be modified to reflect the change in SDF graph. This also means that the code can be largely reused and shorten the delivery time, which is of great importance.

5. Conclusions

5.1 Conclusions

From the discussion and simulation, here are the conclusions from this project:

We proposed a Dynamic Scheduling Implementation to SDF in the DSP network, simulation shows that the proposal could be achieved. The proposal is based on the most recent DSP architecture and new application environment.

A generic Opnet model is created, the same model is valid for real DSP programming.

A way to map the SDF into software is used and proved to be successful.

Dynamic Scheduling Implementation has certain advantages such as code size, maintenance, scalability over Static Scheduling Algorithm.

Finally, Dynamic Scheduling Algorithm is a way to implement in software, while it meets the performance goal got from Static Scheduling Algorithm.

5.2 Difficulties In The Project and What Was Learned

The difficulties of this project lie in the creating of the model in Opnet. All the modules and model are written from scratch. The debugging in Opnet is not as easy as in MSC. There are lot of low level data manipulation in the model, such as matrix traverse, queue traverse, queue insert and delete. All these add to the difficulties of the project.

However, from this project, I become familiar with the Opnet, the simulation, event driven concept, model creation, and various programming kernel functions. These knowledge will definitely have strong effect on the future work.

5.3 Future Work

In the simulation, we showed that the Dynamic Scheduling Implementation to SDF could be done. But there are several uncertainties need to be addressed.

The first one is whether we could always find a Dynamic Scheduling Implementation given a “Static Scheduled” Goal. Suppose the answer is positive, whether the Dynamic Scheduling Algorithm Library is of finite size or could be infinite size? This question may only be of theoretical importance, since we can simply try and exploit. But if this is

the case, we can implement such a library, which means the library can fit for big number of SDF applications (not all). This could be a great news for software people in DSP company.

Another topic is regarding the convergence time. Since the Dynamic Scheduling Algorithm need some time to converge, it may or may not become a draw back of such an implementation. But this is a potential issue.

In this project, we omit the transportation overhead raised from the Dynamic Scheduling Algorithm. But we want to know the impact on the algorithm. Even though the channel speed is very high, it could also be a bottleneck.

6. References

[1] E.A.Lee and D.G.Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Transactions on Computers*, vol. 36, no. 1, pp. 24-35, Jan, 1987.

[2] E. A. Lee and D. G. Messerschmitt, "Pipeline interleaved programmable DSPs: Synchronous Data Flow Programming," *IEEE Transactions on Acoustics, Speech and Signal Processing*, 1987, ASSP-35:1334-1345

[3] G.Liao, G.R.Gao, V. K. Agarwal, "A Dynamically Scheduled Parallel DSP Architecture for Stream Flow Programing," ACAPS Technical Memo 45, June 4, 1993

[4] E. A. Lee and T. M. Parks, "Dataflow Process Networks," *Proceedings of the IEEE*, vol. 83, no. 5, pp. 773-801, May, 1995.

[5] T. W. O'Neil, E.H.-M. Sha, S. Tongsima, "Parallelizing Synchronous Data-Flow Graphs via Retiming," *IEEE 4th International Conference on Algorithms and Architectures for Parallel Processing*, Hongkong, December 2000.

[6] P. K. Murthy, "Scheduling Techniques for Synchronous and Multidimensional Synchronous Dataflow," Technical Report UCB/ERL M96/79, Ph.D. Dissertation, EECS Department, University of California, Berkeley, CA 94720, December 1996.

[7] S. Bhattacharyya, P. K. Murthy, and E. A. Lee, "Synthesis of Embedded Software from Synchronous Dataflow Specifications," *Journal of VLSI Signal Processing Systems*, Vol. 21, No. 2, June 1999.

[8] T. M. Parks, "*Bounded Scheduling of Process Networks*," Technical Report UCB/ERL-95-105. Ph.D. Dissertation. EECS Department, University of California. Berkeley, CA 94720, December 1995.

[9] J. L. Pino, S. S. Bhattacharyya and E. A. Lee, "*A Hierarchical Multiprocessor Scheduling Framework for Synchronous Dataflow Graphs,"* UCB/ERL M95/36, May 30, 1995

Appendix:

A-1 Descriptions of Algorithms Used in Simulation

A-1-1 Earliest Deadline

In this algorithm, the DSP is always trying to finish all the tasks for the packet that comes into the system earliest. The Algorithm will consider tasks of other packets when the task of the earliest packet can not full fill the *SPU* usage.

The Main Body:

```
Set pointer to the top of packet queue;

While pointer is valid {
        For each SPU in the list {
                If SPU is available {
                        Assign a task belongs to the packet;
                        Break;
                }
        }
        set pointer to the next packet in the queue;
}
```

A-1-2 Balanced Schedule

```
Set pointer to the top of packet queue;

While pointer is valid {
        Find the first packet not served now;
        For each SPU in the list {
                If SPU is available {
                        Assign a task belongs to the packet;
                }
        }

}
```

A-2 Selected Code List