Design Space Exploration of Multiprocessor System-on-Chip
Architectures
for Real-Time Multimedia Applications

Bipasa Chattopadhyay* and Samarjit Chakraborty†

Department of Computer Science,
School of Computing,
National University of Singapore

ABSTRACT
We have identified the performance impact associated with different design parameters of
an on-chip ARM processor in the specific context of implementing Real-Time Multimedia
applications. We have mapped different tasks on to a processor, scheduled the tasks
according to a Time Division Multiple Access (TDMA) scheduler, changed the frequency
of the processor and the amount of time available for each task to run on the processor.
We have done a design space exploration to study the impact of processor and scheduler
configurations on the performance of a processor and behavior of multimedia applications.
We have measured the performance of a given processor design by finding out total buffer
size required. Measuring the performance is difficult because multimedia streams have high
variability in their compression factor and in their execution requirements. However, this
variability can be modeled using Variable characterization curves.

1. INTRODUCTION
Over the recent years there has been an increase in the complexity of multimedia applica-
tions. To keep up with this complexity, there has been a lot of interest in high-performance
multiprocessor System-on-Chip (MpSoC) architectures where several processors, memory
and communication subsystems are integrated into a single chip. A considerable amount of
research has already been done in order to tune or configure such a platform for a specific
application at hand. The Kahn Process Network (KPN) has been used to map the tasks of
a MPEG 2 decoder to the given processor architecture (Dwivedi, Kumar, & Balakrishnan,
2004b, 2004a). In a KPN, processes communicate via unbounded FIFO channels. Processes
read and write atomic data elements or tokens from and to channels. Integrating the KPN
model with the Quality of Service (QoS) constraints has proven to be an effective technique
to tune MpSoC architectures (Dhand, Dwivedi, & Balakrishnan, 2006). Research in this
field has shown that incorporating QoS constraints allows one to achieve performance which
is sometimes as much an order of the magnitude better in relation to considering worst case
behavior (Marculescu & Nandi, 2001; Marculescu, Pedram, & Henkel, 2004).

Another approach used in (Maxiaguine, Chakraborty, Knzli, & Thiele, 2004a; Maxi-
aguine, Zhu, Chakraborty, & Weng-Fai, 2004c; Maxiaguine, Knzli, & Thiele, 2004b) to
map tasks onto a MpSoC architecture is with the help of Variable Characterization curves
(VCCs). VCCs are used to quantify best-case and worst-case characteristics of sequences.
The sequences may consist of consecutive stream objects, consecutive executions of a task

*Student
†Supervisor
implemented on a processor or of consecutive time intervals of some specified length. In this project we have used VCCs to model the arrival of the multimedia stream and to model the workload of the tasks. We have used TDMA to model the scheduling of tasks on the processor.

1.1. System Model
The system model we have used is as shown in Figure 1. There can be 2 or more processing elements (PEs) but for simplicity we have fixed the number of PEs to 2. The PEs communicate with each other with the help of unidirectional buffers. There are 2 inputs to this system both of which are multimedia streams. Both the inputs are processed by different set of tasks which are scheduled on the processor using TDMA.

Different tasks of a multimedia application can be mapped on to the 2 PEs. In this project we map different tasks of a MPEG 2 encoder. The MPEG 2 encoder application has various tasks like Forward Search (FS), Motion Compensation (MC), Discrete cosine transform (DCT) and Quantization (Q). Each PE implements some task. Certain tasks require more processor cycles than the others i.e. the workload of some tasks is more than others. Depending upon the workload, tasks can be mapped on to the PEs. This has a great influence on the performance. We determine the workload of a given task using Variable characterization curves (Section 2).

1.2. Modeling Multimedia Streams
In Figure 1, two multimedia streams that consist of raw images arrive at buffer b1a and b1b. They get processed by PE1. The partially encoded macroblocks produced by PE1 are stored in buffer b2a and b2b and eventually get processed by PE2. PE2 returns fully encoded macroblocks. Encoding is useful because encoded videos are much smaller in size than the raw counterparts. Sharing and storing encoded video requires lesser bandwidth and memory.

Within any time interval of length $\Delta$, it is guaranteed that atleast $\alpha^l(\Delta)$ and atmost $\alpha^u(\Delta)$ number of macroblocks will arrive at the PE and that the PE will be able to process at least $\beta^l(\Delta)$ and atmost $\beta^u(\Delta)$ number of macroblocks (Section 2). Once we determine the number of macroblocks arriving in time $\Delta$ and the number of macroblocks being processed in time $\Delta$ for a given processor design we can mathematically calculate the total buffer size required by the system. For optimal performance the buffer size should be such that there is no overflow in the buffers.
2. CHARACTERIZING VARIABILITY

As mentioned earlier VCCs are used to quantify best-case and worst-case characteristics of sequences. A VCC $\nu$ is composed of a tuple $(\nu^l(k), \nu^u(k))$. Both these functions take an integer $k$ as the input parameter, which represents the length of a sequence. $\nu^l(k)$ then returns a lower bound on some property that holds for all subsequences of length $k$ within some larger sequence. Similarly, $\nu^u(k)$ returns the corresponding upper bound that holds for all subsequences of length $k$ within the larger sequence. Let the function $P$ be a measure of some property over a sequence $1, 2, \ldots$. If $P(n)$ denotes the measure of this property for the first $n$ items of the sequence (i.e. $1, \ldots, n$), then $\nu^l(k) \leq P(i + k) - P(i) \leq \nu^u(k)$ for all $i, k \geq 1$. Applications of VCC are described below (Maxiaiguine et al., 2004c).

2.1. Characterizing Arrival Rate

We know that multimedia streams are complex and bursty in nature. The number of macroblocks seen over any fixed time is variable. We use the concept of VCC to determine the arrival rate of the macroblocks. There is a fixed upper bound and lower bound on the number of macroblocks that can arrive in any time interval delta. Let $r(t)$ be the number of macroblocks within time interval $[0, t]$ and $\alpha(\Delta)$ be the number of macroblocks that arrive in any time interval $[t, t+\Delta]$, then $\alpha^u(\Delta)$ is the maximum number of macroblocks arriving within any instance of length $\Delta$ and $\alpha^l(\Delta)$ minimum number of macroblocks arriving in any instance of length $\Delta$ such that $\alpha^l(\Delta) \leq r(t+\Delta) - r(t) \leq \alpha^u(\Delta)$. The arrival rate is thus modeled by the tuple $(\alpha^l(\Delta), \alpha^u(\Delta))$ (for convenience we will refer to this tuple as $\alpha(\Delta)$). This tuple can be determined by examining the multimedia stream and computing the maximum and minimum macroblocks that arrive in any time interval $\Delta$.

2.2. Characterizing Workload

Once the macroblocks have arrived at the PE, the PE provides some service to the macroblocks. The number of cycles required to process a fixed number of macroblocks, which is also referred to as the workload, is variable. We use the concept of Variable Characterization curves to determine the variability in the workload. Let $c(n)$ be the total number of cycles required to process the first $n$ macroblocks ($1, 2, 3 \ldots n$). Let $\gamma(k)$ be the total number of cycles required to process any $k$ consecutive macroblocks ($n5, n6, n7 \ldots n(5+k)$), then $\gamma^u(k)$ is the maximum number of cycles required to process any $k$ consecutive macroblocks and $\gamma^l(k)$ is the minimum number of cycles required to process any $k$ consecutive macroblocks such that $\gamma^l(k) \leq c(n+k) - c(n) \leq \gamma^u(k)$. The workload is thus modeled by the tuple $(\gamma^l(k), \gamma^u(k))$ (for convenience we will refer to this tuple as $\gamma(k)$). If we take the inverse of $\gamma(k)$ and divide it by the frequency of the processor we will get $\beta^l(\Delta) and \beta^u(\Delta)$ which is the minimum and maximum number of macroblocks that will get processed in time $\Delta$ (for convenience we will refer to this tuple as $\beta(\Delta)$).

3. TIME DIVISION MULTIPLE ACCESS

Let us consider two streams being scheduled on a PE by a TDMA scheduler, which has a period equal to $T$ time units. The weights associated with the two streams 1 and 2 are $w1$ and $w2$ respectively, with $w1 + w2 \leq 1$. Let the service offered by a scheduler to a particular stream be specified by the service curve $\sigma(\Delta)$ where the tuple $(\sigma^l(\Delta), \sigma^u(\Delta))$ denote the minimum and the maximum number of processor cycles available to the stream.
within any time interval of length $\Delta$. From the definitions of $\gamma(k)$ and $\sigma(\Delta)$ we can derive that, $\gamma^{u-1}(\sigma^u(\Delta))$ is the maximum number of macroblocks that can be processed within any time interval $\Delta$ and $\gamma^{l-1}(\sigma^l(\Delta))$ is the minimum number of macroblocks that can be processed within any time interval $\Delta$.

The scheduler works as follows: time is divided into periods of length $T$. Within any period of time $T$, $w_1 \times T$ consecutive time units are allocated to stream 1 and $w_2 \times T$ consecutive time units are allocated to stream 2. If a stream does not have sufficient number of stream objects to exhaust the processor time allocated to it, then the unutilized processor time is wasted. Let us first assume that the period $T$ is infinitesimally small and we neglect the effects of a finite sampling on the processor cycles. Then the service offered to the two streams in terms of processor cycles is calculated as follows. If $c$ is the number of processor cycles available from the PE per unit time (i.e. $c$ is the frequency of the PE), then the total service offered by the processor is given by the service curve $\sigma^l(\Delta) = \sigma^u(\Delta) = c\Delta$. The service offered to stream 1 can be specified by the service curve $\sigma^l_1(\Delta) = \sigma^u_1(\Delta) = w_1 \times c\Delta$. Similarly, the service curve for stream 2 is $\sigma^l_2(\Delta) = \sigma^u_2(\Delta) = w_2 \times c\Delta$. Therefore, the lower and the upper service curves for both the streams coincide and are straight lines with slopes $w_1c$ and $w_2c$ for streams 1 and 2 respectively. When the period $T$ has a finite value, the service curves take the form of a staircase function and the lower and upper curves no longer coincide.

4. DESIGN SPACE EXPLORATION

4.1. Basic Design Space Exploration

Figure 2 gives us an overview of the Design Space exploration. Let us consider a raw video. A raw video is essentially made up of raw frames which are in turn made up of macroblocks. The encoder treats every raw frame (macroblock) either as an I, B or P frame (macroblock). Whether a given frame (macroblock) is of type I, B or P is determined by an encoder state machine or automaton. An example of an automaton is shown in Figure 3. Let us assume that the automaton of our encoder is as shown in Figure 3 (Chakraborty, Mitra, Roychoudhury, & Thiele, 2008). For each of the tasks we do a simulation and find out the maximum number of cycles required by the task to process each kind of macroblock. These values are then used by the automaton to determine the maximum number of cycles required by the task to process any $k$ consecutive macroblocks for a given set of design parameters which is nothing but $\gamma(k)$. There are 3 design parameters: task mapping, weights assigned to tasks and processor frequency. We will map the tasks onto the PEs, assign certain weights to the tasks which determine what portion of the processor time the task will get to execute and assign a frequency to the processor. For each set of the design parameters we will calculate $\gamma(k)$. As mentioned earlier, we can compute $\beta(\Delta)$ from $\gamma(k)$. After having computed $\beta(\Delta)$ and $\alpha(\Delta)$ we calculate the total buffer size required and then validate the design parameters for a given processor.

In the basic design space exploration we map all the different permutations of the tasks (FS, Q, MC and DCT) onto the 2 processors. Each task is given a weight and each processor is run at a given frequency. From the experiments conducted in this project we determine which design parameters produce the best performance i.e. least buffer requirement.
encoded

Figure 2. Overview of the Design Space Exploration

Figure 3. Automaton specifying the possible frame patterns according to which a raw video frame is encoded

4.2. Advancements to Basic Design Space Exploration

In order to determine the design parameters that give the best performance, one possibility would be to check all possible configurations (i.e. exhaustive design space exploration). The other possibility is to prune the design space. We have used the following search space pruning techniques:

1) If one design configuration already results in a large buffer size or delay on PE1, then we do not run the simulation for PE2.

2) In case one task requires less cycles (less computationally intensive task) than any of the other tasks, we combine the less computationally intensive tasks and a more computationally intensive task and run the simulation for the combined tasks. This reduces the number of simulations to be performed for a given architecture and given raw video.

3) Since we only need the minimum and maximum number of cycles required to process I, B or P macroblocks we only simulate how the processor processes a few macroblocks and not the whole video (which contains thousands of macroblocks). This method results in a massive decrease in simulation time.

5. EXPERIMENT

In order to study the performance of the design as shown in Figure 1 we will simulate how many cycles each task takes to process each kind of macroblock for a given processor
Table 1. Video Stream 1: Fast paced video. Each entry in the table corresponds to the number of processor cycles required to process a given macroblock (I,B,P) by a given task (FS+Q, MC, DCT).

<table>
<thead>
<tr>
<th>Task</th>
<th>Cycle Count</th>
<th>Cycle Count</th>
<th>Cycle Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>FS+Q</td>
<td>9699</td>
<td>0</td>
<td>10782</td>
</tr>
<tr>
<td>MC</td>
<td>3670</td>
<td>4237</td>
<td>11736</td>
</tr>
<tr>
<td>DCT</td>
<td>5518</td>
<td>4222</td>
<td>11018</td>
</tr>
</tbody>
</table>

Table 2. Video Stream 2: Medium paced video. Each entry in the table corresponds to the number of processor cycles required to process a given macroblock (I,B,P) by a given task (FS+Q, MC, DCT).

<table>
<thead>
<tr>
<th>Task</th>
<th>Cycle Count</th>
<th>Cycle Count</th>
<th>Cycle Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>FS+Q</td>
<td>4210</td>
<td>0</td>
<td>9465</td>
</tr>
<tr>
<td>MC</td>
<td>3876</td>
<td>3008</td>
<td>9674</td>
</tr>
<tr>
<td>DCT</td>
<td>4619</td>
<td>3345</td>
<td>9643</td>
</tr>
</tbody>
</table>

architecture of a ARM processor. The simulation is done using the sim-safe option of the SimpleSim simulator. Sim-safe has been edited for this project so that we can determine the number of cycles required for each task as opposed to the default output which produces the number of cycles required for all tasks. We use a direct-mapped instruction cache with 256 cache lines and 32 bytes block size as the given ARM architecture. We model the encoder application with four tasks FS, Q, MC and DCT.

5.1. Determining arrival rate

Each processor has 2 encoders running on it and each encoder performs some task on a separate raw video stream. The 2 incoming raw video streams 1 and 2 are encoded into a sequence of I, B and P frames, where the possible patterns of I, B and P are determined by the automaton given in Figure 3. The raw video streams are played at a rate of 30 frames per second. The resolution of the incoming frames for both the streams is 64*64 pixels, each frame is composed of 16 macroblocks each of size 16*16 pixels. One of the video streams is of a fast paced video and the other is a slow paced video.

The raw video streams entering PE1 consist of raw macroblocks. The raw macroblocks of video streams 1 and 2 entering PE1 arrive at a constant rate of 480 mbs/sec (30 frames / sec * 16 mbs / frame). Since the number of macroblocks per second is a constant the number of macroblocks that arrive in any time interval $\Delta$ is also a constant and is equal to $480 \times \Delta$. We can thus determine $a^n(\Delta) = a^l(\Delta) = 480 \times \Delta$ for video streams 1 and 2, which is the maximum and minimum number of macroblocks that arrive at PE1 in any given interval $\Delta$. After getting processed by PE1 the macroblocks of streams 1 and 2 are partially encoded. The partially encoded video streams are the input to PE2. The service received by each macroblock in PE1 will depend upon the kind of macroblock, the task scheduled on PE1, frequency of PE1 and the weight assigned to the task. Thus the number of encoded macroblocks that are being produced by PE1 per second is no longer a constant. This also implies that the number of macroblocks entering PE2 per second is not a constant. However in this project we assume that we have some method of making the number of macroblocks...
entering PE2 a constant. Thus, \( \alpha^u(\Delta) = \alpha^l(\Delta) = 480 \times \Delta \) for PE2 for both partially encoded video streams 1 and 2.

5.2. Determining workload
As a sequence of macroblocks get processed (or encoded), different tasks get executed. The worst case execution times of the four different tasks (in terms of processor cycles) for processing macroblocks of different types is given in Table 1 and 2. In Tables 1 and 2 we have combined the processor cycles for the tasks FS and Q because task FS is a computationally intensive task and requires more processor cycles when compared to the other tasks and task Q is not very computationally intensive. We also notice that the I macroblock does not get any service from the MC task. This is because there is no motion compensation required for an I frame. An I frame is an anchor frame. It is encoded as it is. The P and B frames are encoded relative to the I frame. Since we know the maximum number of cycles required to process each macroblock and the encoder automaton (Figure 3), we can determine the maximum number of cycles required by each task to process any k consecutive macroblocks \( \gamma^u(k) \) for a given video stream and given processor design. We divide \( \gamma^u(k) \) by the frequency at which the processor is running to get the maximum amount of time required to process any k number of macroblocks. The inverse of this gives us \( \beta^l(\Delta) \) which is the minimum number of macroblocks that get processed in time \( \Delta \).

5.3. Calculating Buffer Size
The frequency or the number of cycles available on a given processor per second is varied to study the effect of change in frequency on the buffer fill level. If there is only one task running on the processor all the cycles are available to that task. However when there are more than one task, we need to schedule the tasks. There are 2 encoder tasks running on our processors and we use TDMA to schedule the tasks. In our experiment we assign weights \( w_1 \) and \( w_2 \) to the 2 tasks running on the processors. The number of processor cycles per second available for the 2 tasks is \( w_1 \times \text{frequency} \) and \( w_2 \times \text{frequency} \) respectively. If we represent the time period of the scheduler as \( T \) (\( T = 1/\text{frequency} \)) then the time for which the tasks run on the processor are \( w_1 \times T \) and \( w_2 \times T \).

For each task mapping, weights and frequency of the processor we determine \( \beta^l(\Delta) \). Since we know \( \alpha^u(\Delta) \) of the streams and \( \beta^l(\Delta) \) of the processor we can determine the buffer fill level of buffer b1a, b1b, b2a and b2b. For any given time interval \( \Delta \) buffer size required is equal to \( \alpha^u(\Delta) - \beta^l(\Delta) \). Tables 3 and 4 summarize the results of the experiments conducted.

5.4. Validating the design
We will assume that we allow a maximum buffer size of 1800 macroblocks for all buffers. Any design that does not cause an overflow in the buffer is a valid design. In Tables 3 and 4 a "*" following a buffer value represents a buffer overflow. "nil" refers to the situation where no buffer is required because number of macroblocks that arrive in time interval \( \Delta \) is less than number of macroblocks that get processed in time interval \( \Delta \). "-" refers to the situation in PE2 where there is no need to calculate the buffer size because the buffer for the complementary task in PE1 has resulted in a buffer overflow (pruning technique 1). We have combined tasks FS, Q and MC because the total number of cycles required to process all these tasks is comparable to the number of cycles required to process the task DCT. DCT is a computationally intensive task (pruning technique 2). We notice that there are a
Table 3. Buffer fill level $b_1a$, $b_1b$. Task $T_1$ refers to tasks FS, MC and Q of encoder 1. Task $T_2$ refers to task DCT of encoder 2. Size of buffer required is expressed as number of macroblocks.

<table>
<thead>
<tr>
<th>Frequency of PE1</th>
<th>wt for Task $T_1$ on PE1</th>
<th>wt for Task $T_2$ on PE1</th>
<th>Buffer size for Task $T_1$</th>
<th>Buffer size for Task $T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 MHz</td>
<td>0.5</td>
<td>0.5</td>
<td>180</td>
<td>650</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>0.6</td>
<td>112</td>
<td>nil</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>0.7</td>
<td>2096*</td>
<td>nil</td>
</tr>
<tr>
<td>7.2 MHz</td>
<td>0.5</td>
<td>0.5</td>
<td>645</td>
<td>1085</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>0.6</td>
<td>112</td>
<td>302</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>0.7</td>
<td>2387*</td>
<td>nil</td>
</tr>
<tr>
<td>6.4 MHz</td>
<td>0.5</td>
<td>0.5</td>
<td>112</td>
<td>1220</td>
</tr>
<tr>
<td></td>
<td>0.4</td>
<td>0.6</td>
<td>1906*</td>
<td>824</td>
</tr>
<tr>
<td></td>
<td>0.3</td>
<td>0.7</td>
<td>2690*</td>
<td>128</td>
</tr>
</tbody>
</table>

Table 4. Buffer fill level $b_2a$, $b_2b$. Task $T_1$ refers to task DCT of encoder 1. Task $T_2$ refers to tasks FS, MC and Q of encoder 2. Size of buffer required is expressed as number of macroblocks.

<table>
<thead>
<tr>
<th>Frequency of PE2</th>
<th>wt for Task $T_1$ on PE2</th>
<th>wt for Task $T_2$ on PE2</th>
<th>Buffer size for Task $T_1$</th>
<th>Buffer size for Task $T_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 MHz</td>
<td>0.5</td>
<td>0.5</td>
<td>1298</td>
<td>nil</td>
</tr>
<tr>
<td></td>
<td>0.5</td>
<td>0.6</td>
<td>577</td>
<td>165</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
<td>0.3</td>
<td>-</td>
<td>1375</td>
</tr>
<tr>
<td>7.2 MHz</td>
<td>0.5</td>
<td>0.5</td>
<td>1097</td>
<td>nil</td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>0.4</td>
<td>1001</td>
<td>652</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
<td>0.3</td>
<td>-</td>
<td>1736</td>
</tr>
<tr>
<td>6.4 MHz</td>
<td>0.5</td>
<td>0.5</td>
<td>2037*</td>
<td>1738</td>
</tr>
<tr>
<td></td>
<td>0.6</td>
<td>0.4</td>
<td>-</td>
<td>1133</td>
</tr>
<tr>
<td></td>
<td>0.7</td>
<td>0.3</td>
<td>-</td>
<td>2103*</td>
</tr>
</tbody>
</table>

few valid designs. Some are over optimistic and do not require any buffer at all. We choose the lowest frequency that does not cause an overflow as the best design.

For PE1 we get the best performance when processor frequency is 7.2 MHz and weights of the 2 sets of tasks, FS+Q+MC of encoder 1 and DCT of encoder 2, are 0.5 and 0.5 respectively. For PE2 we notice that we get the best performance when processor frequency is 7.2 MHz and weights of the 2 sets of tasks, DCT of encoder 1 and FS+Q+MC of encoder 2, are 0.6 and 0.4 respectively.

6. CONCLUDING REMARKS

Design Space exploration of multiprocessor system-on-chip processors is very important because it gives us an idea of the performance of the processor before we actually go ahead and fabricate it. Design space exploration helps us determine the appropriate frequency, task mapping and task scheduling. The knowledge of how these design parameters affect the system-on-chip processors helps in making more efficient MpSoC processors. Less buffer size makes the system less bulky and more portable. Frequency is an important design parameter because power consumption of system-on-chip processors depends upon the frequency of the processors (Luo & Jha, 2002; Huang, Chakraborty, & Wang, 2005). Higher frequency leads to higher power consumption. However, higher frequency leads to better performance (lesser buffer size). This is a design trade off and we need to choose a design by keeping in mind the application and resources in hand. In this project the application is the MPEG 2 encoder.
It is ideal to have as less buffering as possible but since we have buffers that can hold up to 1800 macroblocks we can make use of this resource to reduce the frequency at which the processor is running.

7. REFERENCES


