On-Chip Communication in Run-Time Assembled Reconfigurable Systems

Pete Sedcole[1], Peter Y. K. Cheung[1], George A. Constantinides[1], and Wayne Luk[2]

[1]Dept. of Electrical & Electronic Engineering, Imperial College London, UK
[2]Dept. of Computing, Imperial College London, UK

Abstract— **Embedded systems in Field-Programmable Gate Arrays can be customised and adaptive if assembled from modular components at run time. This paper describes techniques for modelling inter-module channel behaviour based on statistical time division multiplexing. Where modules communicate over shared media, the proposed techniques enable systematic development of on-chip communication infrastructure to support run-time instantiation of components. Our techniques also allow system designers to guarantee that logical communication requirements between the adjunct modules can be satisfied by the infrastructure. An in-depth analysis is presented, and then verified with cycle-accurate simulations for the Sonic-on-chip reconfigurable platform for real-time video applications.**

I. Introduction

The sedulous efforts of the semiconductor industry have ensured an unrelenting increase in VLSI transistor density over the last several decades. The user-exposed transistor density of Field-Programmable Gate Arrays (FPGAs), while lagging behind that of ASICs, is nevertheless reaching levels where complete embedded systems can be implemented in a single device. To mitigate the design complexity of these systems new methodologies have been created. *Platform-based design* is one such methodology characterised by a modular architecture and extensive and planned design reuse [1].

In ASIC development, specific systems are derived from the generic platform architecture at design-time. The reconfigurability of FPGAs offers the possibility of derivative systems assembled at run-time. This we term *late integration*. Late integration enables an instance of a system to be customised to the environment in which it is deployed and adaptable to changes in the environment.

To achieve late integration additional requirements are imposed on the platform design. The communication architecture must support the run-time instantiation of modules. Moreover, the system integrator must be able to determine that the requirements of inter-modular communication channels are satisfied once the system is assembled. This is challenging, given that the requirements are not known until run-time.

This paper proposes the imposition of constraints on the architecture design such that the communication behaviour becomes analysable. We present an analysis of a constrained system, demonstrating that it is possible to guarantee throughput as well as determining the maximum channel latency and the maximum required channel buffering. The analysis is verified through the use of a cycle-accurate simulation model.

II. Related Work

On-chip communication architectures have been extensively studied. Several bus standards exist for on-chip communication (e.g., [2], [3]). Determining communication scheduling is commonly treated as a synthesis task. In bus-based systems, static scheduling is possible for fully deterministic traffic between simple low-level processes [4]. In [5], abstract communication channels (characterised by average and peak data rates) are assigned to shared media using an allocation algorithm. More complex models use trace-based analysis, which establishes a bus access pattern for each channel using the bus [6].

The fundamental disadvantage of buses is their lack of scalability, which can only be partially mitigated with bus hierarchies. This has engendered research into alternative schemes based on on-chip networks [7], [8]. The increased scalability of networks comes at a price; additional hardware is required for routing and communication scheduling is more complex. Dynamic routing may be supported with complicated routing hardware [9]. As an alternative, in [10] a discrete number of static schedules are determined, which are dynamically switched between as required. In complicated systems, on-chip network traffic exhibits time-variant behaviour, which may be modelled using statistical methods [11]. This can then be used to determine buffer sizes and reduce simulation times.

Our architecture work occupies an interesting space between pure buses and on-chip networks. In systems using late integration, the communication requirements are not known until run-time. The use of computationally intensive simulation- or trace-based methods at run-time is impractical, and therefore an alternative approach is required.

III. A Reconfigurable Platform Architecture

In this section the constrained platform architecture is described. Details of this architecture, called *Sonic-on-a-Chip*, have been published previously [12], [13]. This architecture, which targets real-time video image processing applications, has been implemented and tested in Xilinx Virtex-II Pro and Virtex-4 FPGAs. Only the main features relevant to the analysis of Section IV are summarised here.

The architecture comprises a number of processing element modules (PEs) which communication across a set of shared *SonicBuses*. SonicBuses are connected via fully buffering bridges. A series of *ChainBuses* connect each PE to its adjacent neighbours, making use of physical locality to bypass

1-4244-0155-0/06/$20.00 ©2006 IEEE

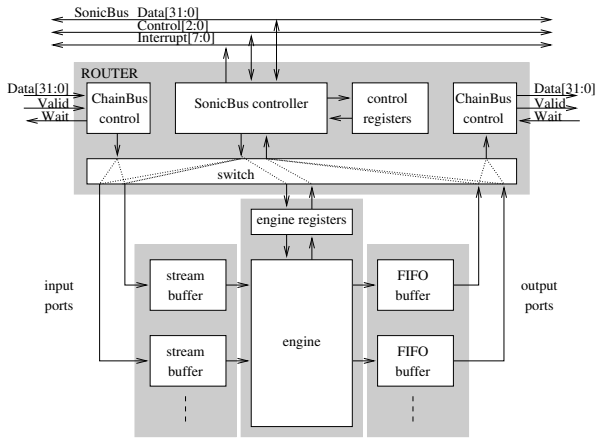

Fig. 1. The internal components and data-path of a processing element.

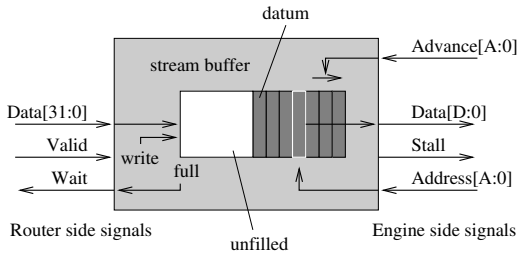

Fig. 2. The details of a stream buffer.

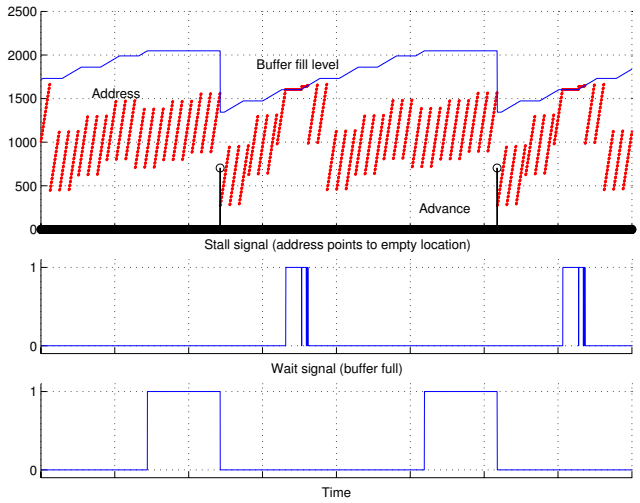

Fig. 3. The motion vector stream buffer behaviour.

the shared bus for fast local data transfers. Video data are processed as they stream through the processing elements; the Sonic processing subsystem is managed via a microprocessor-based control subsystem, which may additionally perform other processing, control and I/O tasks.

At design-time, a specific platform is created by designing the microprocessor subsystem and fixing the number of SonicBuses. Derivative systems are instantiated at run-time by selecting and attaching specific PE modules to the SonicBuses via a process of modular dynamic reconfiguration [14].

Internally, a PE comprises a router, an engine and buffering as depicted in Fig. 1. The engine processes data provided by the input stream buffer(s) and writes the resulting data to the output buffer(s). The engine design is fully user-defined; the remainder of the PE is fixed in design but has some limited scope for customisation, for example the number and size of the input and output buffers can be modified.

The router provides the interface between the input and output buffer stages and the inter-module interconnect and communication protocols. Communication in the Sonic subsystem is entirely source-driven: data are pushed from producer outputs to consumer inputs. Data-flow is programmed using the router control registers.

The output buffers are simple FIFOs, however the input stream buffers are modified so they can be used for data-reuse as well as normal buffering. This optimisation is particularly beneficial in FPGA designs where on-chip memory is significantly limited. As shown in Fig. 2, data are streamed in serially to the input buffer as in a normal FIFO, while on the output side the engine supplies the address of a queued element relative to the front of the queue to read. A `Stall` signal is asserted if the address points to an empty location, and indicates that the engine operation should stall. At each cycle, the engine-supplied `Advance[A:0]` signal will cause the queue to be shifted forward by the given number of positions.

Example 1: Stream buffers in motion vector estimation. Consider a PE which performs block-based motion vector estimation (MVE), a computationally demanding operation which is used in MPEG video compression algorithms. The task is as follows: for a given square macro-block (typically 16×16 pixels) of a reference image, find the coordinates of the most similar block in (a search window within) a target image. Thus a motion vector estimation PE could be constructed with two input stream buffers (one for the reference macro-blocks, one for the search windows). Fig. 3 is a graph of the state of the search window buffer in a simulation of a motion vector estimator PE which uses a three-step coarse-fine search algorithm. Note that since the search windows overlap, only some of the data are discarded from the buffer after performing each search. □

The trichotomy of the engine, router and buffering within each PE ensures that computation and inter-module communication are kept entirely separate, ensuring predictability in communication behaviour, and is a necessity in the analysis to follow in Section IV.

Two communication protocols are supported for transmission of information on the shared SonicBus. The first is in essence a statistical time division multiplexing (STDM) scheme. A channel is formed by transmission of data from an output buffer of one PE to an input stream buffer of another PE. A number of consecutive bus cycles are allocated to a specific channel; if the channel becomes inactive during its allotted time (either by exhausting data to transmit or a lack of space to at the consumer end) the bus is released early for re-arbitration. This is illustrated in Fig. 4. Note that the overhead

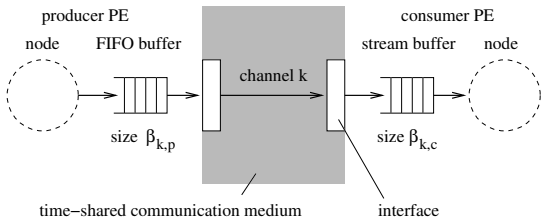

Fig. 5. Components in a communication channel mapped to a shared medium. The channel is buffered on the producer side by a FIFO of depth $\beta_{k,p}$ and on the consumer side by a stream buffer of depth $\beta_{k,c}$. The channel interfaces regulate the access to the communication medium.

of the STDM scheme is 3 cycles per channel transfer.

A second protocol is based on message passing, and is used infrequently, typically for initialising registers during instantiation of a PE module.

The routers within each PE are responsible for implementing the protocols, with the assistance of an arbitration unit (one for each bus). The arbitration unit includes a table which is programmed with the time-slot size of each channel sharing the bus.

IV. Communication Analysis

One of the most challenging issues in automatic run-time assembly of derivative systems is mapping communication channels to shared resources while ensuring performance targets are met. In this section we develop an analysis of the communication system based on the platform architectural template described above. The objectives of this analysis are, for each channel, (a) to determine the arbitration parameters, (b) estimate the minimum required amount of buffering, and (c) estimate the maximum latency.

The analysis begins with a description of the necessary assumptions regarding the communication system. It should be noted that the analysis is not limited to Sonic-on-a-Chip, or even video processing, but may be applied to any communication system designed with similar constraints.

A. Scenario and Assumptions

We start with the assumption that the processing system is a process network comprises a number of processing element nodes (PEs) connected by a series of communication channels, which is to be mapped to a system of buses connected by bridges. The features of a channel are illustrated in Fig. 5. Assume that each node has been assigned to a bus. By using bridges which buffer data, the behaviour of each bus can be isolated and studied separately. Channels assigned to ChainBuses are ignored as they are of no interest in this analysis. For a particular bus, we need to set the size of the time-slot for each channel to ensure throughput is met.

In the analysis which follows, the processing nodes are assumed to have a common pattern of behaviour: One or more streams of data are stored in input buffers; the engine performs a number of accesses on the stored data and outputs results to the output buffer; input data which are no longer needed are discarded. This pattern is repeated indefinitely, such that the processing node has a base-line periodicity. Note that in some cases a node may exhibit different input and output periodicity; for example, a node which computes a histogram of the intensity values of an image may access and discard pixels one at a time (input periodicity of 1 pixel), whereas the results are only presented to the output buffer once per frame (output periodicity of 1 frame).

TABLE I
Characteristics of video processing algorithms

Algorithm	Periodicity Input	Periodicity Output	Advance	Stored data
Window function (5×5 window) e.g. median filter	25	1	1	$4 \times c + 5$
2D convolution (3×3 kernel)	9	1	1	$2 \times c + 3$
Histogram (3 colour channels, 256 points)	1	768	1	1
1D DFT (parallel)	c	c	c	c
Motion vector estimation (16×16 macro-block)	6912	1	704 / 256	1936 / 256

Table I lists some sample video processing algorithms and shows how they may be parameterised in pixels. All algorithms (with the exception of the motion vector estimator) process non-interlaced raster-scanned images of height r and width c.

B. First Approximation

A first approximation analysis is formulated by assuming unlimited buffer sizes. The aim is to determine, for a given mapping of channels to a bus, what size time-slot to allocate to each channel, and the required amount of channel buffering.

Consider a bus with maximum bandwidth Γ supporting N channels. Each channel k has a required average throughput ϕ_k, and is allocated ω_k consecutive bus cycles for each data transfer (at one word per cycle) excluding the STDM overhead of h cycles. The average bandwidth required must be less than that available:

$$\sum_{k=1}^{N} \phi_k = \Phi < \Gamma \tag{1}$$

If data are produced and consumed at a constant rate (ϕ_k for each channel k) and there are no buffer overflows, then the service period (the time taken for all channels to have completed one transfer each, see Fig. 4) is:

$$\tau = \frac{1}{\Gamma} \sum_{k=1}^{N} (\omega_k + h) = \frac{1}{\Gamma} \left(\sum_{k=1}^{N} \omega_k + Nh \right) \tag{2}$$

During this time, $\phi_k \tau$ data are produced for channel k. In steady state $\phi_k \tau = \omega_k$. So we can solve for τ:

$$\tau = \frac{1}{\Gamma} (\tau \Phi + Nh) = \frac{Nh}{\Gamma - \Phi} \tag{3}$$

Also:

$$\omega_k = \frac{\phi_k N h}{\Gamma - \Phi} \tag{4}$$

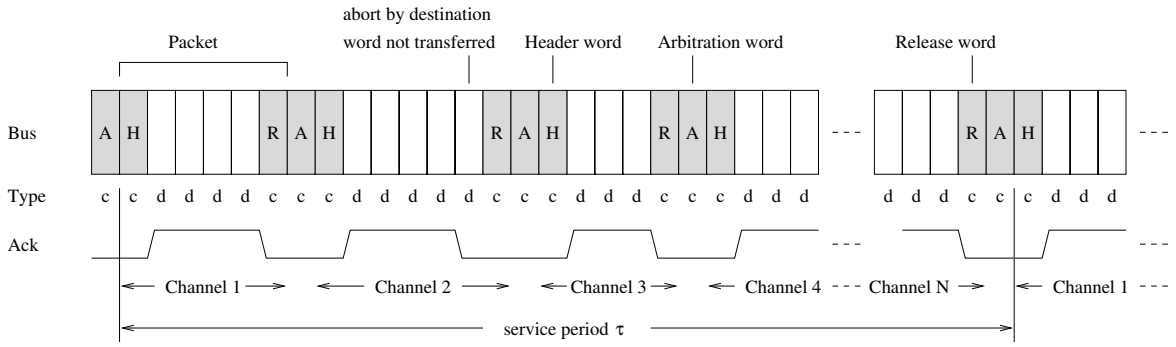

Fig. 4. STDM communication protocol.

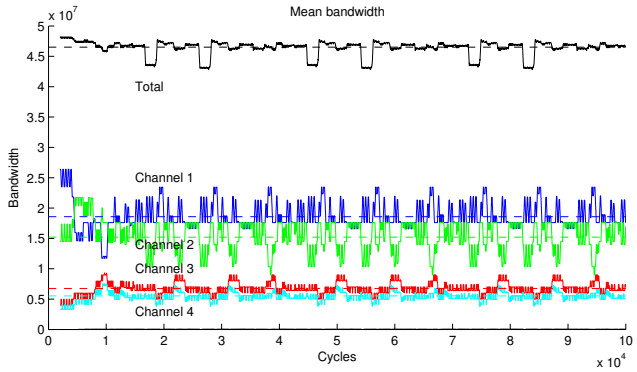

Fig. 6. Graph of motion vector estimation search window buffering. The address pattern shows all possible address accesses over one fundamental period. Buffer fill levels for non-saturating and saturating buffer conditions are shown.

This value ω_k is the time-slot size for channel k, and is used as the cycle count limit in the programming of the arbitration unit. The minimum source buffer size $\beta_{k,\mathrm{p}}$ for each channel is the number of words which need to be stored during the time the channel does not have control of the bus:

$$\beta_{k,\mathrm{p}} = \omega_k \left(1 - \frac{\phi_k}{\Gamma}\right) \tag{5}$$

However, avoiding buffer saturation comes at a cost of greater than necessary consumer side buffering; this is highly non-desirable as on-chip memory is limited, and particularly so in FPGAs.

C. Size-limited Buffers

In order to account for limited buffer sizes, now assume that buffers may saturate. We will determine the time-slot sizes and requirements for the buffers which do not saturate.

In this case, channel throughput is no longer constant, but has inactive periods (when the consumer-side buffer is full), which must be balanced by periods where the throughput is higher than average. This is shown in Fig. 6 for the motion vector estimator of Example 1. The graph shows the number of words in the input stream buffer (the fill-level) for the cases where there is no buffer saturation and where the buffer does saturate. The rate at which the buffer is filled is slightly higher in the saturating case. There are several important features to be noted:

1) In the analysis which follows, we also take into account the pattern of locations accessed in the buffer. In Fig. 6 all possible accesses are plotted for the MVE example. The buffer must be filled sufficiently quickly such that data accesses are all within the available buffered data.
2) To simplify the calculation of the required fill rate for a given access pattern, not all addressed locations need to be considered. The required fill rate can be calculated from the envelope of possible accessed locations.
3) It is assumed that the engine processing rate will be at least as fast as the overall required throughput rate, and potentially faster. This is accounted for by introducing an allowable 'stall time' per fundamental period when determining the required fill rate. This can be seen to be 1000 cycles in the example of Fig. 6.

Now divide the N channels into M channels whose activity is time-variant (denoted V), due to buffer saturation, and $N - M$ time-invariant (I) channels. One can visualise the activity of the time-variant channels as being a cyclic pattern, with a period where there is a burst of activity followed by a period where there is no bandwidth demand once the consumer-side buffer has saturated. The time-variant channels have a required peak throughput $\phi'_k,\ 1 \le k \le M$. The bus must be able to support concurrent peak demands of the V-channels:

$$\Phi_{\mathrm{V}} = \sum_{k=1}^{M} \phi'_k < \Gamma \tag{6}$$

If the peak demand on the bus, including the time-invariant channels, is less than the bus capacity:

$$\Phi_{\mathrm{peak}} = \sum_{k=1}^{M} \phi'_k + \sum_{k=M+1}^{N} \phi_k < \Gamma \tag{7}$$

then (4) can be used to calculate the STDM time-slot parameters ω_k by substituting ϕ'_k for ϕ_k and Φ_{peak} for Φ. If the inequality of (7) does not hold, let us term the bus usage *critical*. In a bus with a critical level of usage, bandwidth demands vary over time. During periods of peak activity by the V-channels the remaining I-channels are starved of bandwidth.

This is compensated for during off-peak times. As a result the I-channels have increased buffering requirements.

Consider the case where $M = 1$: there is one time-variant, saturating channel b. The peak demand on bus bandwidth is:

$$\Phi_{\text{critical}} = \Phi_{\text{V}} + \hat{\Phi}_{\text{I}} \tag{8}$$

where $\hat{\Phi}_{\text{I}}$ is the reduced total bandwidth available to the $N-1$ I-channels. Rearranging (4) and substituting variables:

$$\Phi_{\text{critical}} = \Gamma - \frac{Nh\phi'_b}{\omega_b} \tag{9}$$

(b is the time-variant channel) and also:

$$\omega_k = \frac{\hat{\phi}_k Nh}{\Gamma - \Phi_{\text{critical}}}, \quad M < k \leq N \tag{10}$$

for the I-channels. The variable $\hat{\phi}_k$ is the bandwidth available to channel k during the peak demand times, and is given by:

$$\hat{\phi}_k = \phi_k \frac{\hat{\Phi}_{\text{I}}}{\Phi_{\text{I}}}, \quad M < k \leq N \tag{11}$$

From these equations it will be possible to determine the time-slot size (ω_k) to allocate to each of the time-invariant channels, provided a value can be found for Φ_{critical} first.

Using Eq.s (8), (9), (10) and (11) we can derive:

$$\omega_k = \frac{\phi_k \omega_b}{\Phi_{\text{I}} \phi'_b} \left(\Gamma - \frac{\phi'_b Nh}{\omega_b} - \Phi_{\text{V}} \right) \tag{12}$$

Now, the service period τ in the critical bus usage case also varies over time. During peak activity periods by the V-channels the service time will be longer than when these same channels are idle. For the $M = 1$ case, during the off-peak period (when channel b is idle) the service period is given by:

$$\tau_{\text{offpeak}} = \frac{1}{\Gamma} \left(\sum_{k=M+1}^{N} \omega_k + Nh \right) = \frac{Nh}{\Gamma - \check{\Phi}_{\text{I}}} \tag{13}$$

where $\check{\Phi}_{\text{I}}$ is the off-peak bandwidth demand of the I-channels. Now substitute (12) in (13), and simplify, noting that $\Phi_{\text{V}} = \phi'_b$ in this case and $\Phi_{\text{I}} = \sum_{k=M+1}^{N} \phi_k$. Solve for ω_b:

$$\omega_b = \frac{\phi'_b Nh\Gamma}{(\Gamma - \phi'_b)(\Gamma - \check{\Phi}_{\text{I}})} \tag{14}$$

An expression for $\check{\Phi}_{\text{I}}$ must now be found. The channel b will be active for ϕ_b/ϕ'_b of the time, during which each time-invariant channel k has bandwidth $\hat{\phi}_k$. In order for the average bandwidth requirement for channel k to be met:

$$\frac{\phi_b}{\phi'_b} \hat{\phi}_k + \left(1 - \frac{\phi_b}{\phi'_b} \right) \check{\phi}_k = \phi_k, \quad M < k \leq N \tag{15}$$

Therefore:

$$\check{\Phi}_{\text{I}} = \sum_{k=M+1}^{N} \check{\phi}_k = \frac{1}{1 - \frac{\phi_b}{\phi'_b}} \sum_{k=M+1}^{N} \left(\phi_k - \frac{\phi_b}{\phi'_b} \hat{\phi}_k \right) \tag{16}$$

$$= \frac{1}{1 - \frac{\phi_b}{\phi'_b}} \left(\Phi - \frac{\phi_b}{\phi'_b} \Gamma + \frac{\phi'_b Nh}{\omega_b} \right) \tag{17}$$

TABLE II
CHANNEL CHARACTERISTICS FOR EXAMPLE 2.

Channel	Mean bandwidth ϕ_k (Mwords/s)	Peak bandwidth ϕ'_k (Mwords/s)	ω_k
1	18.59	24.84	235
2	15.21	15.30	145
3	6.76	6.76	40
4	5.53	5.53	33
5	0.03	0.03	1
6	0.03	0.03	1
Total	46.1	52.5	

So finally:

$$\omega_b = \left(\frac{\phi'_b Nh}{\Gamma - \Phi} \right) \left(\frac{\Gamma - \phi_b}{\Gamma - \phi'_b} \right) \tag{18}$$

This can be generalised for situations where $M > 1$:

$$\omega_b = \left(\frac{\phi'_b Nh}{\Gamma - \Phi} \right) \left(\frac{\Gamma - \sum_{b=1}^{M} \phi_b}{\Gamma - \sum_{b=1}^{M} \phi'_b} \right) \tag{19}$$

All the variables in this equation are known, so ω_b can be calculated for all V-channels $b \leq M$. One of these channels is then used to find Φ_{critical} using (9). This can be used to find the values for ω_k for the remaining I-channels $M < k \leq N$. This is illustrated in the following example.

Example 2: Calculation of arbitration parameters. A system comprises two of the motion vector estimation process nodes (Example 1) processing 640×480 sized images at different frame rates (22 and 18 frames-per-second). Each node has two input channels (the reference block and the search window) and one output channel (the vectors), making six channels in total, with an overall mean bandwidth of 46.1Mw/s, mapped to a bus with capacity 50Mw/s.

On inspection of the address patterns, buffer sizes and consumption behaviour of the channels it is determined that two of the destination channel buffers will saturate, increasing the peak bandwidth demand to 52.5Mw/s, as shown in Table II.

Using (19), and $N = 6$, $h = 3$, $\Gamma = 50$Mw/s, $\Phi = 46.1$Mw/s, $M = 2$, we find $\omega_1 = 210.6$ and $\omega_2 = 129.7$. The critical bandwidth demand is $\Phi_{\text{critical}} = 47.9$Mw/s from (9), and from (8) we find that $\hat{\Phi}_{\text{I}} = 7.74$Mw/s. Therefore, using (11) and (10) we find the values $\omega_k = \{35.9, 29.4, 0.1, 0.1\}$ for $k = \{3, 4, 5, 6\}$. These are rounded up to integer values, while ensuring that the ratio $\omega_k : (\sum_{i=3}^{6} \omega_i + Nh)$ for each k does not decrease in the process, giving $\omega_k = \{40, 33, 1, 1\}$ for $k = \{3, 4, 5, 6\}$. After similarly rounding and adjusting ω_1 and ω_2 we obtain the values for ω_k as shown in the right column of Table II. □

D. Buffer Sizing and Latency

We have so far found a method for determining the time-slot sizes to use in the bus arbitration, including situations in which limited buffering causes time-variant behaviour on some channels. We now must determine the required buffer space on the producer-side of these channels and the effect on the

size of the buffers for the remaining channels in the system. If the bus usage is critical, channels which do not exhibit time-variant behaviour require extra buffer space to compensate for periods where their bandwidth is temporarily restricted.

Consider a channel k which is a time-invariant channel: its bandwidth demand is constant. Due to the changes in bandwidth demands by time-variant channels, the *actual* throughput of channel k will be time-varying: $\phi_k(t)$. On the consumer side of the channel, there must be extra buffer space $\Delta\beta_k$ sufficient to prevent supply the processing node engine without causing stalls during deviances from the average throughput rate ϕ_k. Thus:

$$\Delta\beta_{k,\mathrm{c}} \geq \int_{t_1}^{t_2} \phi_k - \phi_k(t)dt \quad \forall\{t_1, t_2\} : t_1 < t_2 \tag{20}$$

Determining the buffer sizes requires finding the worst cases for (20). This occurs when the throughput $\phi_k(t)$ reduces, due to all V-channels being active concurrently. Assuming the active channels are not source-limited and therefore (using the STDM protocol) consume the maximum amount of bandwidth available to them when active. The V-channels when idle due to buffer saturation consume a single bus cycle of their allocated time-slot before releasing the bus for arbitration. By inspection of Fig. 4, one can observe that the time-varying throughput of channel k is therefore:

$$\phi_k(t) \approx \frac{\Gamma\omega_k}{\left[\sum_{i=1}^{M} \alpha_i(t)\right] + \left[\sum_{i=M+1}^{N} \omega_i\right] + Nh} \tag{21}$$

where for time-variant channel i, $\alpha_i(t) = \omega_i$ when the channel is active and $\alpha_i(t) = 1$ at other times. Therefore, the evaluation of the integral of (20) is computationally not difficult, since the worst-case (approximate) $\phi_k(t)$ is piecewise constant. However, it is necessary to determine the active and inactive times for each channel, and the interval (t_1, t_2). Assume that all M burst channels become active at time $t_1 = 0$, and each channel i has periodicity T_i, determined by the periodicity of the node it supplies. The procedure for determining the channel active times is relatively straightforward:

1) At time $t = 0$, each active channel $i \leq M$ starts with a number of words $r_i(0) = \phi_i T(i \to n)$ to be transfered before the channel will become inactive again.
2) For each channel calculate the transfer bandwidth $\phi_k(0^+)$ from (21).
3) Determine the time for the first channel to become inactive:

$$d_1 = \min\left(T_i \frac{\phi_i}{\phi_i(0^+)}\right), \quad \forall i \leq M \tag{22}$$

This channel is marked as inactive for $t > d_1$.

4) Record the number of words remaining to be transfered at time $t = d_1$ in the other channels:

$$r_i(d_1) = r_i(0) - \phi_i(0^+)d_1 \tag{23}$$

5) For each subsequent stage $n = \{1, 2, \dots\}$, the duration of the stage is given by:

$$d_{n+1} = \min \begin{cases} \dfrac{r_i(d_n)}{\phi_i(d_n^+)} & \text{active channels} \\ q_i T_i - d_n & \text{inactive channels} \end{cases} \tag{24}$$

where $\phi_i(d_n^+)$ can be calculated from (21) and

$$r_i(d_n) = r_i(d_{n-1}) - \phi_i(d_{n-1}^+)(d_n - d_{n-1}) \tag{25}$$

In the term $q_i T_i - d_n$, q_i is an integer value that is incremented each time channel i becomes inactive. At each stage, one channel becomes active or inactive, depending on which term in (24) is minimum.

For each channel k there will be a time d_p at which $\phi_k(d_p^+) > \phi_k$. The integral of (20) is therefore calculated between $(0, d_p)$.

On the source side, the equation is slightly different. The producer FIFOs must be large enough to contain data generated by the node without causing a stall, even when the data generation rate is not constant. If the producer for channel k is node n and generates data at a rate $p_n(t)$, then the equation for the buffer space required is:

$$\Delta\beta_{k,\mathrm{p}} \geq \int_{t_1}^{t_2} p_n(t) - \phi_k(t)dt \quad \forall\{t_1, t_2\} : t_1 < t_2 \tag{26}$$

To simplify this, we will compute a conservative estimate for the upper bound, by setting $p_n(t)$ to a periodic function:

$$p_n(t) = \begin{cases} p'_n & 0 < t < \frac{\phi_k T_n}{p'_n} \\ 0 & \frac{\phi_k T_n}{p'_n} < t < T_n \end{cases} \tag{27}$$

Here p'_n is the peak rate at which node n can produce data, and T_n is the periodicity of the node. Equation (26) is now a piecewise constant function and can be computed in a similar way to (20).

For a time-variant channel b, the source side buffer must be sufficiently large to hold the data produced while the consumer-side buffer has saturated. Again, (26) must be evaluated, however in this case we find the worst case conditions by assuming that the destination buffer saturates at time $t = 0$ and $\phi_k(t)$ is the periodic function:

$$\phi_k(t) = \begin{cases} 0 & 0 < t < T_k\left(1 - \frac{\phi_k}{\phi'_k}\right) \\ \phi'_k & T_k\left(1 - \frac{\phi_k}{\phi'_k}\right) < t < T_k \end{cases} \tag{28}$$

In addition to the buffer space required resulting from variations in throughput, the buffer levels also ripple up and down over the duration of each service period τ. The height of this ripple is given by:

$$\Delta\gamma_k \geq \frac{\phi_k}{\Gamma}\left(\sum_{i \neq k} \omega_i + Nh\right) \tag{29}$$

The total spare buffer space required is found by adding $\Delta\beta_k$ and $\Delta\gamma_k$. Finally, the maximum latency introduced by the

TABLE III
SPARE BUFFER SPACE AND LATENCY FOR EXAMPLE 3.

Channel	$\Delta\gamma_k$	$\Delta\beta_k$	Total (words)	Latency (μs)
1	178	89	267	14.4
2	100	5	105	6.9
3	59	72	131	19.4
4	49	58	107	19.4
5	1	0	1	37.9
6	1	0	1	46.3

channel can be approximated by the buffer size and the average throughput rate:

$$l_k \leq \frac{\Delta\beta_k + \Delta\gamma_k}{\phi_k} \tag{30}$$

Example 3: Buffer sizing and latency. We now calculate the required buffer space for each channel from Example 2. Assume that data are fed into channels 1 to 4 at a constant rate, such that for input node n and corresponding channel k, $p'_n = \phi_k$. Thus, $\Delta\beta_{k,\mathrm{c}} = \Delta\beta_{k,\mathrm{p}} = \Delta\beta_k$. Channel 1 has periodicity $T_1 = 37.9\mu$s, and for channel 2, $T_2 = 46.3\mu$s. We find $d_1 = 28.3\mu$s (channel 1 becomes inactive) and $d_2 = 37.3\mu$s (channel 2 becomes inactive). For channel 3, from (21),

$$\phi_3(t) = \begin{cases} 4.23\text{Mw/s} & 0 < t < d_1 \\ 8.37\text{Mw/s} & d_1 < t < d_2 \end{cases}$$

Therefore:

$$\Delta\beta_3 = \left\lceil \int_0^{d_1} \phi_3 - \phi_3(t)dt \right\rceil = \left\lceil d_1(\phi_3 - \phi_3(0^+)) \right\rceil = 72$$

The ripple for channel 3:

$$\Delta\gamma_3 = \left\lceil \frac{\phi_3}{\Gamma} (\omega_1 + \omega_2 + \omega_4 + \omega_5 + \omega_6 + 6 \times 3) \right\rceil = 59$$

Other values for the buffers are listed in Table III. Note that for channels 1 and 2 (where consumer-side buffers saturate) the buffering values are the minimum producer-side buffer sizes. For channels 3 to 6, the totals are the minimum producer-side buffer sizes, and the total spare capacity required before saturation of the consumer-side buffers. □

E. Method Summary

The aim of the analysis is to show how the system designer can ensure that derivative designs constructed at run-time will achieve required performance when sharing communication media. The process is summarised as follows. At design time, the system designer collects the following information about each processing node:

1) The envelope of the address pattern, including the baseline repeat period.
2) The magnitude of the consume delta function.
3) The maximum theoretical processing throughput, assuming no stalling due to lack of data or output buffer space.
4) The input and output buffer sizes.

Algorithms are created from communicating clusters of nodes. At run-time, the processing nodes are assigned to SonicBuses. Application software determines the requirements for the communication between nodes. The run-time operating system then verifies the performance requirements can be fulfilled by executing the following steps:

1) Calculate the required throughput for each node and channel, based on algorithmic throughput requirements.
2) Verify mean demand on each bus does not exceed available bandwidth.
3) Determine for each node the stall time for the engine.
4) From the stall time, the required throughput, and the address pattern envelope, determine for each channel if the destination buffer will saturate, or if not, the spare capacity in the buffer.
5) Based on the buffer saturation, divide channels by their bandwidth demand into time-invariant and time-variant.
6) Calculate the peak bandwidth demand of the time-variant channels, and verify the aggregate peak bandwidth demand is less than the bandwidth available.
7) Calculate the time-slot size (arbitration count) for each channel (ω_k).
8) Calculate the required spare buffer capacity for all source-side buffers and all time-invariant channel destination buffers, and verify this is less than the available capacity.
9) Verify the latency of each channel is acceptable.

If each verification step in the process is successful, the required performance will be achieved. If any step fails, a number of options are available. A different bus assignment may be selected, if the bandwidth requirements of different buses are mismatched. An alternative algorithm may be used with lower performance requirements (with a lower quality of service for example), or an alternative algorithm implementation. The high-level decision made here is system-dependent, and not covered in the scope of this paper.

It is emphasised that the run-time evaluation of communication parameters involve calculations with low computational intensity, and moreover the evaluation is performed infrequently relative to the operation time of the algorithms (which process continuous streams of video data). Thus the overhead incurred is slight.

V. EXPERIMENTAL RESULTS

To verify empirically the validity of the analysis presented above, cycle-accurate simulations of the communication system have been developed, modelled on implementations of Sonic-on-a-Chip in Virtex-II Pro and Virtex-4 FPGAs. The example system described in Example 2, comprising two motion vector estimation nodes, is used again in the simulations. The parameters calculated in Example 2 and Example 3 and listed in Table II and Table III are used as nominal values. The simulated system has four input nodes; the rate at which these node supply data to the system is independently adjustable.

Fig. 7 is a graph of the time-averaged bandwidth of the four main channels (1 to 4) and the overall bus bandwidth usage over a period of 2ms (10^5 bus cycles). For this simulation all input nodes were set to supply data at the maximum output

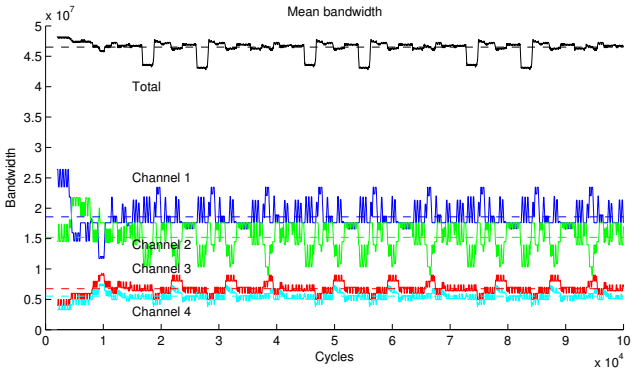

Fig. 7. Bandwidths of the channels in the simulated system, averaged over 2000 cycles.

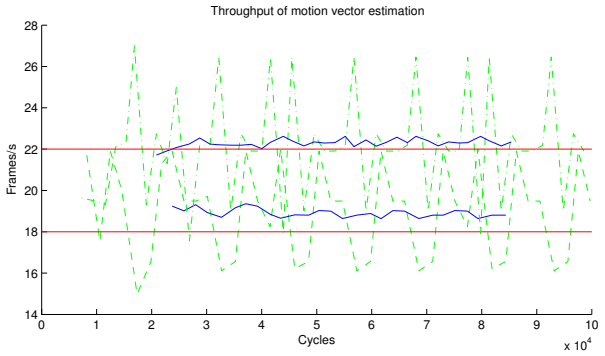

Fig. 8. The throughput of the motion vector estimators from the simulation.

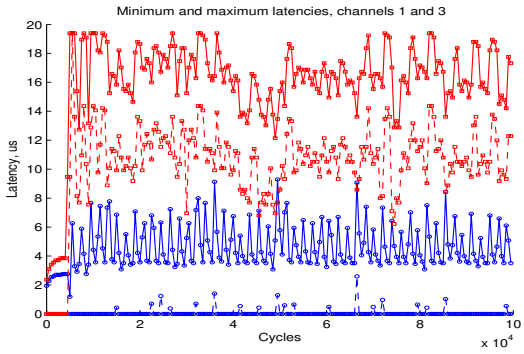

Fig. 9. The maximum (solid lines) and minimum (dashed lines) latencies of channels 1 (circles) and 3 (squares).

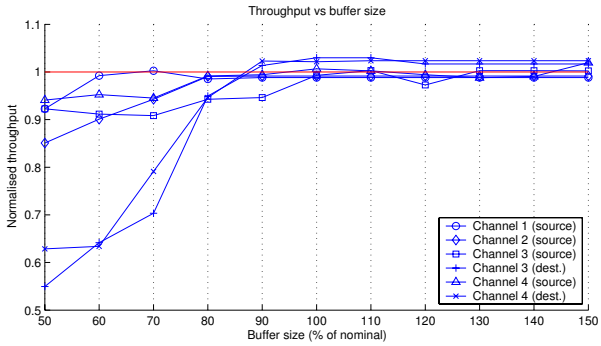

Fig. 10. Effect of varying buffer sizes on system throughput.

rate (one word per cycle). The graph shows that the STDM arbitration scheme is able to cope with high overall bandwidth utilisation and allocate bandwidth to each channel appropriately despite the time-variable demands of the channels. Note the mean bandwidth for each channel is as expected from Table II.

The block-by-block frame rates achieved by the motion vector estimators are plotted in Fig. 8; again all input nodes supply data at an unmoderated rate. Variability in the throughput rate is expected due to the variation in latency in communication. The latency for the channels supplying the 22 fps MVE processing node (channels 1 and 3) was calculated to be between 0μs and 19.4μs (in Table III), and confirmed in the simulation (see Fig. 9). Note that the time-averaged throughputs of the two MVE nodes meets the designed rates of 22 and 18 fps.

In real systems, data would not necessarily be supplied at uncontrolled rates. The communication system must cope with situations in which some channels have rate-limited data sources, while others are unmoderated. Regardless of the source data-rate, the communication parameters can be calculated based on the desired system throughput. This was tested by measuring the performance of the communication system while moderating the supplied data rate of each of the four inputs to the system. Sixteen combinations of source-rate limiting were simulated. The expected MVE throughputs were calculated and compared with the measured performance. Over the 16 different experiments, the achieved throughput ranged from -1.74% to +5.49% of the expected throughput, with a mean of +0.57% and a variance of 0.02%. This demonstrated that the communication scheme could correctly arbitrate between channels regardless of the supplied data rate, and the time-slot size calculations remain valid.

The buffer size values in Table III were verified by varying the values by $\pm 50\%$ of the nominal value and measuring the corresponding throughput for the affected motion vector estimator. The outcomes are plotted in Fig. 10, normalised to the buffer sizes of Table III. It can be seen that the calculated required buffer sizes are sufficient to avoid degrading the system throughput performance. In addition, the buffer sizes calculated are not significantly larger than necessary in this instance, with the exception of the source buffer for channel 1, which appears to be oversized by around 50%. In general, the calculated required buffer sizes are based on worst-case conditions, which may never occur in a given system, and therefore the calculations result in conservative estimates.

Thus far, for consistency all examples and experiments have been based on a single node type, namely motion vector estimators. To demonstrate the applicability of our approach with a wider variety of node types, in addition to the two-MVE system described above (from now denoted `sys1`) four other sample systems (`sys2` to `sys5`) were designed and simulated. As shown in Table IV, each system has a different mixture of processing node types providing different communication

TABLE IV
CHARACTERISTICS OF THE SIMULATED SYSTEMS.

	sys1	sys2	sys3	sys4	sys5
Subsystem	*Node throughput rate, fps*				
MVE 1	22.0			20.0	22.0
MVE 2	18.0				
Block 2D DCT		22.0	32.0	32.0	
Foreground separation		22.0	18.0		
Median filter			20.0		
Histogram				12.0	22.0
	Bus parameters				
Channels (N)	6	5	7	7	5
Time-variant channels (M)	2	0	1	2	1
Average total bandwidth Φ (Mw/s)	46.1	33.8	48.5	46.4	32.1
Peak total bandwidth Φ_{peak} (Mw/s)	52.5	33.8	50.9	50.9	38.4
Critical	yes	no	yes	yes	no

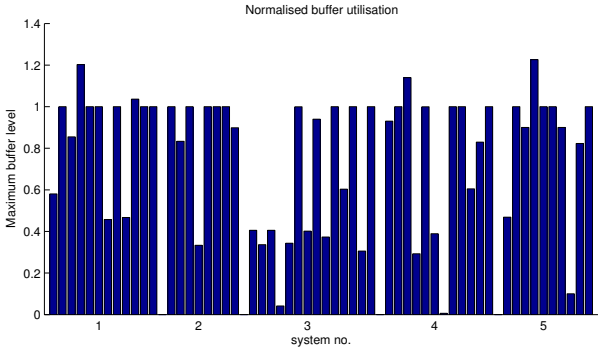

Fig. 11. A graph of the maximum fill levels of every channel buffer in each of the five simulated systems, normalised to the expected values for each channel.

behaviour.

Arbitration values (ω_k) and buffer sizes were calculated for all systems as per the method given in Section IV-E. In these simulations, the buffer size estimates were verified by oversizing all non-saturating buffers and recording the maximum utilisation point for every source and destination buffer. The results are plotted in Fig. 11. In 52 of the 56 buffers the fill level never exceeds the calculated expected buffer size. For buffers where the expected maximum level was exceeded, further investigation of the simulations revealed this was due to transient behaviour during the start-up of the system. When restricted in size, the temporary saturation of these buffers did not affect the steady-state throughput of the system.

VI. CONCLUSION

Platform-based design in Field-Programmable Gate Arrays offers the unique prospect of derivative systems created automatically at run-time. Such systems can be customised and adaptable to variations in the operating conditions. However, ensuring that communication requirements are met between run-time instantiated modules is a non-trivial task.

In this paper it was shown how inter-modular communication behaviour can be limited by separating the communication interface from the computational component within each module. Limiting the expressible behaviour allows communication to become analysable.

An analysis of a communication scheme based on statistical time division multiplexing was presented. This resulted in simple calculations, suitable for executing at run-time, for determining arbitration parameters, minimum buffer sizes and maximum latencies. The analysis was verified by cycle-accurate simulations.

Current and future work include refining our analysis techniques, and extending them to cover a wide range of multimedia systems.

ACKNOWLEDGMENTS

P. S. gratefully acknowledges the financial support provided by the Commonwealth Scholarship Commission and the New Zealand Vice Chancellors' Committee. The support from Xilinx Inc. is also greatly appreciated. The authors would like to thank Kostas Masselos for his helpful suggestions.

REFERENCES

[1] H. Chang, L. Cooke, M. Hunt, G. Martin, A. J. McNelly, and L. Todd, *Surviving the SOC Revolution: A Guide to Platform-Based Design*. Norwell, MA, USA: Kluwer Academic Publishers, 1999.
[2] *AMBA*, ARM Ltd. Spec., Rev. 2.0, 1999.
[3] *The CoreConnect Bus Architecture*, IBM Inc. White paper, 1999.
[4] M. Gasteier and M. Glesner, "Bus-based communication synthesis on system level," *ACM Trans. Design Automation of Electronic Systems*, vol. 4, no. 1, pp. 1–11, Jan 1999.
[5] J.-M. Daveau, T. B. Ismail, and A. A. Jerraya, "Synthesis of system-level communication by an allocation-based approach," in *Proc. Int. Symp. System Level Synthesis*, 1995.
[6] K. Lahiri, A. Raghunathan, and S. Dey, "System-level performance analysis for designing on-chip communication architectures," *IEEE Trans. Computer-Aided Design*, vol. 20, no. 6, pp. 768–83, June 2001.
[7] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. Design Automation Conf.*, 2001.
[8] L. Benini and G. De Micheli, "Networks on chips: A new SoC paradigm," *IEEE Computer*, vol. 35, no. 1, pp. 70–78, January 2002.
[9] E. Waingold, M. Taylor, D. Srikrishna, V. Sarkar, W. Lee, V. Lee, J. Kim, M. Frank, P. Finch, R. Barua, J. Babb, S. Amarasinghe, and A. Agarwal, "Baring it all to software: RAW machines," *IEEE Computer*, vol. 30, no. 9, pp. 86–93, Sept 1997.
[10] J. Liang, A. Laffely, S. Srinivasan, and R. Tessier, "An architecture and compiler for scalable on-chip communication," *IEEE Trans. VLSI Syst.*, vol. 12, no. 4, pp. 711–26, July 2004.
[11] G. V. Varatkar and R. Marculescu, "On-chip traffic modeling and synthesis for MPEG-2 video applications," *IEEE Trans. VLSI Syst.*, vol. 12, no. 1, pp. 108–119, Jan 2004.
[12] N. P. Sedcole, P. Y. K. Cheung, G. A. Constantinides, and W. Luk, "A reconfigurable platform for real-time embedded video image processing," in *Proc. Int. Conf. Field-Programmable Logic and Applications*, 2003.
[13] P. Sedcole, P. Y. K. Cheung, G. A. Constantinides, and W. Luk, "A structured methodology for System-on-an-FPGA design," in *Proc. Int. Conf. Field-Programmable Logic and Applications*, 2004.
[14] P. Sedcole, B. Blodget, J. Anderson, P. Lysaght, and T. Becker, "Modular partial reconfiguration in Virtex FPGAs," in *Proc. Int. Conf. Field-Programmable Logic and Applications*, 2005.