

# Performance and Area Modeling of Complete FPGA Designs in the presence of Loop Transformations

K.R. Shesha Shayee, Joonseok Park, and Pedro C. Diniz

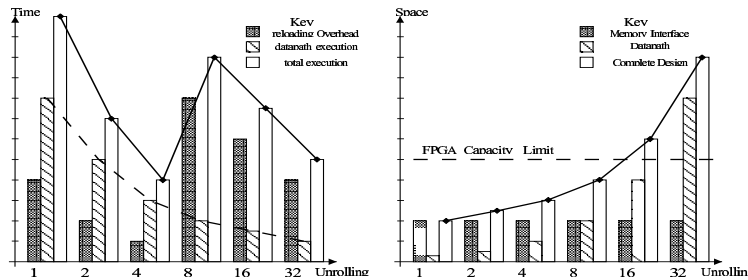
University of Southern California / Information Sciences Institute  
4676 Admiralty Way, Suite 1001  
Marina del Rey, California 90292, USA,  
{shesha, joonseok, pedro}@isi.edu

**Abstract.** Selecting which program transformations to apply when mapping computations to FPGA-based architectures leads to prohibitively long design exploration cycles. An alternative is to develop fast, yet accurate, performance and area models to understand the impact and interaction of the transformations. In this paper we present a combined analytical performance and area modeling for complete FPGA designs in the presence of loop transformations. Our approach takes into account the impact of input/output memory bandwidth and memory interface resources, often the limiting factor in the effective implementation of these computations. Our preliminary results reveal that our modeling is very accurate allowing a compiler tool to quickly explore a very large design space resulting in the selection of a feasible high-performance design.

## 1 Introduction

The application of loop-level transformations, important to expose vast amounts of fine-grain parallelism and to promote data reuse, substantially increases the complexity of mapping computations to FPGA-based architectures. Figure 1 illustrates a typical behavior for a design mapped to an FPGA exploring loop unrolling. At first as the unrolling factor increases the execution time decreases and the amount of consumed space resources increases. However, with additional unrolling, there is the need to exploit more memory resources and memory parallelism. Before the FPGA capacity limitation is reached (in this case for an unrolling of 16) another limit is reached for an unrolling factor of 8. At this point, the design requires more memory channels than the implementation can provide. As such, these resources must be time-multiplexed leading to a non-trivial increase in the execution time (illustrated by the heavy-shaded bar).

Understanding, and mitigating the impact of hardware limitations is important in deriving the parameters of the loop transformations in the quest for the best possible design. In the example illustrated below if a compilation tool is not aware of the fact that memory channels resources are limited, it incorrectly selects the design with an unrolling factor of 8. This selection is only better than the design without any unrolling and far from the actual best design which corresponds to an unrolling amount of 4.



**Fig. 1.** Qualitative Execution and Space Plots in the Presence of Limited I/O Resources

As this example illustrates, ignoring the real memory interface resource limitation can lead extremely poor design choices. Given the extremely large number of possible loop transformations and their interaction, a solution to the problem of finding the best performing (and feasible in terms of space) design is to develop performance and area estimation for the designs resulting from the application of a sequence of loop transformations. To this extent we focus on the development of a set of analytical models combined with behavioral estimation techniques for the performance and estimation of complete FPGA designs. In this work we model the application of a set of important loop transformations, *unrolling*, *tiling*, and *interchange*. We explicitly take into account the impact of the transformations on the limited resources of the design’s memory interfaces.

This paper makes the following specific contributions:

- It presents an area and performance modeling approach that combines analytical and estimation techniques for complete FPGA designs.
- It describes the application of the proposed modeling to the mapping of computations to FPGAs in the presence of loop *unrolling*, *tiling*, *interchange* and *fission*.
- It validates the proposed modeling for a sample case study application on a real Xilinx Virtex<sup>TM</sup> FPGA device. This experience reveals our model to be very accurate even when dealing with the vagaries of synthesis tools.

Overall this paper argues that performance and area modeling for complete designs, either for FPGA or not, is an instrumental technique in handling the complexity of finding effective solutions in the current reconfigurable as well as future architectures.

This paper is organized as follows. In section 2 we describe the analysis and modeling approach in detail. In section 3 we present experimental results for a sample image processing kernel – a binary image correlation. In section 4 we describe related work and conclude in section 5.

## 2 Modeling

We now describe our analytical execution time and area modeling for complete designs as implemented in our target FPGA system.

## 2.1 Target Design Architecture and Execution Model

The designs considered in our modeling are composed of two primary components as illustrated in Figure 2(a). The first component is the *core datapath* that implements the core of the computation and is generated by the synthesis tools from a high-level description such as VHDL. This datapath is connected to an external memory via an *memory interface* component. This memory interface (see [1]) is responsible for generating the physical memory addresses of various data items as well as the electrical signaling with the external memory.

In many computations, such as image processing kernels, there is a substantial opportunity to reuse data in registers across iterations of a given loop. This is the case, for example, in window-based algorithm in which a computation is performed over shifted windows of the same image. the overlap between windows allows for a subset of the data to be reused in registers, therefore avoiding to load all of the data from memory. For computations with these features, our datapath implementations include an internal *tapped-delay* line.

Our model exploits the pipelined execution *mode* of memory accesses and the core datapath. To adequately support this execution mode for the class of regular image processing computations, our memory interface supports the concept of *streamed data channels* or simply *streams*. Associated with each of these *stream* the memory interface include resources to generate the corresponding sequence of memory addresses. Setting up and resetting these resources whenever the datapath requires data from channels not currently set-up (or set up at a different base address in memory) incurs an overhead. The important parameters for the performance modeling of the pipelined execution mode of the datapath are its *initiation interval* and *latency*. As to the memory interface, its important parameters are the *reloading overhead*, the *read* and *write* latencies and the *setup* for reading and writing in pipelined memory access mode.

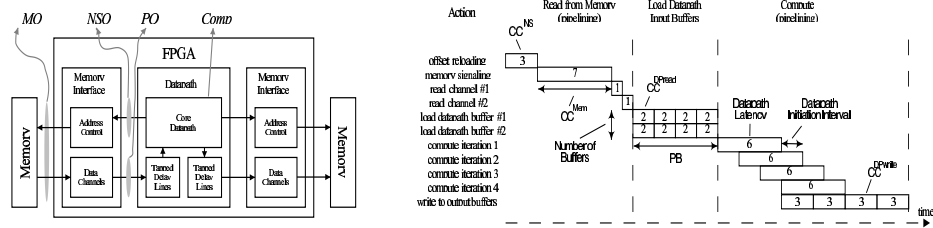


Fig. 2. (a) FPGA Design Architecture (b) Execution Mode

Figure 2(b) illustrates the basic parameters of the execution model, which will be the basic parameters for the performance modeling described in the next section. We show a 2-channel implementation with 2 buffers in the datapath, each with a depth of 4 elements and corresponding to 2 datapath input ports. The computation executes 4 loop iterations per block of data read from memory.

## 2.2 Performance Modeling

As with other compile-time modeling approaches we assume known compile-time loop bounds. In addition we assume that in the presence of control-flow

the expressions must capture the longer execution path and that the datapath implementation will preemptively fetch all of the data items required in each of the possible control paths. While in general these assumptions would lead to excessively inflated performance results, for the target set of digital image applications, and due to their amenability to be modeled into perfectly loop nests without severe control flow, this potential phenomenon is rare, if at all observed. Under these assumptions we split the overall execution into 5 components summarized below and whose analytical expressions are presented below.

$$\begin{aligned}
Comp &= LoopCnt * initInter \\
NSO &= \sum_{0 \leq i \leq m} LoopCnt_i^{NS} * CC^{NS} \\
PO &= \prod_{0 \leq i \leq m} LoopCnt_i^P * max(BD_j^B) * CC^{DPread} \\
MO &= \sum_{0 \leq i \leq m} LoopCnt_i^{Mem} * CC^{Mem} \\
Exec &= (Comp + NSO + MO + PO) * Clk
\end{aligned}$$

**Computation Time (Comp)** models the aggregate time the datapath spends actively computing results. We define it as the product of the overall number of iterations times the datapath initiation interval.

**Non-Streaming Access Overhead (NSO)** models the aggregate overhead in the reloading of the base and offset for non-streaming memory accesses and is defined by the  $NSO$  expression. In this expression  $LoopCnt_i^{NS}$  captures the number of iterations the datapath performs a non-streaming memory accesses; and  $CC^{NS}$  is the individual cost of such accesses. The value  $m$  is the total number of array variables in the particular implementation with non-streaming accesses. For streaming memory access this metric is defined as 0.

**Prologue Overhead (PO)** models the overhead of filling buffers associated with a data stream before the datapath is ready to start its pipelined execution. The component  $PO$  defined above captures this cost of buffer preloading where  $LoopCnt_i^P$  corresponds to the iteration count of the transformed loop nest that requires prologue loading.  $BD_i^B$  is the buffer depth of the  $B$  buffer (in the datapath) for variable  $i$ . The factor  $max(BD_i^B)$  defines the maximum length of time required to fill the longest buffer.

**Memory access overhead (MO)** models the latency on the memory interface for retrieving data from the memory. In the  $MO$  expression  $LC_i^{Mem}$  is the ratio of the number of memory accesses (or loop count accounting the memory access) to that of the granularity, for a variable  $i$ . Granularity is defined as the ratio of the data width of memory access to that of the data consumed in the datapath. Granularity is 1 if the data width of the memory access is the same as that of the data consumed in the datapath.  $CC^{Mem}$  is the number of cycles required for a memory data access.

**Clock rate (Clk)** We include this metric in the evaluation of the performance of a design as different designs will have can exhibit a wide disparity of clock rates due to radically different internal hardware implementations. Conducting relative comparison based on the clock cycles alone could lead to the wrong decision in selecting the best performing design.

**Execution Time (Exec)** simply defines the aggregate execution time and is simply the product of the number of execution cycles with the actual values of  $Clk$  as derived from the synthesis tools.

### 2.3 Area Modeling

This modeling is essential so that a compiler can use the area estimates derived from the synthesis tool and combine the predicted area for the memory interface when judging the area of a complete FPGA design.

To achieve this goal, we have derived a linear model using linear regression for a set of data points for various choices of number of memory interface channels. For 32-bit wide data channels the overall FPGA (Xilinx Virtex™ 1K device) area (in slices) for the memory interface can be approximated by the expression  $Area_{32} = 137 * NumberChannels + 1514$ . A similar empirical approach can be used for other channels widths and FPGA devices.

As to the modeling of the area for the datapath we use estimation results provided by synthesis tools such as the Mentor Graphics' Monet™ tool. The overall area estimation therefore combines the empirical model for the memory interface with the synthesis estimates.

### 2.4 Deriving the Model Parameters

In the current implementation we manually apply the various loop transformations, and derive the parameters of our modeling effort are derived using a crude emulation of the actual execution by actually running the computation in software and accumulating the number of occurrences of each metric. We are in the process of automating the application of the transformations and developing more sophisticated, and automated, approach to extract the values of the modeling parameters.

The modeling parameters that depend on the target FPGA device such as the maximum clock rate or the actual implementation of the memory interfaces (*e.g.*, the latency or the number of cycles for non-pipelined operations) are architecture dependent and known at compile time. The remaining model parameters are either derived from analysis of the source code, as is the case of the classification of which array data accesses are non-sequential, where the compiler can use data dependence analysis as described in [2].

## 3 Case Study: Binary Image Correlation

We now validate the proposed performance and area modeling for one image processing kernel — a binary image correlation (BIC) computation.

### 3.1 Original Computation and Role of Loop Transformations

Figure 3 depicts the pseudo-code for the example under study. This computation consists of a 4-loop nest with known loop bounds and implements binary image correlation using a `mask` variable (2D `mask` array) over an input image (2D `image` array). The computation scans the input image by sliding an  $e \times e$  window over

the input and accumulates the values of the corresponding image window for non-zero `mask` values in `th` (another 2D array variable).

The input image(`image[m+i][n+j]`) is accessed in a row-wise fashion. Unrolling the `j`-loop fully,  $e - 1$  of the `e` values in a single row, can be reused in consecutive iterations of the `n`-loop if the `e` values are stored in a *tapped-delay line*. And, as the data access is row-wise, unrolling of the `i`-loop enhances the performance by increasing the reuse to 2 dimensions. As a consequence, after the first iteration (where  $e \times e$  data are loaded), only `e` data values corresponding to `e` data streams need to be loaded in the sub-subsequent iterations of the `n`-loop.

```

for m = 0 to m < (t-e)
  for n = 0 to n < (t-e)
    for i = 0 to i < e
      for j = 0 to j < e
        if(mask[i][j] != 0)
          th[m][n] += image[m+i][n+j];

```

(a) Original code.

```

for m = 0 to m < t
  for q = 0 to q < t by e
    for p = 0 to p < s by f
      for n = q to e-1
        for i = p to p+f-1 // unrolled loop
          if(mask[i][0] != 0)
            th[m][n] += image[m+i][n];
          ...
          if(mask[i][s-1] != 0)
            th[m][n] += image[m+i][n+s-1];

```

(b) Using  $e \times f$  tiling.

```

for p = 0 to s/k
  for m = 0 to t
    for n = 0 to t
      for i = p*(s/k) to p*(s/k)+k // fully unrolled
        for j = 0 to s // fully unrolled
          if(mask[i][j] != 0)
            temp[p][m][n] += image[m+i][n+j];
    // loop-fission and loop interchange
    for m = 0 to t
      for n = 0 to t
        for p = 0 to s/k // fully unrolled
          th[m][n] += temp[p][m][n];

```

(c) Loop fission after interchange (loops `m` and `p`)

**Fig. 3.** BIC codes after applying loop transformations.

Full unrolling of the two inner most loops, as suggested above, may not always be possible as unrolling leads to the replication of the loop body operators generating very large designs. As such we can apply tiling of the innermost loops as illustrated in Figure 3(b). By tiling the `i` and `j`-loops by a factor of `e` and `f` respectively, we reduce the amount of hardware resources devoted to the datapath. The trade-off, however, is in terms of reduced reuse. The tapped-delay lines that, in case of unrolling, held the data of a row of `image`, now holds data of length `e`. But, of greater significance is the fact that the execution time increases as the memory interface resources, associated with address generation, requires reloading of address (and as a consequence increased data access) for the data streams associated with each tile. The performance of the overall design is thus substantially reduced.

In the presence of associative operations it is possible to use loop interchange with loop tiling to avoid the need of reloading the data in a tile. The implementation saves partial results of the computation in temporary datapath buffers thereby avoiding the need to reload data from external memory. This strategy comes at the cost of a, potentially large, multi-dimensional array for storing the partial results in the datapath. The size of this temporary array is in the order of the input image size, making it an infeasible solution for most cases.

To mitigate this issue of buffer space, we use yet another set of transformation: array privatization (to privatize temporal variable) and loop fission. The privatized temporal value is now stored in the external memory rather internal buffers (we pay the price for memory access). A second loop generates final results by adding up temporal values in the correct order.

The application of these 3 loop transformations to this case study illustrates the point of this paper. As these transformations affect the data reuse patterns and, in turn, the way the data needs to be streamed in and out of the corresponding datapath implementation, the overall performance in FPGA-based implementations is not only affected by the number of iterations executed but also, and infact more significantly, by the number of times the memory interface resources need to reset.

### 3.2 Experimental Results

We have manually derived behavioral specification for the BIC code applying the loop transformations described above for a variety of parameter choices. We then use Mentor Graphics’ Monet high-level synthesis tool to derive the structural VHDL specification and obtain the estimated area and initiation interval and latency for their pipelined implementation. To derive a complete design we merge the structural VHDL with the structural code of the memory channel interfaces. Given a complete VHDL design we used Synplicity Synplify Pro 6.2 and Xilinx ISE 4.1i tool sets for logic synthesis and Place-and-Route (P&R) [3] targeting a Xilinx Virtex<sup>TM</sup> XCV 1000 BG560 device.

**Performance Model Validation** We validate the performance modeling describe in section 2 for the VHDL code resulting from the application of the various loop transformations described above. In this modeling we use the numerical parameters as,  $CC^{DPRead} = 2$ ,  $CC^{Mem} = 7$  and  $CC^{NS} = 3$  and omit the symbolic loop expressions here for space considerations.

Figure 4 plots the measured vs. predicted performance for the various, loop unrolled (for the inner loop); tiled (the i and j loops and tiled-with-interchange. The vertical axis corresponds to the simulated clock cycle counts. We limit our experimental set to tiling versions of  $1 \times f$  because other transformations do not deliver additional performance.

These results indicate that our performance modeling tracks the overall performance for the various implementations very well. The gap between the predicted performance and simulation results is due to our channel controller behavior. The channel controller used in our memory interface uses a round-robin scheduling strategy which introduces a small number of additional cycles, as we verified through simulation.

**Area and Implementation results** We now validate the overall area estimation approach using the empirical model for the memory interface and the area estimation extracted from behavioral synthesis. In this validation we compare the results a compiler would obtain off-line (*i.e.*, without actually synthesizing any designs) with the real results synthesizing the actual complete designs.

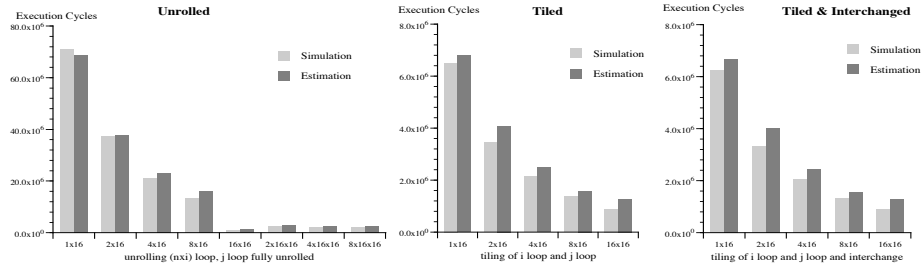


Fig. 4. Performance Estimation .vs. Simulation Results for BIC.

Table 1 presents the synthesis results and complete design results for the various tiled and tiled&interchange designs. For the tiled&interchange design we present here only the design corresponding to the first loop that carries out the bulk of the computation. Area estimates produced by Monet behavioral synthesis tool are not compatible, in terms of units, with those of the P&R tools. However, these estimates to a large extent do capture the 'trend' of the results(area) produced by the synthesis and P&R tools as is evident from table 1. Table 1 compares the estimation numbers with those of the synthesis and P&R numbers for the tiled implementations. Estimation numbers for datapath and interfaces are provided in columns 2 and 4 while columns 3 and 6 are the synthesis and P&R results for datapath only and full design (datapath+interface) respectively. Column 5 is the estimation number for the full design (col. 2 + col. 4).

Tile	Datapath Only		Interface Estimates (Monet)	Full Design		
	Estimates (Monet)	P&R (Slices)		Estimates (Monet)	P&R	
					Slices(%)	Clk (MHz)
(1x1)	14621	4214	1788	16409	4558 (37)	35.3
(1x2)	15804	3808	2602	18406	6661 (54)	29.9
(1x4)	18191	5163	2610	20801	7070 (57)	21.7
(1x8)	22868	6369	3706	26574	9734 (79)	28.3
(1x16)	19704	5053	5898	25602	10417 (84)	24.2

Tile	Datapath Only		Interface Estimates (Monet)	Full Design		
	Estimates (Monet)	P&R (Slices)		Estimates (Monet)	P&R	
					Slices(%)	Clk (MHz)
(1x1)	1411.1	408	1788	3199.1	2326 (19)	43.4
(1x2)	2599.1	725	2602	5201.1	2984 (24)	36.1
(1x4)	4963.1	1333	2610	7573.1	4038 (33)	26.6
(1x8)	9694.9	2620	3706	13400.9	6119 (50)	16.7
(1x16)	19704.0	5053	5898	25602.0	10417 (84)	24.2

Table 1. Synthesis results for tiling (top) and tiling with interchanging and fission (bottom).

As can be seen the combined estimation results for the complete designs track very well the real area results even for the very large designs that occupy more than 50% of the FPGA area. This is important as the application of loop transformations that expose vast amount of instruction level parallelism require the replication of functional operators and invariably lead to large designs. For some of the implementations the internal tapped-delay lines used to reduce the number of memory accesses is an important factor on the area as well.

We investigated the fact that the  $(1 \times 8)$  tiled-version maps to more slices than the  $(1 \times 16)$  version (see table 1 (top) col. 3). We concluded that although the computational for the  $(1 \times 8)$  version are approximately half of those for the  $(1 \times 16)$  version, the area consumed by the registers used to store intermediate results across the m-loop were responsible for this area anomaly.

### 3.3 Discussion

Experimental results reveal that our performance and area modeling correlates well with our simulation/synthesis results for various implementations of BIC. Our performance model is capable of identifying effects of various loop transformations, which ultimately lead to the best combination of loop transformations. In the BIC case study our model accurately captures the effect of tiling with different tiling factors, and we can find the best tiling shape/s. The results also reveal that our modeling successfully captures the performance behavior of the more sophisticated combination of loop interchange, fission and privatization.

Overall, we believe the modeling approach described in this paper to be applicable to a wider range of reconfigurable architectures as our model does not exploit any feature of the underlying architecture. In fact, our case study reveals that the major source of discrepancies was either due to characteristics of FPGA synthesis (*e.g.* the large impact of temporary buffers) or due to FPGA implementation execution features (*e.g.*, scheduling of memory accesses).

## 4 Related Work

Other researchers have also recognized the need for fast area and performance estimation to guide the application of compiler high-level loop transformations for deriving alternative designs. Derrien and S. Rajoupydyhe[4] describe a processor array partitioning that takes into account the memory hierarchy and I/O bandwidth and apply tiling to maximize the performance of the mapping of a loop nest onto FPGA-based architectures. In this context they use an analytical performance model to determine the best tile size. So *et. al.* [5] have expanded the loop transformations to include unrolling and use behavioral synthesis estimates directly from commercially available synthesis tools in an integrated compilation and synthesis. The PICO project [6] takes functions defined as C loop nests to a synchronous array of customizable VLIW processors. PICO uses estimates from its scheduling of the iterations of the nest to determine which loop transformation(s) lead to shorter scheduling time and therefore minimal completion time. The MILAN project [7] provides design space exploration and simulation environments for System-on-Chip(SoC) architecture. MILAN evaluates several possible partitions of the computation among the various system components (processor, memory, special purposed accelerators) using simulation techniques to derive estimates used in the evaluation of a given application mapping.

The work presented in this paper differs from these approaches in several respects. While other projects have focused on more general computation we have focused on the domain of image processing algorithms specified as tight loop nests. Second, rather than using profiling or simulation based estimates we

have analytically modeled the performance and relied on the accuracy of behavioral synthesis estimation commercial tools for area modeling. In terms of loop transformations we have focused not only on loop unrolling and tiling but also on loop interchange in the presence of associative operators. Loop interchanging introduces the complication of having to save intermediate results for subsequent computations. Designs with large set of registers for temporary values lead to an explosion of area and substantial degradation of clock estimates.

## 5 Conclusion

In this paper we presented a performance and area modeling approach using analytical and empirical techniques for complete FPGA designs. Our modeling is geared towards computations expressed as loop nests in high-level programming languages such as C. Using the proposed modeling, compiler tools can evaluate the impact of multiple loop transformations on FPGA resources as well as that of the memory interface resources, often the limiting factor in the effective implementation of these computations. The preliminary results reveal that our approach delivers area and performance estimations that correlate very well with the corresponding metrics from the actual implementation. This experience suggests the proposed modeling approach to be an effective technique that allow compilers to quickly explore a wider range of loop transformations for selecting feasible and high-performance FPGA designs.

## References

1. Park, J., Diniz, P.: Synthesis of Memory Access Controller for Streamed Data Applications for FPGA-based Computing Engines. In: Proc. of the 14th Intl. Symp. on System Synthesis (ISSS'2001), IEEE Computer Society Press (2001)
2. Diniz, P., Park, J.: Automatic synthesis of data storage and control structures for FPGA-based computing machines. In: In Proc. IEEE Symp. on FPGAs for Custom Computing Machines (FCCM'00), IEEE Computer Society Press (2000)
3. : (Virtex 2.5v FPGA product specification. ds003(v2.4)) Xilinx, Inc., 2000.
4. Derrien, S., Rajoupydyhe, S.: Loop tiling for reconfigurable accelerators. In: Proc. of the Eleventh Int. Symp. on Field-Programmable Logic (FPL2001). (2001)
5. So, B., Hall, M., Diniz, P.: A compiler approach to fast hardware design space exploration for fpga systems. In: Proc. of the 2001 ACM Conference on Programming Language Design and Implementation (PLDI'00), ACM Press (2001)
6. Kathail, V., Aditya, S., Schreiber, R., Rau, B., Cronquist, D., Sivaraman, M.: PICO: Automatically designing custom computers. In: IEEE Computer. (2002)
7. Bakshi, A., Prasanna, V., Ledeczi, A.: Milan: A model based integrated simulation framework for design of embedded systems. In: Proc. of the ACM Workshop on Languages, Compilers, and Tools for Embedded Systems (LCTES 2001). (2001)