From PDF to GDS: Designing the RoboBee SoC

Brandon Reagen, Xaun Zhang, Gu-Yeon Wei, David Brooks
Harvard University, Cambridge, MA, USA

I. Introduction

Developing the Robobee was a multi-discipline, 5-year project funded by a National Science Foundation Expeditions in Computing award with the goal of achieving autonomous flight with a bee-sized micro-robot [2]. The intent of the research was to help re-stabilize the declining bee population which researchers have shown could have devastating effects on the earths ecosystem.

Bees are remarkably efficient; their skeleton weighs almost nothing: requiring minimal lift to takeoff and sustain flight; their brains are small: pre-programmed with a minimal set of instincts necessary for the colonies survival. Their capabilities under such stringent weight and compute limitations makes them a prime target for pushing what modern robotics and computer systems can do.

The weight and power limits require a custom System-on-Chip (SoC) be built. Conventional off-chip voltage regulators are heavy and bulky, and thus cannot fit under the weight and form factor of the robotic bee. Commercial Off-The-Shelf parts (COTS) micro-controllers consume too much power to perform the required computation for autonomous flight. The solution is to pack as much IP onto a single die. SoCs have been the trend of all semi-conductor companies over the past decade from mobile and embedded to server grade solutions.

In this paper we recount our experiences designing such a chip. We highlight the major challenges faced when designing for such a unique form factor, how designs and specifications were set by each collaborating lab, the difficulties of integrating a plethora of IP consisting of in-house digital and analog blocks, and the design flows we used. We also discuss how invaluable HLS was in reducing the engineering burden, focusing design efforts at higher levels of abstraction, and an overall successful tape-out.

II. Workload Characterization: How Bees Fly

Bees navigate using only optical information. Each wing flap generates roll, pitch, and yaw movements to sustain flight and move according to the objective. The magnitude of the force applied to each degree of freedom is computed by analyzing how the bees surroundings change over discrete, adjacent periods of time. This method of motion estimation is known as optical flow. Flying using only optical flow requires a lot of compute power. To sustain flight, each force calculation must resolve at a frequency the bee's wings flap: 100Hz.

III. From System-in-a-Lab to System-on-Chip

Before the design of the brain SoC the Harvard Robotics lab had prototyped a bee-scaled robot and demonstrated flight. What the robot lacked in size the control system around it made up for. The first flight demonstration was composed of a bee with 4 wire tethers hanging off of it and a VICON system consisting of 4 high-definition cameras each aimed at the bee, which formed an arena of sorts. The cameras tracked the nodes, feeding live video streams into a powerful desktop computer running Simulink code to analyze the differences between images. The computation outputs power signals conveyed through wires tethered to the bee where voltages contort the piezo-electric actuators that flap the bee's wings.

The challenge is to move this entire experimental setup– from the HD-cameras to the desktop computer and power outlet–onto the robotic bee. Two back-to-back vision sensors are required, each facing outwards to gather enough data about the bees movement to compute the three forces correctly. Weight limitations only permit noisy, 8-bit, 128-by-128 pixel cameras. To meet the performance requirements and fit within the power budget we designed custom IP for any piece of the workload with enough definition that a design specification could be drawn up.

IV. The Frankenstein Design Process

One of the most difficult challenges faced when defining the specification and implementing the brain SoC was figuring out what was needed. Over the course of 3 years bi-weekly meetings were held to communicate the necessary information on achieving flight with wing-flapping micro-robots and the power/performance trade-offs of chip designs.

Traditional design flows such as the waterfall model, agile development, and general top-down verses bottom-up methods do not work in this context. The design specification consisted of PDFs of Simulink blocks, Latex documents describing basic math and filter computations, and photographs of chalk board sketches. Despite the specificity of the application, it is surprisingly difficult to settle on specifications. The solution was to not settle on perfectly precise, detailed specifications but rather establish the necessary blocks and then implement them in a way such that they are easily modified.

With all the parts known, project team members went out and started building. Analog designers built the ADCs and Integrated Voltage Regulator, digital designers built accelerators, and architects focused on general purpose cores, AMBA buses, and the memory system. For this reason, architecting an SoC is more ad-hoc than the more traditional chip design process. Where simulators and base designs are typically iterated on each generation, the uniqueness and novelty of the SoC requires the system be stitched together from a barrage of sources.

V. High-Performing, Low-Power, Flexible, Configurable, Application Specific Hardware

To meet the requirements, many iterations had to be made to the hardware accelerators. The major accelerator blocks are for Optical Flow, Convolution, and Linear Algebra. Here we give examples of how the accelerators were refined to meet requirements.

High-Performance: A common issue with hardware acceleration is keeping them fed with data. The image processing needs of the optical flow workload requires a lot of bandwidth and data movement. To limit this, we carefully partition the image SRAMs to overlap convolution and optical flow computations. As data comes in from the camera, it is put into a partition of one of the SRAMs. The convolution completes and writes results to a different partition. There is a small controller which MUXs addresses from the optical flow accelerator to always get the data from where it was written, maximizing the performance of the system by limiting the amount of data movement required.

Low-Power: Floating point computation is abused, wasteful, and ill-understood by non-computer architects. All the workload specifications provided called for double-precision floating point computation. This is impractical as a single multiplier can consume more energy than the entire general purpose core. We have standards for a reason; when we converted all the computation to fixed-point no support was offered with respect to the precision of the computations. For this reason, all the accelerators emulate fixed-point by tracking it in software. The linear algebra computations simply read in 32-bit integers and complete the computation. When the core reads back the result it knows how to shift the product appropriately to track the decimal. While non-ideal, integer operation can save orders of magnitude of energy verses double precision floating point.

Flexibility: Another design challenge with the accelerators was not committing hardware to something that may turn out to be useless. The Convolution accelerator can be programmed via a memory mapped register to perform different computations. The filter constants can be re-programmed to perform different operations: parts of the image can be discarded: there is also support to conduct vertical or horizontal convolutions on the image data. In this way hardware accelerators were able to be built without having the specifications as detailed as in an ideal situation.

Configurable: Optical flow is a generic technique with many different algorithms deriving from it. Since the bee had never actually flown using 2 outward facing cameras it was unclear which solution is best. To overcome this, there are multiple versions of optical flow supported by the accelerator. The accelerator is configurable to return a 2D vector, a vector field, or a set of vectors averaged over the images. It also supports two algorithms for computing Optical flow. In addition, it is flexible to only operate on part of the image and has both 1D and 2D operation modes.

What we found was that our accelerators still provide the magnitudes of performance and energy savings over general purpose cores, they just need to be designed and reasoned about differently. In [1] the authors demonstrated fixed-function accelerators provide two orders of magnitude in performance and energy compared to the cortex M0. When designing the accelerators for the RoboBee we were able to achieve comparable improvements, up to 723× dynamic energy reduction [3], by focusing on how flexibility and configurability were incorporated. Looking forward, this is is how accelerators should be designed as assuming perfect workload specification is unpractical, though ideal to maximize efficiency.

VI. Instrumental CAD Tools

All of the hardware accelerators were developed using Xilinxs Vivado High-Level Synthesis tool. The countless design iterations, specifications that changed constantly, and amount of testing required to insure the chips functionality would not have been possible with the limited man-power and design time. Developing accelerators in C makes adding and testing new features a function of minutes to hours as opposed to days to weeks. Moreover, HLS automatically generates test benches and wave forms, the C implementations for accelerators are compatible with GCC, simplifying the task of verify design correctness at a high level.

While designing accelerators with HLS we found the *HLS Tax*—the performance difference between RTL generated by HLS and hand-coded RTL)— to be negligible compared to the practical engineering benefits. Also, appropriately tuning designs with directives helps eke out high-performing, efficient designs from a high-level representation. Among the most useful directives were the capabilities to interface with the system and bus protocols. Vivado HLS supports the AMBA protocols and has simple ways of setting up memory mapped registers for external FSM control by general purpose cores.

Our experience with the HLS approach to accelerator design was exceptionally positive. As system design becomes increasingly more complex more innovations like HLS are necessary to allow architects to focus their effort not on designing individual pieces but how they are put together. Specifically, we see a blatant gap in system-level optimization tools. The Frankenstein design process is not a good one, but it works. Being able to balance all the parts that go into an SoC would likely expose inefficiencies in our design we are ignorant of. The impact emergent properties from integrating SoC blocks have on system performance is too complex to exhaustively evaluate. Existing CAD tools, like HLS, are a first step in improving and formalizing the SoC design process. However, architects are still unequipped to handle the degree of optimization and design space exploration necessary to holistically optimize SoCs with levels of heterogeneity similar to the Robobee Brain SoC.

VII. Conclusion

The chip consists of 12 distinct SRAM blocks, used by both the accelerators and the general purpose cores. There are two general purpose cores: a simple ARM M0 micro-controller and the Intel SiskiyouPeak. There are hardware blocks for the optical flow and convolution functionality which consume a considerable amount of the total die area. In addition, 3 of the signal processing/linear algebra blocks are included to assist the general purpose cores with handling common operations. More details are measurements are included in the presentation and can be found in the conference paper published in VLSI 2015 [3].

References

[1] B. Reagen, Y. Shao, G.-Y. Wei, and D. Brooks. Quantifying acceleration: Power/performance trade-offs of application kernels in hardware. In *ISLPED*, 2013.

[2] R. Wood, R. Nagpal, and G.-Y. Wei. Controlled flight of a biologically inspired, insect-scale robot. In *Science*, 2013.

[3] X. Zhang, M. Lok, T. Tong, S. Chaput, S. Lee, B. Reagen, H. Lee, D. Brooks, and G.-Y. Wei. A Multi-Chip System Optimized for Insect-Scale Flapping-Wing Robots. In *VLSI*, 2015.