Interconnection Synthesis of MPSoC Architecture for Gamma Cameras

Tianmiao Wang[1], Kai Sun[1], Hongxing Wei[1], Meng Wang[2], Zili Shao[2*], and Hui Liu[3]

[1] Robot Research Institute, Beihang University, Beijing 100083, China, {wtm, mounthorse, whx}@me.buaa.edu.cn
[2] Department of Computing, The Hong Kong Polytechnic University, Hong Kong, {csmewang, cszlshao}@comp.polyu.edu.hk
[3] Software Engineering Institute, Xidian University, Xi'an, China, liuhui@xidian.edu.cn

Abstract. MPSoC (Multi-Processor System-on-Chip) architecture is becoming increasingly used because it can provide designers much more opportunities to meet specific performance and power goals. In this paper, we use MPSoC architecture to solve real-time signal processing problem in gamma camera. We propose an interconnection synthesis algorithm to reduce the area cost of the Network-on-Chip for an MPSoC architecture we propose in [14]. We implement our interconnection synthesis algorithm on FPGA, and synthesize Network-on-Chip using Synopsys Design Compiler with a UMC 0.18um standard cell library. The results show that our technique can effectively accelerate the processing and satisfy the requirements of real-time signal processing for 256 × 256 image construction.

1 Introduction

MPSoC (Multi-Processor System-on-Chip) architecture is becoming increasingly used. It can provide high throughput while keeping power and complexity under control and give designers more opportunities to meet specific performance and power goals. With a heterogeneous MPSoC architecture, different types of cores can be built on the same die, and each type of core can effectively and efficiently process specific tasks. These flexible combinations make MPSoC systems very powerful and be able to implement complex functions with high performance and low power by integrating multiple heterogeneous processors, hierarchy memory systems, custom logic, and on-chip interconnection. With such advantages, MPSoC architecture has been widely applied in various fields such as network[10], multimedia[11] and HDTV[6]. We focus on the MPSoC design and synthesis for real-time signal processing for biomedical applications in this paper.

In biomedical applications, most medical electronic devices heavily rely on image processing techniques to process large scale data. Therefore, more powerful

* The corresponding author.

techniques are needed in order to improve processing speed and precision. MPSoC architecture brings these systems more opportunities to achieve their goals. For example, in [8], Khatib et al. propose an application-specific MPSoC architecture for real-time ECG (Electrocardiogram) analysis. The advanced industrial components for MPSoC design (multi-issue VLIW DSPs, system interconnect from STMicroelectronics, and commercial off-the-shelf biomedical sensors) are employed in their architecture so real-time ECG analysis can be achieved with high sampling frequencies.

In this paper, we solve the real-time digital signal processing for gamma cameras, most commonly used medical imaging devices in nuclear medicine. In a gamma camera, images are generated by detecting gamma radiation. One of the key components in a gamma camera is PMT (PhotoMultiplier Tube). Basically, PMT is used to detect fluorescent flashes generated by a crystal and produce current. Then the corresponding voltage signals are converted to digital signals by ADC (Analog to Digital Converter) behind a PMT, and finally the digital signals are processed to generate images by a digital signal processing system. To generate images, multiple PMTs are placed in hexagon configurations behind the absorbing crystal. In a typical scheme, a PMT array consisting of more than 30 PMTs is used in a gamma camera. Using a serial 2D images obtained by gamma cameras from the different angles, 3D information can be acquired by SPECT (Single Photon Emission Computed Tomography).

To process the data generated by the PMT array, DSP (Digital Signal Processing) boards based on PC platforms are widely used in current gamma cameras. With such platforms, typically, it takes about 15 - 30 seconds to generate one 64×64 image and 15 - 20 minutes to finish a complete scan in SPEC. The platforms can not efficiently produce higher-quality images such as 256×256. And their slow processing speed and big size limit the effective use of gamma cameras, in particular, for portable gamma cameras [12, 13] that work with new room-temperature nuclear radiation detector. To improve image construction speed, a technique called PMT-PSPMT (Position Sensitive PhotoMultiplier Tube) [7] is proposed. PMT-PSPMT is very effective in optimizing image construction times. But it reduces the image quality and cannot construct 256×256 image dynamically.

To solve these problems, we propose an MPSoC architecture for PMT data processing in a gamma camera in [14]. Our MPSoC architecture consists of the following four parts: one general-purpose embedded processor, a high speed data interface (HSDI), application-specific DSP cores and a Network-on-Chip with an interconnection bus. In this paper, we develop an interconnection synthesis algorithm to reduce the area cost of the Network-on-Chip for the MPSoC architecture in [14]. We implement our interconnection synthesis algorithm with FPGA, and synthesize DSP cores and Network-on-Chip using Synopsys Design Compiler with a UMC 0.18um standard cell library. The results show that our technique can effectively accelerate the processing and implement communication with small area cost. It can satisfy the requirements of real-time signal processing for 256×256 image construction.

The rest of this paper is organized as follows: in Section 2, we introduce necessary backgrounds related to gamma camera technique. In Section 3, we present the MPSoC system architecture. Section 4 provides the experimental results and discussions. In Section 5, we conclude the paper.

2 Background

Gamma camera is a commonly used medical imaging device in nuclear medicine. In a gamma camera, images are generated by detecting gamma radiation. Basically, the counts of gamma photons that are absorbed by a crystal are accumulated, and the crystal produces a faint flash of light at the same time. The PMT array behind the crystal detects the fluorescent flashes and generates current. The current signal generated by the PMT is captured by the ADC, and two corresponding voltage signals are converted into digital signals. Digital signals are used to calculate the coordinate and energy of the gamma photons. With these coordinate and energy data, the final image can be produced.

During the whole medical imaging procedure, three algorithms, the integral, coordinate and amendment algorithms, are applied to the collected data.

The integral algorithm is to calculate the energy of the voltage signal. In this algorithm, the serial data of each PMT is accumulated based on system status conditions. The coordinate algorithm includes the calculation for two parts, position and energy. With the position and energy data, the gamma photon pulse can be determined. The amendment algorithm is used to amend energy and position data with three table-lookup operations. This algorithm consists of two parts, energy and linearity emendation.

To reduce the image construction time, we can increase the pulse frequency of gamma photons. With the limitation of the device, the maximum pulse frequency currently we can achieve is 500KHz-1 MHz. Correspondingly, we have to improve the speed of digital signal processing in order to generate image with such high pulse frequency. In this paper, our goal is to design an MPSoC architecture that can generate one 256×256 image in less than one second for gamma cameras with 1 MHz pulse frequency.

3 MPSoC System Design

In this section, we first introduce the MPSoC architecture in Section 3.1. Then we propose our interconnection synthesis in Section 3.2, respectively.

3.1 Architecture Overview

Our MPSoC architecture is a typical heterogeneous multi-core architecture targeting on the application of gamma camera. It is specially designed for processing PMT data in parallel with multi-processors. In order to achieve these goals, an MPSoC architecture, as shown in Figure 1 is proposed to speed up the image generation and improve image quality.

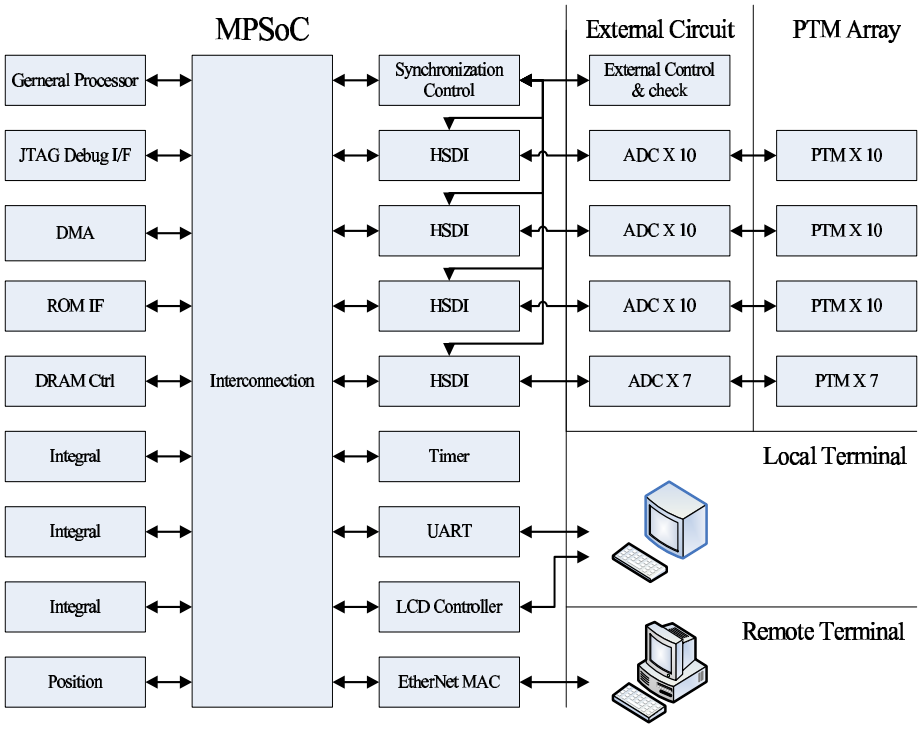
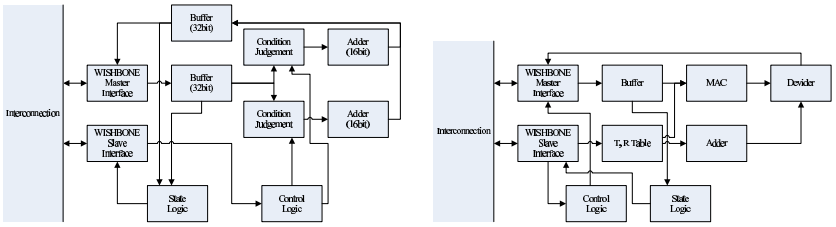


Fig. 1. The MPSoC Architecture

As shown in Figure 1, our MPSoC architecture consists of four parts: general processor, HSDI (High Speed Data Interface), DSP, and interconnection synthesis. In this architecture, the processor speed and the 32-bit on-chip interconnection are 200MHz, which are compatible with the 0.18um ASIC technology and the 32-bit bus interface IP cores. Next, we present the design issues for each key part of MPSoC architecture.

In the general processor part, there are one general purpose processor and some necessary IP cores, such as timer, UART, and SPI etc. Among these IP cores, the most important components are the on-chip RAM, SRAM/Flash controller, SDRAM controller and Ethernet MAC controller. The amendment algorithm and other general purpose computing are implemented in the general processor.

The customized DSPs used in our MPSoC architecture are designed for implementing the integral algorithm and the coordinate algorithm. We design two types of DSP, integral and coordinate, to implement the integral and coordinate algorithm, respectively. The corresponding block diagrams of the integral DSP and coordinate DSP are shown in Figure 2(a) and Figure 2(b), respectively.



(a) (b)

Fig. 2. The block diagrams of two DSP Cores a) The Integral arch structure (b) The Coordinate arch structure.

The integral DSP has two bus interfaces, *Master* and *Slave*. The *Master* interface implements the data load/store, and the *Slave* interface implements the control and status logic accessing from other devices. The main components of the coordinate DSP are *MAC* (Multiply Accumulate) and *Divider*. The coordinate DSP has two bus interfaces, *Master* and *Slave*, which are as same as those of the integral DSP.

3.2 Interconnection Synthesis

In this section, we first compare and analyze several bus structures, and then propose an algorithm to get a better interconnection synthesis for MPSoC architecture.

In MPSoC architecture, since the customized DSP and other components have enough buffer or cache to debase the infection of the bus latency, we can ignore the effect of the buffer to the system. Furthermore, since the communication throughput of the *Slave* interface is very low for the integral DSP, coordinate DSP and DMA controller, in this section, we only focus on the *Master* interface when considering the design of interconnection synthesis.

In most on-chip bus standards, such as AMBA[2], CoreConnect[1], STBus[5] and WISHBONE[4], the share structure is used in the embedded processor as shown in Figure 3(b). In this structure, the total bandwidth of the interconnection is limited to the bandwidth of each node since all buses are connected to one node and only one master can access the interconnection simultaneously.

In order to fulfill the bandwidth requirement, we employ the crossbar structure in which different masters can access the slaves at the same time as shown in Figure 3(c). This structure improves the capability of the interconnection, and it is suitable for our architecture to process the integral algorithm in parallel. However, the area cost is very high in this structure. We have implemented these two structures in Wishbone protocol based on the open source IP core[3], and the results show that the crossbar structure uses more than 8 times area compared with that using the share structure. To reduce the area, the reduced

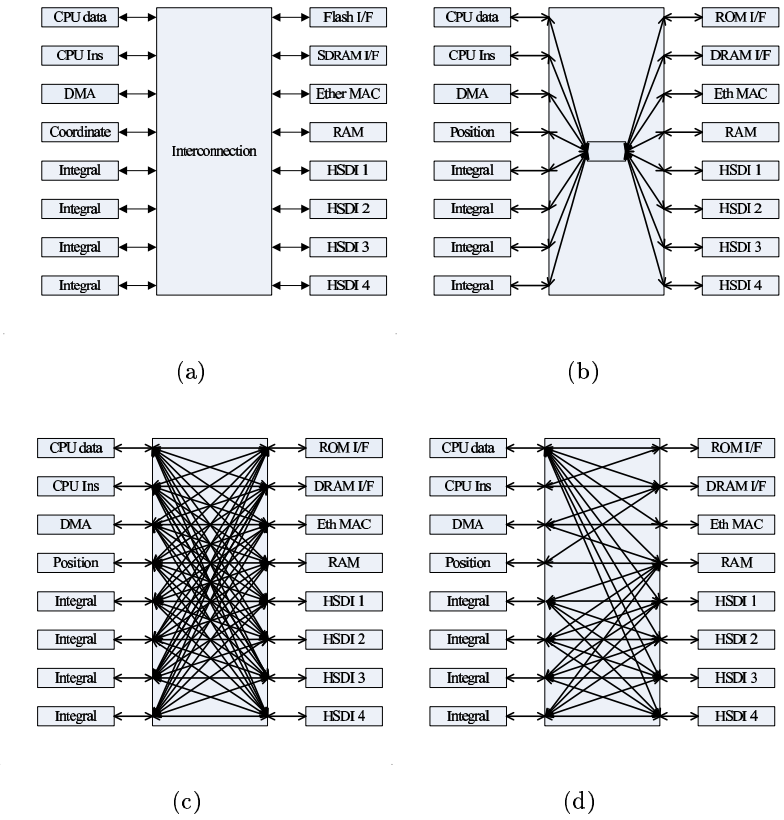


Fig. 3. Interconnection Structure (a) The main components (b) The share structure (c) The crossbar structure (d) The reduced crossbar structure.

crossbar structure which reduces the unnecessary connections between masters and slaves (in Figure 3(d)) is employed in our implementation. The results show that the reduced structure uses more than about four times more area compared with that using the share structure as shown in Table 5.

In order to further reduce the area cost, we design a novel algorithm which can reduce a generic-reduced matrix, thus to achieve our goals to reduce the area cost. The basic idea is to combine the *Slaves* and let the combined *Slaves* use only one *Slave* interface with the conditions that the total bandwidth of the combined *Slaves* can not exceed the bandwidth of every single bus, and the maximum number of buses can be reduced after the combination. We do not combine the *Masters* since such combination may cause more complicated problems [9]. We ignore the communication conflicts as the bandwidth of every single bus is low and the DSP and HSDI have enough buffers. Our MBRA algorithm (The Maximum-Bus-Reduction Algorithm) is shown in Figure 3.1.

Algorithm 3.1 The Maximum-Bus-Reduction Algorithm (MBRA)

Require: $M \leftarrow$ The number of masters; $S \leftarrow$ The number of slaves; $MAX \leftarrow$ The upper-bound of the bandwidth of a single bus; $MS[M][S]$: The communication matrix where $MS[i][j]$ denotes the communication between master i and slave j ($MS[i][j] == 0$ denotes no bus);

Ensure: The communication matrix with the minimum number of slaves after the slave combination.

```
1: //Assign a flag/number to each slave to denote if it has been combined/to which .
2: For i = 0 to S-1, Flag_Slave[i]=NO; To_Slave[i]=-1;
3: //Find the pair of slaves with the maximum cost (the reduced bus number) and combine the pair.
4: while {1} do
5:   //Find the pair of slaves with the maximum cost.
6:   max_cost = -1;
7:   for i=0 → S-2 do
8:     for j=i+1 → S-1 do
9:       combined_bus=total_bus=0; combined_flag=YES
10:      if Flag_Slave[i] == NO and Flag_Slave[j] == NO then
11:        for k=0 → M-1 do
12:          If MS[k][i] > 0, total_bus ++; If MS[k][j] > 0, total_bus ++;
13:          If MS[k][i] > 0 or MS[k][j] > 0, combined_bus ++;
14:          If MS[k][i] + MS[k][j] > MAX, combined_flag=NO;
15:        end for
16:        if combined_flag==YES then
17:          cost ← total_bus - combined_bus;
18:          if cost > max_cost then
19:            max_cost ← cost; combined_slave[0]=i; combined_slave[1]=j;
20:          end if
21:        end if
22:      end if
23:    end for
24:  end for
25:  //Combine the slaves with the maximum cost (always combine Slave j to salve i).
26:  if min_cost != -1 then
27:    i = combined_slave[0]; j=combined_slave[1];
28:    For k=0 to M-1, MS[k][i]+ = MS[k][j];
29:    Flag_Slave[j]=YES; To_Slave[j]=i;
30:  else
31:    Break;
32:  end if
33: end while
```

In the inputs of the algorithm, M and S are the numbers of the master and slaves in the network, MAX is the maximum bandwidth of a single bus, and MS is the communication matrix where $MS[i][j]$ denotes the communication between master i and slave j. We place the communication bandwidth between

the *Masters* and *Slaves* into $MS[i][j]$, in which if $MS[i][j] = 0$, it denotes that there is no bus between master i and slave j. The output of the algorithm is the optimized bus architecture by combining slaves as much as possible.

In the algorithm, for each pair of slaves, we first calculate the total numbers of buses from all masters to this slave pair before and after combining slaves. We then check if the combination is possible by comparing the combined combination with the upper bound of bandwidth of a single bus. Next, if the combination is possible, we calculate the cost that is defined as the reduced number of buses after the combination. The cost is compared with the current recorded maximum cost. If it is larger than the current maximum cost, we record the slave pairs $< i, j >$ into an array. After all possible slave pairs have been checked, we combine the slave pair with the maximum cost, which means that we can reduce the maximum number of buses by combining this slave pair. Then we record which slave has been combined and combined into which one. We set a flag for the slave that has been combined into others so it will not be considered in the further combination. The above procedure will be repeated until we could not find any possible combination. The MBRA algorithm is a polynomial-time algorithm. It takes at most $O(|S|^3|M|)$ to finish where S is the number of slaves and M is the number of masters.

4 Experimental Results and Discussions

To compare our MPSoC architecture with the general architecture, we have implemented our interconnection with WISHBONE protocol and our bus interconnection synthesis algorithm. We compare our technique with the crossbar and the reduce crossbar structure in terms of the area cost.

The communication array without optimization between the masters and slaves in the interconnection is shown in Table 1. In this array, for example, the number 60 in column 3 and row 4 denotes that the communication request between master 4 and slave 3 is 60MBps. After applying our MBRA algorithm in 3.1, the original 8 slaves are combined into 4 slave groups, and the area is reduced accordingly. The reduced array is shown in Table 2. In this reduced array, the columns have been reduced from 8 to 4, and the slaves 1, 2, 5, and slaves 3, 8, 4 have been partitioned into 2 groups, which means that slaves of the two groups can be put into one single bus. With the reduced array, the final structure is shown in Figure 4.

The three interconnection structures are coded in Verilog HDL, and are synthesized to gate-level circuits using Synopsys Design Compiler and a UMC 0.18um standard cell library. The area cost comparison of the cross, the reduced cross and the final structure is show in Figure 5. The results show that the algorithm 3.1 reduces 12% of the area.

	s1	s2	s3	s4	s5	s6	s7	s8
m1	50	25	100	1	2	2	2	2
m2	10	50	0	0	0	0	0	0
m3	0	50	55	50	0	0	0	0
m4	0	4	60	0	0	0	0	0
m5	0	0	19	0	150	150	150	105
m6	0	0	19	0	150	150	150	105
m7	0	0	19	0	150	150	150	105
m8	0	0	19	0	150	150	150	105

Table 1. The unreduced array.

	s1,2,5	s3,8,4	s6	s7
m1	79	105	4	4
m2	60	0	0	0
m3	50	105	0	0
m4	4	6	0	0
m5	150	124	150	150
m6	150	124	150	150
m7	150	124	150	150
m8	150	124	150	150

Table 2. The reduced array.

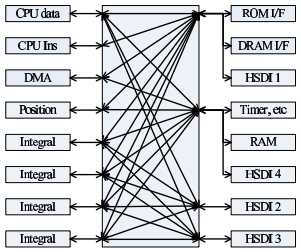


Fig. 4. The final crossbar structure.

Structure	**Area(K um^2)**
Crossbar	1608
Shared crossbar	786
Our MBRA Algorithm	694

Fig. 5. Area Comparison

5 Conclusion

In this paper, we have proposed an interconnection synthesis algorithm for an MPSoC architecture for implementing real-time signal processing in gamma camera in [14]. We synthesized DSP cores and Network-on-Chip using Synopsys Design Compiler with a UMC 0.18um standard cell library. The results show that our technique can effectively accelerate the processing and satisfy the requirements of real-time signal processing for 256×256 image construction.

Acknowledgment

The work described in this paper was partially supported by the grants from the Research Grants Council of the Hong Kong Special Administrative Region, China (PolyU A-PH13, PolyU A-PA5X, PolyU A-PH41, and PolyU B-Q06B), the National Nature Science Foundation of China (60525314), the 973 Program of China (2002CB312204-04) and the 863 Program of China (2006AA04Z206).

References

1. Ibm on-chip coreconnect bus architecture. *www.chips.ibm.com.*
2. Arm amba specification (rev2.0). *www.arm.com*, 2001.

3. Wishbone interconnect matrix ip core. rev. 1.1. *www.opencores.org*, 2002.
4. Wishbone system-on-chip (soc) interconnection architecture for portable ip cores revision: B.3. *www.opencores.org*, 2002.
5. Stbus communication system: Concepts and definitions, reference guide. *STMicroelectronics*, 2003.
6. A. Beric, R. Sethuraman, C. A. Pinto, H. Peters, G. Veldman, P. van de Haar, and M. Duranton. Heterogeneous multiprocessor for high definition video. In *Consumer Electronics, 2006. ICCE '06. 2006 Digest of Technical Papers. International Conference on*, pages 401– 402, 2006.
7. M. H. Jeong, Y. Choi, Y. H. Chung, T. Y. Song, J. H. Jung, K. J. Hong, B. J. Min, Y. S. Choe, K.-H. Lee, and B.-T. Kim. Performance improvement of small gamma camera using nai(tl) plate and position sensitive photo-multiplier tubes. *Physics in Medicine and Biology*, 49(21):4961–4970, 2004.
8. I. A. Khatib, F. Poletti, D. Bertozzi, L. Benini, M. Bechara, H. Khalifeh, A. Jantsch, and R. Nabiev. A multiprocessor system-on-chip for real-time biomedical monitoring and analysis: architectural design space exploration. In E. Sentovich, editor, *DAC*, pages 125–130. ACM, 2006.
9. S. Pasricha, N. D. Dutt, and M. Ben-Romdhane. Constraint-driven bus matrix synthesis for mpsoc. In F. Hirose, editor, *ASP-DAC*, pages 30–35. IEEE, 2006.
10. P. G. Paulin, C. Pilkington, E. Bensoudane, M. Langevin, and D. Lyonnard. Application of a multi-processor soc platform to high-speed packet forwarding. In *DATE '04: Proceedings of the conference on Design, automation and test in Europe*, page 30058, Washington, DC, USA, 2004. IEEE Computer Society.
11. V. Reyes, W. Kruijtzer, T. Bautista, G. Alkadi, and A. Nnuez. A unified system-level modeling and simulation environment for mpsoc design: Mpeg-4 decoder case study. In *DATE '06: Proceedings of the conference on Design, automation and test in Europe*, pages 474–479, 3001 Leuven, Belgium, Belgium, 2006. European Design and Automation Association.
12. F. Sanchez, J. M. Benlloch, B. Escat, N. Pavon, E. Porras, D. Kadi-Hanifi, J. A. Ruiz, F. J. Mora, and A. Sebastia. Design and tests of a portable mini gamma camera. *Medical Physics*, pages 1384–1397, 2004.
13. F. Sanchez, M. M. Fernandez, M. Gimenez, J. M. B. J.M., M. J. Rodriguez-Alvarez, F. G. D. Quiros, C. W. Lerche, N. Pavon, J. A. Palazon, J. Martinez, and A. Sebastia. Performance tests of two portable mini gamma cameras for medical applications. *Medical Physics*, pages 4210–4220, 2006.
14. K. Sun, M. Wang, and Z. Shao. Mpsoc architectural design and synthesis for real-time biomedical signal processing in gamma cameras. In *International Conference on Biomedical Electronics and Devices*, 2008.