Low-latency Hermite Polynomial Characterization of Heartbeats using a Field-Programmable Gate Array

Kartik Lakhotia[1], Gabriel Caffarena[2], Alberto Gil[3], David G. Marquez[3], Abraham Otero[2], and Madhav P. Desai[1]

[1] Indian Institute of Technology (Bombay),
Powai, Mumbai 400076,
India
kartik@ee.iitb.ac.in
madhav@ee.iitb.ac.in
[2] University CEU-San Pablo,
Urb. Monteprincipe, 28668, Madrid, Spain
gabriel.caffarena@ceu.es
alberto.gilf@gmail.com
david.gonzalez.marquez@usc.es
abraham.otero@gmail.com
[3] Centro Singular de Investigacion en Tecnologias da Informacion (CITIUS),
University of Santiago de Compostela, 15782 Santiago de Compostela, Spain

Abstract. The characterization of ECG heartbeats is a computationally intensive problem, and both off-line and on-line (real-time) solutions to this problem are of great interest. In this paper, we consider the use of a dedicated hardware implementation (using a field-programmable gate-array (FPGA)) to solve a critical component of this problem. We describe an implementation of real-time best-fit Hermite approximation of a heartbeat using six Hermite polynomials. The implementation is generated using an algorithm-to-hardware compiler tool-chain and the resulting hardware is characterized using an off-the-shelf FPGA card. The single beat best-fit computation latency is under $0.5ms$ with a power dissipation of 3 watts.

Keywords: Hermite approximation, ECG, QRS, Arrhythmia, FPGA, Parallelization

1 Introduction

Automatic ECG analysis and characterization can be of great help in patient monitoring. In particular, the characterization of the QRS complex by means of Hermite functions seems to be a reliable mechanism for automatic classification of heartbeats [1]. The main advantages seem to be the low sensitivity to noise and artifacts, and the compactness of the representation (e.g. a 144-sample QRS can be characterized with 7 parameters [2]). These advantages have made the

Hermite representation a very common tool for characterizing the morphology of the beats [1–5].

ECG analysis using Hermite functions has a substantial amount of parallelism. Solutions to the problem have been investigated using processors (and multi-cores) and graphics processing units (GPU's). In this paper, we consider the alternative route of using an FPGA to implement the computations. In particular, our work is motivated by the potential of an FPGA (or eventually, a dedicated application-specific circuit) for low-latency energy efficient heart-beat analysis.

In generating the hardware for heart-beat analysis, we make extensive use of algorithm-to-hardware techniques. By this we mean that the hardware is generated from an algorithmic specification that is written in a high-level programming language (**C** in this case), which is then transformed to a circuit implementation using the AHIRV2 algorithm to hardware compilation tools [6–8]. The resulting hardware is then mapped to an FPGA card (the ML605 card from Xilinx, which uses a Virtex-6 FPGA). The circuit is then exercised through the PCI-express interface and used to classify beats. The round-trip latency of a single beat classification was found to be under $0.5ms$.

2 QRS approximation by means of Hermite polynomials

The aim of using the Hermite approximation to estimate heartbeats is to reduce the number of dimensions required to carry out the ECG classification, without sacrificing accuracy. The benchmarks used in this work come from the MIT-BIH arrhythmia database [9] which is made up of 48 ECG recordings whose beats have been manually annotated by two cardiologists. Each file from the database contains 2 ECG channels, sampled at a frequency of 360 Hz and with a duration of approximately 2000 beats.

Before doing the Hermite approximation, the ECG signal is processed to remove the base-line drift. The QRS complexes for each heartbeat are extracted by finding the peak of the beat (e.g. the R wave) and selecting a window of 200 ms centered on the peak. The beat-window is further extended to 400 ms by padding 100-ms sequences of zeros at each side of the complex. Thus, the QRS beat data used as an input to the Hermite polynomial approximation consists of individual beats described as a 144-sample vector $\boldsymbol{x} = \{x(t)\}$ of double precision floating point numbers. This vector is to be estimated with a linear combination of N Hermite basis functions (for the work reported in this paper, we use $N = 6$).

The goal then is to find the best minimum-mean-square-error (MMSE) approximation to $\{x(t)\}$ as

$$\hat{x}(t) = \sum_{n=0}^{N-1} c_n(\sigma)\phi_n(t,\sigma), \tag{1}$$

with

$$\phi_n(t,\sigma) = \frac{1}{\sqrt{\sigma 2^n n! \sqrt{\pi}}} e^{-t^2/2\sigma^2} H_n(t/\sigma) \tag{2}$$

where $H_n(t/\sigma)$ is the n^{th} Hermite polynomial. These polynomials can be computed recursively as

$$H_n(x) = 2xH_{n-1}(x) - 2(n-1)H_{n-2}(x), \tag{3}$$

where $H_0(x) = 1$ and $H_1(x) = 2x$. The parameter σ controls the width of the polynomials. In [1] the maximum value of σ for a given order n is estimated. As the value of n increases, the value of σ_{MAX} decreases.

The Hermite polynomials are orthornormal. Thus, the optimal coefficients that minimize the estimation error for a given σ are

$$c_n(\sigma) = \sum_t x(t) \cdot \phi_n(t,\sigma) \tag{4}$$

The best fit is calculated by comparing the MMSE approximation for each σ, and keeping the one with the smallest value. Once the best σ and the corresponding fit coefficients $\boldsymbol{c} = \{c_n(\sigma)\}$ ($n \in [0, N-1]$) are found for each heartbeat, it is possible to use only these figures to perform morphological classification of the heartbeats [1].

3 Beginning the FPGA implementation: the algorithm

The algorithm used in the FPGA implementation is illustrated in Figure 1.

```
void HermiteBestFit()
{
    receiveHermiteBasisFunctions();

    while(1)
    {
        receiveHeartBeat();
        innerProducts();
        findBestFit();
        reportResults();
    }
}
```

Fig. 1. High-level view of algorithm mapped to the FPGA

The implementation first receives the values of the Hermite polynomial basis functions, and stores them in distinct arrays in the hardware. In the current implementation, we use six arrays to store the basis functions for order $n = 0$ to

$n = 5$. For each n, basis functions for ten different values of σ are stored in the corresponding array. The values of σ used range from 1/120 to 1/90.

After this initialization step, the hardware executes a continuous loop. In the loop body, the hardware first listens for heart beats. When a complete heart-beat (144 samples) is received, the inner products of the heart-beat with all the basis functions are calculated in a double loop. After all inner products are calculated, the inner product coefficients are used to compute the best fit among the different values of σ. The best-fit σ index and the fitted values are then written out of the hardware.

The algorithm as described above is purely sequential and does not contain any explicit parallelization. The AHIRV2 compiler is intelligent enough to extract parallelism from the two critical loops (in the inner-product and best-fit functions).

Even with this simple coding of the hardware algorithm, we observe that excellent real-time performance is observed (in comparison with CPU/GPU implementations). Going further, it is possible to specify explicit parallelism by and exploit it by using multiple function units in hardware in order to reduce the processing latency. These investigations are currently in progress.

3.1 The inner product loop

The inner product loop is shown in Figure 2. The outer loop is over the samples, and the inner loop across the σ values. There is a high-level of parallelism in the inner loop which can be further boosted by unrolling the outer loop. When translating this to hardware, the entire function uses one single double-precision multiplier and one single double-precision adder. Further note that the arrays hFn and $dotPn$ are declared on a per-n basis (for $n = 0$ to $n = 5$). This allows the arrays to be mapped to distinct memory spaces, thus increasing the memory access bandwidth in the hardware.

3.2 The minimum-mean-square loop

The MMSE calculation hardware uses the algorithm shown in Figure 3. Note that the inner loop again has considerable parallelism. One pipelined double precision multiplier and one pipelined double precision adder are used to implement the loop. The arrays referred to in the loop $dotPn$ and hFn are all implemented in disjoint memories to give high memory access bandwidth.

3.3 Further optimizations

The current implementation uses a simple sequential specification. We have investigated the impact of further optimizations. In particular, we find that outer-loop unrolling (up to four) and inner-loop pipelining have substantial impact on the performance of the generated hardware. This data is reported in Section 5.

```
void  innerProduct()
{
  int I;
  for (I=0; I < NSAMPLES; I++)
  { // outer-loop
     double x = inputData[I];
     for(SI = 0; SI < NSIGMAS; SI++)
     { // inner-loop
        int I0 = I + Offset[SI];
        double p0 = (x0*hF0[I0]);
        double p1 = (x0*hF1[I0]);
        double p2 = (x0*hF2[I0]);
        double p3 = (x0*hF3[I0]);
        double p4 = (x0*hF4[I0]);
        double p5 = (x0*hF5[I0]);
        dotP0[SI] += p0;
        dotP1[SI] += p1;
        dotP2[SI] += p2;
        dotP3[SI] += p3;
        dotP4[SI] += p4;
        dotP5[SI] += p5;
     }
}
```

Fig. 2. Inner-product loop

```
void computeMSE()
{
  int I, SI;
  best_mse = 1.0e+20;
  best_sigma_index = -1;
  for (I=0; I<NSAMPLES; I=I+4)
  { // outer-loop
     for (SI=0; SI<NSIGMAS; SI++)
     { // inner-loop
        int fetchIndex0 = I + Offset[SI];
        double p0 = (dotP0[SI]*hF0[fetchIndex0]);
        double p1 = (dotP1[SI]*hF1[fetchIndex0]);
        double p2 = (dotP2[SI]*hF2[fetchIndex0]);
        double p3 = (dotP3[SI]*hF3[fetchIndex0]);
        double p4 = (dotP4[SI]*hF4[fetchIndex0]);
        double p5 = (dotP5[SI]*hF5[fetchIndex0]);
        double diff = (inputData[I]-
                        ((p0+p1) + (p2+p3) + (p4+p5)));
        err[SI] += (diff*diff);
      }
    }
    for (SI=0; SI<NSIGMAS; SI++)
    {
        if(err[SI] <  best_mse)
        {
           best_mse = err[SI];
           best_sigma_index = SI;
        }
    }
}
```

Fig. 3. MMSE calculation loop

4 Hardware Implementation Details

The overall system has 3 major components:

- A host computer, which is used to calculate the Hermite basis functions, initialize the FPGA card, send beat data to the FPGA card and receive the best-fit coefficients from the FPGA card.
- The FPGA card, on which the best-fit algorithm is implemented. We use the Xilinx ML605 card which features a Virtex-6 FPGA and an 8-lane PCI express interface.
- The FPGA card driver, which is based on the RIFFA infrastructure [10].

The algorithm mapped to the FPGA is first described in a C program (code fragments descibed in Sections 3, 3.1 and 3.2). The architecture of the hardware produced is shown in Figure 4. In the initialization phase, the hardware-side listens on an input FIFO to acquire the Hermite polynomials (these are stored in six disjoint memories, one for each order $n = 0, 1, 2, \ldots 5$). After this step, the unit that receives the beat samples is triggered. This unit listens on the input FIFO and receives a 144 sample heartbeat (coded as 144 double-precision floating point numbers). After receiving the sample, it triggers the inner-product stage. In the inner-product stage, the hardware computes, for each σ and n, an inner product of the received beat with the Hermite polynomials $\phi_n(t, \sigma)$. We are using ten values of σ and 6 values of n. Thus, 60 inner-products are computed in this phase. The inner products are stored in ten disjoint memories, one for each σ. After this is done, the MMSE stage is triggered. In the MMSE stage, the inner-products are used to find the best fit σ. The computed best-fit coefficients are sent back to the host using the output FIFO. The hardware unit which listens for the next beat is then triggered (wait for the next beat).

The VHDL hardware for this design is generated using AHIR-V2 toolchain [8]. The generated VHDL is instantiated in the FPGA together with the RIFFA wrappers, and the resulting design is synthesized and mapped to the Virtex-6 FPGA using the Xilinx ISE 14.3 toolset.

5 Results

We measure the round-trip delay and FPGA core power consumption for processing one beat. The round-trip delay is the time interval between the beginning of transmission of beat-data from the host to the hardware and the beginning of reception of best fit coefficients from the hardware. The test feeds a single beat at a time to the FPGA and measures the latency.

In the implementation, the two outer-loops described in Sections 3.1 and 3.2 were unrolled to different extents to see the impact of unrolling on the system performance. Three levels of unrolling were tried: one-way, two-way and four-way. The four-way unrolling gave the best performance, as expected. The results are summarized in Table 1 (the reported latency is the average value observed across 100 beats). The minimum latency achieved with four-way-unrolling was

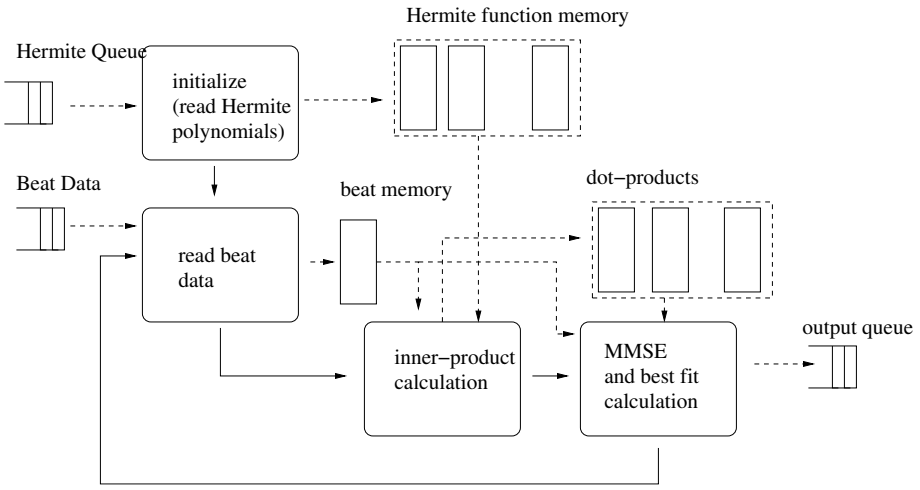

Fig. 4. Hardware Architecture

observed to be 0.39 ms (for the processing of a single 144-sample beat). The power dissipation values are measured using hardware monitoring while the beats are being processed, and represent peak power dissipation.

Table 1. Results: FPGA utilization and latency for different loop-unrolling levels

Unroll-level	Slice LUT Utilization	Slice Register Utilization	Avg. Processing Latency	FPGA core Power Consumption
1-way	56839	65995	1.39ms	2.75W
2-way	65895	80709	0.80ms	2.88W
4-way	84331	110165	0.44ms	3.09W

It is clear that FPGA can be used for real-time processing since the computation time required to process a pair of beats (corresponding to a heart-beat sampled on two channels) is less than $1ms$, which is much smaller than time-interval between actual heart beats (which is about $1s$). The observed power dissipation is 3.1W. Hardware utilization in 4-way unrolled system is less than 55% of the FPGA resource.

5.1 Comparison with GPU/CPU implementations

Hermite basis fitting has been evaluated on GPU and CPU implementations as well. For example, in [11], the authors demonstrate a GPU implementation

that shows excellent scaling behaviour, so that 100K beats can be processed in 15.7 seconds. However, when it comes to the latency needed to process a pair of beats, the FPGA can process a beat-pair in under $1ms$, whereas the CPU and GPU implementations can process a single beat-pair in $15ms$ and $5ms$ respectively. Thus, the FPGA beat-pair processing latency shows a 15X improvement relative to the CPU and a 5X improvement relative to the GPU. These comparisons are made against same algorithm executed on Intel-i7 PC(1.6GHz) and NVIDIA TESLA C2050 (1.15GHz) [11]. The operating frequency on the FPGA card was 100MHz which is less than one-tenth of that used on CPUs and GPUs. Further, the peak power consumption (while actually processing the beats) on the FPGA is 3W as compared to $100W+$ on Core i7 processors and $200W+$ on the GPU. Thus, the FPGA is an attractive option for low energy real-time ECG classification applications in portable health monitoring devices.

6 Conclusions

In this paper, a solution to the problem of Hermite polynomial based heartbeat characterization using FPGAs is presented. We have mapped the problem to hardware using algorithm-to-hardware techniques (with the AHIRV2 tools). The Xilinx ML605 card with a Virtex-6 FPGA was used as the platform and the RIFFA host-interface was used to communicate with the FPGA card.

This methodology is presented as an alternative to existing software oriented approaches, for real-time latency-sensitive signal processing. When using the FPGA, a substantial latency reduction in single-beat processing was observed (in comparison with both GPU and CPU implementations of the same algorithm). Further, the peak power dissipation in the FPGA is almost two orders of magnitude lower than that observed in the CPU/GPU case.

The highly parallel GPU and CPU architectures are very effective in off-line processing (processing of a large number of beats, not necessarily in real-time). When it comes to real-time, online beat processing, our work demonstrates that dedicated hardware implementation using an FPGA offers a very competitive platform for ECG signal processing.

References

1. Lagerholm, M., Peterson, C., Braccini, G., Edenbr, L., Sörnmo, L.: Clustering ECG complexes using Hermite functions and self-organizing maps. IEEE Trans. Biomed. Eng **47** (2000) 838–848
2. Márquez, D.G., Otero, A., Félix, P., García, C.A.: On the Accuracy of Representing Heartbeats with Hermite Basis Functions. In Alvarez, S., Solé-Casals, J., Fred, A.L.N., Gamboa, H., eds.: BIOSIGNALS, SciTePress (2013) 338–341
3. Braccini, G., Edenbrandt, L., Lagerholm, M., Peterson, C., Rauer, O., Rittner, R., Sornmo, L.: Self-organizing maps and Hermite functions for classification of ECG complexes. In: Computers in Cardiology 1997. (1997) 425–428

4. Linh, T.H., Osowski, S., Stodolski, M.: On-line heart beat recognition using Hermite polynomials and neuro-fuzzy network. Instrumentation and Measurement, IEEE Transactions on **52**(4) (2003) 1224–1231
5. Linh, T.H., Osowski, S., Stodolski, M.: On-line heart beat recognition using Hermite polynomials and neuro-fuzzy network. Instrumentation and Measurement, IEEE Transactions on **52**(4) (2003) 1224–1231
6. Sahasrabuddhe, S.D.: A competitive pathway from high-level programs to hardware. PhD thesis, IIT Bombay (2009)
7. Sahasrabudhe, S.D., Subramanian, S., Ghosh, K., Arya, K., Desai, M.P.: A c-to-rtl flow as an energy efficient alternative to the use of embedded processors in digital systems. In: DSD 2010. (2010) 147–154
8. Rinta-Aho, T., Karlstedt, M., Desai, M.: The clicktonetfpga tool-chain. In: USENIX ATC-2012, USENIX Association, Berkeley CA (2012)
9. Moody, G.B., Mark, R.G.: The impact of the MIT-BIH arrhythmia database. Engineering in Medicine and Biology Magazine, IEEE **20**(3) (2001) 45–50
10. Jacobsen, M., Kastner, R.: RIFFA 2.0: A reusable integration framework for FPGA accelerators. Volume 23. (2013) 1–8
11. Gil, A., Caffarena, G., Marquez, D., Otero, A.: Hermite polynomial characterization of heartbeats with graphics processing units. IWBBO 2014 (2014)