A CMOS Image Sensor with On-Chip Image Compression based on Predictive Boundary Adaptation and Memoryless QTD Algorithm

Shoushun Chen, Member, IEEE, Amine Bermak, Senior Member, IEEE, and Yan Wang, Student Member, IEEE

Abstract—This paper presents the architecture, algorithm and VLSI hardware of image acquisition, storage and compression on a single-chip CMOS image sensor. The image array is based on time domain digital pixel sensor technology equipped with non-destructive storage capability using 8-bit Static-RAM device embedded at the pixel level. The pixel-level memory is used to store the uncompressed illumination data during the integration mode as well as the compressed illumination data obtained after the compression stage. An adaptive quantization scheme based on Fast Boundary Adaptation Rule (FBAR) and Differential Pulse Code Modulation (DPCM) procedure followed by an on-line, least storage Quadrant Tree Decomposition (QTD) processing is proposed enabling a robust and compact image compression processor. A prototype chip including 64 x 64 pixels, read-out and control circuitry as well as an on-chip compression processor was implemented in 0.35μm CMOS technology with a silicon area of 3.2 x 3.0mm² and an overall power of 17mW. Simulation and measurements results show compression figures corresponding to 0.6 – 1 Bit-per-Pixel (BPP), while maintaining reasonable PSNR levels.

I. INTRODUCTION

With the development of network and multimedia technology, real time image acquisition and processing is becoming a challenging task because of higher resolution, which imposes very high bandwidth requirement. New applications in the area of wireless video sensor network and ultra low power biomedical applications have created new design challenges. For example, in a wireless video sensor network, limited by power budget, communication links among wireless sensor nodes are often based on low bandwidth protocols [1], such as ZigBee (up to 250 kbps) and Bluetooth (up to 1 Mbps). Even at the data rate of Bluetooth, conventional image sensor can barely stream an uncompressed 320 x 240 8-bit video at 2 frame/s. To avoid communication of raw data over wireless channels, energy efficient single chip solutions that integrate both image acquisition and image compression are required. Discrete wavelet transform (DWT), among various block-based transforms, is a popular technique used in JPEG-2000 image/video compression standard. However, implementation of image/video compression standards in cameras is computationally expensive, requiring a dedicated digital image processor in addition to the image sensor [2], [3]. A single chip solution is also possible by integrating compression functions on the sensor focal plane. This single-chip system integration offers the opportunity to reduce the cost, system size and power consumption by taking advantage of the rapid advances in CMOS technology. A number of CMOS image sensors with focal plane image compression have been proposed [4], [5], [6], [7], [8], [9], [10], [11]. In [5], an 8 x 8 point analog 2D-DCT processor is reported with fully switched capacitor circuits. In [6], floating gate technology is used to compute the DCT coefficients. However, the aforementioned designs do not actually implement compression on the focal plane since the entropy coding stage is located off-chip to limit chip size and cost. In [8], HAAR wavelets transforms are implemented by adopting a mixed-mode design approach to combine the benefits of both analog and digital domains. The CMOS image compression sensor features a 128 x 128 pixel array, a compression processor area of 1.8mm² and a total chip area of 4.4mm x 2.9mm while the total power consumption was reported to be 26.2mW [8]. In [9], a 44 x 80 CMOS image sensor integrating a complete focal-plane standard compression block with pixel prediction circuit and a Golomb-Rice entropy coder is reported. The chip has an average power consumption of 150mW and a size of 2.596mm x 5.958mm in 0.35μm CMOS technology. These various reported implementations are the results of trade-offs between the level of complexity and functionalities of the focal-plane compression block and the associated silicon area and power consumption overheads for a given resolution of the imager. In [11], a compression processor is proposed, whose complexity and power consumption are to some extent independent of the resolution of the imager, making it very attractive for high resolution high-frame rate image sensors [11]. The single chip vision sensor integrates an adaptive quantization scheme followed by a quadrant tree decomposition (QTD) to further compress the image. The compression processor exhibits a significantly lower power consumption (e.g. 6.3mW) and occupies a silicon area of 1.8mm². The compression sensor permits to compress the data to 0.6 – 1 Bit-per-Pixel (BPP). The imager uses a Morton(Z) [12] block-based scanning strategy. The transition from one quadrant to the next involves jumping to a non-neighboring pixel, resulting in spatial discontinuities or image artifacts. In this paper, we propose a second generation prototype with the following main contributions: a) new Hilbert scanning technique and its hardware implementation to avoid spatial discontinuities in the block-based scanning strategy; b) the 1-bit Fast Boundary Adaptation Rule (FBAR) algorithm is performed on the predictive error rather than the pixel itself.
using Differential Pulse Code Modulation (DPCM), which results in improved performance; c) introduction of memory reuse technique enabling over a three-fold reduction in silicon area and d) improved pixel structure for the DPS sensor. The proposed second generation imager with focal plane image compression is successfully implemented using Alcatel 0.35μm CMOS technology.

The remainder of the paper is organized as follows. Section II introduces the design of a digital Time-to-first-Spike (TFS) image sensor. Section III discusses the algorithmic considerations for the FBAR algorithm combined with the predictive coding technique and presents the simulation results showing the improvements. Section IV describes the imager architecture and discusses design strategies used for implementing the Hilbert scan as well as the QTD processing involving the memory reuse concept. Section V reports the experimental results and provides a comparison with other compression processors. Section VI concludes this work.

II. PIXEL DESIGN AND OPERATION

The proposed system integrates the image sensor with pixel level ADC and frame storage together with the array-based stand-alone compression processor. The sensor array adopts a time domain digital pixel sensor (DPS) [13], in which the image is captured and locally stored at the pixel level. The image array consists of 64 × 64 digital pixel sensors. Fig. 1(a) illustrates the circuit diagram of the pixel, which includes 4 main building blocks, namely the photodetector PD with its internal capacitance $C_d$, followed by a reset transistor M1, a comparator (M2-M8) and an 8-bit SRAM. The comparator’s output signal (Out) is buffered by (M9-M10) and then used as a write enable signal (“WEn”) for the SRAM.

Fig. 1(b) illustrates the operation timing diagram of the proposed pixel, which is divided into two separate stages denoted as Acquisition stage and Read-out/Store stage. The first stage corresponds to the integration phase, in which the illumination level is recorded asynchronously within each pixel. The voltage of the sensing node $V_N$ is first reset to $V_{ddA}$. After that, the light falling onto the photodiode discharges the capacitance $C_d$ associated with the sensing node, resulting in a decreasing voltage across the photodiode node. Once the voltage $V_N$ reaches a reference voltage $V_{ref}$, a pulse is generated at the output of the comparator Out.

The time to generate the first spike is inversely proportional to the photocurrent [13] and can be used to encode the pixel’s brightness. A global off-pixel controller operates as a timing unit, which is activated at the beginning of the integration process and provides timing information to all
the pixels through "Data Bus". The pixel’s "WEn" signal is always valid until the pixel fires. Therefore, the SRAM will keep tracking the changes on the "Data Bus" and the last data uploaded is the pixel’s timing information. Once the integration stage is over, the pixel array turns to Read-out/Store stage. During this operating mode, the pixel array can be seen as a distributed static memory which can be accessed in both read or write modes using the Row and Column addresses. The on-chip image processor will first readout the memory content, compress the data and reuse the on-pixel memory as storage elements. With the external global control signal "R/W" and the row and column select signals RSel and CSel, the pixel’s SRAM can be accessed in both read or write, namely:

- When the "R/W" signal is "1", the pixel will drive the "Data Bus" and the memory content will be readout.
- When the "R/W" signal turns to "0", transistor M11 and M12 will be turned on and the "WEn" signal is enabled again. The memory can therefore be accessed for write mode again and can be used as storage element for the processor.

This new feature differs significantly from previous DPS implementations, in which the on-pixel memory is only used for storing the raw pixel data. In our proposed design, the on-pixel memory is used to store the uncompressed illumination data during integration mode, as well as the compressed illumination data obtained after the compression stage. The memory is therefore embedded within the pixel array but also interacts with the compression processor for further processing. Moreover, the new pixel design also reduces the number of transistors from 102 to 84 compared to the pixel reported in [13]. This is achieved by removing the self-reset logic for the photodiode and the reset transistor for each bit of the on-pixel SRAM. In addition, the current pixel only requires two stages of inverter to drive the write operation for the memory. This is made possible because the SRAM's "WEn" signal is no longer pulse width sensitive.

III. IMAGE COMPRESSION - ALGORITHMIC CONSIDERATIONS

The image compression procedure is carried-out in three different phases. In the first phase, the image data is scanned out of the array using Hilbert scanning then compared to the previously reconstructed pixels. The comparison result, 0 or 1, is taken as a codeword BPP(n) to update the boundary point y delimiting two quantization intervals, with R0 = [0, y] and R1 = [y, 255]. At each time step, the input pixel intensity will fall into either R0 or R1. BP is shifted to the direction of the active interval by a quantity η. After that, the BP itself is taken as the reconstructed value. With this adaptive quantization procedure, the BP tracks the input signal and since BP itself is used as the reconstructed value, a high resolution quantization is obtained even when using a single bit quantizer

\[
d(x, Q(x)) = \sum_{i=1}^{N} |x - y_i|^r p(x)dx
\]

It has been shown that using Fast Boundary Adaptation Rule [14] can minimize the r-th power law distortion, e.g., the mean absolute error when \( r = 1 \) or the mean square error when \( r = 2 \). At convergence, all the N quantization intervals \( R_i \) will have the same distortion \( D_r(i) = D_r/N \) [14]. This property guarantees an optimal high resolution quantization. For a 1-bit quantizer, there will be just one adaptive boundary point \( y \) delimiting two quantization intervals, with \( R0 = [0, y] \) and \( R1 = [y, 255] \). At each time step, the input pixel intensity will fall into either R0 or R1. BP is shifted to the direction of the active interval by a quantity η. After that, the BP itself is taken as the reconstructed value. With this adaptive quantization procedure, the BP tracks the input signal and since BP itself is used as the reconstructed value, a high resolution quantization is obtained even when using a single bit quantizer

\[
\text{BP} = \text{Reg}0 \times 1.375 - \text{Reg}1 \times 0.75 + \text{Reg}2 \times 0.375
\]

Compared to the scheme reported in [11], BPp is now a function of three neighboring pixels and the estimated pixel value (prediction) is compared with the real incoming value. The comparison result, 0 or 1, is taken as a codeword \( u(n) \), which is further used to update the boundary point:

\[
\text{if } (u(n) = 1), \text{BP} = \text{BP}p + \eta; \text{ else } \text{BP} = \text{BP}p - \eta
\]

The newly obtained BP is feed back to update Reg0 and to predict the next pixel’s value. The codeword \( u(n) \) is also used to adjust another very important parameter η. Indeed, the adaptation step size parameter η is found to affect the quantizer’s performance [11]. On one hand, a large η is preferred so as to track rapid fluctuations in consecutive pixel
values. On the other hand, a small $\eta$ is preferred so as to avoid large amplitude oscillations at convergence. To circumvent this problem, we propose to make $\eta$ adaptive using a heuristic rule described as follows:

- **case 1**: If the active quantization interval does not change between two consecutive pixel readings, we consider that the current quantization parameters are far from the optimum and $\eta$ is then multiplied by $\Lambda > 1$.
- **case 2**: If the active quantization interval changes between two consecutive pixel readings, we consider that the current quantization parameters are near the optimum and thus $\eta$ is reset to its initial value $\eta_0$ (typically a small value).

This rule can be easily implemented by simply comparing two consecutive codewords, namely $u(n)$ and $u(n-1)$. Codeword values that are consecutively equal can be interpreted as a sharp transient in the signal as the $BP$ is consecutively adjusted in the same direction. In this situation, a large $\eta$ is used. Consequently, when $u(n) = u(n-1)$, $\eta$ is updated as $\eta = \eta \times \Lambda$. Otherwise, i.e., when $u(n) \neq u(n-1)$, $\eta = \eta_0$.

**B. Hilbert scanning**

The adaptive quantization process explained earlier permits to build a binary image on which quadrant tree decomposition (QTD) can be further employed to achieve higher compression ratio. The QTD compression algorithm is performed by building a multiple hierarchical layers of a tree which corresponds to a multiple hierarchical layers of quadrants in the array. To scan the image data out of the pixels array, many approaches can be employed. The most straightforward way is, for example, raster scan. However the choice of the scan sequence is very important as it highly affects the adaptive quantizer and QTD compression performance. Generally speaking, block based scan can result in higher PSNR and compression ratio because it provides larger spatial correlation, which is favorable for the adaptive quantization and QTD processing.

Fig. 3.(a) illustrates a typical Morton (Z) scan [12] which is used to build the corresponding tree as reported in [11]. In this approach, transition from one quadrant to the next involves jumping to a non-neighboring pixel, which results in spatial discontinuity, which gets larger and larger when scanning the array due to the inherent hierarchical partition of the QTD algorithm. This problem can be addressed by taking the boundary point from the physically nearest neighbor of the previous quadrant rather than the previously scanned pixel [12]. Unfortunately, this solution comes at the expense of two additional 8-bit registers for each level of the quadrant. As shown in Fig. 3.(a), two registers ($A4, B4$) are needed to store the boundary point for the $4 \times 4$ quadrant level and two other registers ($A8, B8$) are needed to store those related to the $8 \times 8$ quadrant level.

Fig. 3.(b) illustrates an alternative solution using Hilbert scan sequence. In this scheme, multi-layers hierarchical quadrants are sequentially read-out while maintaining spatial continuity during transitions from quadrant to the next. The storage requirement issue is also addressed in this scheme as for the adaptive quantization processing, the neighboring pixel values are the ones just consecutively scanned. Hardware implementation of Hilbert scanning can be quite straightforward using hierarchical address mapping logic.

In summary, the compression scheme proposed in this paper can be generally interpreted as the cascade of 2 basic blocks namely the boundary adaptation block and the QTD processor. The first stage is a lossy compression for which there is a trade-off between the compression ratio and the quality of the image. The compression performance is therefore controllable because the user can define the required number of bits at the output of the first block. The second stage (QTD processor) is a lossless compression as it processes a binary image and looks for removing spatial redundancy within the block. The compression quality in the second block is not controllable and is highly dependant on the input image. The main trade-off involved in this design are related to the first stage in which the number of bits at the output of the adaptive quantizer.

A larger number of bits enables improved signal to noise...
C. Simulation Results

The performance of our proposed compression scheme i.e. adaptive $\eta$ with DPCM using Hilbert scan ($\eta$-Hilbert+DPCM), is compared with other operating modes, namely fixed $\eta$ raster scan ($\eta$-R), adaptive $\eta$ raster scan ($\eta$-R), adaptive $\eta$ Morton (Z) scan ($\eta$-MZ), adaptive $\eta$ smooth boundary Morton (Z) scan ($\eta$-SmoothMZ), adaptive $\eta$ Hilbert scan ($\eta$-Hilbert) and adaptive $\eta$ with DPCM using Hilbert scan ($\eta$-Hilbert+DPCM). $M = \frac{1}{2^{n}}$ [dB/BPP]. For each operating mode, $\eta_0$ was optimized in order to achieve the best possible performance. The $\eta$-Hilbert+DPCM mode presents the best PSNR and BPP figures compared to the first generation compression algorithm [11] and for all possible operating modes.

![Image 72x404 to 526x505]

Fig. 4. Simulation results illustrating the compression performance (PSNR and BPP) as function of $\eta_0$. The left and right y-axes illustrate the PSNR and BPP, respectively. The simulation is reported for two image sizes namely: (a) image size of 256 $\times$ 256 and (b) image size = 512 $\times$ 512.

<table>
<thead>
<tr>
<th>Operation modes</th>
<th>64 $\times$ 64</th>
<th>128 $\times$ 128</th>
<th>256 $\times$ 256</th>
<th>512 $\times$ 512</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PSNR</td>
<td>BPP</td>
<td>M</td>
<td>$\eta_0$</td>
</tr>
<tr>
<td>$\eta_0$-R</td>
<td>21.00</td>
<td>1.01</td>
<td>20.83</td>
<td>26</td>
</tr>
<tr>
<td>$\eta$-K</td>
<td>21.60</td>
<td>1.09</td>
<td>20.79</td>
<td>24</td>
</tr>
<tr>
<td>$\eta$-SmoothMZ</td>
<td>21.48</td>
<td>0.97</td>
<td>22.15</td>
<td>18</td>
</tr>
<tr>
<td>$\eta$-Hilbert</td>
<td>22.77</td>
<td>0.98</td>
<td>23.37</td>
<td>17</td>
</tr>
<tr>
<td>$\eta$-Hilbert+DPCM</td>
<td>23.36</td>
<td>0.88</td>
<td>26.14</td>
<td>19</td>
</tr>
<tr>
<td>DCT[16]</td>
<td>19.28</td>
<td>0.91</td>
<td>21.18</td>
<td>16</td>
</tr>
<tr>
<td>Wavelet[16]</td>
<td>26.78</td>
<td>0.75</td>
<td>35.70</td>
<td>30</td>
</tr>
<tr>
<td>QD[11]</td>
<td>31.49</td>
<td>0.56</td>
<td>4.80</td>
<td>30.34</td>
</tr>
</tbody>
</table>

TABLE I

AVERAGE PERFORMANCE OF 20 TEST IMAGES FROM SCIENTIFIC IMAGE DATABASE[15] UNDER DIFFERENT OPERATING MODES, NAMELY DCT[16], WAVELET TRANSFORM[16], QTD[7], FIXED $\eta$ RASTER SCAN ($\eta_0$-R), ADAPTIVE $\eta$ RASTER SCAN ($\eta$-R), ADAPTIVE $\eta$ MORTON (Z) SCAN ($\eta$-MZ), ADAPTIVE $\eta$ SMOOTH BOUNDARY MORTON (Z) SCAN ($\eta$-SmoothMZ), ADAPTIVE $\eta$ HILBERT SCAN ($\eta$-Hilbert) AND ADAPTIVE $\eta$ WITH DPCM USING HILBERT SCAN ($\eta$-Hilbert+DPCM). $M = \frac{1}{2^{n}}$ [dB/BPP]. For each operating mode, $\eta_0$ was optimized in order to achieve the best possible performance. The $\eta$-Hilbert+DPCM mode presents the best PSNR and BPP figures compared to the first generation compression algorithm [11] and for all possible operating modes.
configuration. Both PSNR and BPP are highly dependant upon the value of $\eta$. Using a large value for $\eta$ enables faster tracking of sharp transients in the signal and hence improved compression ratios are obtained when combined with QTD. However $\eta$ cannot be increased indefinitely as it will result in a rapidly degraded PSNR performance. For image sizes of $256 \times 256$ and $512 \times 512$, the optimum $\eta$ was found to be around 13 and 10, respectively. From Table I, one can notice the benefit of adaptive $\eta$ by comparing the performance figures in the first and second rows, where both modes are based on conventional row and column raster scan. With adaptive $\eta$ (second row), the PSNR is increased by about 0.4dB. Morton (Z) enables better performance as compared to raster scan because it is a block based scan improving the spatial correlation, which is exploited by both the adaptive Q and QTD processing blocks (third row). However, in Morton (Z) scan, the transition from one quadrant to the next involves transitions to a non-neighboring pixel resulting in spatial discontinuity. In [11], a smooth boundary point propagation scheme is proposed, enabling to solve this spatial discontinuity issue resulting in a PSNR improvement of about 1-1.5dB (fourth row). Hilbert scan provides another interesting block-based scan strategy featuring spatial continuity. It is clearly shown that the performance of Hilbert scan are superior to that of raster, Morton Z and even smooth Morton Z scanning strategies (fifth row). It is important to note from this table that using predictive boundary adaptive coding combined with Hilbert scanning ($\eta$-Hilbert+DPCM) enables about 25% improvement in terms of performance (expressed by the PSNR to BPP ratio) compared to the first generation design [11]. From Table I, we can also note that performance improvements are obtained when using large size images. For our proposed algorithm ($\eta$-Hilbert+DPCM), using $512 \times 512$ format instead of $64 \times 64$ enables a 23% and a 14% improvements in terms of PSNR and BPP, respectively. This represents a significant improvement suggesting that the proposed algorithm is much more effective for large size images. Table I also illustrates a comparison of the proposed algorithm to other standards. One can note that the performance of our processor are clearly superior to a stand-alone QTD and comparable to DCT based compression DCT [16] but clearly inferior to that of wavelet based compression DCT [16]. It is however important to note that the hardware complexity is an order of magnitude simpler when compared to both DCT and wavelet-based compression. This is due to the inherent advantage of boundary adaptation processing requiring simple addition, substraction and comparison for $\eta$ adaptation. The storage requirement is however quite demanding for QTD processing since a tree construction and storage is required, however, this issue and some of the hardware optimization techniques will be addressed in our proposed system, as will be explained in section IV.

IV. VLSI IMPLEMENTATION

A. Imager Architecture

Fig. 5.(a) shows the block diagram of the overall system featuring the CMOS image sensor integrated together with the compression processor including the adaptive DPCM quantizer and the QTD processor. The image array consists of $64 \times 64$ digital pixel sensors. The pixel array is operated in two separate phases. The first phase corresponds to the integration phase, in which the illumination level is recorded and each pixel sets its own integration time which is inversely proportional to the photocurrent. A timing circuit is used in order to compensate for this non-linearity by adjusting the quantization times using a non-linear clock signal which is fed to the counter [13]. Moreover, proper adjustment of the quantization timing stamps stored in a $16 \times 256$-bit on-chip SRAM memory enables to implement various transfer functions including a log-response [17].

During the integration phase, the row buffers drive the timing information to the array, using gray code format. Once
the longest permitted integration time is over, the imager turns into the read-out mode. The row buffers are disabled and the image processor starts to operate. First, the QTD processor will generate linear quadrant address which is then translated into Hilbert scan address by the Hilbert Scanner block. The address is decoded into “Row Select Signal (RSel)” and “Column Select Signal (CSel)”. The selected pixel will drive the data bus and its value will be first quantized by the DPCM Adaptive Quantizer then the binary quantization result will be compressed by the QTD processor.

**B. Hilbert Scanner**

Hilbert scan is actually composed of multiple levels of four basic scanning patterns as shown in Fig. 6.

These are denoted as **RR**, **−RR**, **−CC**, and **CC**, respectively. **RR** represents a basic scanning pattern featuring a relationship between its linear scanning sequence and the physical scanning address described as follows:

\[
\begin{align*}
&\text{Linear Add : } ('b00) \rightarrow ('b01) \rightarrow ('b10) \rightarrow ('b11) \\
&\text{Hilbert Add : } ('b00) \rightarrow ('b01) \rightarrow ('b11) \rightarrow ('b10)
\end{align*}
\]

**CC** represents another basic scanning pattern with the following address mapping relationship:

\[
\begin{align*}
&\text{Linear Add : } ('b00) \rightarrow ('b01) \rightarrow ('b10) \rightarrow ('b11) \\
&\text{Hilbert Add : } ('b00) \rightarrow ('b10) \rightarrow ('b11) \rightarrow ('b10)
\end{align*}
\]

For an array of \(2^m \times 2^n\) pixels, the whole Hilbert scan can be represented by \(m\) levels of scanning patterns. For an intermediate level, its scanning pattern is determined by its parent quadrant’s pattern. At the same time, its scanning pattern can also determine its child quadrants’ patterns, as illustrated in Fig. 7. If a quadrant is in the **RR** format, then its four children quadrants must be in the **RR** \(\rightarrow\) **−CC** formats, respectively. Using this strategy, it is possible to implement Hilbert scan in a top-down approach. Firstly, a linear address is used to segment the whole array into quadrant levels. Each quadrant level is addressed by a 2-bit address. Secondly, the scanning pattern for each quadrant level is retrieved. For the very top quadrant level, the scanning sequence is predefined as either **RR** or **CC**. If the current scan sequence is **RR**, then the scanning sequences of the four children quadrants should be **CC** \(\rightarrow\) **RR** \(\rightarrow\) **CC**, respectively. The 2 Most Significant Bits (MSB) of the address are used to decode one out of four largest quadrants being scanned. If the 2-bit MSB are equal to ‘b11, the fourth quadrant is being scanned and its scanning pattern is set to **−CC** format. Consequently, its four sub-quadrants are set to be **−RR** \(\rightarrow\) **−CC** \(\rightarrow\) **−CC** \(\rightarrow\) **RR** formats, respectively. Furthermore, the decoding of the sub-quadrants is performed using the second 2 MSB bits of the linear address. Applying the same procedure on the subsequent hierarchical levels enables the mapping of all the linear address into Hilbert scan address. The above mapping only involves bitwise manipulation and therefore, no sequential logic is needed, which results in very compact VLSI implementation.

**C. QTD Algorithm with Pixel Storage Reuse**

For our \(64 \times 64\) array, the tree information is to be stored in registers with a total number of \(1024 + 256 + 64 + 16 + 4 + 1 = 1407\). In [11] the QTD tree is built out of the pixel array, which occupies significant silicon area. A possible solution to save area is based on the following observation: The proposed 1-bit FBAR algorithm compresses the original 8-bit pixel array into a binary image with only 1-bit per pixel. QTD tree can therefore be stored inside the array by reusing the storage elements of the DPS pixels.

The QTD algorithm is based on the fact that if a given quadrant can be compressed, only its first pixel’s value and its root are necessary information. All the other pixels in the quadrant and the intermediate level nodes on the tree can be compressed. The only storage requirement outside the pixel
The single chip image sensor and compression processor is implemented using 0.35μm Alcatel CMOS digital process (1-poly 5 metal layers). Fig. 5.(a) illustrates the architecture of the overall imager including the sensor and the processor. Fig. 5.(b) illustrates the corresponding microphotograph of the chip with a total silicon area of $3.2 \times 3.0 \mu m^2$. The 64 x 64 pixel array was implemented using a full-custom approach. The main building blocks of the chip are highlighted in Fig. 5.(b). The photosensitive elements are $n^+ p$ photodiodes chosen for their high quantum efficiency. Except for the photodiode, the entire in-pixel circuitry (Fig. 1(a)) is shielded from incoming photons to minimize the impact of light-induced current resulting in parasitic light contribution to the signal. Guard rings are extensively used to limit substrate coupling and as means to shield the pixels from the outer array digital circuitry. Power and ground buses are routed using top layer metal. Fig. 5.(c) illustrates the layout of the pixel. Each pixel occupies an area of $39 \times 39 \mu m^2$ with a fill-factor of 12%. The digital processor was synthesized from HDL and implemented using automatic placement and routing tools. The digital processor occupies an area of $0.25 \times 2.2 = 0.55 \mu m^2$. It should be noted that the newly proposed design achieves an area reduction of over 70% as compared to [11] (1.8mm$^2$). This is mainly due to the optimization of the storage requirement for the QTD tree using “Pixel Storage Reuse” technique, which saves a large number of flip-flops. Table II, reports the number of flip-flops used in this processor compared to that reported in [11].
FPGA Board

Host PC

Chip mounted on PCB

Fig. 9. FPGA based test platform which is composed of a host computer, FPGA board (Memec Virtex-4 MB) and the chip under test.

FPGA platform and transferred to the PC for display. Once the data is received, the total number of bits per frame ($B_F$) is counted and the BPP is calculated as:

$$BPP = \frac{B_F}{64 \times 64}$$

(4)

BPP=8
8 8 8 8 8

BPP=0.68
0.82 1.02 0.85 1.03

BPP=8
8 8 8 8 8

BPP=1.23
0.65 1.00 1.03 0.85

Fig. 10. Captured images from the prototype chip. The first and third rows show the sample images captured without compression while the second and fourth rows represent the reconstructed compressed images using the proposed image compression processor. Table IV compares the performance of the our proposed scheme presented in this paper with the first generation processor [11] as well as other imagers with compression processors reported in the literature [6], [7], [8], [9], [10]. One should note that the comparison of different compression processors is not obvious as the target performance is different for different designs and therefore computational requirements and circuit complexities, image quality and compression performance as well as imager resolution and specifications may vary significantly. In addition, some designs implement only certain building blocks of the compression algorithm on the focal plane, while an external post-processing is still required to realize a full compression system. Some other implementations only focus on the compression processing ignoring the sensor, the ADC circuitry and the frame storage and buffering. This renders the comparison of different designs very subjective and non-conclusive. One can however notice that our proposed chip does not require any post-processing and the compression processor is successfully integrated together with the sensor achieving quite low silicon area and reasonably low power consumption.

VI. Conclusion

This paper reports a single chip CMOS image sensor with on-chip image compression processor, based on a hybrid predictive boundary adaptation processing and QTD encoder. Hilbert scan is employed to provide both spatial continuity and quadrant based scan. The proposed compression algorithm enables about 25% improvement in terms of performance (PSNR to BPP ratio) compared to the first generation design. Reported performance are clearly superior to that of a stand-alone QTD and quite comparable to DCT-based compression. The hardware complexity is however an order of magnitude simpler when compared to both DCT and wavelet based compression. This is due to the inherent advantage of boundary adaptation processing requiring simple addition, substraction and comparison for $\eta$ adaptation. The storage requirement is however quite demanding for QTD processing since a tree construction and storage is required, however, this issue is addressed in this paper by introducing a QTD algorithm with pixel storage reuse technique. The memory is therefore embedded within the pixel array but also interacts with the compression processor for further processing storage. This technique has enabled an area reduction of the compression processor by about 70%. The proposed hardware friendly algorithm has therefore enabled a complete system implementation which integrates the image sensor with pixel level ADC and frame storage together with the full stand-alone compression processor including predictive boundary adaptation and QTD. A prototype chip including a 64x64 pixel array was successfully implemented in 0.35 $\mu$m CMOS technology with a silicon area of $3.2 \times 3.0 \text{mm}^2$. A very interesting fact about this design is that compression is performed on-the-fly while scanning out the data using Hilbert scanner. This results in reduced timing overhead while the overall system consumes less than 18mW of power.

ACKNOWLEDGMENT

The authors would like to thank Dr. Dominique Martinez for helpful discussions. This work was supported by the Research Grant Council of Hong Kong under grant references (610405 and 610509).


**Comparison of Our Design with Some Imagers with On-Chip Compression Reported in the Literature. These Designs Are Based on Different Compression Scheme Such as DCT, Wavelet Transform, Predictive Coding. Estimated Areas Are Marked in Asterisk (*).**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Chip Level</td>
<td>Pixel</td>
<td>Column</td>
<td>Column</td>
<td>Chip</td>
<td>Chip</td>
<td>Chip</td>
<td>Chip</td>
</tr>
<tr>
<td>Lossy Type</td>
<td>Lossy</td>
<td>Lossy</td>
<td>Lossless</td>
<td>Lossy</td>
<td>Lossy</td>
<td>Lossy</td>
<td>Lossy</td>
</tr>
<tr>
<td>Technology</td>
<td>0.5μm</td>
<td>0.35μm</td>
<td>0.35μm</td>
<td>0.5μm</td>
<td>0.35μm</td>
<td>0.35μm</td>
<td>0.35μm</td>
</tr>
<tr>
<td>Array Size</td>
<td>104×128</td>
<td>32×32</td>
<td>128×128</td>
<td>80×44</td>
<td>33×25</td>
<td>64×64</td>
<td>64×64</td>
</tr>
<tr>
<td>Processor Area</td>
<td>1.5mm²</td>
<td>0.4mm²</td>
<td>1.6mm²</td>
<td>0.11mm²</td>
<td>0.36mm²</td>
<td>1.8mm²</td>
<td>0.55mm²</td>
</tr>
<tr>
<td>Power</td>
<td>80μW/frame</td>
<td>26.2mW/chip</td>
<td>24.4mW/chip</td>
<td>150mW/chip</td>
<td>0.25mW/chip</td>
<td>28mW/chip</td>
<td>6.3mW/proc.</td>
</tr>
<tr>
<td>post-proc requirement</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

**TABLE IV**

**REFERENCES**


