Stand-alone Memory Controller for Graphics System

Tassadaq Hussain[1], Oscar Palomar[1], Osman S. Ünsal[1], Adrian Cristal[1], Eduard Ayguadé[1], Mateo Valero[1], and Amna Haider[2]

[1] Barcelona Supercomputing Center
[2] Unal Center of Education Research and Development
[1]{first}.{last}@bsc.es, [2]amna@ucerd.com

Abstract. There has been a dramatic increase in the complexity of graphics applications in System-on-Chip (SoC) with a corresponding increase in performance requirements. Various powerful and expensive platforms to support graphical applications appeared recently. All these platforms require a high performance core that manages and schedules the high speed data of graphics peripherals (camera, display, etc.) and an efficient on chip scheduler. In this article we design and propose a SoC based Programmable Graphics Controller (PGC) that handles graphics peripherals efficiently. The data access patterns are described in the program memory; the PGC reads them, generates transactions and manages both bus and connected peripherals without the support of a master core. The proposed system is highly reliable in terms of cost, performance and power. The PGC based system is implemented and tested on a Xilinx ML505 FPGA board. The performance of the PGC is compared with the Microblaze processor based graphic system. When compared with the baseline system, the results show that the PGC captures video at 2x of higher frame rate and achieves 3.4x to 7.4x of speedup while processing images. PGC consumes 30% less hardware resources and 22% less on-chip power than the baseline system.

1 Introduction

Graphics systems are now being used in different engineering sectors such as artificial intelligence, robotics, telecommunication, etc. Hand-held devices, such as personal digital assistants (PDAs), or cellular phones have embedded cameras, and most of them have mega-pixel image sensor cameras and interactive applications such as symbol recognition, etc. As the image resolution of these devices grows, application specific and high-performance hardware are required to run complex graphics code.

A typical FPGA based SoC comprises one or two processor cores (master core and the dedicated core), a programmable crossed bus arbitration module, an interrupt and direct memory control module along with control bus and data bus. The master core can be used for real time operation, and the dedicated core is reserved for graphics operation, whereas low power devices have a single processor and perform camera processing in software. In line with Moore's law [1], the architecture of graphics devices have been developing from simple micro-controller to multi-core processors. The computing power and memory component speed improves at the rate of around 50% and 25% respectively per year. This is much smaller compared to the speed of processor [2]. This speed gap widens the gap between power and memory wall in the graphics industry and affects the performance of graphics applications.

In this work, we intend to design a low-power and high-performance bus controller called the Programmable Graphics Controller (PGC) for FPGA based SoC. The PGC resides in

on-chip *Bus Unit* and holds data transfer patterns and control instructions in its program memory. The PGC provides on-chip and off-chip bus interconnects and controls data transfer without the support of complex bus matrix, DMA and master processor. The PGC reduces the master/slave arbitration delay, bus switching time, balances the work load, and gives a promising interconnection approach for multi-peripherals with the potential to exploit parallelism while coping the memory/network latencies. PGC bus scheduler provides low-cost and simple control that arranges multiple bus access requests and communicates with integrated processing units. We integrated dedicated hardware accelerators in the design as they have low footprint and low power consumption and gives high performance computation. The PGC supports multi-peripherals (camera, display) and processor core without support of the master cores and operating system. The integration of PGC with peripherals facilitates the graphics system to overcome wire (interconnection) and memory read/write delays and improves the performance of application kernels by arranging complex on-chip data transfers.

2 Related Work

Rowe et al. [3] proposed an FPGA based camera system for sensor networking applications. The design dealt with an 8-bit microprocessor and FPGA based hardware accelerator to capture and process the images. The system does not support high resolutions due to limited RAM and can process 2 frame per second. Petouris et al. [4] proposed an architecture that is used to implement and test advanced image processing algorithms. The system gives control to the user to control camera system through LCD panel, and the system maintains contact with a PC through a JTAG interface for storing the images on it. The design is evaluated on Altera's DE2 development board and is designed to be a low cost proposal for academia. Murphy et al. [5] proposed a low cost stereo vision system on an FPGA, based on the census transform algorithm. The design uses a camera projection model to represent the image formation process that occurs in each of the two cameras which is suitable for independent vehicles for agricultural applications. The PGC system does not require a master core to manage the graphics system. The data movement is controlled by an on-chip scheduler at higher frame rate which reduces the power and cost of system. A light weight 16-bit processor core is also proposed in the design to perform basic image processing.

Matthew proposed an optical imaging system [6] having multiple high sensitivity cooled CCD cameras. The system gives desired representation of point source metastases and other small injuries. Shi et al. [7] presented a camera that can not only see but also perform recognition called Smart Camera. The proposed camera system recognizes simple hand gestures. The camera was built using a single chip of FPGA as processing device. The PGC system has ability to read data from multiple image sensors and provide it to processing core in arranged formate. The PGC bus management allows the processing core to perform computation (recognition, transformation, etc) on run-time video at higher frame rates. 16-bit or 32-bit processing cores can be used to perform complex algorithms.

To solve the on-chip bus bandwidth bottleneck, several types of high-performance on-chip buses have been proposed. The multi-layer AHB (ML-AHB) bus-matrix proposed by ARM [8] has been used in may SoC designs due to its simplicity, the good architecture and low power. The ML-AHB bus-matrix interconnection scheme provides parallel access paths between multiple masters and slaves in a system. Hussain et al. [9–12] discussed the architecture of a memory controller and its implementation on a Xilinx Virtex-5 in order

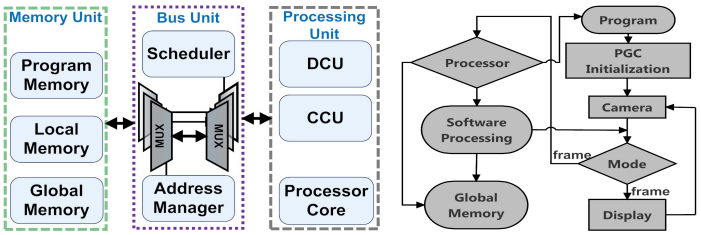


Fig. 1. PGC : (a) Internal Structure (b) Flowchart

to establish a fast communication with the host. The PLB crossbar switch (CBS) from IBM [13] allows communication between masters on one PLB and slaves on the other. The CBS supports concurrent data transfers on multiple PLB buses along with a prioritization method to hold multiple requests to a generic slave port. Like other on-chip Bus Units, AHB (ML-AHB) and PLB (CBS) use a master core that manages on-chip bus transactions. The PGC controls processing units and manages data transfer between them without support of complex bus-matrix and processor core. This reduces request/grant time and bus arbitration time. Moreover, the support for strided and scatter/gather data transfers allows the PGC system to manage complex data transactions.

3 PGC Graphics System Specification

In this section, we describe the specification of PGC system and design its architecture. The section is further divided into four subsections: *Overview of PGC System*, the *Processing Units*, the *Memory Unit* and the *Bus Unit*.

3.1 Overview of PGC System

The PGC graphics system architecture is pipelined from the sensor chip over the wire to the processing chip. The PGC inner architecture is shown in Figure 1(a), which shows the interconnection with the processing units and memory. The system uses a combined hardware/software solution that includes hardware accelerators and a RISC processor core. The camera control unit (CCU) and display control unit (DCU) are custom hardware accelerators and control the camera sensor and the display unit respectively. The *Local Memory* holds the CCU/DCU data for basic image/video processing using the *Processor Core*. To store high resolution images the *Global Memory* is integrated. The *Program Memory* is used to hold CCU/DCU program description and data transfer information. Depending upon the data transfer the *Address Manager* takes single or multiple instructions from *Program Memory* and schedules the data movement for *CCU* and *DCU*. The PGC *Scheduler* handles the concurrent bus request by the CCU and DCU and rearranges multiple bus access requests and arbitrates data transfer without creating bus contention.

We define two use cases of graphic system (shown in Table 1); the *Video Mode* and the *Snapshot Mode*. The *processing* step is used to perform filtering, compression, transformation, etc. on the input image. Each use case has two variants: with-processing and without-processing. The resolution of *Video Mode* is selected to fit in *Local Memory* of the target device. In our current design, the *Video Mode* supports up to 640×480 image resolution.

Table 1. Graphics System: Use Case, Mode of Operations

Use Case	Processing	Pixel $_{Depth}$	Resolution	frame/sec (fps)
Video Mode				
Single-Camera Video	With/Without	24-bit	VGA = 640 × 480	variable up to 150
Dual-Camera Video	With	24-bit	VGA = 640 × 480	variable
Snapshot Mode	With/Without	24-bit	QSXGA = 2560 × 2048	1

It reads multiple frames (images) per second (*fps*) from the camera sensor and transfers them to display unit. The *Video Mode* is further divided into two modes. The *Single-Camera Video* uses a single image sensor and *Dual-Camera Video* supports two image sensors. The *Snapshot Mode* of operation takes a still image from the image sensor, performs software processing if required and writes to *Global Memory*. The *Snapshot Mode* supports a maximum resolution of 2560×2048 with 24-bit pixels (16 Mega colors) depth.

The working operation of the PGC system is shown in Figure 1(b). During programming-time, the program memory of PGC is initialized. The program memory holds the instructions of CCU/DCU program registers and data transfer. During initialization, the PGC programs the CCU and DCU according to the different use cases (*Video Mode* or *Snapshot Mode*).

3.2 Processing Unit

The PGC supports two types of cores: the *Application Specific Accelerator Core* and the *RISC Core*.

3.2.1 Application Specific Accelerator Core Camera Control Unit (*CCU*) and Display Control Unit (*DCU*) *Application Specific Accelerator Cores* are used in the design to control camera and display units respectively. The *CCU* grabs raw data from Image sensor, processes it and transfers it to the system via on-chip bus. The major function blocks of *CCU* are Camera Interface Front-End, Image Signal Processor, Color processing, Scaling, Compression, and Bus controller. Each *CCU* block has memory mapped internal registers that can be initialized and programmed by the processor core.

The *DCU* is used to control and display image data on LCD panel. The *DCU* supports LCD 16bpp up to 24bpp colors and user defined resolution from VGA to QSXGA. Programming is done by register read/write transactions using a slave interface. The *DCU* consumes 425 registers and 312 LUTs on a V5-Lx110T FPGA device.

The CCU and DCU data rate is given by Formula *Output*$_{data\ rate}$shown in Figure 2. The *Analog Interface*$_{width}$ represents the analog port width of Image sensor and display. Both CCU and DCU support a 32-bit parallel interface (Analog Interface$_{width}$) to communicate with image sensor and display (LCD). When the graphics system architecture is finalized, a top-level software API for the target product is provided. Each hardware block of *DCU* and *CCU* is invoked by the processor using memory mapped register sets which change the operation of the internal hardware architecture.

3.2.2 RISC Core A low power and light weight RISC processor core is used to provide programmability, flexibility and software data processing. The processing core changes the

$$Output_{data\ rate} = Resolution * fps * \frac{Pixel_{Depth}}{Analog\ Interface_{width}}$$

Fig. 2. CCU and DCU Data Rate

features by programming the PGC system using a software API. The API can be used to correct design errors, update the system to a new graphic standard and add more features to the graphics system. The proposed processor core has 16-bit data bus, 16-general purpose registers, custom instruction set, non-pipelined Load/Store access, hardwired control unit, 64KBytes address space 16 interrupts and memory mapped I/O. 1KBytes of memory is allocated for display and camera control units using chip select. On a V5-Lx110T FPGA device, the core uses 481 registers, 1496 LUTs and 4 Brams.

3.3 Memory Unit

The PGC graphics system memory [14] is organized into two sections: the *Local Memory*, and the *Global Memory*.

3.3.1 Local Memory The *Local Memory* is used to support run-time video processing. It also reduces wire delay, data access latency and provide parallel read/write accesses to the processing core. The memory is shared between processor, CCU and DCU. During *Video Mode*, two frames buffers are required: one for processing and other for displaying. Each VGA frame has 900KBytes of size therefore 2MBytes of *Local Memory* are reserved. To save the image in *Snapshot Mode* we use *Global Memory*.

3.3.2 Global Memory The slowest type of memory in the graphics system is *Global Memory* and is accessible by the whole system. The *Global Memory* has SDRAM, SD/SDHC cards, etc. interfaces to read/write data. Even though the PGC system has an efficient way of accessing *Global Memory* that best utilizes the bandwidth, it still has substantially higher latency with respect to the *Local Memory*.

3.4 Bus Unit

Two buses are used in the graphics system which are the *Graphics Bus* and the *System Bus*. The *Graphics Bus* is used for internal communication between the processing units and *Local Memory*. The *System Bus* is used to communicate with external peripherals such as global memories. Both buses can operate in parallel. This section is further divided into three subsections: *Bus Specification*, *Bus Control Unit* and *Bus Interconnect Network*.

3.4.1 Bus Specification It is important to calculate the required data-rate for each *use case* before selecting and configuring the *Bus Unit*. This section is further divided into three subsections: *Graphics Bus Specification*, *System Bus Specification* and *Bus Usage*.

Graphics Bus Specification: The clock of the camera and display is directly synchronized with the output data hence define the bandwidth of *Graphics Bus*. The actual theoretical data rate of the *Graphics Bus* (GBB) is the total bandwidth of master sources (shown in Figure 3(a)). For example, during *Video Mode* (without-processing) the PGC reads streaming data from CCU and writes directly to DCU. For *Video Mode* with-processing, the PGC takes video frames from CCU, writes them to *Local Memory* for processing and then transfers the processed frames to DCU. In this case the PGC operates CCU and DCU in parallel, therefore the bandwidth of the *Graphics Bus* is the sum of CCU and DCU data-rates. For dual camera, the PGC takes two video frames and transfers them to CCU. The required *Graphics Bus* bandwidth (shown in Figure 3(a)) with single camera without processing and with processing is given by the formula GBB_{SC} and GBB_{SCP} respectively. Figure 3(a) also presents the

$$GBB = \sum_{n=1}^{max} Master\ Source\ Bandwidth_n$$

$$GBB_{SC} = Image\ Sensor[Pixel_{Depth} * Resolution_{max} * fps_{max}]$$

$$GBB_{SC} = Image\ Sensor[24 * 640 \times 480 * 150]$$

$$\simeq 1.05Gb/s \simeq 132MB/s$$

$$GBB_{SCP} \simeq 264MB/s$$

$$GBB_{DC} = Image\ Sensor_a + Image\ Sensor_b$$

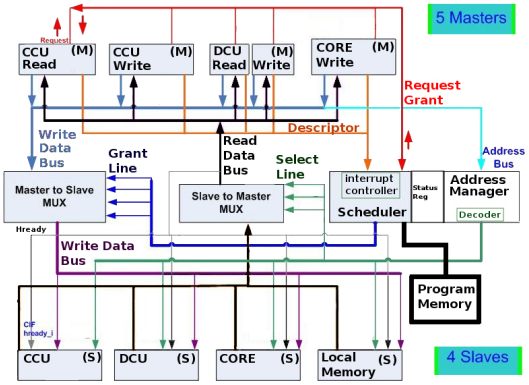
$$\simeq 2.1Gb/s \simeq 264MB/s$$

$$SBB_{SN} = Image\ Sensor[24 * 2560 \times 2048 * 1]$$

$$\simeq 120Mb/s \simeq 15MB/s$$

$$Percentage\ of\ Bus_{usage} = \frac{Source_{Bandwidth}}{MasterBus_{freq}} * 100$$

(a)



(b)

Fig. 3. (a) Graphics Bus Required Bandwidth (b) PGC Graphics Bus Unit

bandwidth of dual camera GBB_{DC}. The *Local Memory* provides high bandwidth and has 2 cycles of latency for an individual transfer. The *Bus Latency* contains the on-chip/off-chip memories read/write and on-chip bus delays. The *Graphics Bus* manages multiple read/write access transactions in a single transfer and pipelines the multiple stream, thus reducing the overhead of $Local\ Memory_{Latency}$ and improving the bus performance. After calculating bus bandwidth and considering the different use cases we selected a bus with 100 Mhz of clock speed and 32 bit-width.

System Bus Specification: The *System Bus* manages data transfers during *Snapshot Mode*. The PGC reads data from image sensor and writes it to *Global Memory*. The bandwidth requirements of *System Bus* for *Snapshot Mode* are given by the Formula (SBB_{SN}) (shown in Figure 3(a)).

Bus Usage: The PGC *Graphics Bus* has 400 MB/s of bandwidth, so it takes 10 nsec to transfer 1 pixel (32bit). For example, the graphics bandwidth for video of 30 *fps* (without-processing) is 9 Mega pixels per second. This means each pixel takes 111 nsec and occupies *Graphics Bus* for 9% of its time, given by the formula $Percentage\ of\ Bus_{usage}$ (shown in Figure 3(a)). For *Video Mode* with-processing, the *Graphics Bus* takes 111 nsec to transfer one pixel from CCU to *Local Memory*, and it takes the same time to transfer it to DCU. Similarly *Video Mode* needs 18 Mega pixels of bus bandwidth, that takes 56 nsec to transfer a pixel between image and display accelerators. The display camera interface for *Video Mode* utilizes graphics for approximately 14% of total bus time. The *Snapshot Mode* requires bus bandwidth of 5 Mega pixels to transfer one image (QSXGA resolution without-processing) from CCU to the *Global Memory*.

3.4.2 Bus Control unit The PGC control unit uses *program memory*, *scheduler* and *address manager*, to manage the processing units and memory units. The *program memory* holds *descriptors* that define the data movement between CCU/DCU, processor core and memory unit. The *descriptors* allow the programmer to describe the shape of memory patterns and its location in memory. A single *descriptor* is represented by parameters: command, source address, destination address, priority, stream and stride. The command specifies data

transfers between single/multiple masters and single/multiple slaves. The address parameters also specify the master and slave cores. The priority describes the selection and execution criteria of data transfer by a master core. It also defines the order in which memory accesses are entitled to be processed. Stream defines the number of pixels to be transferred. Stride indicates the distance between two consecutive memory addresses of a stream. PGC manages a complex data transfer protocol using single or multiple *descriptors*. Each *descriptor* transfers a strided burst, by using multiple *descriptors* the PGC transfers more complex data. C/C++ function calls are provided to define a complex pattern in software.

The PGC bus *scheduler* along with *address manager* (shown in Figure 3(b)) arrange requests coming from single or multi-bus masters and arbitrate master processing units. A *bus master* provides address and control information to initiate read and write operations. A *bus slave* responds to a transfer that is initiated by the masters core. The *address manager* holds the address and control information of *bus slaves*. The *scheduler*'s *interrupt controller* reads requests from master cores and routes them to the slave. The *address manager*'s *decoder* determines for which slave a transfer is destined for. The PGC bus holds two types of status registers: the source status and the slave status registers. The status registers indicate the state of each master and slave, such as request, ready, busy and grant. The *scheduler* and *address manager* administer the status register of master and slave cores respectively. The PGC *scheduler* emphasizes on priority and incoming requests of the master core. At compile-time a number of priority levels are configured for each data transfer. At run-time the *scheduler* picks a master core to transfer data, only if it is ready to run and there are no higher priority data patterns that are ready. If same priorities are assigned for more than one data pattern, the PGC scheduler executes them in first-in first-out (FIFO) order.

At run-time, a master core generates a request, the *interrupt controller* reads the request and updates its status registers. The *scheduler* reads data transfer information of the master and slave cores from program memory and transfer slave core information to *address manager*. The PGC *address manager* decodes the address of each transfer and provides a select signal for the slave that is involved in the transfer and provides a control signal to the multiplexers. A single *master-to-slave* multiplexer (*MUX*) is controlled by the *scheduler*. The *master-to-slave MUX* multiplexes the write data bus and allocate data bus for a single master after getting the response signal from the *slave-to-master MUX*. A *slave-to-master MUX* multiplexes the read data bus and response signals from the slaves to the master. Multiple *master-to-slave* and *slave-to-master* multiplexers can be added to implement a multi-layer *Bus Unit*. The PGC *Bus Unit* can be programmed up to eight layer bus which requires eight pairs of *master-to-slave* and *slave-to-master* multiplexers.

3.4.3 Bus Interconnect To connect the graphics components together, a bus interconnection is described (shown in Figure 3(b)). We select a double layer *Bus Interconnect* (*System Bus* and *Graphics Bus*) for the design due to its design simplicity and low power consumption. Each layer is controlled by a pair of *master-to-slave* and *slave-to-master* multiplexers. The PGC *scheduler* and *address manager* control the pairs of multiplexers. The *PGC Bus Interconnects* can be configured according to the requirements of hardware accelerator, master and slave ports. The *System Bus* has a simple design that uses a single master and slave port. The bus is used to read/write high resolution image to global memory.

The *Graphics Bus* is employed to provide high speed link between the CCU, DCU, processor and *Local Memory* components. Current PGC *Graphics Bus* has 5 Masters and

4 Slaves therefore the *Bus Unit* is configured accordingly. The proposed *Bus Unit* provides standard communication protocol and implements the features required for high-performance.

4 Experimental Framework

In this section, we describe and evaluate the PGC based graphics system. In order to evaluate the performance of the PGC system, the results are compared with a generic graphics system managed by the MicroBlaze processor. The Xilinx Integrated Software Environment and Xilinx Platform Studio are used to design the graphic systems. The power analysis is done by Xilinx Power Estimator (XPE). A Xilinx ML505 [15] development board is used to test the systems. For the implementation of graphics system the THDB-D5M Camera and the TRDB-LTM LCD Touch Panel by Terasic have been chosen. This section is divided into two subsections: the *MicroBlaze based Graphics System* and the *PGC based Graphics System*.

4.1 MicroBlaze based Graphics System

The FPGA based MicroBlaze system is proposed (Figure 4 (a)) to operate graphics system. The design (without CCU & DCU) uses 9547 flip-flops, 11643 LUTs and 51 BRAMs in a Xilinx V5-Lx110T device. The system architecture is further divided into the *Processor Core*, the *Shared Peripheral Unit*, and the *Bus Unit*.

4.1.1 The Processor Unit The MicroBlaze processor [16] has Harvard memory architecture where instruction and data accesses have separate 32-bit address spaces. Two MicroBlaze cores are used in the graphics system which are the Master core and the Graphics Core. The Master core is used to program, schedule and manage the system components. The camera and display hardware scheduling and data memory management are controlled by Graphics processor. Both cores use local memory Bus (LMB) [17] to link with local-memory (FPGA BRAM) that offers single clock cycle access to the local BRAM.

4.1.2 The Bus Unit In the design, a Processor Local Bus (PLB) [18] provides connection between peripheral components and microprocessors. The PLB has 32 bit-width and is connected to a bus control unit, a watchdog timer, separate address read/write data path units, and an optional DCR (Device Control Register) slave interface that provides access to a bus error status registers. Bus is configured for single master (MicroBlaze) and multi slaves. The PLB provides maximum of 400 MBytes of bandwidth while operating at 100Mhz and 32-bit width, with byte enables to write byte and half-word data.

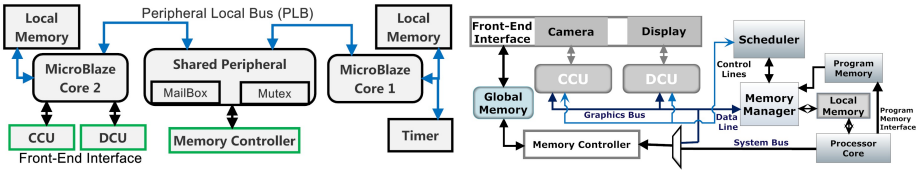


Fig. 4. Graphics System: (a) MicroBlaze Core System (b) PGC System

4.1.3 The Shared Peripheral Unit The Mutex core is used to provide synchronization when accessing shared resources. The core has a configurable number of mutexes and can lock the scheme. The Mailbox core is used to pass messages between processor cores in FIFO. The mailbox core offers an interrupt line if a core wants to indicate the presence of data.

4.2 PGC based Graphics System

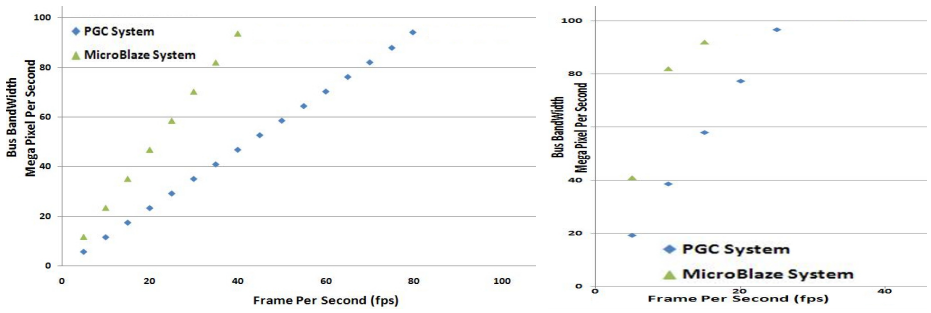
The PGC based graphics system is shown in Figure 4(b) having components similar to the MicroBlaze based graphic system. The implementation details of PGC based graphics system are addressed in Section 3. The main difference between PGC and MicroBlaze based systemis that PGC system takes instructions during initialize-time and at run-time it manages and schedules data transfer without the support of the processor. The processor core and *System Bus* remain free for the use cases which do not involve processing. The design (without CCU and DCU) uses 5547 flip-flops, 6643 LUTs and 35 BRAMs in a Xilinx V5-Lx110T device.

5 Results and Discussion

This section analyzes the results of different experiments conducted on the different graphic systems. The experiments are classified into four subsections: *Bus Performance*, *Snapshot Mode Performance*, *Applications Performance*, and *Area & Power*.

5.1 Bus Performance

To measure the bus performance, the graphic systems are executed on *Video Mode* (without-processing) having fixed resolution (640×480) and variable frame rate (*frame per second - fps*). The image sensor is programmed to transfer variable frames (*fps*) and each frame has VGA quality. Inside DCU we integrated a controller that detects video frame rate, the speed at which frames are coming. A hardware timer is added to the on-chip bus controller that measures clocks used to transfer frames between master to slave peripherals. This section discusses results for *Single-Camera Bus Bandwidth* and *Multi-Camera Bus Bandwidth*.



(a) (b)

Fig. 5. Bandwidth Required For Different Frame Rate: (a) Display Camera Video Transfer Time (b) Dual Camera System

5.1.1 Single-Camera Bus Bandwidth In this section, we compare the bus performance of graphic systems while using single image and display units. Figure 5 (a) shows the on-chip data bus transfer speed of PGC and Microblaze systems for different video frame rates. A single THDB-D5M image sensor is used. It can operate up to 150 *fps* with VGA resolution. The X-axis presents video of different *fps*. The Y-axis shows measured bandwidth for different videos frame rates. To measure the bandwidth we calculate the time to transfer video from CCU to DCU. Theoretically the PLB and the graphic bus support video of VGA quality more than 100 *fps*. In practice there are on-chip bus arbitration and request grant time delays. By using the PGC system, the results show that the system manages video for higher *fps*. While the MicroBlaze based graphic system supports video up to 40 *fps*, with higher *fps* the video starts flicking. The system uses a separate processor core that manages the data movement of CCU and DCU. The PGC allows graphics system to operate *Video Mode* up to 85 *fps*. The PGC resides in on-chip bus unit and has direct interconnection with CCU and DCU. The PGC *control unit* controls the CCU and DCU without the intervention of processor core which reduces the master/slave request/grant time.

5.1.2 Multi-Camera Bus Bandwidth A multi-camera graphics system can be used for 3D-graphics using geometric transformation and projection plane [19]. In this section, two THDB-D5M image sensors are used that generate two separate, simultaneous video streams and apply *Alpha blending* application that evaluate the performance of system. Each camera is operating at VGA color resolution. The video of dual image sensors is combined into a single stream, processed by graphics core and then displayed. The key issue of the dual-camera system is receiving the images synchronously, in the right format and on the right bus. The graphic system sends the configuration data to both image sensors and ensure that they are properly configured and synchronized. Once both sensors are set up and synchronized, both sensors begin to transmit image data. The graphic system looks for the appropriate control characters so it recognizes the start of the frame and start of line for each sensor. The PGC system performs it by looking for a control character and sequence of sensors commands. *Alpha blending* is applied to give a translucent effect to the incoming video stream. The application blends the color value of the consecutive pixels of image sensors of the same position. This blending is done according to the alpha value associated with the pixel. The alpha value represents the capacity of the given pixel. After blending, the result color value is updated to the frame buffer of the DCU. Results show (Figure 5(b)) that PGC system handles dual camera system and support system up to 30 fps. The MicroBlaze based dual-camera graphics system supports videos only up to 15 *fps*. The PGC on-chip *scheduler* and *decoder* update multi-camera information in status register. This allows both cameras to synchronize without using extra clocks.

5.2 Snapshot Mode Performance

In this section the performance is measured by executing PGC and MicroBlaze systems in *Snapshot Mode*. During *Snapshot Mode* the system reads one still image of QSXGA resolution from CCU using *Graphics Bus* and writes it to *Global Memory* using *System Bus*. The MicroBlaze based system and PGC take 22.17M and 7.07M clocks respectively to transfer an image. The *Snapshot Mode* results show that the PGC system achieves 3.1x of speed compared to MicroBlaze based system. The PGC directly controls the *Graphics Bus*, *System Bus*, *CCU* and *Global Memory*, therefore it takes less clocks to read data from *CCU*

to synchronize different units, transfer data from *Graphics Bus* to *System Bus*, and write data to *Global Memory*. The MicroBlaze based system uses a separate bus controller that controls bus system and a DMA controller that transfers data from *CCU* to *Global Memory*.

5.3 Applications Performance

In this section we execute some application kernels that perform image processing. The image is saved in *Global Memory* (SDRAM), the processor core reads the 4KBytes image, performs computation and then writes it back to *Global Memory*. To achieve low power the application kernels are executed by the 16-bit core on PGC system. Alternatively, a 32-bit MicroBlaze core is also used with PGC system to get higher performance. Figure 6 shows time (clock cycles) to process application kernels. The X and Y axis represent application kernels and number of clock cycles, respectively. The Y-axis has logarithmic scale. Each bar represents the application kernel's execution time with 16/32 bit cores and memory access time. By using the PGC system with 16-bit and 32-bit cores, the results show that thresholding (Thresh) applications achieve 4.6x and 4.7x of speedup respectively over the MicroBlaze graphics system. This application kernel requires single pixel element and very few operations, therefore it achieves almost the same performance on 16-bit and 32-bit cores. The FIR application has streaming data access pattern and perform multiplication and addition. The PGC 16-bit and 32-bit cores achieve 3.4x and 4.7x of speedup respectively. The FFT application kernel reads a 1D block of data, perform complex computation and writes it back to *Global Memory*. This application achieves 4.4x and 4.8 of speedup. The Laplacian application processes over 2D block of data and achieve 5.2x and 7.4x of speedup. The PGC places access patterns on *program memory* at program-time and are programmed in such a way that few operations are required for generating addresses at run-time. The MicroBlaze based system uses multiple load/store or DMA calls to access complex patterns. The speedups are possible because PGC is able to manage data transfers with a single *descriptor*. At run-time, PGC takes *descriptor* from *program memory* and manages them, whereas the baseline system is dependent on the processor core that feeds data transfer instructions. The stand-alone working operation of PGC removes the overhead of processor/memory system request/grant delay.

5.4 Power

In comparison with on-chip power in a Xilinx V5-Lx110T device, the Microblaze based system dissipates 3.45 watts and the PGC system 2.7 watts. Results show that PGC system consumes 30% fewer slices than the Microblaze system. While comparing on-chip power of

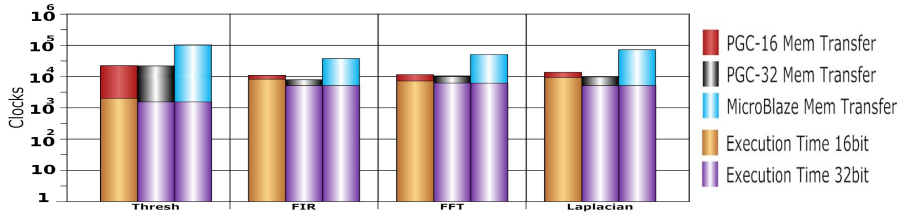


Fig. 6. Application Performance

Microblaze graphics system with the PGC, results show that PGC system consumes 22% of less on-chip power.

6 Conclusion

With the increase of image resolution the graphics system demands a low power, low cost and high performance architecture. In this paper we have suggested a Programmable Graphics Controller (PGC) for low cost and low power graphics system. The system takes high resolution images and supports video at higher frame rate without the support of a processor. The PGC system provides strided, scatter/gather and tiled access pattern that eliminates the overhead of arranging and gathering address/data. In the future, we plan to execute some complex and high performance image processing applications which include image recognition, image transform and image compression.

7 Acknowledgments

We thankfully acknowledge the support Microsoft Research though the BSC-Microsoft Research Centre, the European Commission through the HiPEAC-3 Network of Excellence, the Spanish Ministry of Education (TIN2007-60625, and CSD2007-00050), the Generalitat de Catalunya (2009-SGR-980) and Unal Center of Education Research and Development.

References

1. Gordon E Moore et al. Cramming more components onto integrated circuits. 1965.
2. Lecturer Brian Towles et al. Memory systems and memory latency. 2000 Citeseer.
3. Anthony Rowe et al. Cmucam3: an open programmable embedded vision sensor. In *International Conferences on Intelligent Robots and Systems*, 2007.
4. M Petouris et al. An fpga-based digital camera system controlled from an lcd touch panel. In *Signals, Circuits and Systems, 2009. ISSCS 2009. International Symposium on*.
5. Chris Murphy et al. Low-cost stereo vision on an fpga. In *Field-Programmable Custom Computing Machines, 2007. FCCM 2007. 15th Annual IEEE Symposium*.
6. Matthew A Lewis et al. A multi-camera system for bioluminescence tomography in preclinical oncology research. *Diagnostics, Multidisciplinary Digital Publishing Institute*, 2013.
7. Yu Shi et al. An fpga-based smart camera for gesture recognition in hci applications. In *Computer Vision–ACCV 2007*.
8. AMBA 4 AXI. `http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.ihi0022e/index.html`, 2013.
9. T. Hussain and others. Recongurable Memory Controller with Programmable Pattern Support. *HiPEAC WRC*, Jan, 2011.
10. T. Hussain and others. PPMC: A Programmable Pattern based Memory Controller. In *ARC 2012*.
11. T. Hussain and others. PPMC : Hardware Scheduling and Memory Management support for Multi Hardware Accelerators. In *FPL 2012*.
12. T. Hussain and others. APMC: Advanced Pattern based Memory Controller. In *FPGA 2014*.
13. IBM CoreConnect. PLB Crossbar Arbiter Core. 2001.
14. T. Hussain and others. Implementation of a Reverse Time Migration Kernel using the HCE High Level Synthesis Tool. *FPT 2012, The International Conference*.
15. Xilinx University Program XUPV5-LX110T Development System. `http://www.xilinx.com/univ/xupv5-lx110t.htm`.
16. Embedded Development Kit EDK 10.1i. *MicroBlaze Processor Reference Guide*.
17. Xilinx LogiCORE IP. *Local Memory Bus (LMB)*, December, 2009.
18. Embedded Development KitEDK 10.1i. *MicroBlaze Processor Reference Guide*.
19. Richard Hartley et al. *Multiple view geometry in computer vision*, volume 2. Cambridge Univ Press, 2000.