Design and Implementation of a DSP based Inkjet Printer Motion Control System for Dynamic Print Mode Control

A.V. Deshpande[a], M. Kamasak[b], K.L. Thoon[b]
G.T.-C. Chiu[a], C. Bouman[b], and J. Allebach[b]
[a] School of Mechanical Engineering,
[b] School of Electrical and Computer Engineering,
Purdue University, West Lafayette, Indiana

S. Fedigan, D. Schafer, and C. Cole
Texas Instruments, Dallas, Texas

Abstract

Current inkjet printing systems are based on a RISC+ASIC architecture for image processing and printer control. Time critical control function such as print masking is implemented on an ASIC and computationally intensive tasks such as image processing and motion control are executed on a RISC processor. This paper considers the development of a DSP based media handling and cartridge motion control system for an inkjet printer, in which a single high performance DSP is desired to execute the printer control as well as image processing tasks and print with the novel inkjet printing approach of *Dynamic Print Mode Control* (DPMC). A motion control architecture is implemented to maximize the use of DSP processing power and system memory while taking into account the latencies of motion controllers in printing a swath. The architecture utilizes various real-time services of DSP/BIOS to effectively manage DSP and printer engine while increasing printer throughput and it can be easily extended to include image processing activities on the DSP. Experimental results show the efficacy of the proposed motion control architecture and the control scheme for cartridge motion control.

Introduction

In the fastest growing embedded computing market, Digital Signal Processors (DSP) have made their presence felt in a range of applications from power hungry, hardware intensive devices like printers, scanners and networking switches to power efficient, extremely lightweight handheld devices such as cell phones, internet audio players, cameras, video games. A DSP is a special-purpose microcontroller, whose architecture and instruction sets have been optimized specifically for signal processing algorithms.[1,2] Since their inception in early 1980s, DSPs have gone through a lot of architectural modifications to suit the changing market demands for better price-performance ratio with the main focus on boosting efficiency and performance in signal processing applications. Also, DSP architects have begun to experiment with the new architectures such as Harvard, VLIW architecture from the earlier highly specialized, compiler unfriendly architecture to the general-purpose processor designs,[2] so that DSPs will be better compiler targets, code compatible with their predecessors.

Current inkjet printing systems are based on a RISC+ASIC structure for image processing and printer control. Time critical control functions such as open-loop media[3] or cartridge motion control[4] print masking[5] are implemented on an ASIC and computationally intensive tasks such as image processing and closed loop printer engine control are executed on a RISC processor.[6] This structure offloads some of the computational burden on the main processor. Also, hardware implementation of time critical and repetitive functions reduces firmware size, keeping the manufacturing and development costs to a minimum. However, this structure also has a long turn around time and it is less flexible to late cycle modification due to firmware revisions.

High performance DSPs coupled with code-efficient compilers and guaranteed real-time execution through real-time operating systems lead to the possibility of executing imaging, printer engine control as well as time critical functions on a single DSP. Hence with DSP implementation of the whole inkjet system, design modifications can be implemented as a firmware revision, which considerably reduces development time. It also gives the flexibility to explore novel printing approaches, which are hereto been impossible to implement with the ASIC implementation. For example, in the case of nozzle failures, an active feedback from inkjet cartridge allows changing print mask and low-level control function dynamically. An aging printer is also characterized by parameter variations. Closed

loop system monitoring and control with the DSP implementation results in a robust platform.

This paper considers the development of an inkjet system, where a single high performance DSP is desired to perform printer control as well as image processing tasks and print with the novel inkjet printing approach of *Dynamic Print Mode Control* (DPMC). With the capability of dynamically changing media and cartridge motion control parameters, a motion-control architecture for the inkjet system is implemented which can programmatically print either uni-directionally or bi-directionally or combination of both depending upon the specifications of image processing output. The architecture utilizes various real-time services of the real-time operating system DSP/BIOS to effectively manage DSP processing power, system memory and printer engine while increasing printer throughput and it can be easily extended to include image processing activities on the DSP. Experimental results show the efficacy of the proposed motion control architecture.

Dynamic Print Mode Control

Typical commercially available inkjet printers print with a variety of user-selectable print modes such as "draft", "normal", "high" and "maximum". These print modes control following media, cartridge and image processing attributes: number of passes, print direction, print-head (cartridge) speed, print mask and halftoning technique. Print quality of various page contents such as continuous tone image, graphics, line art and text is directly affected by the specific print mode chosen. Tables 1 and 2 show the measured values of some of these attributes for the black and color cartridge from an off-the-shelf printer.

Given a print mode, all the abovementioned attributes get statically defined. Hence, we can observe the tradeoff between print quality and printing speed in selecting the print mode. Computationally simple print modes like "draft" and "normal" modes result in higher printing speed, but reduce the quality of continuous tone images and graphics while computationally expensive modes like "high" and "maximum" modes increase print quality but reduce print speeds. Also, much of the computational power is wasted in printing line art, text in high quality modes, for which low quality "draft" or "normal" mode printing is sufficient.

Dynamic print mode control (DPMC) helps to optimize this tradeoff between the print quality and print speed. In DPMC, the page is first segmented into different regions containing continuous tone image, graphics, line art and text. Then, aforementioned printing attributes are selected dynamically for each of these regions. For example, for page regions containing images and graphics, parameters pertaining to "high" or "maximum" modes can be selected, while line art and text can be printed with parameters of "draft" or "normal" modes. This method of printing results in a much better tradeoff between print quality and print speed rather than using a single print mode for the entire page.

Table 1. Measurements of print attributes for the color cartridge (Nozzle height is 192 nozzles, 64 nozzles each for cyan, magenta and yellow)

Print mode / Attributes	Draft Mode	Normal Mode	High Mode	Max. Mode
No. of passes (nozzle height)	1 64 nz.	2 32 nz.	4 16 nz.	8 8 nz.
Printing Direction	bi-dir ctional	bi-dir ctional	bi-dir ctional	bi-dir ctional
Cartridge speed (ips)	40	40	30	30

Table 2. Measurements of print attributes for the black cartridge (Nozzle height is 208 nozzles)

Print Mode / Attributes	Draft Mode	Normal Mode	High Mode	Maxi. Mode
No. of passes (nozzle height)	1 208 nz.	2 104 nz.	4 52 nz.	8 26 nz.
Printing Direction	bi-dir ectional	bi-dir ectional	bi-dir ectional	bi-dir ectional
Cartridge speed (ips)	30	20	20	20

Experimental Inkjet Printer System

The experimental inkjet printer system described herein uses Lexmark Z-52 printer engine and TIger 2.0 RIP reference system containing TMS320C6211 DSP from Texas Instruments' C6x roadmap. C6211 DSP is based upon VelociTI – modified VLIW – architecture where 8 instructions can be executed in parallel at a clock speed of 150 MHz resulting in the MIPS rating of 1200. It consists of direct mapped 2-level 64 KB cache and it supports pre-emptive multitasking real-time operating system DSP/BIOS to facilitate real-time execution.

There are four main components of the experimental inkjet system: inkjet cartridges, media handling mechanism, cartridge transport mechanism (carriage) and DSP interface to the abovementioned components. Media handling mechanism and cartridge transport mechanism – from Lexmark Z52 inkjet printer – constitute the printer engine. Engine interface connects printer engine with the C6211 DSP motherboard, while pen driver interface and video interface connects the inkjet cartridges with C6211 DSP motherboard.

A 7.5^{o} two-phase bipolar stepper motor actuates the media handling mechanism in an open loop to carry out pick, advance and eject sequences. One half step (3.75^{o} rotation of stepper motor) corresponds to media advancement of 1/600 inch, which is equal to the vertical separation between two successive nozzles in black and

color inkjet cartridge. Therefore, as print modes change from draft mode to maximum mode, media advancement decreases progressively by a factor of two for either black or color cartridge. Stepper motor runs at a sampling frequency of 5 kHz in hardware interrupt service routine (ISR), mapped to the external interrupt EXT_INT7 of the DSP. On-chip Timer 1 is the hardware source for the interrupt EXT_INT7.

Cartridge transport mechanism helps black and color inkjet cartridges disperse ink droplets on the media in a direction orthogonal to the media movement. A DC motor controls to and fro movement of carriage on the printer rail, while position of carriage on the rail is sensed through a linear incremental encoder having a resolution of 1/600 inch/count after quadrature decoding. Carriage motion control loop runs at a sampling frequency of 5 kHz in hardware interrupt service routine (ISR), mapped to the external interrupt EXT_INT6 of the DSP. *Period* register in Engine interface interrupts the DSP on EXT_INT6 at the set sample rate.

Pen driver interface generates programmable nozzle firing address signals and data signals along with some control signals for each cartridge. Video interface module in TIger2.0 system is used to transfer the nozzle-mapped data from system SDRAM to the inkjet cartridge. Video interface converts the generic unpacked nozzle-mapped data into printer engine specific data bus signals, when sending it to the cartridges. Detailed experimental inkjet printer system description can be found in Ref. [7].

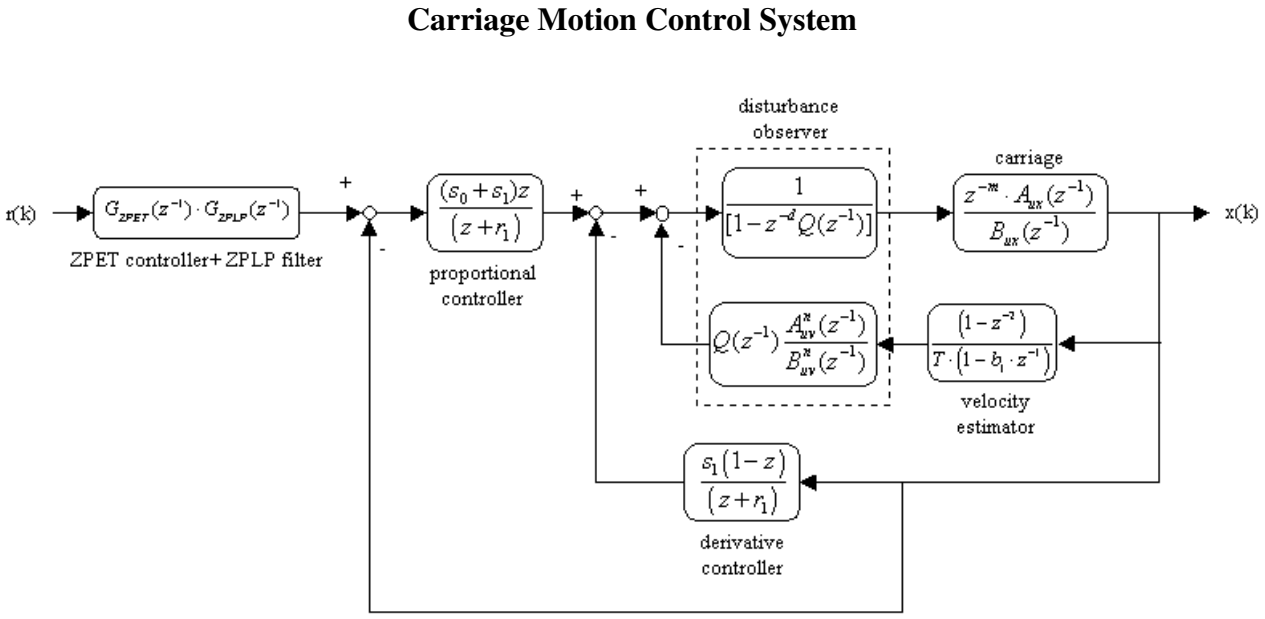

Figure 1. Schematic block diagram of carriage motion controller

Carriage transport mechanism is affected by a variety of disturbances such as stiction, coulomb friction, sensor noise and ignored dynamics of the belt transmitting the torque from the DC motor to the carriage. A computationally simple disturbance observer was designed to compensate for these disturbances and combined with a two-degree-of-freedom (TDOF) controller as shown in figure 1. In the TDOF controller structure, a feedback controller (PD controller) was designed by pole placement approach to guarantee adequate stability and dynamic lag of the closed loop system was compensated with a ZPET based feed-forward controller. To prevent the amplification of high frequency components near the nyquist frequency due to high frequency gains of ZPET controller, a zero phase low pass (ZPLP) filter was convolved with the ZPET controller. To minimize the computation time, controller was implemented in fixed-point, which entailed its reconfiguration to lessen the quantization noise in the system. Carriage motion controller design details and carriage position trajectory planning for DPMC can be found in Ref. [7].

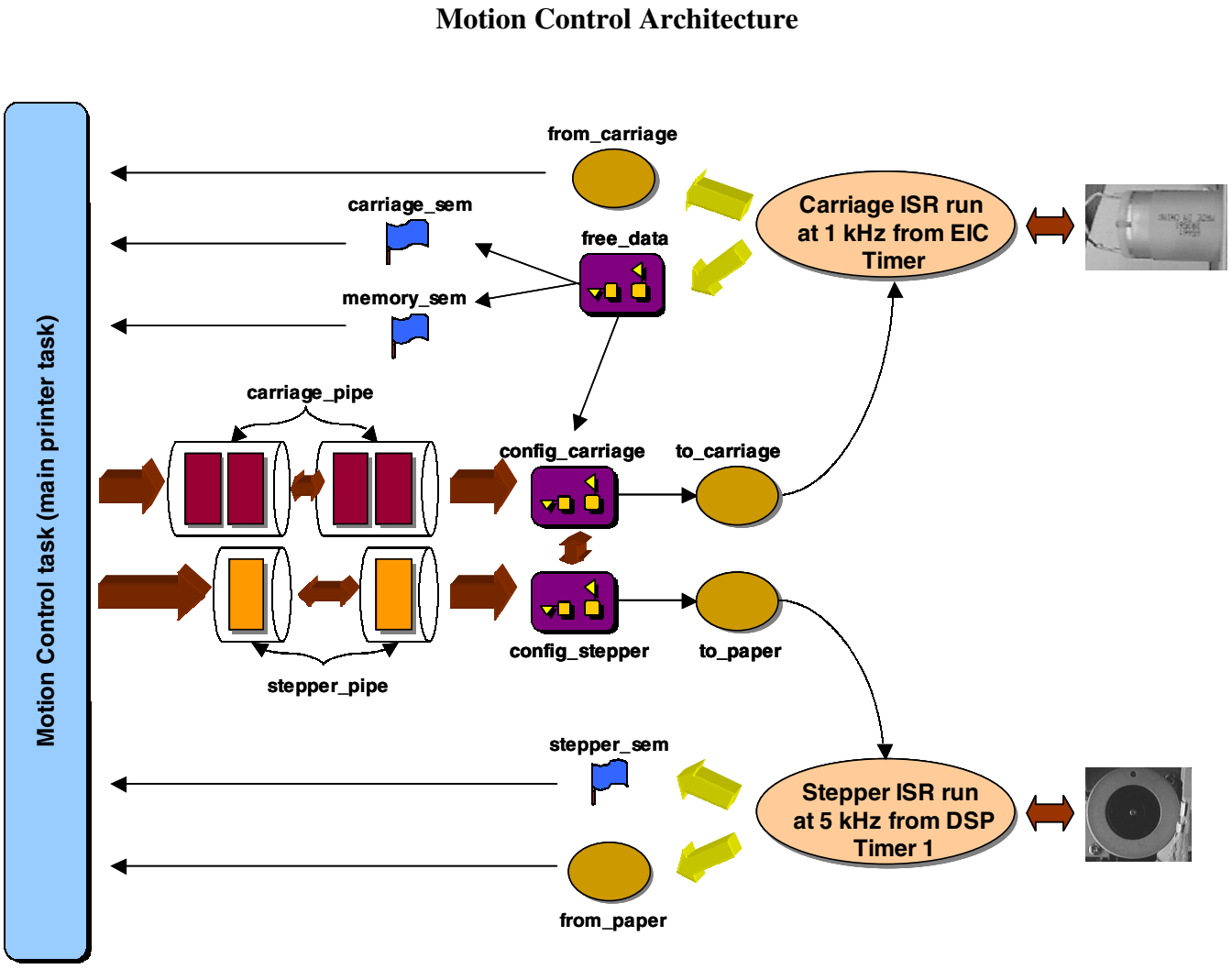

Figure 2. Motion control architecture for Lexmark Z-52 printer

The main objective in the motion control architecture is to maximize the use of CPU processing power and system memory while taking into account the latencies of motion controllers in printing a swath. Also, in the high-end inkjet and multi-function devices, the DSP performs computationally intensive tasks such as PDL conversion, raster image processing in addition to the printer engine control. Since, the DSP/BIOS kernel is based upon foreground/background scheduling,[8] a motion control framework can be designed where the DSP can execute other soft real-time tasks such as image processing for next swath, PDL conversion in the background when the printer engine is printing the current swath in foreground. An architecture such as the one in the figure 2 helps to realize such a goal.

Hardware ISRs: In the multi-rate inkjet system, cartridge motion controller periodically runs at 1 kHz from Engine interface *period* register and stepper motor controller at 5 kHz from C6211 DSP timer 1. If these control loops cannot meet these sampling time requirements, stability of carriage motion will be compromised which will directly affect print quality and also, paper advancement will not be accurate. Hence, these control loops are executed in hardware ISR triggered by the respective timers and the processing in hardware ISR is kept to a minimum to help meeting hard real-time deadlines.

Motion Control Task: Computations such as carriage position trajectory generation for the next swath from imaging output (page structure), calculating paper advancement for next swath have less critical deadlines and can be deferred in favor of the hardware ISRs. Therefore, these computations are carried out in the background motion control task. In order to print a given swath, motion control task first reads the swath information from the downloaded page structure, then generates carriage position

trajectory (in accordance with the principle of DPMC) from the swath information and calculates the media advancement necessary for printing the swath. Subsequently to instruct the carriage to print the corresponding swath, the task puts the required position trajectory information and swath information in the form of a job in the carriage_pipe and the required media advancement information as a job in the stepper_pipe for the stepper motor.

Data Pipes: The advantage of putting the necessary motion control information in a data pipe rather than directly transferring to the respective ISR is that the motion control task can go ahead to process the next swath and to put the jobs for the corresponding swath while the carriage ISR is printing the current swath and hence, DSP processing power is better utilized while increasing the overall printer throughput. Since data pipe objects in DSP/BIOS are flexible in terms of the contents of the buffers (or frames) being filled up, if the motion control information has an array of data (e.g. position trajectory), only array pointers are transferred avoiding copy of contents. Also, when the data pipe buffers are being read or written, only pointers are passed to the reader or writer. Hence, there is no restriction on the size and contents of the buffers and transfer time between writer and reader is constant, a must for a real-time system.

Since, the data flow from the task to the respective ISRs is non-continuous, dependent upon the processing time of the task, a data synchronization and notification mechanism is needed, which avoids the polling by the reader (to receive the data) or writer (to put the data in pipe). DSP/BIOS utilizes data notification functions[8] to synchronize the writer (motion control task) and reader (hardware ISRs) with the underlying data pipes. Once the pipe frame is filled up by the writer or read by the reader, these notification functions (notifyReader or notifyWriter respectively) are triggered to notify the counterpart that a frame is available or free. In the inkjet system, DSP processing time will be optimally utilized if the hardware ISRs are enabled and run only, when needed by the background task. In this setting, inadvertent context switching is avoided by posting a software interrupt from the notifyReader function (i.e. from the writer side), once the task puts the frame in the data pipe.

Software ISRs: Software interrupts are instantiated in software (not associated with any physical device) and are less time-critical than the hardware interrupts. As shown in the figure 2 to synchronize motion control task and hardware ISRs, config_carriage software interrupt is posted when a carriage_pipe frame buffer is filled with a job for carriage and config_stepper software interrupt is triggered once, a stepper_pipe frame buffer is written to by the motion control task. Once posted, the software interrupt copies data from the pipe frame to a global structure, to_carriage for the carriage mechanism and to_paper for the media handling mechanism. Also, since copying data to global structures is infrequent, posting a software interrupt for copying operation is more efficient than copying the data in hardware ISR itself.

Another software interrupt, free_data is used to free the memory used temporarily for storing the nozzle-mapped data for the current swath after printing, corresponding carriage position trajectory and scan start and scan stop information (in to_carriage structure). It is important to free the memory in a software ISR than in the carriage hardware ISR because memory allocation and freeing are non-deterministic,[8] since DSP/BIOS maintains a linked list of free memory blocks and allocation or freeing up of memory requires traversing this linked list. If done in the carriage ISR, carriage motion will be severely affected.

Semaphores: Completion of important events such as (a) carriage has traversed the entire command position trajectory (b) stepper motor has put the media at the required swath start position and (c) memory used for printing the current swath is freed, is indicated with the counting semaphores carriage_sem, stepper_sem and memory_sem respectively. Semaphores are inter-task synchronization and communication primitives and they are needed to synchronize the motion control task with hardware ISRs, since the task might have to suspend its execution until some events are completed. For example, if there is insufficient system memory available for further processing or no frame in either data pipe is available for writing the motion control information. Posting a semaphore releases the task, waiting for the corresponding semaphore to be posted.

Data Communication: Carriage hardware ISR *reads* to_carriage structure to execute carriage position trajectory and print the corresponding swath, while carriage is motion. to_carriage structure contains the pointer to carriage command position trajectory located in the heap section, its length and the direction of printing. The swath information consists of number of scans to be printed and start and stop position of each scan in the swath being printed. This information is used to configure pen driver interface registers on-the-fly for printing each scan when carriage is in motion. Motion control task knows about the status of carriage through from_carriage structure, which carriage ISR *writes* to. Actual carriage position, corresponding command carriage position, current index in command position trajectory, present carriage state along with the number of scans printed in the current swath are indicated in from_carriage structure.

Similarly, stepper hardware ISR *reads* to_paper structure parameters for media advancement needed to print a swath. to_paper structure consists of pointer to the step table or excitation pattern for each phase of the stepper motor and its length, pointer to the time table used to generate precise stepping velocity profile and its length. Number of times to repeat the given time table to advance the media and command paper state is also included in the to_paper structure. Stepper hardware ISR reflects its execution state by *writing* to from_paper structure. Present

step table and time table index, the number of time table repeat counts remaining, present paper state and number of half steps moved on the page are stored in the from_paper structure.

Synchronization Between Carriage and Stepper Motor: Synchronization between carriage and stepper motor is necessary when media is still being advanced at the moment carriage is ready to print the swath. In all other cases, where stepper motor is performing paper pick or eject sequence and carriage is being retracted to properly position it before printing the next swath or when it is being docked to establish its home position initially, carriage ISR and stepper ISR can run independently. Config_carriage software ISR at the time of filling to_carriage structure checks if carriage is commanded to print a swath and media is already positioned to print it. If media is not properly aligned, config_carriage requests stepper hardware ISR to enable the carriage hardware ISR, otherwise carriage ISR is enabled to process current job in to_carriage structure. Similarly, to increase the printer throughput, carriage hardware ISR posts stepper software ISR config_stepper as soon as carriage starts decelerating at the end of swath.

Experimental Results

Table 3. Swath parameters used for testing carriage motion controller

Scan No.	Scan start position (inch)	Scan stop position (inch)	Velocity (ips)
1	0.75	1.75	20
2	2.50	4.50	20
3	5.00	7.00	30

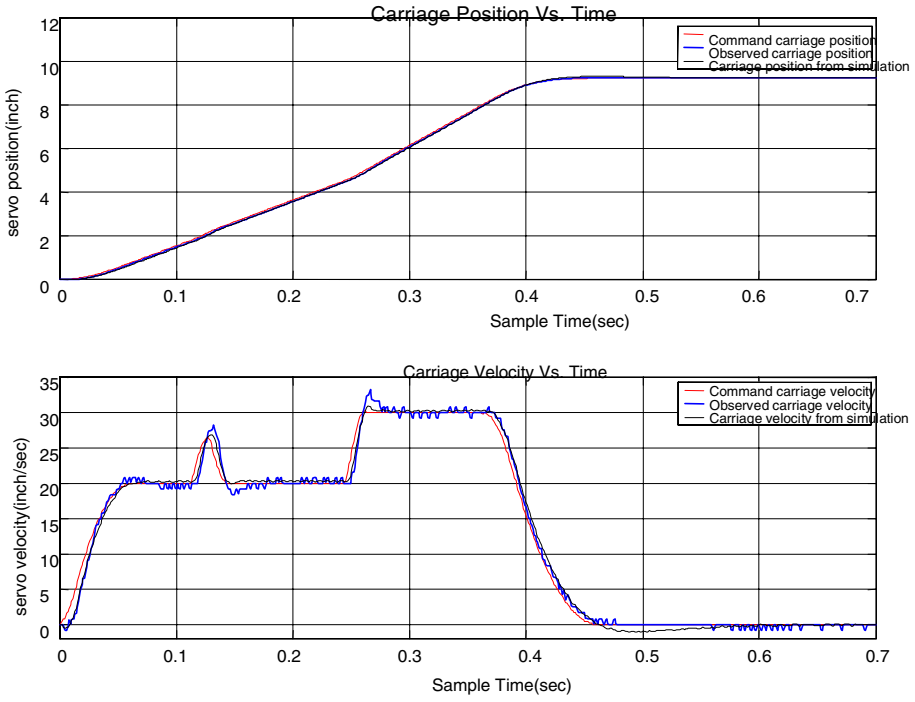

Figure 3. Comparison between experimental and simulated carriage position (inch) and velocity (inch/sec) for swath parameters in table 3

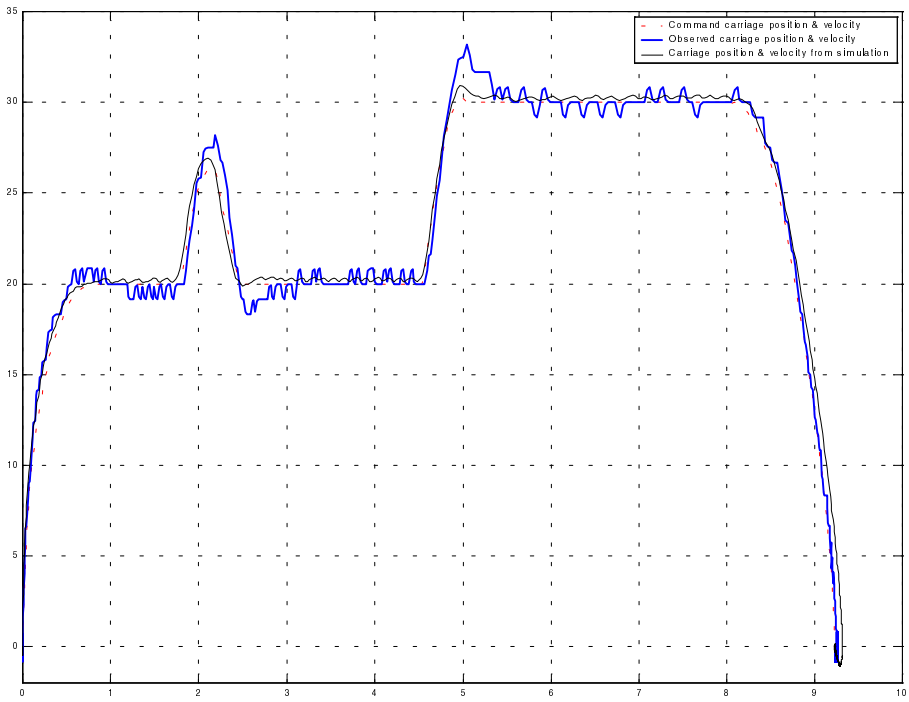

Figure 4. Experimental and simulated carriage velocity (inch/sec) vs. position (inch) for swath parameters in table 3

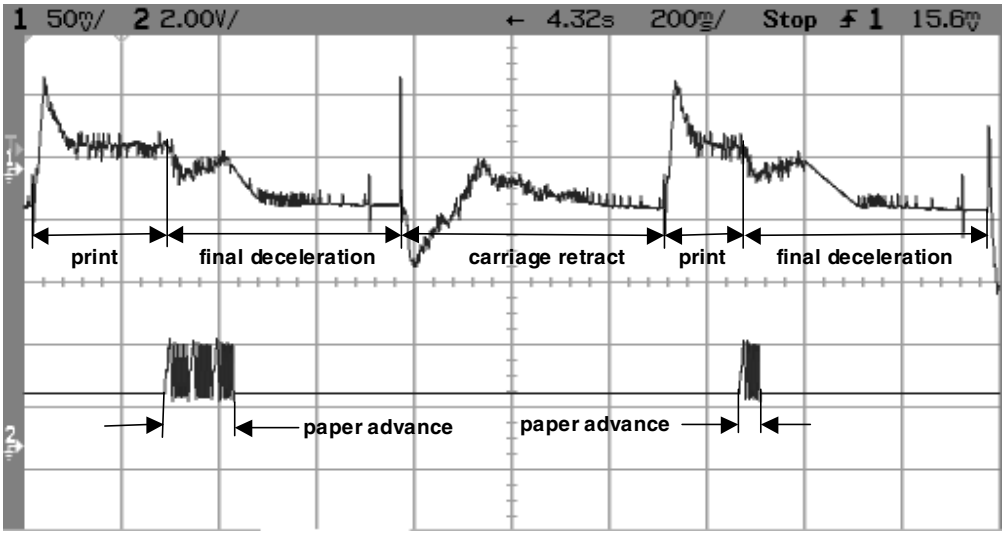

Figure 5. Carriage DC motor current and stepper motor current in phase A for printing a test pattern in uni-directional print mode at 300 dpi

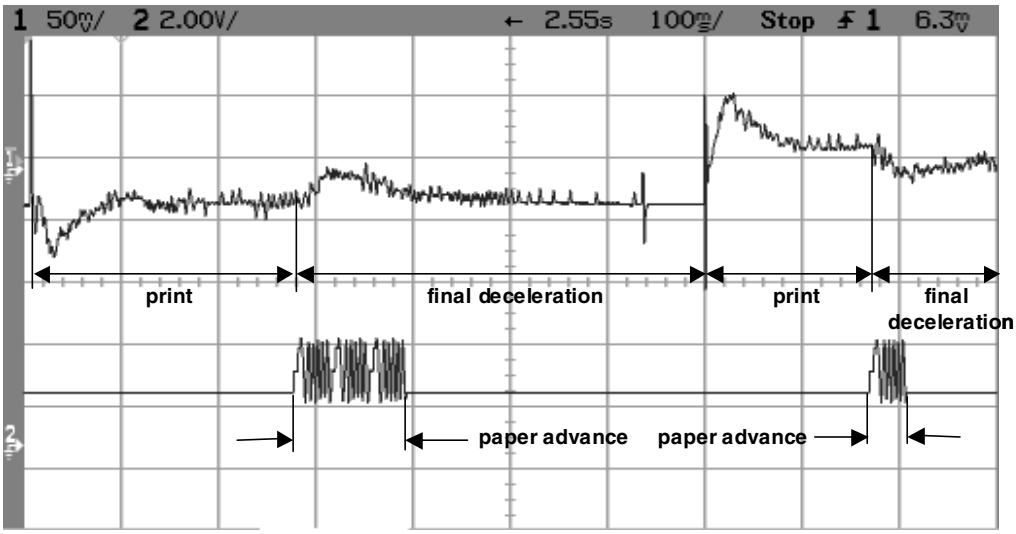

Figure 6. Carriage DC motor current and stepper motor current in phase A for printing a test pattern in bi-directional print mode at 300 dpi

Figure 3 and 4 show the experimental and simulated carriage position and velocity profiles for fixed-point implementation of carriage motion controller in figure 1. Representative swath parameters used for experiments are given in table 3, where cartridge speeds for black printing (table 2) are used for each scan. Carriage accelerates to an intermediate velocity of 28 ips before decelerating back to 20 ips printing velocity for scan 2, thus performing horizontal white-space skipping. After printing scan 2, carriage accelerates to 30 ips printing velocity for scan 3 within an approach distance of 0.5 inch in accordance with the principles of DPMC. In inkjet printing, phase plot of carriage velocity vs. carriage position (figure 4) is especially more relevant and important as carriage is required to reach a certain command velocity at a given scan start position. As evident, carriage speed and position requirements for given swath parameters are very well met for the proposed carriage motion controller (figure 1).

Figure 5 and 6 show the carriage DC motor current profile and stepper motor current profile in one of the phases for printing a test pattern at 300 dpi in uni-directional and bi-directional print modes respectively. The advantages of implementing motion control architecture in figure 2 are immediately apparent from the figures. Paper advancement for the next swath starts immediately after printing of current swath. This can be done only if motion control task can simultaneously process the next swath in the background and keep the paper advancement and carriage trajectory information ready in stepper_pipe and carriage pipe respectively when the carriage ISR finishes printing current swath in the foreground. Also due to this motion control framework, there is no time gap between carriage DC motor current for printing and retracting (in uni-directional print mode) or printing the next swath (bi-directional print mode), which helps to increase printer throughput.

Table 4 details the overall motion control system performance for printing a test page in uni-directional (row 1) and bi-directional print mode (row 2). Average execution time for carriage hardware ISR is 165 µsec and 40 µsec for stepper hardware ISR, well within their real-time deadline requirements. Motion control task's average computation time per swath depends upon the test page specifications. For bi-directional printing, carriage-retract may or may not be required and hence, corresponding computation time per swath is less in bi-directional printing.

Table 4. Motion control architecture performance for printing a test page at 600 dpi in uni-directional and bi-directional print mode

Print Time (sec)	Interrupt, Task	No. of Interrupts	Total DSP Clock Counts	Avg. Time (µsec)
75	Carriage	79215	2.13588×10^{9}	179.75
	Stepper	37239	2.13429×10^{8}	38.21
	MC Task	64	2.00502×10^{8}	20885.61
52	Carriage	46277	1.28309×10^{9}	184.84
	Stepper	38500	2.20753×10^{8}	38.23
	MC Task	64	1.35721×10^{8}	14137.62

Conclusion

In this paper, a motion control architecture for DPMC is proposed and implemented on TMS320C6211 DSP using Lexmark Z-52 printer engine and DSP/BIOS real-time operating system. The architecture utilizes various real-time services of DSP/BIOS to effectively manage DSP processing power, system memory and printer engine while increasing printer throughput and it can be easily extended to include image processing activities on the DSP. Experimental results show the efficacy of the proposed

motion control architecture and the control scheme for cartridge motion control.

References

1. M.K. Masten and I. Panahi, *Control Eng. Practice*, Vol. **5**, No. 4 (1997), pg. 449-458
2. J.L. Hennessy et.al., Computer Architecture: A Quantitative Approach, Morgan Kaufmann Publishers, San Francisco, CA USA, 2001
3. T.J. Radermacher and R.D. Mayo, Programmable Stepper Motor Controller and Method therefore, U.S. Patent: 6,124,696 (2000)
4. S.H. Lee, Technique for Controlling the Position of a Driving Motor and a Printhead, U.S. Patent: 6,000,869 (1999)
5. N. Nicoloff Jr., Mixed-Density Print Masking in a Mixed-swath-height Printer, U.S. Patent: 6,017,113 (2000)
6. S. Akula et.al., Advantages of DSPs for Ink Imaging Systems, TI World-wide RIP white paper, Dallas, TX USA, 2000
7. A.V. Deshpande, DSP Based Inkjet Printer System for Dynamic Print Mode Control, M.S. Thesis, Purdue University, 2001
8. DSP/BIOS Workshop – A Real-time Software Designer's Workshop, DSP/BIOS – Notes 2.0, Texas Instruments Inc. (2000)