JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 26, 1289-1305 (2010)

Hardware Context-Switch Methodology for Dynamically Partially Reconfigurable Systems*

TRONG-YEN LEE, CHE-CHENG HU, LI-WEN LAI AND CHIA-CHUN TSAI+
Department of Electronic Engineering and Institute of Computer and Communication Engineering
National Taipei University of Technology
Taipei, 106 Taiwan
E-mail: tylee@ntut.edu.tw
+*Department of Computer Science and Information Engineering*
Nanhua University
Chiayi, 622 Taiwan
E-mail: chun@mail.mhu.edu.tw

Nowadays, the hardware of field programmable gate arrays (FPGAs) can be reconfigured both dynamically and partially. A dynamically and partially reconfigurable system can share hardware contexts among various hardware tasks. However, such FPGA systems require much memory to save the hardware context. To solve this problem, this work proposes a methodology for switching hardware context in dynamically partially reconfigurable FPGA systems. This method can reduce the reconfiguration time and memory size of hardware context-switching by analyzing the characteristic of bit indexes and frame addresses. The experimental results show that the proposed method reduces 10.674% in hardware reconfiguration time, 47.47% in memory size, and 41.25% in resource overhead.

Keywords: context-switching, dynamically partially reconfigurable system, *Readback*, reconfiguration time, memory size, resource overhead

1. INTRODUCTION

The hardware in dynamically partially reconfigurable systems can normally be reconfigured in the run-time using the early access partial reconfiguration technique [1]. An engineer can design various hardware modules in a single FPGA (Field Programmable Gate Array), which dynamically changes hardware modules to suit the system requirements. The ability to release the hardware space, however, has yet to be developed. Restated, hardware should have context-switching capability, as does software in operating system (OS). This capability enables hardware to be arbitrarily swapped with other hardware in the reconfigurable system. Accordingly, the hardware resource can be replaced when other highly-prioritized tasks are being executed. Tasks can also be reloaded to complete unfinished work while a request is made for swap-in by context-switching. Consequently, the dynamic reconfigurable system has advantages of both hardware and software in short execution time, concurrency and flexibility [2, 3]. However, a major disadvantage of the reconfiguration system is the time delay associated with the hardware configuration [1, 3].

Received June 2, 2008; revised March 16 & June 1, 2009; accepted July 17, 2009.
Communicated by Chung-Ping Chung.
* This paper was partially supported by the National Science Council of Taiwan, R.O.C., under Grants No. NSC 98-2221-E-027-071.

The FPGA has a more flexible structure than application specific integrated circuit (ASIC) because the former can change the hardware task arbitrarily. FPGA systems can perform partial reconfiguration and share resources among hardware tasks. These capabilities can save and restore register information in the FPGA system, it referred to as hardware context-switching. To realize hardware context-switching, a mechanism to suspend and restart different spatial or temporal hardware tasks [4, 5] in FPGA reconfigurable systems is developed.

Dynamic switching or relocation in hardware design falls into one of two categories that are *reconfiguration-based* (*configuration port access*) [6-10] and *design-based* (*task specific access structure*) [11-14]. Reconfiguration-based dynamic hardware context switching utilizes the *Readback* function based on the FPGA configuration port. The advantage of the reconfiguration-based technique is that it does not require extra hardware circuits in the design of the hardware task. Restated, it does not require greater design effort nor consume FPGA resources. Another benefit is that the designer does not need knowledge of the internal behavior or communication infrastructure associated with task. Furthermore, if the task module is an IP core (Intellectual Property), then the hardware context can be completely saved and restored. However, the greatest disadvantage of the reconfiguration-based technique is the low data efficiency [1, 6, 14, 15] and huge memory requirement. These issues have been partially resolved in other works [6] and [7].

Specifically, Kalte *et al*. [6] only stores locations of the register that contain information on rows and columns in FPGA. The information on rows and columns is applied to calculate register location which then filter the current register information by *Readback* procedures. This method avoids the reading of the entire CLB (configurable logic block) column of the hardware during the *Readback* operation, because all state registers of a CLB column are only grouped into two frames (a CLB column has 22 frames) in Virtex-E/II/Pro series. With respect to memory width, Kalte *et al.* [6] used a 19-bit data width in the Virtex-E series to store content, location of a register and a 17-bit data width in the Virtex-II series because the method of Kalte *et al*. [6] depends on the maximum row and column length of the CLB in the Virtex series along with other important information. Clearly, when reconfiguration occurs very frequently, more memory is required to store hardware context. Therefore, reducing memory size is a very important issue.

The design-based technique requires an extra hardware interface compared to original hardware circuit, such as one associated with read/write interface [12-14] or software API (application programming interface) [11], to save and restore all state registers. Moreover, the designer must have detailed knowledge of the internal behavior and communication infrastructure associated with the task because each task does not have a standard or general interface. Therefore, the greatest disadvantage of the design-based technique is the increase in the required additional design effort and consumption of extra resources. Another drawback is that, if the task module as an IP core comes from a third party, adding an extra read/write interface is impossible, if no RTL model of the IP is available. However, the advantage of the design-based technique is the high data efficiency [6, 14, 15] because all context switching is performed by the designed hardware, such as the read/write interface. Moreover, only the required data rather than all of the frame data are read. Consequently, the reconfigurable computing in the design-based technique is faster than that of the reconfiguration-based technique.

A method of hardware context-switching is proposed to save and restore the significant frame and register data for hardware modules swap-in and swap-out in an FPGA. The significant frame and register data are stored in the same memory space [3]. The memory structure differs with that of our method proposed elsewhere [6]. The method proposed herein uses only an 11-bit data width to store content and the location of a register because only the frame address value, bit index value and others important information can be recorded (see subsection 3.3 for detail). Therefore, the proposed method reduces the memory size in hardware context-switching.

Readback is applied to save and restore hardware context when hardware modules are replaced. To increase the configuration/reconfiguration speed, the SelectMAP interface [16] is selected to perform *Readback* procedures. The proposed method is suitable for the Xilinx Virtex-II/Pro FPGA series. The Virtex-II/Pro FPGA platform is used to implement our proposed method.

The proposed method is a reconfiguration-based technique. Not only are the reconfiguration time and memory size compared with those of other reconfiguration-based technique, but also its resource overhead is compared with those of design-based technique.

The rest of this paper is organized as follows. Section 2 will describe the FPGA structure and bitstream format for Xilinx Virtex-II/Pro. Section 3 will describe the proposed method and architecture for hardware context-switching. Section 4 presents eight design examples to verify the accuracy of our proposed method, and estimate the performance in terms of reconfiguration time and memory size. The resource overhead is compared with those of other design techniques. Finally, section 5 draws conclusions.

2. FPGA STRUCTURE

The FPGA structures of current reconfigurable systems fall into three categories that are single context, multi-context and partially reconfigurable [15, 17]. To implement run-time reconfiguration on a single context FPGA, configuration data must be grouped into full contexts. Complete contexts are swapped in and swapped out of the hardware as needed. This swapping is a main cause of delay in hardware configuration. Therefore, two techniques have been applied to reduce the configuration delay these are the multi-context and partially reconfigurable structures. Multiple context configuration memory can be active at a different point in time. Partial configuration allows selective access to the configuration memory. The method proposed herein based on the partially reconfigurable FPGA structure.

The Xilinx Virtex-II/Pro FPGA structure comprises six column types. They are IOB, IOI, CLB, BRAM, BRAM_INT and GCLK, as shown in Fig. 1 [16, 18, 19]. Configuration memory frames are grouped into six column types and are organized in vertical rectangles with one-bit width on the entire FPGA column. A frame is the smallest addressable unit in configuration memory. The configuration memory is loaded to FPGA using a column-based approach [20]. The columns all have different amounts of column and frame lengths [16, 21], which depend on the FPGA series. In this work, Xilinx Virtex-II XC2V1000 and Virtex-II Pro XC2VP20 are selected for target platforms, which have 22 frames in a CLB (configurable logic block) column, and 106 and 146 words in a frame, respectively.

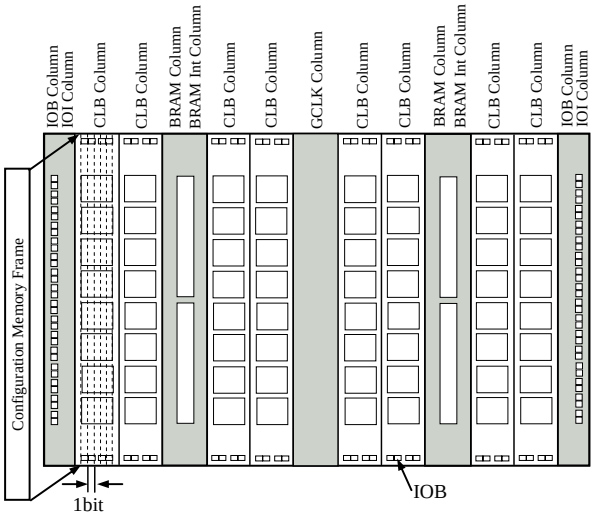


Fig. 1. Xilinx Virtex-II/Pro configuration column and frame overlay [16, 18, 19].

The behavior of an FPGA system is determined by a bitstream in the planning of configuration memory [22]. Therefore, bitstream must include information on addressing the frame and on the configuration register, which will be discussed in subsections 2.1 and 2.2.

2.1 Frame Addressing

The frame address in Xilinx Virtex-II/Pro FPGA has 32 bits. It is divided into the block address (BA), the major address (MJA), the minor address (MNA) and a byte number, as shown in Table 1 [23, 24]. The BA represents the column type in the configuration memory. The MJA specifies a particular column location within a configuration memory. The MNA is related to the location of the frames within a column. The byte number (nine LSBs) and five MSBs of the frame address must always be set to '0'.

Table 1. Frame address for Virtext-II FPGA.

		BA	MJA	MNA	Byte Number
Bits	31-27	26-25	24-17	16-9	8-0
Content	0	x	x	x	0

Fig. 2 shows the Xilinx Virtex-II/Pro configuration memory map. In Fig. 2, BA0 comprises GCLK, IOB, IOI and CLB configuration columns. BA1 and BA2 represent the BRAM and BRAM interconnect columns, respectively. Each column has an exclusive MJA. The symbol of *n* is the number of CLB columns and *m* is the number of BRAM columns. During configuration, frames are programmed according to in the order of the BA and MJA addresses (from left to right) [16, 18, 19].

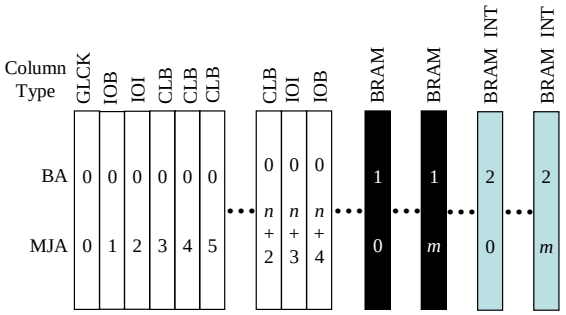


Fig. 2. Column-level configuration memory map [16, 18, 19].

2.2 Configuration Registers

In an FPGA, the configuration logic is controlled by the 32 bit configuration register. All configuration operations (configuration and *Readback*) are executed by reading or writing the configuration register data [16]. The application and function of these configuration registers are briefly described below.

The configuration register consists of the CRC (Cyclic Redundancy Check) register, command register (CMD), the frame address register (FAR), the frame data input register (FDRI), and the frame data output register (FDRO). The CRC register supports the error check function using a cyclic redundancy checking (CRC) algorithm [25, 26]. The CMD register is utilized not only to instruct the configuration control logic to handle the global signal, but also to perform other configuration functions. The commands SHUTDOWN, GCAPTURE, RCFG and START are supported in the CMD register. The FAR assigns an address to the configuration frame when data are written to FDRI or read from FDRO. The configuration data are written into configuration memory by shifting frame data to the FDRI register. The FDRO register can read, write, or capture configuration data from FPGA. The reading/writing operation is pipelined via the frame buffer. Therefore, the smallest reading/writing operation of both FDRI and FDRO requires two frames data [16].

3. PROPOSED HARDWARE CONTEXT-SWITCH

This section presents a hardware context-switching method for dynamically partially reconfigurable systems. The proposed method uses the SelectMAP interface and the *Readback* procedure to control FPGA configuration memory data and register information.

The *Readback* procedure can arbitrarily read frame data in configuration memory, including the contexts of registers and RAM. Therefore, the bitstream can be entirely or partially read by the *Readback* function. In the *Readback* operation, the state information is filtered and saved in the output of *Readback* stream. During the allocation step, the state information, such as the flip-flop data or the RAM contents in the FPGA structure, can be restored into suitable for register and RAM locations. The following subsections will introduce the proposed context-switch architecture and method.

3.1 Context-Switch Architecture

The proposed method for hardware context-switching can arbitrarily read back the bitstream via the configuration port of FPGA. Therefore, the hardware context can be extracted and saved using the *Readback* command if the hardware module is swap-in and swap-out, respectively.

Fig. 3 presents the proposed hardware context-switching architecture. The architecture consists of five modules: command ROM, database memory, state memory, controller and state filter. The command ROM module stores the commands according to which the controller communicates with SelectMAP. The database memory module saves frame addresses and the bit index parameters, both of which are used to calculate the location of hardware registers. The state filter module consists of the following; (1) "Bit index parameters RAM" temporarily stores the bit index parameter of the second frame, which obtained from the database memory module; (2) "Bit index calculation" unit is used to filter the register location within the frame. The state memory module saves register information from the state filter module before the hardware module is swapped out. The controller module handles all communication flows that include *Readback* and configuration via the SelectMAP interface.

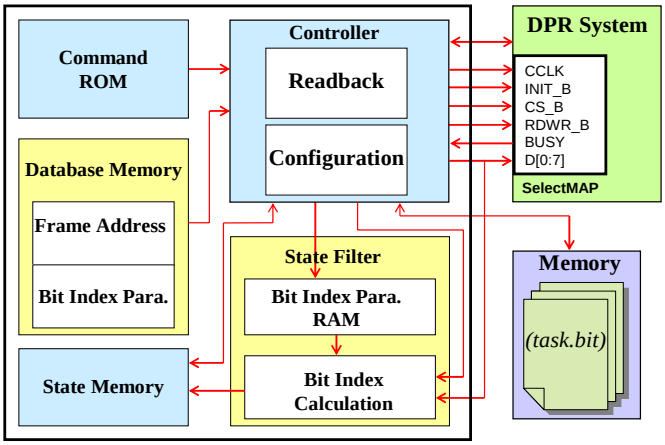


Fig. 3. Hardware context-switching architecture.

3.2 Database Memory and State Filter

The database must store all significant information for each hardware task. This important information is saved in a logic allocation file. Fig. 4 shows the generation flow of the task context database. A logic allocation file (task.ll) is generated by BitGen (bitstream generator) [16, 24], whose content includes the frame address and bit index of the used registers. The logic allocation file is parsed by an auxiliary tool to extract important information, and is established in the database memory. Afterward, the state filter extracts the locations of the registers within each frame. If the registers have the same frame address, only one of them will be saved in database memory to reduce the computation time

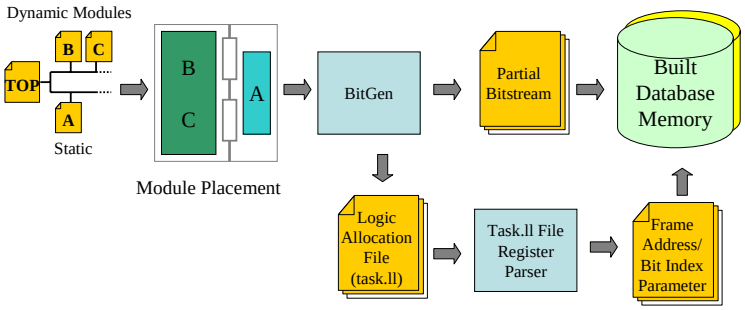


Fig. 4. Task context database generation flow.

Table 2. Frame address and bit index of slice in XC2V1000.

Frame Address (column) / Frame Bit Index (row)		C1			
		MJA = 3			
		X0		X1	
		XQ (hex)	YQ (hex)	XQ (hex)	YQ (hex)
		MNA = 1	MNA = 2	MNA = 1	MNA = 2
		00060200	00060400	00060200	00060400
R40	Y79(dec)	118	118	116	116
	Y78(dec)	158	158	156	156
R39	Y77(dec)	198	198	196	196
	Y76(dec)	238	238	236	236
⋮	⋮	⋮	⋮	⋮	⋮
R1	Y1(dec)	3238	3238	3236	3236
	Y0(dec)	3278	3278	3276	3276

and memory occupied by the frame addresses. The characteristic for bit index and frame address, and the equation for the state filter, are introduced below.

One CLB consists of four similar slices. One slice has two D-type registers (Flip-Flops) whose respective outputs are XQ and YQ [27], respectively. Each slice occupies two frame addresses in a single CLB column. Therefore, one CLB has eight registers [16]. The relationship among rows, columns, frame addresses and bit index parameters (*Y_row*) is presented in the logic allocation file, as shown in Table 2.

Each column (such as C1), *X_column*, can be separated into an even column (*X_even*) and an odd column (*X_odd*) [23]. Each row (such as R40), *Y_row*, can be further separated into an even row (*Y_even*) and an odd row (*Y_odd*). For example, the eight registers that occupy two frame addresses in C1 column are 00060200(hex) and 00060400(hex), as shown in Table 2. *X_even* and *X_odd* in Table 2 are utilized to derive the MJA of the frame address of the register, as shown in Eq. (1). The MNA depends on XQ and YQ. If XQ is used, then MNA equals 1. If YQ is used in the design, then MNA equals 2.

$$MJA = X\_enen/2 + 3 = (X\_odd - 1)/2 + 3 \quad (1)$$

Table 2 illustrates the rules for computing the bit indexes of registers in a frame. For example, eight registers in the C1R40 (column 1, row 40) CLB occupy two frame addresses in the sub-column (such as X0) and the adjacent sub-column (such as X1).

Moreover, the register data are distributed over bit indexes 118, 158, 116 and 156. In Eq. (2) is derived from these rules when the column number is odd. Similarly, Eq. (3) is derived if the column number is even. In Eqs. (2) and (3), *CLB_Rows* is the number of CLB rows.

$$fm\_bit\_idx\_Xodd = 116 + 40 \times (2CLB\_Rows - 1 - Y\_row) \quad (2)$$

$$fm\_bit\_idx\_Xeven = 118 + 40 \times (2CLB\_Rows - 1 - Y\_row) \quad (3)$$

According to Eqs. (2) and (3), the bit indexes are related to the number of rows of CLB. Therefore, instead of the bit index, the *Y_row* datum must be stored in database memory. The bit indexes of the registers within a frame are obtained by Eqs. (2) and (3). Afterward, the register data within the frame are stored in the state memory.

In the Xilinx Virtex-II/Pro series FPGA, the maximum numbers of CLB column are 104 (XC2V8000) and 94 (XC2VP100) respectively, and the maximum MJA are 106 (104 + 2) and 96 (94 + 2) respectively. Each CLB column has maximum numbers of rows of 112 (XC2V8000) and 120 (XC2VP100), and the maximum *Y_row* values are 223 and 239 respectively. Accordingly, the MJA is stored as 7 bits, and the MNA is stored as 2 bits in database memory. In addition, *Y_row* is stored as 8 bits, and whether a column is odd or even which is represented by a single bit.

3.3 Frame Addresses and Bit Indexes Addressing

The frame addresses and bit indexes are stored in the same part of memory, as shown in Fig. 5. The proposed method initially stores one frame address of a column (such as C1) to database memory. Afterward, bit index parameters are stored.

The database memory has a width of 11 bits to store the frame address and bit index, as shown in Table 3. The two most significant bits (*Bit_Share_Flag*) represent frame address or bit index format. In the frame address format, the MJA requires 7 bits because the maximum value is 106, while MNA uses 2 bits because the maximum value is 2, *Bit_Share_Flag* is permanently as "00". In the bit index format, *Y_row* requires 8 bits because the maximum value is 239, and *X_oe* uses 1 bit. Furthermore, the *Bit_Share_Flag* also represents the utilization condition of bit index of the registers. *X_oe* represents the *X* column attribute. Table 4 presents detailed information on *Bit_Share_Flag* and *X_oe*. Based on *X_oe* in Table 4, Eqs. (2) and (3) can be substituted into Eq. (4).

$$fm\_bit\_idx\_Xeven = 118 + 40 \times (2CLB\_Rows - 1 - Y\_row) - 2X\_oe \quad (4)$$

3.4 Controller

The controller supports *Readback* and configuration functions through the SelectMAP interface in context-switching architecture. The controller also controls the database memory, the state memory, the state filter, the bitstream memory and the command ROM.

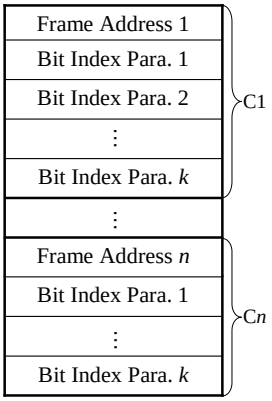


Fig. 5. Form of storage in database memory.

Table 3. Frame address and bit index parameter for addressing the register.

Frame Address										
Bit_Share_Flag		MJA							MNA	
10	9	8	7	6	5	4	3	2	1	0
0	0	X	X	X	X	X	X	X	X	X
Bit Index Parameter										
Bit_Share_Flag		X_oe	Y_row							
10	9	8	7	6	5	4	3	2	1	0
X	X	X	X	X	X	X	X	X	X	X

Table 4. Compatibility of bit index and frame address in database memory.

Bit_Share_Flag	Description
00	The frame address
01	The bit index occupied on the first frame address
10	The bit index occupied on the second frame address
11	The bit index occupied on the both frame addresses
X_oe	Description
0	X Column is in the even
1	X Column is in the odd

The command ROM saves operating commands to enable the controller to communicate with SelectMAP interface. To read data from configuration memory, the *Readback* procedure will give commands to address the frames. The *Readback* procedure consists of SHUTDOWN, CAPTURE and STARTUP sequences, as shown in Fig. 6. In the SHUTDOWN sequences, the hardware task must be suspended before data are captured. In the CAPTURE sequences, the states of the current register are captured. In the STARTUP sequences, the hardware task will be performed again [16] if the hardware task has been shut down by the SHUTDOWN sequences.

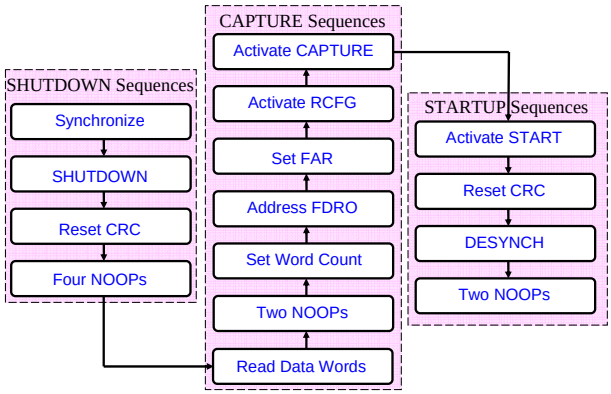
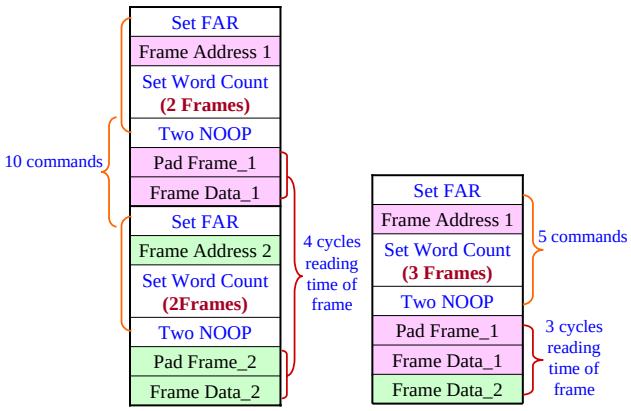


Fig. 6. *Readback* procedure.



(a) [6] and [7]. (b) Our proposed.

Fig. 7. Comparison of reading procedures of CLB column materials.

3.5 Performance Analysis

Kalte *et al*. [6] used a 17-bit data width in the Virtex-II series to store the frame address and bit index information of a register. This information is used to calculate frame addresses and bit indexes in the *Readback* operation. In another work [7], the number of bytes carries significant information about a frame, but depends on the size of the design and frame length of the target device. Therefore, two bytes are used to store significant information about the hardware because all state registers of a CLB column are located in only two frames in the Virtex-II/Pro series. The proposed method uses an 11-bit data width to store frame address and bit index information. Furthermore, it needs to calculate the bit index only while the *Readback* is running.

Fig. 7 compares the methods of Kalte *et al*. [6], Koester *et al*. [7] and this work on reading procedures of CLB column materials. According to the cited works, [6, 7], the reading of a single frame address requires one pad frame to flush the frame buffer. Therefore, when the *Readback* procedure reads two frame addresses, four cycles of reading

time per frame are used, as shown in Fig. 7 (a). However, the proposed method saves one single frame address per CLB column. Therefore, the three frames that are read out by the method include two frames of data and one pad frame, as shown in Fig. 7 (b). As mentioned above, the proposed method reduces the number of commands and memory size to below those in the other works [6, 7].

The reconfiguration time in an FPGA depends on the SelectMAP frequency and *Readback* procedures. In Eq. (5) yields overall command setting time for the *Readback* procedure, where the number of commands is the sum of N_{Shudow} (SHUTDOWN Sequences), N_{Cap_Rcfg}, $N_{Cap\_Far} \times N_{clb}$ (CAPTURE Sequences) and N_{Start} (STARTUP Sequences). N_{clb} is the number of used CLB columns. N_{Cap_Far} is the number of commands required to read a CLB column material (Fig. 7). Since one command has four bytes, the numerator in Eq. (5) must be multiplied by four. $T_{read\_frame\_data}$ is the sum of the reading times of the data of all frames and $T_{instruction}$. In the reading time for data of all frames, the term $3N_{clb}$ represents the three cycles of reading time per frame in the proposed method (Fig. 7 (b)). $N_{Byte/Frame}$ is the frame length of FPGA. Therefore, the total time for reading frame data is given by Eq. (6). A CLB of Xilinx Virtex-II/Pro has 22 frames ($22N_{clb}$). Therefore, Eq. (7) yields an estimate of the configuration time, T_{config}. The reconfiguration time ($T_{reconfig}$) in a task is the sum of $T_{read\_frame\_data}$ and T_{config}, given by Eq. (8).

$$T_{instruction} = \frac{4 \times [N_{Shudow} + N_{Cap\_Rcfg} + (N_{Cap\_Far} \times N_{clb}) + N_{Start}]}{f_{SelectMAP}} \tag{5}$$

$$T_{read\_frame\_data} = \frac{3N_{clb} \times N_{Byte/Frame}}{f_{SelectMAP}} + T_{instruction} \tag{6}$$

$$T_{config} = \frac{22N_{clb} \times N_{Byte/Frame}}{f_{SelectMAP}} \tag{7}$$

$$T_{reconfig} = T_{read\_frame\_data} + T_{config} = \frac{25N_{clb} \times N_{Byte/Frame}}{f_{SelectMAP}} + T_{instruction} \tag{8}$$

4. EXPERIMENTAL RESULTS

In this section, eight design examples, up-counter, LED display control, 16-bit divider, 32-bit divider, *multiple lights controller* (MLC), 32-bit *greatest common divisor* (GCD_{32}), *data encryption standard* (DES), and *data transformation component* (DCT) are used to demonstrate the benefits of the proposed method. The experiment is based on Xilinx XC2V1000 and XC2VP20 FPGA platforms. The XC2V1000 FPGA has a frame length of 424 bytes (= 106-word × 4 byte/word, $N_{Byte/Frame}$), and XC2VP20 FPGA has a frame length of 584 bytes (= 146-word × 4 byte/word, $N_{Byte/Frame}$). The SelectMAP frequency in these platforms is 50 MHz. The reconfiguration time and memory size are measured to evaluate the performance of hardware context-switching in the reconfiguration-based technique.

Table 5 compares the memory sizes in the work of Kalte *et al*. [6], that of Koester *et al*. [7], and the proposed method. Kalte *et al*. [6] and Koester *et al*. [7] used 17 bits and

Table 5. Comparison of memory sizes.

Methods / Design Examples (Total Flip-Flops)	Kalte [6]	Koester [7]	New method	Reduced
Up-Counter (28)	476 Bits	448 Bits	176 Bits	61.87%
16-bit Divider (40)	680 Bits	640 Bits	462 Bits	29.93%
LED Display Control (46)	782 Bits	736 Bits	440 Bits	41.98%
32-bit Divider (73)	1241 Bits	1168 Bits	748 Bits	37.84%
MLC (24)	408 Bits	384 Bits	209 Bits	47.17%
GCD_{32} (83)	1411 Bits	1328 Bits	715 Bits	47.74%
DES (84)	1428 Bits	1344 Bits	704 Bits	49.16%
DCT (1421)	24157 Bits	22736 Bits	8415 Bits	64.08%
Average	47.47%			

Reduced: % of overheads of memory size in the new method compared to [6, 7]. For example (MLC), {[(408 – 209)/408] + [(384 – 209)/384]}/2 = 47.17%

two bytes, respectively, to store parameters of the frame address and the bit index for each register. Our proposed method uses 11 bits to store parameters of each register. The experimental results show that the proposed method requires an average of 47.47% less memory.

Tables 6 and 7 compare the reconfiguration times of the method of Kalte *et al*. [6], that of Koester *et al*. [7], and the proposed method. Table 6 uses XC2V1000 FPGA and Table 7 uses XC2VP20 FPGA. According to "number of frame" and "number of frame data" in Tables 6 and 7, the methods in the other [6, 7] used four cycles of reading time per frame ($4N_{clb}$), but our proposed method only used three cycles reading time per frame ($3N_{clb}$). Clearly, the method proposed herein takes less time than the others [6, 7] in all design examples. According to "number of command" and "setting time of command" in Tables 6 and 7, the other methods [6, 7] used ten *Readback* operations, whereas the new method uses only five. Clearly, our method requires less processing than the others [6, 7] in all design examples. In Eqs. (6) and (8) yields the reading time for all frames and the reconfiguration time, respectively. Consequently, the proposed method requires 10.674% (= (10.739% + 10.608%)/2) less hardware reconfiguration time on average.

Section 1 pointed out that design-based reconfigurable computing is faster than the reconfiguration-based technique, that is, the latter is slower than the design-based technique in terms of reconfiguration time. However, the reconfiguration-based technique has a lower resource overhead than the design-based technique because the latter requires an extra read/write interface. Therefore, Table 8 compares the resource overheads of the reconfigurable-based technique and the design-based technique. In Table 8, although the number of flip-flops in Huang *et al*. [14] ranges from 31 to 2094 in all design examples, the number of flip-flops in the proposed method ranges only from 24 to 1421. Clearly, the proposed method consumes fewer resources than the method of Huang *et al*. [14] in all design examples. Consequently, the proposed method has on average a 41.25% lower hardware resource overhead.

These experimental results show that the reconfiguration time and memory size of our proposed hardware context-switching methodology are better than those of other

reconfiguration-based techniques. Moreover, the proposed reconfiguration-based technique uses fewer resources than the design-based techniques [14].

Table 6. Comparison of reconfiguration times for saving and restoring context.

Design Examples	UP Counter			LED Display Control			16-bit Divider			32-bit Divider		
Method / Items	Kalte [6]	Koester [7]	New method	Kalte [6]	Koester [7]	New method	Kalte [6]	Koester [7]	New method	Kalte [6]	Koester [7]	New method
Reading Frame	8	8	6	28	28	21	40	40	30	56	56	42
Reading Frame Data(Byte)	7008	3392	2544	24528	11872	8904	35040	16960	12720	49056	23744	17808
Number of Commands(Byte)	164	164	124	364	364	224	484	484	284	644	644	364
Time to Set Command(μs) ($T_{instruction}$)	3.28	3.28	2.48	7.28	7.28	4.48	9.68	9.68	5.68	12.88	12.88	7.28
Reading Time of Frame(μs) ($T_{read\_frame\_data}$)	143.44	71.12	53.36	497.84	244.72	182.56	710.48	348.88	260.08	994	487.76	363.44
Reconfig. Time(μs) ($T_{reconfig}$)	516.56	444.24	426.48	1803.76	1550.64	1488.48	2576.08	2214.48	2125.68	3605.84	3099.6	2975.28
Reduced Rate of $T_{reconfig}$	–	–	10.718%	–	–	10.744%	–	–	10.747%	–	–	10.749%
Average	10.739%											

Reduced Rate of $T_{reconfig}$: % of overheads of $T_{reconfig}$ in the new method compared to [6, 7]. For example (UP Counter), {[(516.56 – 426.48)/516.56] + [(444.24 – 426.48)/444.24]}/2 = 10.718%

Table 7. Comparison of reconfiguration times for saving and restoring context.

Design Examples	MLC			GCD_{32}			DES			DCT		
Method / Items	Kalte [6]	Koester [7]	New method	Kalte [6]	Koester [7]	New method	Kalte [6]	Koester [7]	New method	Kalte [6]	Koester [7]	New method
Reading Frame	12	12	9	16	16	12	24	24	18	36	36	27
Reading Frame Data(Byte)	14352	7008	5256	19136	9344	7008	28704	14016	10512	43056	21024	15768
Number of Commands(Byte)	204	204	144	244	244	164	324	324	204	444	444	264
Time to Set Command(μs) ($T_{instruction}$)	4.08	4.08	2.88	4.88	4.88	3.28	6.48	6.48	4.08	8.88	8.88	5.28
Reading Time of Frame(μs) ($T_{read\_frame\_data}$)	291.12	144.24	108	387.6	191.76	143.44	580.56	286.8	214.32	870	429.36	320.64
Reconfig. Time(μs) ($T_{reconfig}$)	1062	915.12	878.88	1415.44	1219.6	1171.28	2122.32	1828.56	1756.08	3182.64	2742	2633.28
Reduced Rate of $T_{reconfig}$	–	–	10.602%	–	–	10.606%	–	–	10.61%	–	–	10.613%
Average	10.608%											

Reduced Rate of $T_{reconfig}$: % of overheads of $T_{reconfig}$ in the new method compared to [6, 7]. For example (MLC), {[(1062 – 878.88)/1062] + [915.12 – 878.88)/915.12]}/2 = 10.602%

Table 8. Comparison of resource overhead.

Design Examples	MLC		GCD_{32}		DES		DCT	
Method / Item	Huang [14]	New method	Huang [14]	New method	Huang [14]	New method	Huang [14]	New method
Flip-Flops	31	24	169	83	207	84	2094	1421
Reduced rate of resource	23%		51%		59%		32%	
Average	41.25%							

5. CONCLUSION

A reconfiguration-based hardware context-switching method to reduce the memory size and reconfiguration time for dynamically partially reconfigurable systems is pre-

sented. The proposed method also reduces the reading time of one pad frame and the complexity of the setting of commands in the *Readback* operation. Moreover, the hardware context switching is suitable for all FPGAs of the Virtex-II/Pro series. The experimental results show that our proposed method reduces 47.47% memory size for saving hardware context and reduces 10.674% hardware reconfiguration time for dynamically partially reconfigurable systems on average over previous reconfiguration-based techniques. The proposed method has on average a 41.25% lower resource overhead than design-based techniques. Future work should attempt to determine the compatibility of FPGA structure among various series of FPGA. Efforts are underway to develop a hardware context-switching method appropriate for various Virtex series of FPGA by utilizing the compatibility of FPGA structure.

REFERENCES

1. J. Phillips, A. Sudarsanam, H. Samala, R. Kallam, J. Carver, and A. Dasu, "Methodology to derive context adaptable architectures for FPGAs," *IET Computers and Digital Techniques*, Vol. 3, 2009, pp. 124-141.
2. S. M. Scalera and J. R. Vazquez, "The design and implementation of a context switching FPGA," in *Proceedings of the IEEE Symposium on FPGAs for Custom Computing Machines*, 1998, pp. 78-85.
3. F. Berthelot, F. Nouvel, and D. Houzet, "A flexible system level design methodology targeting run-time reconfigurable FPGAs," *EURASIP Journal on Embedded Systems*, Vol. 2008, 2008, pp. 1-18.
4. C. H. Huang, S. S. Chang, and P. A. Hsiung, "Generic wrapper design for dynamic swappable hardware IP in partially reconfigurable systems," *International Journal of Electrical Engineering*, 2007, pp. 229-238.
5. C. H. Huang, K. J. Shih, C. S. Lin, S. S. Chang, and P. A. Hsiung, "Dynamically swappable hardware design in partially reconfigurable systems," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, 2007, pp. 2742-2745.
6. H. Kalte and M. Porrmann, "Context saving and restoring for multitasking in reconfigurable systems," in *Proceedings of the International Conference on Field Programmable Logic and Applications*, 2005, pp. 223-228.
7. M. Koester, M. Porrmann, and H. Kalte, "Relocation and defragmentation for heterogeneous reconfigurable system," in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms*, 2006, pp. 70-76.
8. H. Simmler, L. Levinson, and R. Männer, "Multitasking on FPGA coprocessors," in *Proceedings of the 10th International Conference on Field-Programmable Logic and Applications*, 2000, pp. 121-130.
9. S. A. Guccione, D. Levi, and P. Sundararajan, "JBits: A Java-based interface for reconfigurable computing," in *Proceedings of the 2nd Annual Military and Aerospace Applications of Programmable Devices and Technologies Conference*, 1999, http://www.citeulike.org/group/3294/article/812855.
10. H. Simmler, L. Levinson, and R. Männer, "Multitasking on FPGA coprocessors," in *Proceedings of the 10th International Workshop on Field Programmable Gate Arrays*, 2000, pp. 121-130.

11. K. Puttegowda, D. I. Lehn, J. H. Park, P. Athanas, and M. Jones, "Context switching in a run-time reconfigurable system," *The Journal of Supercomputing*, Vol. 26, 2003, pp. 239-257.
12. M. Ullmann, M. Huebner, B. Grimm, and J. Becker, "An FPGA run-time system for dynamical on-demand reconfiguration," in *Proceedings of the 18th International Parallel and Distributed Processing Symposium*, 2004, pp. 135-142.
13. J. Y. Mignolet, V. Nollet, P. Coene, D. Verkest, S. Vernalde, and R. Lauwereins, "Infrastructure for design and management of relocatable tasks in a heterogeneous reconfigurable system-on-chip," in *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, 2003, pp. 986-991.
14. C. H. Huang and P. A. Hsiung, "Software-controlled dynamically swappable hardware design in partially reconfigurable systems," *EURASIP Journal on Embedded Systems*, Vol. 2008, 2008, Article ID 231940.
15. K. Compton and S. Hauck, "Reconfigurable computing: a survey of systems and software," *ACM Computing Surveys*, Vol. 34, 2002, pp. 171-210.
16. Xilinx, Inc., "Virtex-II platform FPGA user guide," 2005, http://direct.xilinx.com/bvdocs/userguides/ug002.pdf.
17. Y. Birk and E. Fiksman, "Dnnamic reconfiguration architectures for multi-context FPGAs," *Computers and Electrical Engineering*, in Press, Corrected Proof, Available online 31 January 2009.
18. Xilinx, Inc., "Virtex-II Pro and Virtex-II Pro X FPGA user guide," 2007, http://www.xilinx.com/support/documentation/user_guides/ug012.pdf.
19. H. Kalte and M. Porrmann, "REPLICA2Pro: Task relocation by bitstream manipulation in Virtex-II/Pro FPGAs," in *Proceedings of the 3rd Conference on Computing Frontiers*, 2006, pp. 403-412.
20. C. Steiger, H. Walder, and M. Platzner, "Operating systems for reconfigurable embedded platforms: online scheduling of real-time tasks," *IEEE Transactions on Computers*, Vol. 53, 2004, pp. 1393-1407.
21. M. Hubner, C. Schuck, and J. Becker, "Elementary block based 2-dimensional dynamic and partial reconfiguration for Virtex-II FPGAs," in *Proceedings of the 20th International Parallel and Distributed Processing Symposium*, 2006, p. 8.
22. H. Kalte, G. Lee, M. Porrmann, and U. Rückert, "REPLICA: A bitstream manipulation filter for module relocation in partial reconfigurable systems," in *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, 2005, p. 151b.
23. Y. E. Krasteva, E. de la Torre, T. Riesgo, and D. Joly, "Virtex II FPGA bitstream manipulation: Application to reconfiguration control systems," in *Proceedings of the International Conference on Field Programmable Logic and Applications*, 2006, pp. 1-4.
24. T. Y. Lee, C. C. Hu, L. W. Lai, C. C. Tsai, and R. S. Hsiao, "Hardware context switching methodology for dynamically partially reconfigurable systems," in *Proceedings of the National Computer Symposium*, 2007, pp. 300-310.
25. Xilinx, Inc., "Virtex series configuration architecture user guide," 2004, http://www.xilinx.com/bvdocs/appnotes/xapp151.pdf.
26. Xilinx, Inc., "Virtex FPGA series configuration and Readback," 2005, http://www.xilinx.com/bvdocs/appnotes/xapp138.pdf.

27. B. Blodget, C. Bobda, M. Huebne, and A. Niyonkuru, "Partial and dynamically reconfiguration of Xilinx Virtex-II FPGAs," in *Proceedings of the International Conference on Field-Programmable Logic and its Applications*, 2004, pp. 801-810.

Trong-Yen Lee (李宗演) received his Ph.D. degree in Electrical Engineering from National Taiwan University, Taipei, Taiwan, R.O.C., in 2001. Since 2002, he has been a member of the faculty in the Department of Electronic Engineering, National Taipei University of Technology, where he is currently an Associate professor. His research interests include hardware-software codesign of embedded systems and FPGA systems design and VLSI design.

Che-Cheng Hu (胡哲郝) received his B.S. and M.S. degrees in Department of Electronic Engineering from Kun Shan University, Tainan, Taiwan, R.O.C., in 2003 and 2005, respectively. He is currently a Ph.D. candidate in the Graduate Institute of Computer and Communication Engineering, National Taipei University of Technology. His research interests include partial reconfigurable of FPGA systems and hardware-software codesign.

Li-Wen Lai (賴麗文) received her B.S. degree in Department of Electronic Engineering from Kun Shan University, Tainan, Taiwan, ROC, in 2005, and M.S. degree in Graduate Institute of Computer and Communication Engineering, National Taipei University of Technology, Taipei, Taiwan, ROC, in 2007.

Chia-Chun Tsai (蔡加春) received his B.S. degree in Industrial Education from National Taiwan Normal University, Taipei, Taiwan, R.O.C., in 1978, and both M.S. and Ph.D. degrees in Electrical Engineering from National Taiwan University, Taipei, Taiwan, R.O.C., in 1987 and 1991, respectively. From 1978 to 1989, he was a specialist teacher of electronic maintenance at Taiwan Provincial Hsin-Hua and Taipei Municipal Nan-Kang Technology High Schools, respectively. From 1989 to 2005, he was an Instructor, an Associate Professor, and then a Full Professor at the Department of Electronic Engineering, National Taipei University of Technology, Taipei, Taiwan, R.O.C. From 1994 to 1995 and 2000 to 2001, he was working on postdoctoral research at University of California, San Diego, and North Carolina State University, respectively, in U.S.A. Since 2005, he has been with the Department of Computer Science and Information Engineering, Nanhua University, Chiayi, Taiwan, R.O.C., where he is currently a Full Professor. His current research interests include VLSI design automation and mixed-signal IC design.